paloalto NETWORKS

TECHDOCS

Prisma Cloud Compute Edition Administrator's Guide

22.06 (EoL)

docs.paloaltonetworks.com

Contact Information

Corporate Headquarters:
Palo Alto Networks
3000 Tannery Way
Santa Clara, CA 95054
www.paloaltonetworks.com/company/contact-support

About the Documentation

- For the most recent version of this guide or for access to related documentation, visit the Technical Documentation portal docs.paloaltonetworks.com.
- To search for a specific topic, go to our search page docs.paloaltonetworks.com/search.html.
- Have feedback or questions for us? Leave a comment on any page in the portal, or write to us at documentation@paloaltonetworks.com.

Copyright

Palo Alto Networks, Inc.
www.paloaltonetworks.com

Last Revised

November 21, 2022

Table of Contents

Welcome

Edit on GitHub

Welcome to Prisma Cloud Compute Edition.

Prisma Cloud Compute is a cloud workload protection platform (CWPP) for the modern era. It offers holistic protection for hosts, containers, and serverless deployments in any cloud, and across the software lifecycle. Prisma Cloud Compute is cloud-native and API-enabled. It can protect all your workloads, regardless of their underlying compute technology or the cloud in which they run.

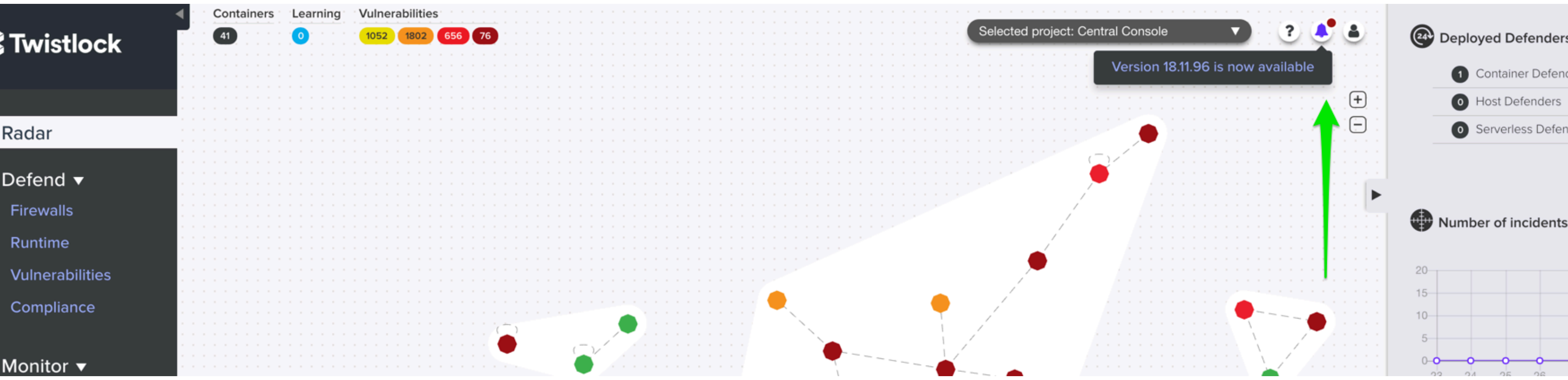
- Releases
- Getting started
- Product architecture
- Support lifecycle
- Security Assurance Policy on Prisma Cloud Compute
- Licensing
- Prisma Cloud Enterprise Edition vs Compute Edition
- Utilities and plugins

Releases

Edit on GitHub

In general, you should stay on the latest major release unless you require a feature or fix from a subsequent maintenance release. We recommend that you upgrade to new major releases as they become available. For more information, see the Prisma Cloud support lifecycle.

The bell icon in Console automatically notifies you when new releases are available:



Downloading the software

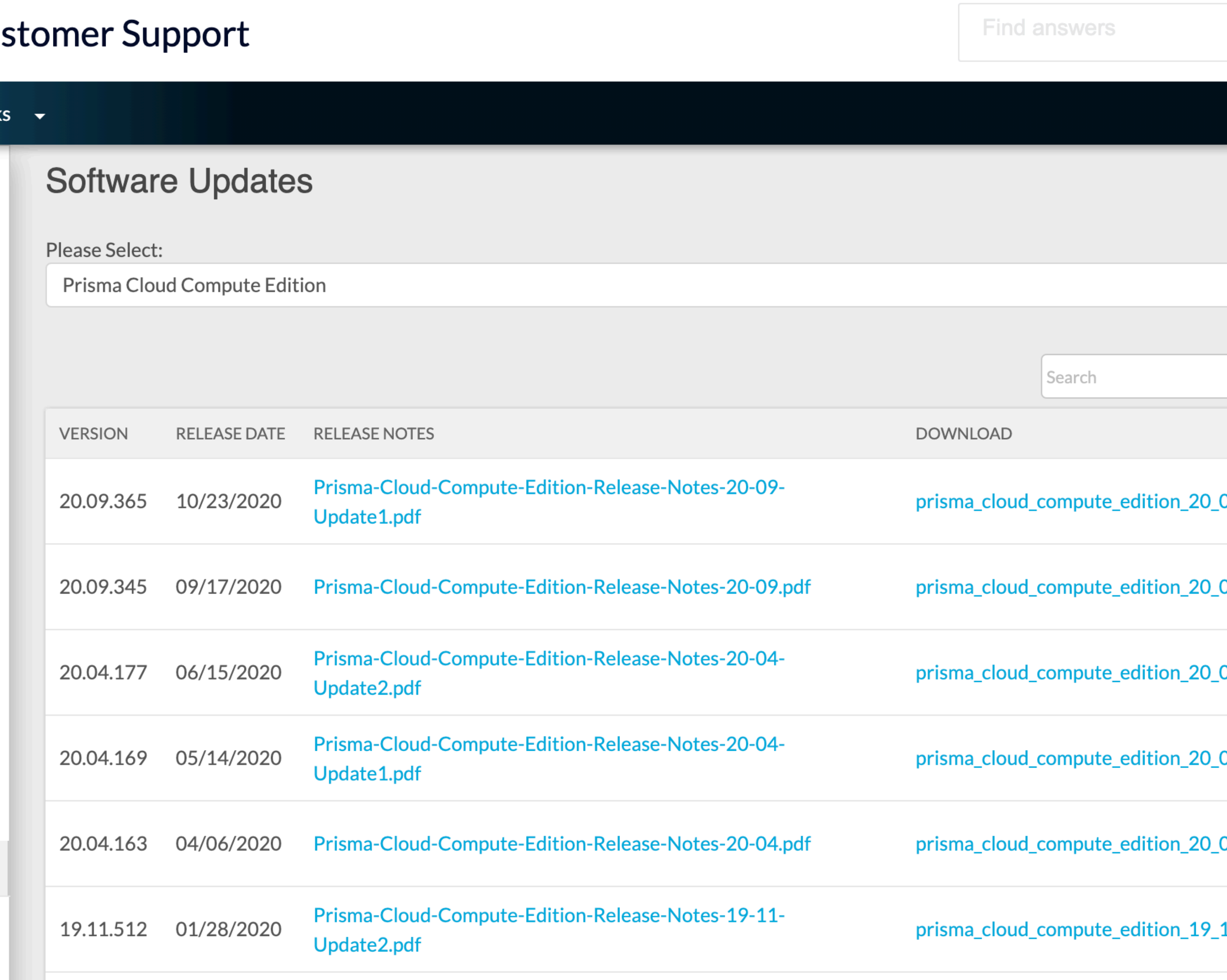
Download the software from the Palo Alto Networks Customer Support portal.

*If you don't see **Prisma Cloud Compute Edition** in the drop-down list, contact customer support. They'll send you a direct link to the download. We are currently working on fixing all accounts that have this issue.*

STEP 1 | Log into the Customer Support portal.

STEP 2 | Go to **Updates > Software Updates**.

STEP 3 | From the drop-down list, select **Prisma Cloud Compute Edition**. All releases available for download are displayed.



Downloading the software programmatically

Besides hosting the download on the Customer Support Portal, we also support programmatic download (e.g., curl, wget) of the release directly from our CDN. The link to the tarball is published in the release notes.

*If you don't see **Prisma Cloud Compute Edition** in the drop-down list, contact customer support. They'll send you a direct link to the download. We are currently working on fixing all accounts that have this issue.*

STEP 1 | Log into the Customer Support portal.

STEP 2 | Go to **Updates > Software Updates**.

STEP 3 | From the drop-down list, select **Prisma Cloud Compute Edition**. All releases available for download are displayed.

STEP 4 | Open the releases notes PDF.

RELEASE DATE	RELEASE NOTES	DOWNLOAD	SIZ
10/23/2020	Prisma-Cloud-Compute-Edition-Release-Notes-20-09-Update1.pdf	prisma_cloud_compute_edition_20_09_365.tar.gz	1.1 GB
09/17/2020	Prisma-Cloud-Compute-Edition-Release-Notes-20-09.pdf	prisma_cloud_compute_edition_20_09_345.tar.gz	1.0 GB
06/15/2020	Prisma-Cloud-Compute-Edition-Release-Notes-20-04-Update2.pdf	prisma_cloud_compute_edition_20_04_177.tar.gz	0.8 GB

STEP 5 | Scroll down to the release information to get the link.

20.09 Update 1 Release Notes

This section lists the issues addressed in this release.

Besides hosting the download on the Palo Alto Networks Customer Support Portal, we also support programmatic download (e.g., curl, wget) of the release directly from our CDN:

https://<LINK>

Improvements, fixes, and performance enhancements

- Adds support for running any minor version of Defender within a major release. In other words, given a major version of Console, Prisma Cloud supports all minor versions of Defender. For example, if

Open source components

Prisma Cloud includes various open source components, which may change between releases. Before installing Prisma Cloud, review the components and licenses listed in *twistlock-oss-licenses.pdf*. This document is included with every release tarball. Changes to components or licenses between releases are highlighted.

A full listing of the open source software and their licenses is also embedded in the Defender image. For example, to extract the listing from Defender running in a Kubernetes cluster, use the following command:

```
kubectl exec -ti -n twistlock <DEFENDER_POD> -- cat /usr/local/bin/
prisma-oss-licenses.txt
```

Code names

We often use code names when referring to upcoming releases. They're convenient to use in roadmap presentations and other forward-looking communications. Code names tend to persist even after a release ships.

Version to code name mapping

Version numbers indicate the date a release first shipped, along with the build number, as follows:

<YY>.<MM>.<BUILD-NUMBER>

For example, 22.01.840 is the Joule release, which first shipped in January, 2022.

The following table maps versions to code names. The table is sorted from newest (top) to oldest release.

Version	Code name
TBD(slated to ship in H1Y22)	Lagrange
22.06.XXX	Kepler
22.01.XXX	Joule
21.08.XXX	Iverson
21.04.XXX	Hamilton
20.12.XXX	Galileo
20.09.XXX	Fermat
20.04.XXX	Euler
19.11.XXX	Dirac

Getting started

Edit on GitHub

Welcome to the Prisma Cloud product documentation site. Start exploring how our technology can secure your environment.

Preinstall check

Ensure your environment meets the minimum system requirements.

Install the software

Download the latest Prisma Cloud release to your Prisma Cloud Console server or cluster controller. Then install Prisma Cloud using one of the dedicated guides.

Register your license key

Open a browser and navigate to the Prisma Cloud Console. Create an initial admin user, then enter your license key.

Your Prisma Cloud Console is available on https://<consoleServer>:8083

Install a test application

Use your own app or check out the Sock Shop.

Explore Prisma Cloud's core features

The following articles will get you started with Prisma Cloud's core features:

- Scan and monitor Docker registries
- Review image scan reports
- Create compliance rules
- Create vulnerability rules
- Learn about runtime protection
- Set up a cloud native application firewall
- Set up connection monitoring and enforcement

Product architecture

Edit on GitHub

Prisma Cloud offers a rich set of cloud workload protection capabilities. Collectively, these features are called *Compute*. Compute has a dedicated management interface, called *Compute Console*, that can be accessed in one of two ways, depending on the product you have.

- **Prisma Cloud Enterprise Edition** — Hosted by Palo Alto Networks. Prisma Cloud Enterprise Edition is a SaaS offering. It includes both the Cloud Security Posture Management (CSPM) and Cloud Workload Protection Platform (CWPP) modules. Access the Compute Console, which contains the CWPP module, from the **Compute** tab in the Prisma Cloud UI.
- **Prisma Cloud Compute Edition** - Hosted by you in your environment. Prisma Cloud Compute Edition is a self-hosted offering that's deployed and managed by you. It includes the Cloud Workload Protection Platform (CWPP) module only. Download the Prisma Cloud Compute Edition software from the Palo Alto Networks Customer Support Portal. Compute Console is delivered as a container image, so you can run it on any host with a container runtime (e.g. Docker Engine).

The following table summarizes the differences between the two offerings:

Capabilities	Prisma Cloud Enterprise Edition	Prisma Cloud Compute Edition
Management interface	Hosted by Palo Alto Networks (SaaS).	Deployed and managed by you in your environment (self-hosted).
Modules	CSPM and CWPP.	CWPP only.
Security agents	Deployed and managed by you.	Deployed and managed by you.
User management	Configure single sign-on in Prisma Cloud.	Configure single sign-on in Prisma Cloud Compute Edition. Compute Console exposes additional views for Active Directory and SAML integration when it's run in self-hosted mode.
Multi-tenancy	Supported by Palo Alto Networks Hub.	Supported by a feature called Projects. Projects is enabled in Compute Edition only. It's disabled in Enterprise Edition.

Accessing Compute in Prisma Cloud Enterprise Edition

In Prisma Cloud, click the **Compute** tab to access Compute Console. Think of Prisma Cloud as the outer management interface, and Compute Console as the inner management interface.

To access the Compute Console UI, users must have the Prisma Cloud (outer management interface) System Admin role. Access is denied to users with any other role.

The following screenshot shows the Prisma Cloud UI, or the so-called outer management interface. It can be accessed directly from the Internet. The format of the URL is:

```
https://app<opt-num>.<opt-region>.prismacloud.io
```

The following screenshot shows Prisma Cloud with the Compute Console open. Compute Console is the so-called inner management interface. Compute Console's GUI cannot be directly addressed in the browser. It can only be opened from within the Prisma Cloud UI. It's important to make the distinction between the inner and outer interfaces because a number of of Compute components directly address the inner interface, namely:

- Defender, for Defender to Compute Console connectivity.
- twistcli
- Jenkins plugin

- Compute API

You can find the address of Compute Console in Prisma Cloud under **Compute > Manage > System > Utilities**. The address for Compute Console has the following format:

```
https://<region>.cloud.twistlock.com/<customer>
```

Accessing Compute in Prisma Cloud Compute Edition

In Compute Edition, Palo Alto Networks gives you the management interface to run in your environment. In this setup, you deploy Compute Console directly. There's no outer or inner interface; there's just a single interface, and it's Compute Console. Compute Console's address, whether an IP address or DNS name, is used for all interactions, namely:

- GUI access from a web browser.

- Defender to Compute Console connectivity.
- twistcli
- Jenkins plugin
- Compute API

Support lifecycle

Edit on GitHub

Because the container ecosystem is rapidly evolving, understanding supportability policies is an important part of keeping your environment supportable and secure. This article describes not only the support policy for Prisma Cloud itself, but also for other software you may integrate it with.

You can always find the most up to date information on available releases on the Releases page.

Definitions

- **Major Releases (X.Y.z)** --

  Include significant new features and changes. These are also known as 'milestones' and include significant new functionality; they are released approximately every four months and include all applicable fixes made in previous releases. These are known by versions such as "20.12" and "21.04".

- **Maintenance Releases (x.y.Z)** --

  Also known as 'updates', these are released to correct specific problems in previous releases. They incorporate all applicable defect corrections made in prior Maintenance Releases. These are known by versions such as "21.04 Update 2".

- **End of Life (EOL)** --

  Versions that are no longer supported by Prisma Cloud. Updating to a later version is recommended.

- **Support** --

  Includes not only resolution of technical issues through interactive assistance, but also fixes delivered in maintenance releases to correct problems.

Prisma Cloud supported versions policy

Prisma Cloud has an 'n-2' support policy that means the current release ('n') and the previous two releases ('n-1' and 'n-2') receive support.

Note that in some cases, resolution of a problem in the n-1 or n-2 version may require upgrading to a current build. Prisma Cloud will make commercially reasonable efforts to work with customers that require porting fixes back to the n-1 or n-2 versions, but sometimes architectural changes are significant enough between versions that this is practically impossible without making the n-1 or n-2 versions essentially the same as the n version.

There will be version-specific API endpoints. With API versioning, as your Console is upgraded to newer versions, you can continue to use older versioned APIs with stability and migrate to newer version APIs at your convenience within the n-2 support lifecycle. As a best practice, update your scripts to use the version-specific API endpoints to ensure that your implementation is fully supported. For the version-specific APIs, you will have access to the API Reference and Release Notes documentation for changes or updates that may impact you.

Third party software

Customers use a diverse set of technologies in the environments that Prisma Cloud Compute protects, including host operating systems, orchestrators, registries, and container runtimes. As the vendors and projects responsible for these technologies evolve them, newly introduced versions and deprecated older versions can impact the scope of what Prisma Cloud supports. For example, Prisma Cloud cannot effectively support third-party software that the vendor (or project) itself no longer supports. Conversely, as new versions of 3rd party software are released, Prisma Cloud must comprehensively test them to be able to provide official support for them.

For each major and maintenance release of Prisma Cloud Compute, we begin testing by evaluating the versions of 3rd party software we list as officially supported in our system requirements. When new supported versions of this software are available, we perform our testing for the release using them. For example, if Red Hat were to release a new version of OpenShift before we begin testing an upcoming Prisma Cloud release, we'll include that new OpenShift release in our testing. If the new version of OpenShift is released after we've begun our testing, we'll instead do this validation in the subsequent Prisma Cloud release. Depending on where we are in the development cycle, this next release may be a maintenance release or the next major release. Typically, new 3rd party releases can be supported with no or minor changes in Prisma Cloud. However, there may be circumstances where a new version of 3rd party software introduces significant breaking changes that require more significant work within Prisma Cloud to maintain compatibility. In these cases, we'll update the system requirements page to clearly note this and will communicate a roadmap for supporting this software in a later release of Prisma Cloud.

While Prisma Cloud does not actively prevent interoperability with unsupported software, with each release we evaluate the versions of software supported by vendors and projects. As older versions are deprecated, Prisma Cloud support will similarly deprecate support for them as well.

Security Assurance Policy on Prisma Cloud Compute

Edit on GitHub

Prisma Cloud adheres to the guidelines outlined in the Palo Alto Networks Product Security Assurance Policy.

In accordance with this policy, Prisma Cloud Compute may have security releases outside of the regular release schedule.

Security releases are used for the sole purpose of remediating vulnerabilities that affect Prisma Cloud Compute, whether in its codebase or its dependencies.

We frequently analyze new vulnerabilities between releases to determine if any issue warrants a security release before the next scheduled release. This section outlines which issues are addressed in security releases.

With each new release of Prisma Cloud Compute, software dependencies are kept up-to-date to eliminate any known and confirmed vulnerabilities in third-party dependencies.

When new vulnerabilities are discovered in Prisma Cloud Compute dependencies after an official release, these vulnerabilities are addressed in the newer releases with the exceptions noted below.

Therefore, as a best practice, always upgrade to the latest release of Prisma Cloud Compute.

Vulnerability Triage

New releases of Prisma Cloud Compute are signed off with up-to-date dependencies. Vulnerabilities that meet the below criteria are analyzed between releases:

Vulnerabilities Analyzed

- Any vulnerability with severity high and above, regardless of having a fix or not.
- Any vulnerability with moderate severity when a fix is available.

Vulnerabilities Not Analyzed

- Any vulnerability with severity lower than high that does not have an existing fix.
- Any vulnerability with severity low or unimportant.

Exceptions

We also review vulnerabilities of any other severity when there is a known exploit or proof-of-concept that is affects Prisma Cloud Compute. Including product vulnerabilities identified during development, reported by customers or third-party researchers. To report a vulnerability in Prisma Cloud Compute, submit the vulnerability details to our PSIRT team.

Frequently Asked Questions

- Which Prisma Cloud Compute releases receive security updates?

Prisma Cloud has an 'n-2' support policy that means the current release ('n') and the previous two releases ('n-1' and 'n-2') receive support. Security fixes will be backported only for supported releases. End of Life (EOL) releases will not receive security fixes. For more information, see support lifecycle.

Are security fixes provided for both Prisma Cloud Enterprise and Compute editions?

Yes, security vulnerabilities are addressed in both the editions.

Do I have to upgrade my console/defender to get security updates?

If security fixes are released, you may be required to upgrade either or both the Console and Defender. We recommend that all security releases are adopted immediately. For the full details of which vulnerabilities were fixed in a release, refer to the

release notes.

What is the minimum severity for vulnerabilities to warrant a security release?

See triage criteria above.

What is the frequency of security releases for Prisma Cloud Compute?

There is no schedule for security releases. Security releases happens anytime a new vulnerability that meets the criteria outlined above is discovered in Prisma Cloud Compute.

Where do you take information on severity and fix details when triaging?

Console and Defender images are based on Red Hat Universal Base Images. For known vulnerabilities that are assigned a CVE identifier, we rely on severity ratings and fixes released by Red Hat. For zero-days or undocumented vulnerabilities (such as PRISMA-IDs), we rely on severity determined by our researchers.

A new vulnerability is affecting Prisma Cloud Compute, but a security release was not issued. If the vulnerability affects the latest release, meets the criteria for a security release outlined above, but it has not yet been addressed: please report it through to Palo Alto Networks Support or to PSIRT.

Licensing

Edit on GitHub

You must procure a license for each resource that Prisma Cloud protects and renew the license before the expiry term.

Licensing on Prisma Cloud uses a metering system based on credits used, and both Prisma Cloud Enterprise Edition (SaaS) and Prisma Cloud Compute Edition (self-hosted) are licensed with the same credits metering system.

Prisma Cloud Compute protects your hosts, containers, and serverless functions using a security agent called Defender, and using an agentless method. The number of credits you consume directly correlates with the type and mix of Defenders you deploy and the agentless security option.

Prisma Cloud also offers twistcli, a command-line configuration tool for which there is no additional credit usage. The credit usage is for the resources that are being protected using an agent or an agentless method.

Resource	Credits per resource	What's counted?
Hosts that don't run containers	1 credit	Host Defender
Hosts that are being scanned by Agentless scanning	1 credit	Host Agentless scan
Hosts that run containers	7 credits	Container Defender
Hosts that run applications	7 credits	Tanzu Application Service Defender
On-demand containers (such as AWS Fargate, Google Cloud Run)	1 credits	App-Embedded Defender
Serverless functions (such as AWS Lambda, Azure Functions, Google Cloud Functions)	1 credits per 6 defended functions	Defended functions: • Functions (only latest version) with a Serverless Defender - including Runtime & WAAS • Functions scanned for vulnerabilities and compliance (only latest version)

Resource	Credits per resource	What's counted?
Web Application and API Security (WAAS)	30 credits per Defender agent associated with protected web-application nodes (container/pod/host/AppID)	• Host Defender • Container Defender • App-Embedded Defender

Defender types

The type of Defender you deploy depends on the resource you're securing.

- **Host Defender** — Secures legacy hosts (Linux or Windows) that don't run containers.
- **Container Defender** — Secures hosts (Linux or Windows) that run containers. These types of hosts have a container runtime installed, such as Docker Engine or CRI-O. Container Defender protects both the underlying host and any containers it runs, and the license (7 credits) includes coverage for both. A container host consumes 7 credits whether it runs one container or a hundred containers.
- **Container Defender - App Embedded** — Secures containers which are run by a managed service, where the service provider maintains all infrastructure required to run the container, including the underlying host and container runtime. For this type of deployment, a Container App Embedded Defender is embedded into each container to be secured.
- **Serverless Defender** — Secures serverless functions. For this type of deployment, a Serverless Defender is embedded into each function to be secured.

Workload fluctuation

Prisma Cloud Compute Defenders are licensed on the honor system. License limits are not 'hard-enforced'. If you exceed your license count, Palo Alto Networks will notify you with a prominent banner at the top of the Prisma Cloud UI, but will neither disable any security functions nor prevent the deployment of additional Defenders. Protection is only disabled when your license expires.

Credit consumption is measured using a 30 day rolling average. To determine if you're within your licensed coverage, the rolling average is compared to the number of credits in your license.

Prisma Cloud samples the number of protected nodes hourly, then creates a daily average based on these samples. The preceding 30 daily averages are averaged to determine the credit consumption. If there is less than 30 days of data available, the average is calculated using the days available.

Example: Assume you've licensed 700 credits to cover 100 container hosts, and usage fluctuates from week to week:

Nov 1-7: Lower demand, uses 90 nodes (630 credits) Nov 8-15: Uses 100 nodes (700 credits) Nov 16-22: Uses 100 nodes (700 credits) Nov 23-30: High demand, uses 110 nodes (770 credits)

Even though you used 770 credits for a short period of time, you're still properly licensed because the 30 day rolling average is 700:

(630 + 700 + 700 + 770) / 4 = 700 credits

Example scenarios

For hosts and containers, the number of credits you need to procure depends on the number of Defenders you intend to deploy.

Example: Assume you have a Kubernetes cluster with 100 nodes (hosts). You deploy a Container Defender to each node. You would procure a license with 700 credits:

100 container hosts * 7 credits per container host = 700 credits

Serverless functions are licensed based on the number of defended functions, and averaged over the period of a month. Every 6 defended functions count as 1 credit. A defended function is either (a) a function with a Serverless Defender embedded or (b) a function scanned for vulnerabilities and compliance.

Example: Assume you have 180 functions, 180 functions are scanned for vulnerabilities and compliance while only 80 functions are defended in runtime (i.e., have a Serverless Defender embedded). Since we count each function only once:

180 defended functions / 6 credits per defended function = 30 credits

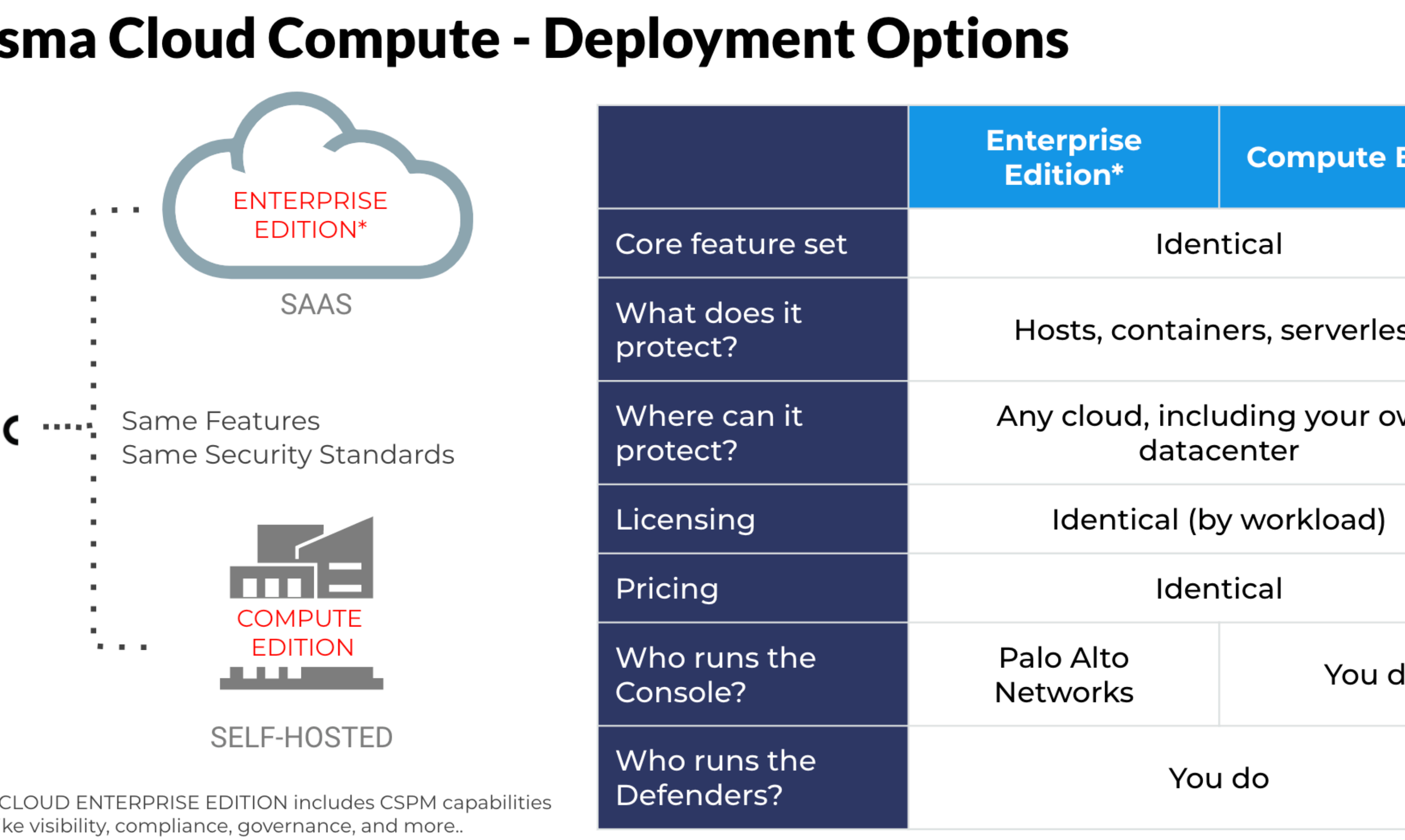
Example: Assume you have a web application running over 50 containers in a 5 node cluster. The containers running the images protected by WAAS rules are running on 2 out of the 5 nodes. You would procure a license with 60 credits.

2 Defenders protected nodes with WAAS protected containers * 30 credits per Defender = 60 credits

Prisma Cloud Enterprise Edition vs Compute Edition

Edit on GitHub

This article describes the key differences between Compute in Prisma Cloud Enterprise Edition and Prisma Cloud Compute Edition. Use this guide to determine which option is right for you.



How is Compute delivered?

Compute is delivered in one of two packages:

- **Prisma Cloud Enterprise Edition (SaaS)** — Single pane of glass for both CSPM (Cloud Security Posture Management) & CWPP (Cloud Workload Protection Platform). Compute (formerly Twistlock, a CWPP solution) is delivered as part of the larger Prisma Cloud system. Palo Alto Networks runs, manages, and updates Compute Console for you. You deploy and manage Defenders in your environment. You access the Compute Console from a tab within the Prisma Cloud user interface.

- **Prisma Cloud Compute Edition (self-hosted)**—Stand-alone, self-operated version of Compute (formerly Twistlock). Download the entire software suite, and run it in any environment. You deploy and manage both Console and Defenders.

What are the similarities between editions?

Both Enterprise Edition (SaaS) and Compute Edition (self-hosted) are built on the same source base. The Console container image we run for you in Enterprise Edition is the exact same container image we give to you in Compute Edition to run in your environment. We are committed to supporting and developing both versions without any feature divergence.

When should you use Enterprise Edition?

Prisma Cloud Enterprise Edition is a good choice when:

- You want a single platform that protects both the service plane (public cloud resource configuration) and the compute plane.
- You want convenience. We manage your Console for you. We update it for you. You get a 99.9% uptime SLA.

When should you use Compute Edition?

Prisma Cloud Compute Edition is a good choice when:

- You want full control over your data.
- You're operating in an air-gapped environment.
- You want to implement enterprise-grade multi-tenancy with one Console per tenant. For multi-tenancy, Compute Edition offers a feature called Projects.

What advantages does Prisma Cloud Enterprise Edition offer over Compute Edition?

When the Prisma Cloud CSPM and CWPP tools work together, Palo Alto Networks can offer economies of scale by sharing data (so called "data overlays"). The Prisma Cloud CSPM tool has always offered the ability to integrate with third party scanners, such as Tenable, to supplement configuration assessments with host vulnerability data. Starting with the Nov 2019 release of Enterprise Edition, the CSPM tool can utilize the host vulnerability data Compute Defender collects as part of its regular scans. Customers that have already licensed one workload for a host can leverage that single workload for configuration assessments by the CSPM tool, host vulnerability scanning (via Compute Defender), and host runtime protection (via Compute Defender).

Customers can expect additional "data overlays" in future releases, including better ways to gauge security posture with combined dashboards.

What are the differences between Prisma Cloud Enterprise Edition and Compute Edition?

There are a handful of differences between Enterprise Edition and Compute Edition. Consider these differences when deciding which edition is right for you.

Projects:

There is no support for Compute projects in the Prisma Cloud Enterprise Edition (PCEE). However, Enterprise Edition (EE) does offer alternatives that support Project's primary use cases.

The use case for projects is isolation, where each team has a dedicated Console so that other teams can't see each other's data. Prisma Cloud EE supports isolation with multiple independent Prisma Cloud tenants, one per team, with one Compute Console per tenant. Within a single PCEE tenant, Compute Console also offers isolation to data access based on cloud account filtering.

Contact Customer Success to create multiple tenants. Note that the license count shown in the Prisma Cloud UI is per tenant, not the aggregate across multiple tenants.

If you want to control tenant deployments yourself, use Compute Edition.

Syslog:

- Prisma Cloud Enterprise Edition Consoles do not emit syslog events for customer consumption. Since we operate the Console service for you, we monitor Console on your behalf.
- Prisma Cloud Enterprise Edition Defenders still emit syslog events that you can ingest. Syslog messages from Defender cover runtime and firewall events. For more details, see the article on logging.

User management:

- In Prisma Cloud Enterprise Edition, user and group management, as well as auth, is handled by the outer Prisma Cloud app in Enterprise Edition.
- As such, Compute Console in SaaS mode disables AD, OpenLDAP, and SAML integration in the Compute tab.
- In Prisma Cloud Enterprise Edition, you can assign roles to users to control their level of access to Prisma Cloud. These roles are mapped to Compute roles internally.
- For the CI/CD use case (i.e. using the Jenkins plugin or twistcli to scan images in the CI/CD pipeline), there's a new permission group called "Build and Deploy Security".

Assigned Collections:

- Prisma Cloud Enterprise Edition supports this via Resource Lists feature. Read more about assigning roles.

How do Defender upgrades work?

Upgrades work a little differently in each edition.

- **Prisma Cloud Enterprise Edition (SaaS)**—Consoles are automatically upgraded by PANW with notification posted in our status page at least 2 weeks in advance of upgrade. For more details, please refer to this article. Auto-upgrade function for Defenders is always turned ON ensuring that Defenders stay compatible with Console in each release.

- **Prisma Cloud Compute Edition (self-hosted)**—You fully control the upgrade process. When an upgrade is available, customers are notified via the bell icon in Console. Clicking on it directs you to the latest software download. Deploy the new version of Console first, then manually upgrade all of your deployed Defenders.

Can you migrate from Compute Edition to Enterprise Edition (SaaS)?

Yes.

See Migrate to SaaS.

Summary

The following table summarizes the key differences between Enterprise Edition (SaaS) and Compute Edition (self-hosted). For gaps, we provide a date we intend to deliver a solution.

Capability	Compute SaaS support
Projects	If you need Projects, use Compute Edition. Projects will not be ported to Prisma Cloud Enterprise Edition.
Syslog	Supported for Defenders only.
User management	Available centrally in the platform for Prisma Cloud Enterprise Edition.
Assigned collections	Available via Resource Lists
Defender backward compatibility	Yes
Compute Edition to Enterprise Edition migration	Available - Must go through Customer Success team.

Utilities and plugins

Edit on GitHub

All Prisma Cloud utilities and plugins can be downloaded directly from the Console UI They are also bundled with the release tarball you download from the Customer Support Portal

To download the utilities from Prisma Cloud Console, go to **Manage > System > Utilities**. From there, you can download:

- Jenkins plugin.
- Linux Container Defender image.
- twistcli for Linux, macOS, and Windows.

Install

Edit on GitHub

Prisma Cloud can be deployed to almost any environment. The guides in this section show you how to deploy Prisma Cloud to a variety of on-prem and public cloud environments.

- Getting started
- System Requirements
- Prisma Cloud container images
- Onebox
- Kubernetes
- OpenShift v4
- Console on Fargate
- Amazon ECS
- Alibaba Cloud Container Service for Kubernetes (ACK)
- Azure Kubernetes Service (AKS)
- Amazon Elastic Kubernetes Service (EKS)
- Google Kubernetes Engine (GKE)
- Google Kubernetes Engine (GKE) Autopilot
- IBM Kubernetes Service (IKS)
- Windows
- Defender types
- Cluster Context
- Install Defender

Getting started

Edit on GitHub

Prisma Cloud software consists of two components: Console and Defender. Install Prisma Cloud in two steps. First, install Console. Then install Defender.

Console is Prisma Cloud's management interface. It lets you define policy and monitor your environment. Console is delivered as a container image.

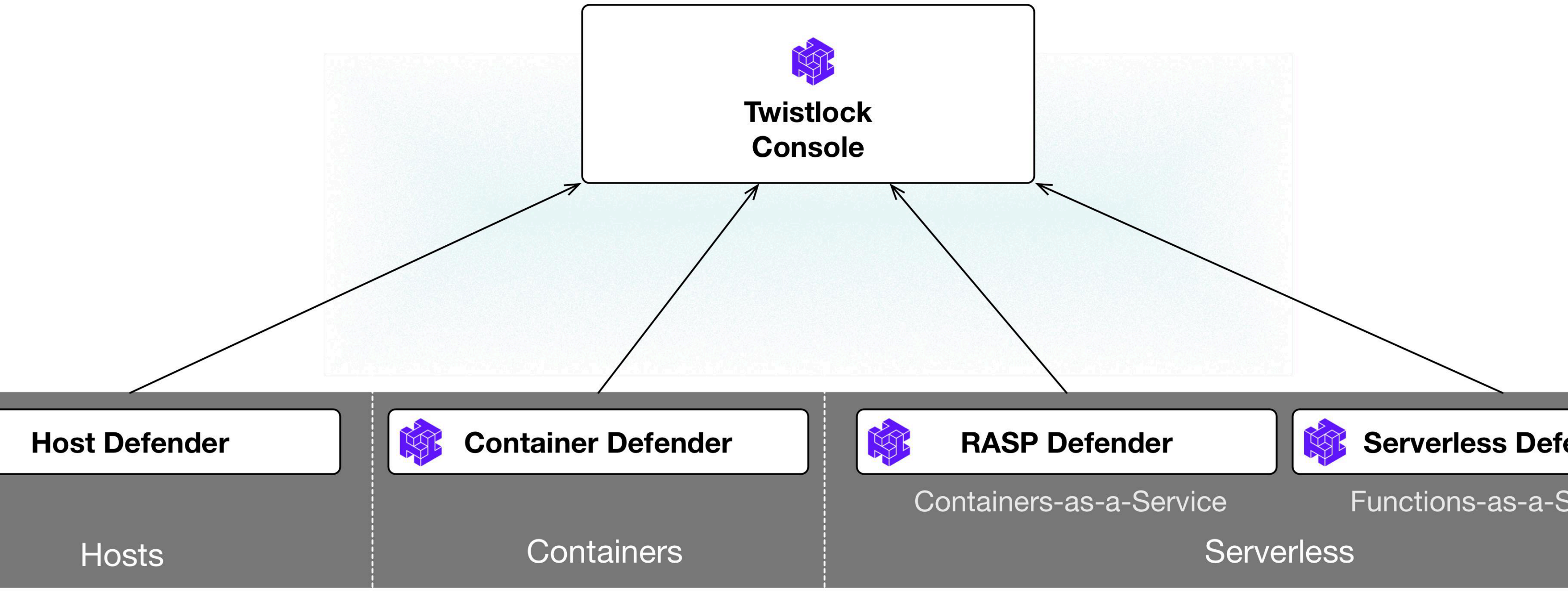
Defender protects your environment according to the policies set in Console. There are a number of Defender types, each designed to protect a specific resource type.

Install one Console per environment. Here, environment is loosely defined because the scope differs from organization to organization. Some will run a single instance of Console for their entire environment. Others will run an instance of Console for each of their prod, staging, and dev environments. Prisma Cloud supports virtually any topology.

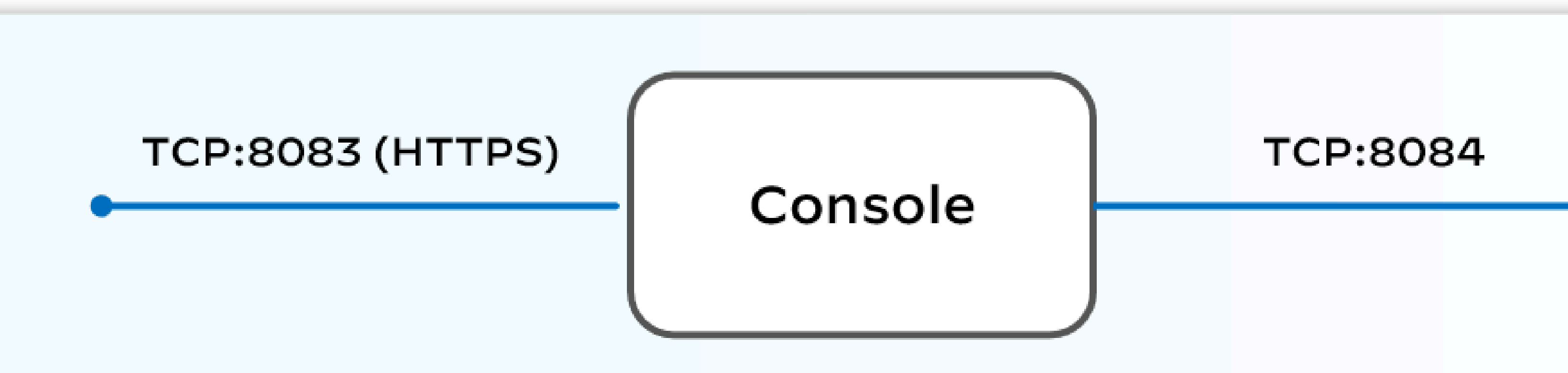
The primary concern for most customers getting started with Prisma Cloud is securing their container environment. To do this, install Container Defender on every host that runs containers. Container orchestrators typically provide native capabilities for deploying an agent, such as Defender, to every node in the cluster. Prisma Cloud leverages these capabilities to install Defender. For example, Kubernetes and OpenShift, offer DaemonSets, which guarantee that an agent runs on every node in the cluster. Prisma Cloud Defender, therefore, is deployed in Kubernetes and OpenShift clusters as a DaemonSet.

In this section, you'll find dedicated install guides for all popular container platforms. Each guide shows how to install Prisma Cloud for that given platform.

As you adopt other cloud-native technologies, Prisma Cloud can be extended to protect those environments too. Deploy the Defender type best suited for the job. For example, today you might use Amazon EKS (Kubernetes) clusters to run your apps. This part of your environment would be protected by Container Defender. Later you might adopt AWS Lambda functions. This part of your environment would be secured by Serverless Defender. Extending Prisma Cloud to protect other types of cloud-native technologies calls for deploying the right Defender type.



All Defenders, regardless of their type, report back to Console, letting you secure hybrid environments with a single tool. The main criteria for installing Defender is that it can connect to Console. Defender connects to Console via websocket to retrieve policies and send data. In Compute Edition (self-hosted), the Defender websocket connects to Console on port 8084 (configurable at install-time). The following diagram shows the key connections in Compute Edition.



Downloading the software

Prisma Cloud Compute Edition software can be downloaded from the Palo Alto Networks Customer Support portal. For more information, see here.

Install guides

Start your install with one of our dedicated guides.

Install procedure	Description
Onebox	Simple, quick install of Prisma Cloud on a single, stand-alone host. Installs both Console and Defender onto a host. Suitable for evaluating Prisma Cloud in a small, self-contained environment. You can extend the environment by installing Defender on additonal hosts.
Kubernetes	Prisma Cloud runs on any implementation of Kubernetes, whether you build the cluster from scratch or use a managed solution (also known as Kubernetes as a service). We've tested and validated the install on: • Amazon Elastic Kubernetes Service (Amazon EKS) • Azure Kubernetes Service (AKS) • Google Kubernetes Engine (GKE) • IBM Kubernetes Service (IKS) • Alibaba Cloud Container Service for Kubernetes In some cases, there is a dedicated section for installing on a specific cloud provider's managed solution. When there is no dedicated section, use the generic install method.
OpenShift 4	Prisma Cloud offers native support for OpenShift.
Amazon ECS	To install Prisma Cloud, deploy Console to your cluster with a task definition. Then configure the launch configuraration for cluster members to download and run Defenders, guaranteeing that every node is protected.
Windows	Install Defender on Windows hosts running containers. Defender is installed using a PowerShell script. Note that while Defenders can run on both Windows and Linux hosts, Console can only run on Linux. Windows Defenders are designed to interoperate with the Linux-based Console to send data and retrieve policy.

Encryption

All network traffic is encrypted with TLS (https) for user to Console communication. Likewise, all Defender to Console communication is encrypted with TLS (WSS).

The Prisma Cloud database is not encrypted at rest, however all credentials and otherwise secure information is encrypted with AES 256 bit encryption. If you require data at rest to be encrypted, then underlying persistence storage /var/lib/twistlock can be mounted with one of the many options that support this.

System Requirements

Edit on GitHub

Before installing Prisma Cloud, verify that your environment meets the minimum requirements.

For information about when Prisma Cloud adds and drops support for third party software, see our support lifecycle page.

Hardware

Prisma Cloud supports **x86_64** and **ARM64** architectures. Ensure that your systems meet the following hardware requirements.

Prisma Cloud Console Resource Requirements on x86_64

The Prisma Cloud Console supports running on x86_64 systems. Ensure your system meets the following requirements.

- For up to 1,000 Defenders connected:
  - 4 vCPUs
  - 8GB of RAM
  - 100GB of persistent storage
- For 1,001 - 10,000 Defenders connected
  - 8 vCPUs
  - 30GB of RAM
  - 500GB SSD of persistent storage
- More than 10,000 Defenders connected:
  - At least 8 vCPUs
  - At least 30GB of RAM
  - At least 500GB SSD of persistent storage
  - 4 vCPUS and 10GB of RAM for every additional 5,000 Defenders For example, 20,000 connected Defenders require a total of 16 vCPUs, 50GB of RAM and 500GB SSD of persistent storage.

The Prisma Cloud Console uses *cgroups* to cap resource usage. When more than 1,000 Defenders are connected, you should disable this cap using the *DISABLE_CONSOLE_CGROUP_LIMITS* flag in the *twistlock.cfg* configuration file.

Defender Resource Requirements

Each Defender requires 256MB of RAM and 8GB of host storage.

The Defender uses *cgroups* to cap resource usage at 512MB of RAM and 900 CPU shares where a typical load is ~1-5% CPU and 30-70MB RAM.

The Defender stores its data in the */var* folder. When allocating disk space for Defender, ensure the required space is available in the */var* folder. Defenders are designed to be portable containers that collect data. Any data that must be persisted is sent to the Prisma Cloud Console for storage. Defenders don't require persistent storage. If you deploy persistent storage for Defenders, it can corrupt Defender files.

If Defenders provide registry scanning they require the following resources:

- Defenders providing registry scanning--
- 2GB of RAM
- 20GB of storage
- 2 CPU cores Defenders that are part of CI integrations (Jenkins, twistcli) require storage space depending on the size of the scanned images. The required disk space is 1.5 times the size of the largest image to be scanned, per executor. For example, if you have a Jenkins instance with two executors, and your largest container image is 500MB, then you need at least 1.5GB of storage space: *500MB x 1.5 x 2*

Virtual Machines (VMs)

Prisma Cloud has been tested on the following hypervisors:

- VMware for Tanzu Kubernetes Grid Multicloud (TKGM)
- VMware for Tanzu Kubernetes Grid Integrated (TKGI)

Cloud Platforms

Prisma Cloud can run on nearly any cloud Infrastructure as a Service (IaaS) platform.

Prisma Cloud has been tested on the following services:

- Amazon Web Services (AWS)
- Google Cloud Platform
- IBM Cloud
- Microsoft Azure
- Oracle Cloud Infrastructure (OCI)

ARM Architecture Requirements

The following setups support Prisma Cloud on ARM64 architecture:

- Cloud provider
  - **AWS** Graviton2 processors
  - **GCP** GKE on ARM using the Tau T2A machine series
- Supported Defenders:
  - Orchestrator Defenders on AWS and GCP
  - Host Defenders including auto-defend on AWS
- The *twistcli* is supported on Linux ARM64 instances.

Learn more in the Supported Operating Systems on ARM64 and Supported Orchestrators on ARM64 sections.

The Prisma Cloud Console doesn't support running on ARM64 systems.

File Systems

When deploying Prisma Cloud Console to AWS using the EFS file system, you must meet the following minimum performance requirements:

- **Performance mode:** General purpose
- **Throughput mode:** Provisioned. Provision 0.1 MiB/s per deployed Defender. For example, if you plan to deploy 10 Defenders, provision 1 MiB/s of throughput.

Host Operating Systems

Prisma Cloud is supported on both x86_64 and ARM64

Supported Operating Systems on x86_64

Prisma Cloud is supported on the following host operating systems on x86_64 architecture:

Distro	Version
Amazon Linux 2	AMI name: amzn2-ami-hvm-2.0.20220426.0-x86_64-gp2 AMI ID: ami-06eecef118bbf9259
Bottlerocket OS	Tested version: 1.7.0 Containerd v1.5.11 Kernel version: 5.10.102 Kubelet version: v1.22.6-eks-b18cdc9 • *Vulnerability and compliance blocking policies are not supported on Bottlerocket.* • *RunC not supported.* • *Prevent is not supported on containerd runtime.* • *Compliance for containerd not supported.* • *Defenders must to be installed as privileged.*
CentOS	CentOS 7 CentOS 8
Debian	Debian 10 Debian 11
GCOOS	Container-Optimized OS on Google Cloud latest

Distro	Version
	GCOOS is purposefully minimalistic. It doesn't support installing new packages or writing new bins. Hence, Prisma Cloud's vulnerability detection on GCOOS only covers Docker and Kubernetes package binary detection. *Runtime prevent capability is supported only for DNS events. Other prevent capabilities are not supported.*
Red Hat Enterprise Linux	Red Hat Enterprise Linux 7, Red Hat Enterprise Linux 8
Red Hat Enterprise Linux CoreOS (RHCOS)	Red Hat Enterprise Linux CoreOS (RHCOS) versions included in OpenShift versions: 4.8, 4.9, and 4.10
SUSE	SLES-12 SP5 SLES 15 - Only Host Defenders are supported. SLES 15 SP1 - SP4 - Only Host Defenders are supported.
Ubuntu	Ubuntu 22.04 LTS Ubuntu 20.04 LTS Ubuntu 18.04 LTS
VMware	Photon OS 3.0 - Runtime scanning supported with kernel version >= 4.19.191-1 Photon OS 4.0 - Runtime scanning not supported *The following use features are currently not supported in Photon 3.0 and 4.0:* • *Detecting binaries without a package manager.* • *Event / incident for WildFire malware* • *SSHD application in host runtime events and empty SSH events on Host observations* • *Vulnerabilities in Layers view*
Windows	Windows Server 2016 Windows Server 2019 Long-Term Servicing Channel (LTSC) Windows on ARM64 architecture is not supported

Distro	Version
	The Console container must be run on a supported Linux operating system. Defender is supported on Windows Server 2016 (vulnerability and compliance scanning), and Windows Server 2019 (vulnerability scanning, compliance scanning, and runtime defense for containers).

Supported Operating Systems on ARM64

Prisma Cloud is supported on the following host operating systems on ARM64 architecture in AWS:

Distro	Version
Amazon Linux 2	AMI Image: amzn-ami-hvm-2018.03.0.20220315.0-x86_64-gp2 AMI ID: ami-0f7691f59fd7c47af
CentOS 8	AMI Image: CentOS-8-ec2-8.3.2011-20210302.1.arm64-a14b8c70-a48b-4a94-87b3-5dc93b3f6be8 AMI ID: ami-0446e1158fe3f255a
Debian 10	AMI Image: debian-10-arm64-20210208-542 AMI ID: ami-08b2293fdd2deba2a
Redhat Enterprise Linux (RHEL)	AMI Image: RHEL-8.4.0_HVM-20210504-arm64-2-Hourly2-GP2 AMI ID: ami-01fc429821bf1f4b4
Ubuntu 18	AMI Image: ubuntu/images/hvm-ssd/ubuntu-bionic-18.04-arm64-server-20211129 AMI ID: ami-0a940cb939351ccca Ubuntu 20 AMI Image: ubuntu/images/hvm-ssd/ubuntu-focal-20.04-arm64-server-20211129 AMI ID: ami-0b49a4a6e8e22fa16

Kernel Capabilities

Prisma Cloud Defender requires the following kernel capabilities. Refer to the the Linux capabilities man page for more details on each capability.

- *CAP_NET_ADMIN*
- *CAP_NET_RAW*
- *CAP_SYS_ADMIN*

- *CAP_SYS_PTRACE*
- *CAP_SYS_CHROOT*
- *CAP_MKNOD*
- *CAP_SETFCAP*
- *CAP_IPC_LOCK*

The Prisma Cloud App-Embedded Defender requires CAP_SYS_PTRACE only.

When running on a Docker host, Prisma Cloud Defender uses the following files/folder on the host:

- */var/run/docker.sock* — Required for accessing Docker runtime.
- */var/lib/twistlock* — Required for storing Prisma Cloud data.
- */dev/log* — Required for writing to syslog.

Docker Engine

Prisma Cloud supports only the versions of the Docker Engine supported by Docker itself. Prisma Cloud supports only the following official mainstream Docker releases and later versions.

- Community Edition (CE):
  - 18.06.1
  - 20.10.7
  - 20.10.13
- Enterprise Edition (EE):
  - 19.03.4
  - 19.03.8

The following storage drivers are supported: * *overlay2* * *overlay* * *devicemapper* are supported.

For more information, review Docker's guide to select a storage driver.

The versions of Docker Engine listed apply to versions you independently install on a host. The versions shipped as a part of an orchestrator, such as Red Hat OpenShift, might defer. Prisma Cloud supports the version of Docker Engine that ships with any Prisma Cloud-supported version of the orchestrator.

Container Runtimes

Prisma Cloud supports the following container runtimes:

Container runtime	Version
Docker	See the Docker section
cri-containerd	Native Kubernetes 1.23.8 (containerd 1.6.6)

Container runtime	Version
	Native Kubernetes 1.24.2 (containerd 1.6.6) Supported versions are listed in the orchestration section
CRI-O	OS 4.8 - CRIO version 1.21.3 OS 4.9- CRIO version 1.22.3 OS 4.10- CRIO version 1.23.1 K8s native - versions 1.23.8, 1.24.2 (x86_64 Arch)

Podman

Podman is a daemon-less container engine for developing, managing, and running OCI containers on Linux. The twistcli tool can use the preinstalled Podman binary to scan CRI images.

Podman v1.6.4, v3.0.1, v4.0.2

Helm

Helm is a package manager for Kubernetes that allows developers and operators to more easily package, configure, and deploy applications and services onto Kubernetes clusters.

Helm v3.9 is supported.

Orchestrators

Prisma Cloud is supported on the following orchestrators. We support the following versions of official mainline vendor/project releases.

Supported Orchestrators on x86_64

Orchestrator	Version
Azure Kubernetes Service (AKS)	Linux on AKS 1.22.6 (containerd 1.5.9+azure-2) Linux on AKS 1.23.5 (containerd 1.5.11+azure-2) Linux on AKS 1.24.3 (containerd 1.6.4+azure-4) Windows on AKS v1.22.6 (containerd 1.5.8+azure) Windows on AKS v1.23.3 (containerd 1.6.1+azure) Windows on AKS 1.24.3 (containerd 1.6.6+azure)
Bottlerocket	Bottlerocket OS 1.7.0 (aws-k8s-1.22) containerd 1.5.11 Kernel version: 5.10.102 Kubelet version: v1.22.6-eks-b18cdc9

Orchestrator	Version
	Bottlerocket OS 1.9.2 (aws-k8s-1.23) containerd 1.6.6+bottlerocket Kubelet v1.23.7-eks-4721010 *The following features are not supported.* • *RunC.* • *Prevent on the containerd runtime.* • *Compliance discovery for containerd.*
Elastic Container Service (ECS)	ECS Fargate Console: Fargate Platform 1.4.0 ECS x86 Console: AMI ID: ami-0002eba4f029226a3 ECS agent version: 1.62.2 Docker version: 20.10.13
Elastic Kubernetes Service (EKS)	EKS 1.23.7 EKS 1.21.9 (containerd 1.4.13) EKS 1.22.6 (containerd 1.4.6) EKS 1.22.9 (containerd 1.4.13) EKS 1.23.9 (containerd 1.6.6)
Google Kubernetes Engine (GKE)	GKE 1.21.11 (containerd 1.4.8) GKE 1.22.8 (containerd 1.5.4) GKE 1.23.7 (containerd 1.5.11) GKE 1.24.2 (containerd 1.6.6)
Google Kubernetes Engine (GKE) autopilot	GKE autopilot 1.21.11 (containerd 1.4.8) GKE autopilot 1.22.11 (containerd 1.5.13) Custom Compliance and Prevent (Runtime) are not supported on GKE autopilot.
Kubernetes (k8s)	k8s 1.23.8 (CRIO 1.23.3) k8s 1.24.2 (CRIO 1.24.1) k8s 1.25.0 (CRIO 1.25.0) k8s 1.23.8 (containerd 1.6.6)

Orchestrator	Version
	k8s 1.24.2 (containerd 1.6.6) k8s 1.25.0 (containerd 1.6.8) k8s 1.23.8 Docker 20.10.17
Lightweight Kubernetes (k3s)	k3s version: 1.21.7+k3s1 (containerd 1.4.12-k3s1) k3s version: 1.23.8+k3s2 (containerd 1.5.13-k3s1) k3s version: v1.23.6+k3s1 (containerd 1.5.11-k3s2) k3s version: v1.23.8+k3s1 (containerd 1.5.13-k3s1) k3s version: v1.24.4+k3s1 (containerd v1.6.6-k3s1)
OpenShift	OpenShift 4.8 (CRIO 1.21.3) OpenShift 4.9 (CRIO 1.22.3) OpenShift 4.10 (CRIO 1.23.1) Openshift 4.11 (CRIO 1.24.1)
Rancher Kubernetes Engine (RKE)	RKE2 v1.22.5+rke2r1 (containerd 1.5.8-k3s) RKE2 v1.24.4+rke2r1 (containerd 1.6.6-k3s1)
VMware Tanzu Application Service (TAS)	v2.11, v2.12, v2.13
VMware Tanzu Kubernetes Grid Integrated (TKGI)	TKGi version: TAS TKGI 1.14.2 Kernel Version: 4.15.0-184-generic containerd version: 1.6.0 OS version: Ubuntu 16.0.4.7 LTS

VMware Tanzu Kubernetes Grid Multicloud (TKGM)	TKG Multicloud 1.5.1
	vSphere 6.7U3
	• Kubernetes version v1.20.14+vmware.1 with:
	  • containerd version: 1.5.9
	  • OS-Image: VMware Photon 3 OS/Linux
	  • VMWare version: v1.20.14+vmware.1
	  • Kernel version :4.19.224-2.ph3

Orchestrator	Version

	• Kubernetes version v1.22.5+vmware.1 with:
	  • containerd version: 1.5.9
	  • OS-Image: Ubuntu 20.04.03 LTS
	  • VMWare version: v1.22.5+vmware.1
	  • Kernel version: 5.4.0-96-generic

Supported Orchestrators on ARM64

Prisma Cloud supports the official releases of the following orchestrators for the ARM64 architecture.

Orchestrator	Version
Elastic Container Service (ECS)	AMI name: amzn2-ami-ecs-hvm-2.0.20220831-arm64-ebs ECS agent 1.62.2 Docker 20.10.13
Elastic Kubernetes Service (EKS)	EKS v1.21.5 (containerd 1.4.6) EKS v1.23.9 (containerd 1.6.6)
GKE on ARM	GKE 1.23.5-gke.2400 (containerd 1.5.11) GKE 1.24.1-gke.1400 (containerd 1.6.2) *Defenders running in GKE on ARM don't support the following features:* • *Prevent for processes* • *Prevent for file system events* *While Prevent is not supported, runtime detection is supported for processes and file system events.*
Kubernetes with containerd	Kubernetes 1.23.5 (containerd 1.5.11)
Kubernetes with Docker	Docker Engine version: 20.10.14 API version:1.41 Go Version: go1.16.15
OpenShift	OpenShift 4.10 (CRI-O 1.23.1) OpenShift 4.11 (CRI-O 1.24.2)

Istio

Prisma Cloud supports Istio 1.13.4.

Jenkins

Prisma Cloud supports Jenkins 2.235.1, 2.319.1, and the 2.319.2 container version.

The Prisma Cloud Jenkins plugin supports Jenkins LTS releases greater than 2.319.1. For any given release of Prisma Cloud, the plugin supports those Jenkins LTS releases supported by the Jenkins project at the time of the Prisma Cloud release.

The Jenkins plugin is not supported on ARM64 architecture.

Image Base Layers

Prisma Cloud can protect containers built on nearly any base layer operating system. Comprehensive Common Vulnerabilities and Exposures (CVE) data is provided for the following base layers for all versions except EOL versions:

- Alpine
- Amazon Linux container image
- Amazon Linux 2
- BusyBox
- CentOS
- Debian
- Red Hat Enterprise Linux
- SUSE
- Ubuntu (LTS releases only)
- Windows Server

If a CVE doesn't have an architecture identifier, the CVE is related to all architectures.

Serverless Runtimes

Prisma Cloud can protect AWS Lambda functions at runtime. Prisma Cloud supports the following runtimes:

Serverless Runtimes Using Lambda Layers

- Node.js 12.x, 14.x
- Python 3.6, 3.7, 3.8, 3.9
- Ruby 2.7

Serverless Runtimes Using Manually Embedded Defenders in AWS

- C# (.NET Core 3.1)
- Java 8, 11

- Node.js 12.x, 14.x
- Python 3.6, 3.7, 3.8, 3.9
- Ruby 2.7

Prisma Cloud can also scan serverless functions for vulnerabilities and compliance benchmarks. Prisma Cloud supports the following runtimes for vulnerability and compliance scans in AWS Lambda, Google Cloud Functions, and Azure Functions:

Serverless Vulnerability and Compliance Scanning

- Node.js 12.x, 14.x
- Python 3.6, 3.7, 3.8, 3.9
- Ruby 2.7

Serverless WAAS Functions

- Java 11
- Node.js 12.x, 14.x
- Python 3.6, 3.7, 3.8, 3.9
- Ruby 2.7

Serverless Runtimes Using Manually Embedded Defenders in Azure

- C# 3 (.NET Core 3.1)
- C# 5 (.NET Core 4.0)
- C# 6 (.NET Core 4.0)

Go

Prisma Cloud can detect vulnerabilities in Go executables for Go versions 1.13 and greater.

Shells

For Linux, Prisma Cloud depends on the Bash shell. For Windows, Prisma Cloud depends on PowerShell.

The shell environment variable *DOCKER_CONTENT_TRUST* should be set to *0* or unset before running any commands that interact with the Prisma Cloud cloud registry, such as Defender installs or upgrades.

Browsers

Prisma Cloud supports the latest versions of Chrome, Safari, and Edge.

For Microsoft Edge, only the new Chromium-based version (80.0.361 and later) is supported.

Cortex XDR

Prisma Cloud Defenders can work alongside Cortex XDR agents. Currently, users need to manually add exceptions in Console for both agents to work together. In a future release, there will be out-of-the-box support for co-existence. Users can disable the Defender runtime defense when a Cortex XDR agent is present.

To allow for both the solutions to co-exist:

1. Add the Cortex agent as a trustable executable. For more information, see to Creating a trusted exeuctable.
2. Suppress runtime alerts from the Cortex agent by adding custom runtime rules that allow the Cortex agent process and file path.

Prisma Cloud container images

Edit on GitHub

Prisma Cloud images are built from the RedHat Universal Base Image 8 Minimal (UBI8-minimal) which is designed for applications that contain their own dependencies. With an active subscription or a valid license key, you can retrieve the images from a cloud registry. This option simplifies a lot of workflows, especially the install flow.

All builds, including private builds, are published to the registry. Private builds temporarily address specific customer issues. Unless you've been asked to use a private build by a Prisma Cloud representative during the course of a support case, you should only pull officially published builds.

You can optionally manage Prisma Cloud images in your own registry. You can push the Prisma Cloud images to your own private registry, and manage them from there as you see fit. The Console image is delivered as a *.tar.gz* file in the release tarball. The Defender image can be downloaded from Console, under **Manage > System > Utilities**, or from the Prisma Cloud API.

There are two different methods for accessing images in the cloud registry:

- Basic authorization.
- URL authorization.

The length of time that images are available on the cloud registry complies with our standard n-1 support lifecycle.

Retrieving Prisma Cloud images using basic auth

Authenticate using *docker login* or *podman login*, then retrieve the Prisma Cloud images using *docker pull* or *podman pull*. For basic authorization, the registry is accessible at *registry.twistlock.com*.

Image names contain a version string. The version string must be formatted as X_Y_Z, where X is the major version, Y is the minor version, and Z is the patch number. For example, 19.07.363 should be formatted as 19_07_363. For example:

registry.twistlock.com/twistlock/defender:defender_19_07_363.

Prerequisites:

- You have your Prisma Cloud access token.

STEP 1 | Authenticate with the registry.

```
$ docker (or podman) login registry.twistlock.com
Username:
Password:
```

Where **Username** can be any string, and **Password** must be your access token.

STEP 2 | Pull the Console image from the Prisma Cloud registry.

```
$ docker (or podman) pull registry.twistlock.com/twistlock/
console:console_<VERSION>
```

STEP 3 | Pull the Defender image from the Prisma Cloud registry.

```
$ docker (or podman) pull registry.twistlock.com/twistlock/
defender:defender_<VERSION>
```

Retrieving Prisma Cloud images using URL auth

Retrieve Prisma Cloud images with a single command by embedding your access token into the registry URL. For URL authorization, the registry is accessible at *registry-auth.twistlock.com*.

By embedding your access token into the registry URL, you only need to run *docker pull* or *podman pull*. The *docker login* or *podman login* command isn't required.

The format for the registry URL is: *registry-auth.twistlock.com/tw_<ACCESS-TOKEN>/<IMAGE>:<TAG>*

Image names contain a version string. The version string must be formatted as X_Y_Z, where X is the major version, Y is the minor version, and Z is the patch number. For example, 19.07.363 should be formatted as 19_07_363. For example:

registry.twistlock.com/twistlock/defender:defender_19_07_363.

Prerequisites:

- You have a Prisma Cloud access token.
- The Docker or Podman client requires that repository names be lowercase. Therefore, all characters in your access token must be lowercase. To convert your access token to lowercase characters, use the following command:

```
$ echo <ACCESS-TOKEN> | tr '[:upper:]' '[:lower:]'
```

STEP 1 | Pull the Console image from the Prisma Cloud registry.

```
$ docker (or podman) pull \
  registry-auth.twistlock.com/tw_<ACCESS-TOKEN>/twistlock/
console:console_<VERION>
```

STEP 2 | Pull the Defender image from the Prisma Cloud registry.

```
$ docker (or podman) pull \
  registry-auth.twistlock.com/tw_<ACCESS-TOKEN>/twistlock/
defender:defender_<VERSION>
```


Onebox

Edit on GitHub

Onebox provides a quick, simple way to install both Console and Defender onto a single host. It provides a fully functional, self-contained environment that is suitable for evaluating Prisma Cloud.

Install Prisma Cloud

Install Onebox with the *twistlock.sh* install script.

Prerequisites:

- Your host meets the minimum system requirements.
- You have a license key.
- Port 8083 is open. Port 8083 (HTTPS) serves the Console UI. You can configure alternative ports in *twistlock.cfg* before installing.
- Port 8084 is open. Console and Defender communicate with each other on this port.

STEP 1 | Download the latest Prisma Cloud release to the host where you'll install Onebox.

STEP 2 | Extract the tarball. All files must be in the same directory when you run the install.

```
$ mkdir twistlock
$ tar -xzf prisma_cloud_compute_<VERSION>.tar.gz -C twistlock/
```

STEP 3 | Configure Prisma Cloud for your environment.

Open *twistlock.cfg* and review the default settings. The default settings are acceptable for most environments.

If your Docker socket is in a custom location, update twistlock.cfg before continuing. By default, Prisma Cloud expects to find the Docker socket in /var/run/docker.sock. If it's not located there on your host, open twistlock.cfg in an editor, find the DOCKER_SOCKET variable, and update the path.

STEP 4 | Install Prisma Cloud.

```
$ sudo ./twistlock.sh -s onebox
```

- *-s* --

  Agree to EULA.
- *-z* --

  (Optional) Print additional debug messages. Useful for troubleshooting install issues.
- ***onebox*** --

  Install both Console and Defender on the same host, which is the recommended configuration. Specify *console* to install just Console.

STEP 5 | Verify that Prisma Cloud is installed and running:

```
$ docker ps --format "table {{.ID}}\t{{.Status}}\t{{.Names}}"
CONTAINER ID        STATUS              NAMES
764ecb72207e        Up 5 minutes
 twistlock_defender_<VERSION>
be5e385fea32        Up 5 minutes        twistlock_console
```

Configure Console

Create your first admin user and enter your license key.

STEP 1 | Open Prisma Cloud Console. In a browser window, navigate to 'https://<CONSOLE>:8083', where <CONSOLE> is the IP address or DNS name of the host where Console runs.

STEP 2 | Create your first admin user.

Consider using *admin* as the username. It's a convenient choice because *admin* is the default user for many of Prisma Cloud's utilities, including twistcli.

STEP 3 | Enter your license key.

Uninstall

Use the *twistlock.sh* script to uninstall Prisma Cloud from your host. The script stops and removes all Prisma Cloud containers, removes all Prisma Cloud images, and deletes the */var/lib/twistlock* directory, which contains your logs, certificates, and database.

STEP 1 | Uninstall Prisma Cloud.

```
$ sudo ./twistlock.sh -u
```

STEP 2 | Verify that all Prisma Cloud containers have been stopped and removed from your host.

```
$ docker ps -a
```

STEP 3 | Verify that all Prisma Cloud images have been removed from your host.

```
$ docker images
```

What's next?

Install Defender on each additonal host you want to protect.

Kubernetes

Edit on GitHub

This topic helps you install Prisma Cloud in your Kubernetes cluster quickly. There are many ways to install Prisma Cloud, but use this workflow to quickly deploy Defenders and verify how information is accessible from the Prisma Cloud Console. After completing this procedure, you can modify the installation to match your needs.

To install Prisma Cloud, you use the command-line utility called *twistcli*, which is bundled with the Prisma Cloud software. The process has the following steps to give you full control over the created objects.

1. The *twistcli* command-line utility generates YAML configuration files or Helm charts for the Prisma Cloud Console and Defender.
2. You create the required objects in your cluster with the *kubectl create* command.

To better understand clusters, read our cluster context topic.

You can inspect, customize, and manage the YAML configuration files or Helm charts before deploying the Prisma Cloud Console and Defender. You can place the files or charts under source control to track changes, to integrate them with Continuous Integration and Continuous Development (CI/CD) pipelines, and to enable effective collaboration.

To ensure a single copy of the Prisma Cloud Console is always available, the Prisma Cloud Console is created as a Kubernetes *Deployment*. Kubernetes deployments are also known as Kubernetes services. To ensure that a Prisma Cloud Defender instance runs on each worker node of your cluster, each Prisma Cloud Defender is deployed as a Kubernetes *DaemonSet*.

When a node goes down, the orchestrator can reschedule the Prisma Cloud Console on a different healthy node. To improve the availability of the Prisma Cloud Console, you must ensure that the orchestrator can run the Prisma Cloud Console on any healthy node. The default configuration files or charts ensure this capability. These default configuration files or charts enable the following features to ensure availability.

- **Deploy a persistent volume (PV), to enable Prisma Cloud Console to save the state.** This configuration ensures that no matter where Prisma Cloud Console runs, it has access to the state of the deployment. For persistent volumes to work, every node in the cluster must have access to the shared storage. Setting up a persistent volume can be easy or hard depending on the following factors.
  - What is your cloud provider?

    For example, Google Cloud Kubernetes Engine (GKE) offers persistent volumes out-of-the box with zero additional configuration required.
  - Is Kubernetes managed or unmanaged?

    If you deploy your clusters manually, you might need to configure a Network File System (NFS).
- **Expose the Prisma Cloud Console to the network through a load balancer.** A load balancer ensures that the Prisma Cloud Console is reachable regardless of where it runs in the cluster. The Prisma Cloud Console must be accessible in your deployment because it serves as a web interface and communicates policy to all the deployed Defenders.

Requirements

To deploy your Defenders smoothly, you must meet the following requirements.

- You have a valid Prisma Cloud license key and access token.
- You provisioned a Kubernetes cluster that meets the minimum system requirements and runs a supported Kubernetes version.
- You set up a Linux or macOS system to control your cluster, and you can access the cluster using the *kubectl* command-line utility.
- The nodes in your cluster can reach Prisma Cloud's cloud registry at *registry-auth.twistlock.com*.
- Your cluster can create PersistentVolumes and LoadBalancers from YAML configuration files or Helm charts.
- Your cluster uses any of the following runtimes. For more information about the runtimes that Prisma Cloud supports, see the system requirements.
  - Docker Engine
  - CRI-O
  - CRI-containerd

Required Permissions

- You can create and delete namespaces in your cluster.
- You can run the *kubectl create* command.

Required Firewall and Port Configuration

Open the following ports in your firewall.

Ports for the **Prisma Cloud Console**:

- Incoming: 8083, 8084
- Outgoing: 443, 53

Ports for the **Prisma Cloud Defenders**:

- Incoming: None
- Outgoing: 8084

Install Prisma Cloud

To use Prisma Cloud as part of your Kubernetes deployment, you need the *twistcli* command-line utility and the Prisma Cloud Defenders.

Use the *twistcli* command-line utility to install the Prisma Cloud Console and deploy the Defenders. The *twistcli* utility is included with every release, or you can download the utility separately. After completing this procedure, the Prisma Cloud Console and Prisma Cloud Defenders run in your Kubernetes cluster.

When you install Prisma Cloud on Amazon Elastic Kubernetes Service (EKS), Azure Kubernetes Service (AKS), or Alibaba Container Service with Kubernetes, additional configuration steps are required.

Download the Prisma Cloud Console

Download the Prisma Cloud software to any system where you run *kubectl* to manage your cluster.

STEP 1 | Download the current release.

STEP 2 | Create the *prisma_cloud* folder and unpack the release tarball.

```
$ mkdir prisma_cloud
$ tar xvzf prisma_cloud_compute_edition_<VERSION>.tar.gz -C
 prisma_cloud/
```

Install the Prisma Cloud Console

Install the Prisma Cloud Console and expose the service using a load balancer.

STEP 1 | On your cluster controller, navigate to the directory where you downloaded and extracted the Prisma Cloud release tarball.

STEP 2 | Generate a YAML configuration file for Console, where <PLATFORM> can be linux or osx.

The following command saves *twistlock_console.yaml* to the current working directory. If needed, you can edit the generated YAML file to modify the default settings.

```
$ <PLATFORM>/twistcli console export kubernetes --service-type
 LoadBalancer
```

If you're using Network File System version 4 (NFSv4) as the persistent storage in your cluster, use the nolock, noatime and bg mount options in your PersistentVolume custom resource definition (CRD). After generating the YAML file in the Prisma Cloud Console, add the mount options to your PersistentVolume CRD as follows.

```
apiVersion: v1
kind: PersistentVolume
metadata:
name: twistlock-console
labels:
app-volume: twistlock-console
annotations:
volume.beta.kubernetes.io/mount-options:
 "nolock,noatime,bg"
```

STEP 3 | Deploy the Prisma Cloud Console.

```
$ kubectl create -f twistlock_console.yaml
```

STEP 4 | Wait for the service to come up completely.

```
$ kubectl get service -w -n twistlock
```

Configure the Prisma Cloud Console

Create your first administrator and enter your license key.

STEP 1 | Get the public endpoint address for the Prisma Cloud Console.

```
$ kubectl get service -o wide -n twistlock
```

STEP 2 | Register a DNS entry for the external IP address of the Prisma Cloud Console. This procedure assumes the registered DNS name is *yourconsole.example.com*.

STEP 3 | If you need to secure the Prisma Cloud Console communication with TLS, set up a custom certificate. (Optional)

STEP 4 | Open a browser window, and navigate to the Prisma Cloud Console. By default, the Prisma Cloud Console is served with the HTTPS protocol on port 8083. You can go to https://yourconsole.example.com:8083 to access the Prisma Cloud Console.

STEP 5 | Create your first administrator.

STEP 6 | Enter your Prisma Cloud license key.

STEP 7 | The Defender communicates with the Prisma Cloud Console using TLS. Update the list of identifiers in the Prisma Cloud Console certificate that Defenders use to validate the identity of the Prisma Cloud Console.

1. Go to **Manage > Defenders > Names**.
2. In the **Subject Alternative Name** table, click **Add SAN**, then enter the Prisma Cloud Console IP address or domain name. Enter the *yourconsole.example.com* domain name. Any Defenders deployed outside the cluster can use this domain name to connect to the Prisma Cloud Console.
3. In the **Subject Alternative Name** table, click **Add SAN** again, then enter *twistlock-console*. Any Defenders deployed in the same cluster as the Prisma Cloud Console can use the *yourconsole.example.com* domain name to access the Prisma Cloud console.

The service name of the Prisma Cloud Console is twistlock-console, but that name is not the same as the pod's name, which is twistlock-console-XXXX.

Install the Prisma Cloud Defender

To install the Prisma Cloud Defender, deploy the Defenders as *DaemonSet* custom resources. This approach ensures that a Defender instance runs on every node in the cluster. To deploy the Prisma Cloud Defender, use a macOS or Linux cluster controller with *kubectl* enabled and follow these steps:

1. Use the *twistcli* command-line utility to generate the *DaemonSet* YAML configuration file for the Defender.
2. Deploy the generated custom resource with *kubectl*.

This approach is called declarative object management. It allows you to work directly with the YAML configuration files. The benefit is that you get the full source code for the custom resources you create in your cluster, and you can use a version control tool to manage and track modifications. With YAML configuration files under version control, you can delete and reliably recreate *DaemonSets* in your environment.

If you don't have *kubectl* access to your cluster, you can deploy Defender *DaemonSets* directly from the Console UI.

This procedure shows you how to deploy Defender *DaemonSets* using the *twistcli* command-line utility and declarative object management. You can also generate the installation commands using the Prisma Cloud Console UI under **Manage > Defenders > Deploy > Defenders**. Installation scripts are provided for Linux and MacOS workstations. Use the *twistcli* command-line utility to generate the Defender *DaemonSet* YAML configuration files from Windows workstations. Deploy the custom resources with *kubectl* following this procedure.

STEP 1 | Generate the DaemonSet custom resource for the Defender.

1. Go to **Compute > Manage > Defenders > Deploy > Defenders**.
2. Select **Orchestrator**.
3. Select **Kubernetes** from **Step 2: Choose the orchestrator type**.
4. Copy the hostname from **Step 3: The name that Defender will use to connect to this Console**.

STEP 2 | Generate the *defender.yaml* file using the following *twistcli* command with the described parameters.

For Defenders deployed in the cluster where Console runs, specify the service name of the Prisma Cloud Console, for example *twistlock-console*.

```
$ <PLATFORM>/twistcli defender export kubernetes \
  --user <ADMIN_USER> \
  --address <PRISMA_CLOUD_COMPUTE_CONSOLE_URL> \
  --cluster-address <PRISMA_CLOUD_COMPUTE_HOSTNAME>
```



```
  --cri
```

- <PLATFORM> can be *linux*, *osx*, or *windows*.
- <ADMIN_USER> is the name of a Prisma Cloud user with the System Admin role.
- <PRISMA_CLOUD_COMPUTE_CONSOLE_URL> specifies the address of the Prisma Cloud Compute Console.
- <PRISMA_CLOUD_COMPUTE_HOSTNAME> specifies the address Defender uses to connect to Prisma Cloud Console. You can use the external IP address exposed by your load balancer or the DNS name that you manually set up.

> - *For provider managed clusters, Prisma Cloud automatically gets the cluster name from your cloud provider.*
> - *To override the cluster name used that your cloud provider has, use the --cluster option.*
> - *For self-managed clusters, such as those built with kops, manually specify a cluster name with the --cluster option.*
> - *When using the CRI-O or containerd runtimes, pass the --cri flag to the twistcli command-line utility when you generate the YAML configuration file or the Helm chart.*
> - *When using an AWS Bottlerocket-based EKS cluster, pass the --cri flag when creating the YAML.*
> - *To use Defenders in **GKE on ARM**, you must prepare your workloads.*

STEP 3 | Deploy the Defender *DaemonSet* custom resource.

```
$ kubectl create -f ./defender.yaml
```

> *You can run both Prisma Cloud Console and Defenders in the same Kubernetes namespace, for example twistlock. However, you must be careful when running kubectl delete commands with the YAML file generated for Defender. The defender.yaml file contains the namespace declaration, so comment out the namespace section if you don't want the namespace deleted.*

STEP 4 | (Optional) Schedule Defenders on your Kubernetes master nodes.

If you want to also schedule Defenders on your Kubernetes master nodes, change the DaemonSet's toleration spec. Master nodes are tainted by design. Only pods that specifically match the taint can run there. Tolerations allow pods to be deployed on nodes to which taints have been applied. To schedule Defenders on your master nodes, add the following tolerations to your DaemonSet spec.

```
tolerations:
- key: "node-role.kubernetes.io/master"
  operator: "Exists"
  effect: "NoSchedule"
```

STEP 5 | In Prisma Cloud Compute, go to **Manage** > **Defenders** > **Manage** > **Defenders** to see a list of deployed Defenders.

Install Prisma Cloud with Helm charts

You can use *twistcli* to create Helm charts for the Prisma Cloud Console and the Defenders. Helm is a package manager for Kubernetes, and a *chart* is a Helm package.

Follow the main install flow, with the following changes.

- Pass the --*helm_ option to _twistcli* to generate a Helm chart. Don't change the other options passed to *twistcli* since they configure the chart.
- Deploy your Defender with the *helm install* command instead of *kubectl create*.

The following procedure shows the modified commands.

STEP 1 | Download the current recommended release.

STEP 2 | Create a Console Helm chart.

```
$ <PLATFORM>/twistcli console export kubernetes \
  --service-type LoadBalancer \
  --helm
```

STEP 3 | Install the Console.

```
$ helm install twistlock-console \
  --namespace twistlock \
  --create namespace \
  ./twistlock-console-helm.tar.gz
```

STEP 4 | Configure Console.

STEP 5 | Create a Defender *DaemonSet* Helm chart.

```
$ <PLATFORM>/twistcli defender export kubernetes \
  --address https://yourconsole.example.com:8083 \
  --helm \
  --user <ADMIN_USER> \
  --cluster-address twistlock-console
```

STEP 6 | Install the Defender.

```
$ helm install twistlock-defender-ds \
  --namespace twistlock \
  --create namespace \
  ./twistlock-defender-helm.tar.gz
```

Install Prisma Cloud on a CRI (non-Docker) cluster

Kubernetes lets you set up a cluster with the container runtime of your choice. Prisma Cloud supports Docker Engine, CRI-O, and cri-containerd.

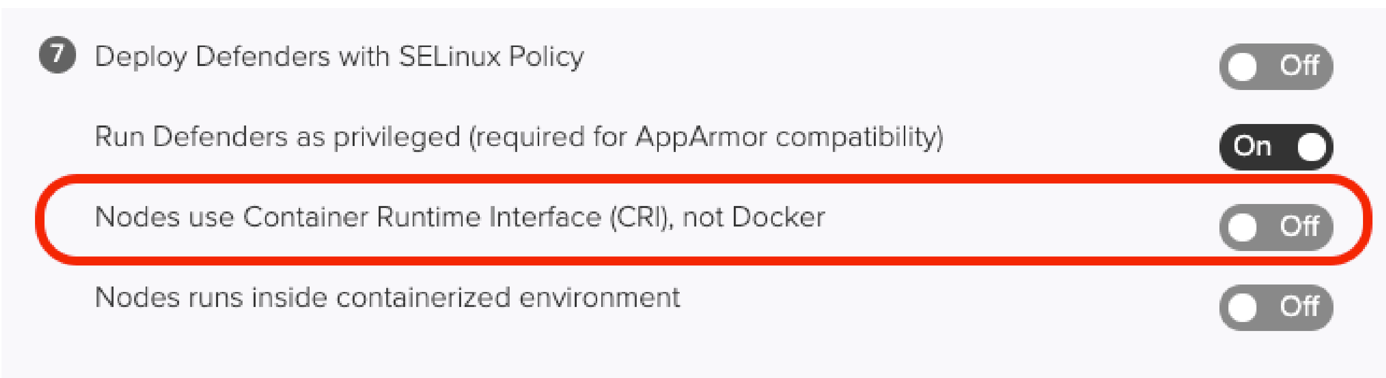
Deploying Console

Irrespective of your cluster's underlying container runtime, you can install Console using the standard install procedure. Console doesn't interface with other containers, so it doesn't need to know which container runtime interface is being used.

Deploying Defender DaemonSets

When generating the YAML file to deploy the Defender DaemonSet, a toggle lets you select your runtime environment. Since Defenders need to have a view of other containers, this option is necessary to guide the communication. By default the toggle is off Prisma Cloud uses Docker Engine. When the toggle is on, Prisma Cloud generates the propper *yaml* for the CRI Kubernetes environment.

*If you use containerd on GKE, and you install Defender without the CRI switch, everything will appear to work properly, but you'll have no images or container scan reports in **Monitor > Vulnerability** and **Monitor > Compliance pages** and you'll have no runtime models in **Monitor > Runtime**. This happens because the Google Container Optimized Operating system (GCOOS) nodes have Docker Engine installed, but Kubernetes doesn't use it. Defender thinks everything is OK because all of the integrations succeed, but the underlying runtime is actually different.*



If you're deploying Defender DaemonSets with twistcli, use the **--cri** option to use to specify the runtime interface. By default (no flag), twistcli generates a configuration that uses Docker Engine. With the **--cri** flag, twistcli generates a configuration that uses CRI.

```
$ <PLATFORM>/twistcli defender export kubernetes \
  --cri
  --address https://yourconsole.example.com:8083 \
  --user <ADMIN_USER> \
  --cluster-address yourconsole.example.com
```

When generating YAML from Console or twistcli, there is a simple change to the *yaml* file as seen below.

In this abbreviated version DEFENDER_TYPE:daemonset will use the Docker interface.

```
...
spec:
  template:
    metadata:
```



```
      labels:
        app: twistlock-defender
    spec:
      serviceAccountName: twistlock-service
      restartPolicy: Always
      containers:
      - name: twistlock-defender-19-03-321
        image: registry-auth.twistlock.com/tw_<token>/twistlock/
defender:defender_19_03_321
        volumeMounts:
        - name: host-root
          mountPath: "/host"
        - name: data-folder
          mountPath: "/var/lib/twistlock"
          ...
        env:
        - name: WS_ADDRESS
          value: wss://yourconsole.example.com:8084
        - name: DEFENDER_TYPE
          value: daemonset
        - name: DEFENDER_LISTENER_TYPE
          value: "none"
          ...
```

In this abbreviated version DEFENDER_TYPE:cri will use the CRI.

```
...
spec:
  template:
    metadata:
      labels:
        app: twistlock-defender
    spec:
      serviceAccountName: twistlock-service
      restartPolicy: Always
      containers:
      - name: twistlock-defender-19-03-321
        image: registry-auth.twistlock.com/tw_<token>/twistlock/
defender:defender_19_03_321
        volumeMounts:
        - name: host-root
          mountPath: "/host"
        - name: data-folder
          mountPath: "/var/lib/twistlock"
          ...
        env:
        - name: WS_ADDRESS
          value: wss://yourconsole.example.com:8084
        - name: DEFENDER_TYPE
          value: cri
        - name: DEFENDER_LISTENER_TYPE
          value: "none"
          ...
```


Troubleshooting

Pod Security Policy

If Pod Security Policy is enabled in your cluster, you might get the following error when trying to create a Defender DaemonSet.

```
Error creating: pods "twistlock-defender-ds-" is forbidden: unable to
 validate against any pod security policy ..Privileged containers are
 not allowed
```

If you get this error, then you must create a PodSecurityPolicy for the Defender and the necessary ClusterRole and ClusterRoleBinding for the twistlock namespace. You can use the following Pod Security Policy, ClusterRole and ClusterRoleBinding:

```
apiVersion: extensions/v1beta1
kind: PodSecurityPolicy
metadata:
 name: prismacloudcompute-service
spec:
 privileged: false
 seLinux:
  rule: RunAsAny
 allowedCapabilities:
  - AUDIT_CONTROL
  - NET_ADMIN
  - SYS_ADMIN
  - SYS_PTRACE
  - MKNOD
  - SETFCAP
 volumes:
  - "hostPath"
  - "secret"
 allowedHostPaths:
  - pathPrefix: "/etc"
  - pathPrefix: "/var"
  - pathPrefix: "/run"
  - pathPrefix: "/dev/log"
  - pathPrefix: "/"
 hostNetwork: true
 hostPID: true
 supplementalGroups:
  rule: RunAsAny
 runAsUser:
  rule: RunAsAny
 fsGroup:
  rule: RunAsAny
```

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
 name: prismacloudcompute-defender-role
rules:
- apiGroups: ['policy']
```



```
  resources: ['podsecuritypolicies']
  verbs: ['use']
  resourceNames:
  - prismacloudcompute-service
```

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
 name: prismacloudcompute-defender-rolebinding
roleRef:
 apiGroup: rbac.authorization.k8s.io
 kind: ClusterRole
 name: prismacloudcompute-defender-role
subjects:
- kind: ServiceAccount
  name: twistlock-service
  namespace: twistlock
```


OpenShift v4

Edit on GitHub

Prisma Cloud Console is deployed as a Deployment, which ensures it's always running. The Prisma Cloud Console and Defender container images can be stored either in the internal OpenShift registry or your own Docker v2 compliant registry. Alternatively, you can configure your deployments to pull images from Prisma Cloud's cloud registry.

Prisma Cloud Defenders are deployed as a DaemonSet, which ensures that an instance of Defender runs on every node in the cluster. You can run Defenders on OpenShift master and infrastructure nodes by removing the taint from them.

The Prisma Cloud Defender container images can be stored either in the internal OpenShift registry or your own Docker v2 compliant registry. Alternatively, you can configure your deployments to pull images from Prisma Cloud's cloud registry.

This guide shows you how to generate deployment YAML files for both Console and Defender, and then deploy them to your OpenShift cluster with the *oc* client.

To better understand clusters, read our cluster context topic.

Preflight checklist

To ensure that your installation on supported versions of OpenShift v4.x goes smoothly, work through the following checklist and validate that all requirements are met.

Minimum system requirements

Validate that the components in your environment (nodes, host operating systems, orchestrator) meet the specs in System requirements.

For OpenShift installs, we recommend using the overlay or overlay2 storage drivers due to a known issue in RHEL. For more information, see https://bugzilla.redhat.com/show_bug.cgi?id=1518519.

Permissions

Validate that you have permission to:

- Push to a private docker registry. For most OpenShift setups, the registry runs inside the cluster as a service. You must be able to authenticate with your registry with docker login.
- Pull images from your registry. This might require the creation of a docker-registry secret.
- Have the correct role bindings to pull and push to the registry. For more information, see Accessing the Registry.
- Create and delete projects in your cluster. For OpenShift installations, a project is created when you run *oc new-project*.
- Run *oc create* commands.

Internal cluster network communication

TCP: 8083, 8084

External cluster network communication

TCP: 443

The Prisma Cloud Console connects to the Prisma Cloud Intelligence Stream (https://intelligence.twistlock.com) on TCP port 443 for vulnerability updates. If your Console is unable to contact the Prisma Cloud Intelligence Stream, follow the guidance for offline environments.

Install Prisma Cloud

Use *twistcli* to install the Prisma Cloud Console and Defenders. The *twistcli* utility is included with every release. After completing this procedure, both Prisma Cloud Console and Prisma Cloud Defenders will be running in your OpenShift cluster.

Download the Prisma Cloud software

Download the latest Prisma Cloud release to any system where the OpenShift oc client is installed.

STEP 1 | Go to Releases, and copy the link to current recommended release.

STEP 2 | Download the release tarball to your cluster controller.

```
$ wget <LINK_TO_CURRENT_RECOMMENDED_RELEASE_LINK>
```

STEP 3 | Unpack the release tarball.

```
$ mkdir twistlock
$ tar xvzf twistlock_<VERSION>.tar.gz -C twistlock/
```

Create an OpenShift project for Prisma Cloud

Create a project named *twistlock*.

Login to the OpenShift cluster and create the *twistlock* project:

```
$ oc new-project twistlock
```

(Optional) Push the Prisma Cloud images to a private registry

When Prisma Cloud is deployed to your cluster, the images are retrieved from a registry. You have a number of options for storing the Prisma Cloud Console and Defender images:

- OpenShift internal registry.
- Private Docker v2 registry. You must create a docker-secret to authenticate with the registry.

Alternatively, you can pull the images from the Prisma Cloud cloud registry at deployment time. Your cluster nodes must be able to connect to the Prisma Cloud cloud registry (registry-auth.twistlock.com) with TLS on TCP port 443.

This guides shows you how to use both the OpenShift internal registry and the Prisma Cloud cloud registry. If you're going to use the Prisma Cloud cloud registry, you can skip this section. Otherwise, this procedure shows you how to pull, tag, and upload the Prisma Cloud images to the OpenShift internal registry's *twistlock* imageStream.

STEP 1 | Determine the endpoint for your OpenShift internal registry. Use either the internal registry's service name or cluster IP.

```
$ oc get svc -n default
NAME                TYPE        CLUSTER-IP       EXTERNAL-IP
 PORT(S)        AGE
docker-registry     ClusterIP   172.30.163.181   <none>        5000/
TCP      88d
```

STEP 2 | Pull the images from the Prisma Cloud cloud registry using your access token. The major, minor, and patch numerals in the <VERSION> string are separated with an underscore. For exampe, 18.11.128 would be 18_11_128.

```
  $ docker pull \
    registry-auth.twistlock.com/tw_<ACCESS_TOKEN>/twistlock/
defender:defender_<VERSION>

  $ docker pull \
    registry-auth.twistlock.com/tw_<ACCESS_TOKEN>/twistlock/
console:console_<VERSION>
```

STEP 3 | Tag the images for the OpenShift internal registry.

```
$ docker tag \
  registry-auth.twistlock.com/tw_<ACCESS_TOKEN>/twistlock/
defender:defender_<VERSION> \
  172.30.163.181:5000/twistlock/private:defender_<VERSION>

$ docker tag \
  registry-auth.twistlock.com/tw_<ACCESS_TOKEN>/twistlock/
console:console_<VERSION> \
  172.30.163.181:5000/twistlock/private:console_<VERSION>
```

STEP 4 | Push the images to the *twistlock* project's imageStream.

```
$ docker push 172.30.163.181:5000/twistlock/
private:defender_<VERSION>
$ docker push 172.30.163.181:5000/twistlock/
private:console_<VERSION>
```

Install Console

Use the *twistcli* tool to generate YAML files or a Helm chart for Prisma Cloud Compute Console. The *twistcli* tool is bundled with the release tarball. There are versions for Linux, macOS, and Windows.

The *twistcli* tool generates YAML files or helm charts for a Deployment and other service configurations, such as a PersistentVolumeClaim, SecurityContextConstraints, and so on. Run the twistcli command with the *--help* flag for additional details about the command and supported flags.

You can optionally customize *twistlock.cfg* to enable additional features. Then run twistcli from the root of the extracted release tarball.

Prisma Cloud Console uses a PersistentVolumeClaim to store data. There are two ways to provision storage for Console:

- **Dynamic provisioning:** Allocate storage for Console on-demand at deployment time. When generating the Console deployment YAML files or helm chart with *twistcli*, specify the name of the storage class with the *--storage-class* flag. Most customers use dynamic provisioning.
- **Manual provisioning:** Pre-provision a persistent volume for Console, then specify its label when generating the Console deployment YAML files. OpenShift uses NFS mounts for the backend infrastructure components (e.g. registry, logging, etc.). The NFS server is typically one of the master nodes. Guidance for creating an NFS backed PersistentVolume can be found here. Also see Appendix: NFS PersistentVolume example.

Option #1: Deploy with YAML files

Deploy Prisma Cloud Compute Console with YAML files.

STEP 1 | Generate a deployment YAML file for Console. A number of command variations are provided. Use them as a basis for constructing your own working command.

Prisma Cloud Console + dynamically provisioned PersistentVolume + image pulled from the OpenShift internal registry.

```
$ <PLATFORM>/twistcli console export openshift \
  --storage-class "<STORAGE-CLASS-NAME>" \
  --image-name "172.30.163.181:5000/twistlock/
private:console_<VERSION>" \
  --service-type "ClusterIP"
```

Prisma Cloud Console + manually provisioned PersistentVolume + image pulled from the OpenShift internal registry. Using the NFS backed PersistentVolume described in Appendix: NFS PersistentVolume example, pass the label to the *--persistent-volume-labels* flag to specify the PersistentVolume to which the PersistentVolumeClaim will bind.

```
$ <PLATFORM>/twistcli console export openshift \
  --persistent-volume-labels "app-volume=twistlock-console" \
  --image-name "172.30.163.181:5000/twistlock/
private:console_<VERSION>" \
  --service-type "ClusterIP"
```

Prisma Cloud Console + manually provisioned PersistentVolume + image pulled from the Prisma Cloud cloud registry. If you omit the *--image-name* flag, the Prisma Cloud cloud registry is used by default, and you are prompted for your access token.

```
$ <PLATFORM>/twistcli console export openshift \
  --persistent-volume-labels "app-volume=twistlock-console" \
  --service-type "ClusterIP"
```

STEP 2 | Deploy Console.

```
$ oc create -f ./twistlock_console.yaml
```

You can safely ignore the error that says the twistlock project already exists.

Option #2: Deploy with Helm chart

Deploy Prisma Cloud Compute Console with a Helm chart.

Prisma Cloud Console Helm charts fail to install on OpenShift 4 clusters due to a Helm bug. If you generate a Helm chart, and try to install it in an OpenShift 4 cluster, you'll get the following error:

```
Error: unable to recognize "": no matches for kind
 "SecurityContextConstraints" in version "v1"
```

To work around the issue, you'll need to manually modify the generated Helm chart.

STEP 1 | Generate a deployment helm chart for Console. A number of command variations are provided. Use them as a basis for constructing your own working command.

Prisma Cloud Console + dynamically provisioned PersistentVolume + image pulled from the OpenShift internal registry.

```
$ <PLATFORM>/twistcli console export openshift \
  --storage-class "<STORAGE-CLASS-NAME>" \
  --image-name "172.30.163.181:5000/twistlock/
private:console_<VERSION>" \
  --service-type "ClusterIP" \
  --helm
```

Prisma Cloud Console + manually provisioned PersistentVolume + image pulled from the OpenShift internal registry. Using the NFS backed PersistentVolume described in Appendix: NFS PersistentVolume example, pass the label to the *--persistent-volume-labels* flag to specify the PersistentVolume to which the PersistentVolumeClaim will bind.

```
$ <PLATFORM>/twistcli console export openshift \
  --persistent-volume-labels "app-volume=twistlock-console" \
  --image-name "172.30.163.181:5000/twistlock/
private:console_<VERSION>" \
  --service-type "ClusterIP" \
  --helm
```

Prisma Cloud Console + manually provisioned PersistentVolume + image pulled from the Prisma Cloud cloud registry. If you omit the *--image-name* flag, the Prisma Cloud cloud registry is used by default, and you are prompted for your access token.

```
$ <PLATFORM>/twistcli console export openshift \
  --persistent-volume-labels "app-volume=twistlock-console" \
  --service-type "ClusterIP" \
  --helm
```

STEP 2 | Unpack the chart into a temporary directory.

```
$ mkdir helm-console
$ tar xvzf twistlock-console-helm.tar.gz -C helm-console/
```

STEP 3 | Open *helm-console/twistlock-console/templates/securitycontextconstraints.yaml* for editing.

STEP 4 | Change *apiVersion* from *v1* to *security.openshift.io/v1*.

```
{{- if .Values.openshift }}
apiVersion: security.openshift.io/v1
kind: SecurityContextConstraints
metadata:
  name: twistlock-console
...
```

STEP 5 | Repack the Helm chart

```
$ cd helm-console/
$ tar cvzf twistlock-console-helm.tar.gz twistlock-console/
```

STEP 6 | Install the updated Helm chart.

```
$ helm install --namespace=twistlock -g twistlock-console-
helm.tar.gz
```

Create an external route to Console

Create an external route to Console so that you can access the web UI and API.

STEP 1 | From the OpenShift web interface, go to the *twistlock* project.

STEP 2 | Go to **Application** > **Routes**.

STEP 3 | Select **Create Route**.

STEP 4 | Enter a name for the route, such as **twistlock-console**.

STEP 5 | Hostname = URL used to access the Console, e.g. *twistlock-console.apps.ose.example.com*

STEP 6 | Path = /

STEP 7 | Service = **twistlock-console**

STEP 8 | Target Port = 8083 → 8083

STEP 9 | Select the **Security** > **Secure Route** radio button.

STEP 10 | TLS Termination = Passthrough (if using 8083)

If you plan to issue a custom certificate for Console TLS communication that is trusted and will allow the TLS establishment with the Prisma Cloud Console, then Select Passthrough TLS for TCP port 8083.

STEP 11 | Insecure Traffic = **Redirect**

STEP 12 | Click **Create**.

Create an external route to Console for external Defenders

If you are planning to deploy Defenders to another cluster and report to this Console, you will need to create an additional external route to Console so that the Defenders can access the Console. You need to expose the Prisma Cloud-Console service's TCP port 8084 as external OpenShift routes. Each route will be an unique, fully qualified domain name.

STEP 1 | From the OpenShift web interface, go to the *twistlock* project.

STEP 2 | Go to **Application > Routes**.

STEP 3 | Select **Create Route**.

STEP 4 | Enter a name for the route, such as **twistlock-console-8084**.

STEP 5 | Hostname = URL used to access the Console, using a different hostname, e.g. *twistlock-console-8084.apps.ose.example.com*

STEP 6 | Path = /

STEP 7 | Service = **twistlock-console**

STEP 8 | Target Port = 8084 → 8084

STEP 9 | Select the **Security > Secure Route** radio button.

STEP 10 | TLS Termination = Passthrough (if using 8084)

The Defender to Console communication is a mutual TLS secure websocket session. This communication cannot be intercepted.

STEP 11 | Insecure Traffic = **Redirect**

STEP 12 | Click **Create**.

Configure Console

Create your first admin user, enter your license key, and configure Console's certificate so that Defenders can establish a secure connection to it.

STEP 1 | In a web browser, navigate to the external route you configured for Console, e.g. *https://twistlock-console.apps.ose.example.com*.

STEP 2 | Create your first admin account.

STEP 3 | Enter your license key.

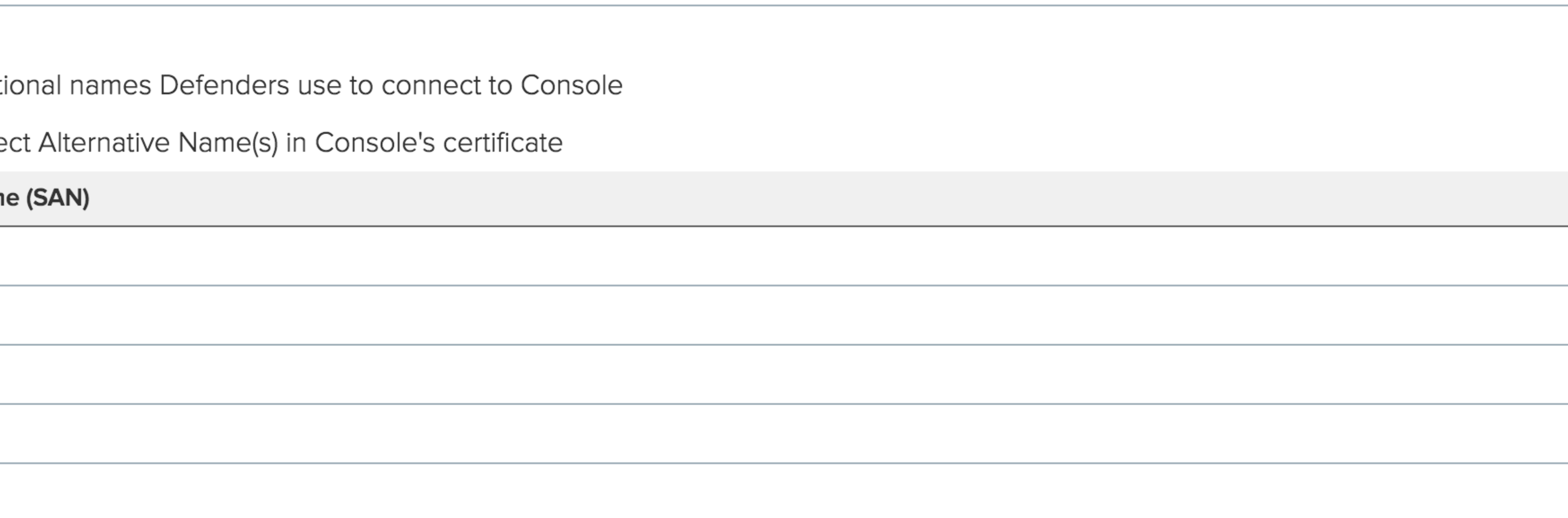
STEP 4 | Add a SubjectAlternativeName to Console's certificate to allow Defenders to establish a secure connection with Console.

Use either Console's service name, *twistlock-console* or *twistlock-console.twistlock.svc*, or Console's cluster IP.

Additionally, if a route for external Defenders was created, add that one to the SAN list too: *twistlock-console-8084.apps.ose.example.com*

```
$ oc get svc -n twistlock
NAME                TYPE           CLUSTER-IP     EXTERNAL-IP
           PORT(S)
twistlock-console   LoadBalancer   172.30.41.62
 172.29.61.32,172.29.61.32   8084:3184...
```

1. Go to **Manage > Defenders > Names**.
2. Click **Add SAN** and enter Console's service name.
3. Click **Add SAN** and enter Console's cluster IP.



Install Defender

Prisma Cloud Defenders run as containers on the nodes in your OpenShift cluster. They are deployed as a DaemonSet. Use the *twistcli* tool to generate the DaemonSet deployment YAML or helm chart.

The command has the following basic structure It creates a YAML file named *defender.yaml* or a helm chart *twistlock-defender-helm.tar.gz* in the working directory.

Example for export of a YAML file:

```
$ <PLATFORM>/twistcli defender export openshift \
  --address <ADDRESS> \
  --cluster-address <CLUSTER-ADDRESS> \
  --cri
```

Example for export of a Helm chart:

```
$ <PLATFORM>/twistcli defender export openshift \
  --address <ADDRESS> \
  --cluster-address <CLUSTER-ADDRESS> \
  --helm \
  --cri
```

The command connects to Console's API, specified in *--address*, to generate the Defender DaemonSet YAML config file or helm chart. The location where you run twistcli (inside or outside the cluster) dictates which Console address should be supplied.

The *--cluster-address* flag specifies the address Defender uses to connect to Console. For Defenders deployed inside the cluster, specify Prisma Cloud Console's service name, twistlock-console or twistlock-console.twistlock.svc, or cluster IP address. For Defenders deployed outside the cluster, specify the external route for the Console over port 8084 created before, *twistlock-console-8084.apps.ose.example.com*, if the external route is not exposing port 8084, specify the port in the address, e.g. *twistlock-console-8084.apps.ose.example.com:443* within the defender daemonSet yaml.

Example: Edit the resulting defender.yaml and change: - name: WS_ADDRESS value: wss://twistlock-console-8084.apps.ose.example.com:8084 to - name: WS_ADDRESS value: wss://twistlock-console-8084.apps.ose.example.com:443

If SELinux is enabled on the OpenShift nodes, pass the *--selinux-enabled* argument to twistcli.

For managed clusters, Prisma Cloud automatically gets the cluster name from the cloud provider. To override the the cloud provider's cluster name, use the *--cluster* option. For self-managed clusters, manually specify a cluster name with the *--cluster* option.

Option #1: Deploy with YAML files

Deploy the Defender DaemonSet with YAML files.

STEP 1 | Generate the Defender DaemonSet YAML. A number of command variations are provided. Use them as a basis for constructing your own working command.

Outside the OpenShift cluster + pull the Defender image from the Prisma Cloud cloud registry. Use the OpenShift external route for your Prisma Cloud Console, *--address https://twistlock-console.apps.ose.example.com*. Designate Prisma Cloud's cloud registry by omitting the *--image-name* flag. Defining CRI-O as the default container engine by using the *cri* flag.

```
$ <PLATFORM>/twistcli defender export openshift \
  --address https://twistlock-console.apps.ose.example.com \
  --cluster-address 172.30.41.62 \
  --selinux-enabled \
  --cri
```

Outside the OpenShift cluster + pull the Defender image from the OpenShift internal registry. Use the *--image-name* flag to designate an image from the OpenShift internal registry. Defining CRI-O as the default container engine by using the *cri* flag.

```
$ <PLATFORM>/twistcli defender export openshift \
  --address https://twistlock-console.apps.ose.example.com \
  --cluster-address 172.30.41.62 \
  --selinux-enabled \
```

```
  --image-name 172.30.163.181:5000/twistlock/
private:defender_<VERSION> \
  --cri
```

Inside the OpenShift cluster + pull the Defender image from the Prisma Cloud cloud registry. When generating the Defender DaemonSet YAML with twistcli from a node inside the cluster, use Console's service name (twistlock-console) or cluster IP in the *--cluster-address* flag. This flag specifies the endpoint for the Prisma Cloud Compute API and must include the port number. Defining CRI-O as the default container engine by using the *cri* flag.

```
$ <PLATFORM>/twistcli defender export openshift \
  --address https://172.30.41.62:8083 \
  --cluster-address 172.30.41.62 \
  --selinux-enabled \
  --cri
```

Inside the OpenShift cluster + pull the Defender image from the OpenShift internal registry. Use the *--image-name* flag to designate an image in the OpenShift internal registry. Defining CRI-O as the default container engine by using the *-cri* flag.

```
$ <PLATFORM>/twistcli defender export openshift \
  --address https://172.30.41.62:8083 \
  --cluster-address 172.30.41.62 \
  --selinux-enabled \
  --image-name 172.30.163.181:5000/twistlock/
private:defender_<VERSION> \
  --cri
```

STEP 2 | Deploy the Defender DaemonSet.

```
$ oc create -f ./defender.yaml
```

Option #2: Deploy with Helm chart

Deploy the Defender DaemonSet with a Helm chart.

Prisma Cloud Defenders Helm charts fail to install on OpenShift 4 clusters due to a Helm bug. If you generate a Helm chart, and try to install it in an OpenShift 4 cluster, you'll get the following error:

```
Error: unable to recognize "": no matches for kind
 "SecurityContextConstraints" in version "v1"
```

To work around the issue, you'll need to manually modify the generated Helm chart.

STEP 1 | Generate the Defender DaemonSet helm chart. A number of command variations are provided. Use them as a basis for constructing your own working command.

Outside the OpenShift cluster + pull the Defender image from the Prisma Cloud cloud registry. Use the OpenShift external route for your Prisma Cloud Console, *--address https://twistlock-console.apps.ose.example.com*. Designate Prisma Cloud's cloud registry by omitting the *--image-name* flag. Defining CRI-O as the default container engine by using the *-cri* flag.

```
$ <PLATFORM>/twistcli defender export openshift \
```

```
  --address https://twistlock-console.apps.ose.example.com \
  --cluster-address 172.30.41.62 \
  --helm \
  --cri
```

Outside the OpenShift cluster + pull the Defender image from the OpenShift internal registry. Use the *--image-name* flag to designate an image from the OpenShift internal registry. Defining CRI-O as the default container engine by using the *-cri* flag.

```
$ <PLATFORM>/twistcli defender export openshift \
  --address https://twistlock-console.apps.ose.example.com \
  --cluster-address 172.30.41.62 \
  --image-name 172.30.163.181:5000/twistlock/
private:defender_<VERSION> \
  --helm \
  --cri
```

Inside the OpenShift cluster + pull the Defender image from the Prisma Cloud cloud registry. When generating the Defender DaemonSet YAML with twistcli from a node inside the cluster, use Console's service name (twistlock-console) or cluster IP in the *--cluster-address* flag. This flag specifies the endpoint for the Prisma Cloud Compute API and must include the port number. Defining CRI-O as the default container engine by using the *-cri* flag.

```
$ <PLATFORM>/twistcli defender export openshift \
  --address https://172.30.41.62:8083 \
  --cluster-address 172.30.41.62 \
  --helm \
  --cri
```

Inside the OpenShift cluster + pull the Defender image from the OpenShift internal registry. Use the *--image-name* flag to designate an image in the OpenShift internal registry. Defining CRI-O as the default container engine by using the *-cri* flag.

```
$ <PLATFORM>/twistcli defender export openshift \
  --address https://172.30.41.62:8083 \
  --cluster-address 172.30.41.62 \
  --image-name 172.30.163.181:5000/twistlock/
private:defender_<VERSION> \
  --helm \
  --cri
```

STEP 2 | Unpack the chart into a temporary directory.

```
$ mkdir helm-defender
$ tar xvzf twistlock-defender-helm.tar.gz -C helm-defender/
```

STEP 3 | Open *helm-console/twistlock-defender/templates/securitycontextconstraints.yaml* for editing.

STEP 4 | Change *apiVersion* from *v1* to *security.openshift.io/v1*.

```
{{- if .Values.openshift }}
apiVersion: security.openshift.io/v1
kind: SecurityContextConstraints
```

```
metadata:
name: twistlock-console
...
```

STEP 5 | Repack the Helm chart

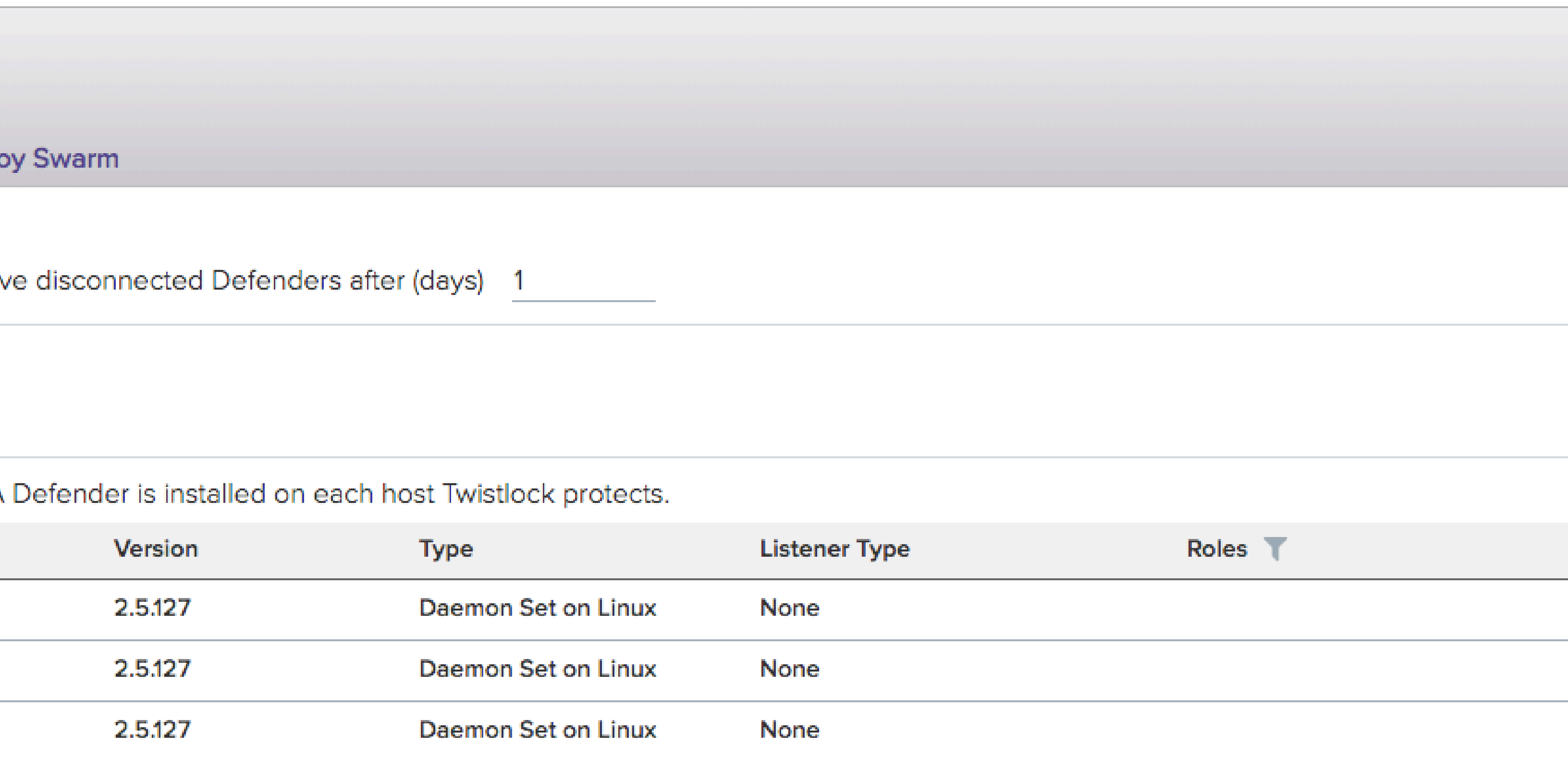
```
$ cd helm-defender/
$ tar cvzf twistlock-defender-helm.tar.gz twistlock-defender/
```

STEP 6 | Install the updated Helm chart.

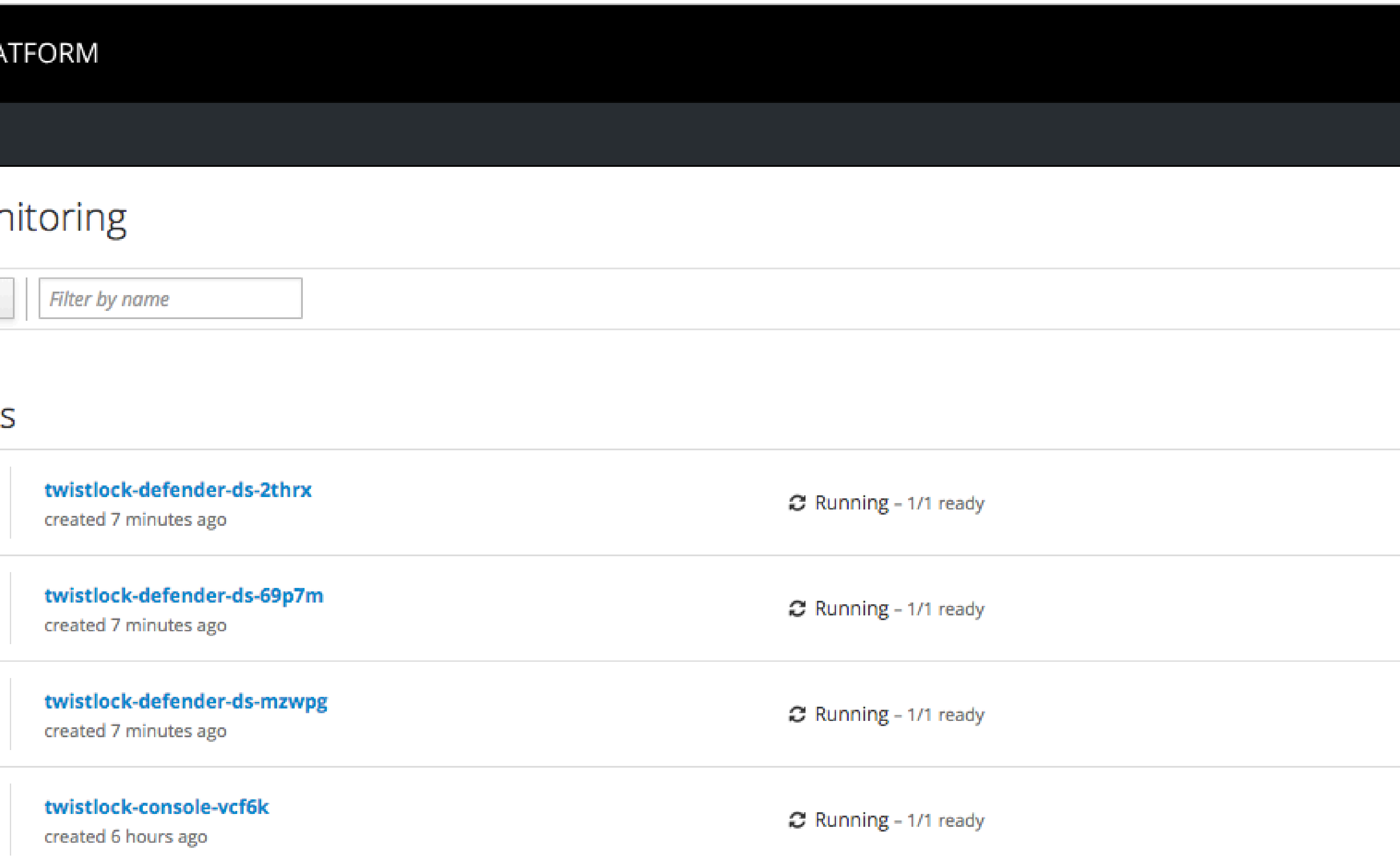
```
$ helm install --namespace=twistlock -g twistlock-defender-
helm.tar.gz
```

Confirm the Defenders were deployed.

1. In Prisma Cloud Console, go to **Manage > Defenders > Manage** to see a list of deployed Defenders.



2. In the OpenShift Web Console, go to the Prisma Cloud project's monitoring window to see which pods are running.



3. Using the OpenShift CLI to see the DaemonSet pod count.

```
$ oc get ds -n twistlock
```

```
NAME                    DESIRED   CURRENT   READY     UP-TO-DATE
 AVAILABLE    NODE SELECTOR    AGE
twistlock-defender-ds   4         3         3         3
 3            <none>           29m
```

The desired and current pod counts do not match. This is a job for the nodeSelector.

Control Defender deployments with taint

You can deploy Defenders to all nodes in an OpenShift cluster (master, infra, compute). OpenShift Container Platform automatically taints infra and master nodes These taints have the NoSchedule effect, which means no pod can be scheduled on them.

To run the Defenders on these nodes, you can either remove the taint or add a toleration to the Defender DaemonSet. Once this is done, the Defender Daemonset will automatically be deployed to these nodes (no need to redeploy the Daemonset). Adjust the guidance in the following procedure according to your organization's deployment strategy.

- **Option 1 - remove taint all nodes:**

```
$ oc adm taint nodes --all node-role.kubernetes.io/master-
```

- **Option 2 - remove taint from specific nodes:**

```
$ oc adm taint nodes <node-name> node-role.kubernetes.io/master-
```

- **Option 3 - add tolerations to the twistlock-defender-ds DaemonSet:**

```
$ oc edit ds twistlock-defender-ds -n twistlock
```

  Add the following toleration in PodSpec (DaemonSet.spec.template.spec)

```
tolerations:
- key: "node-role.kubernetes.io/master"
  operator: "Exists"
  effect: "NoSchedule"
```

Uninstall

To uninstall Prisma Cloud, delete the *twistlock* project, then delete the Prisma Cloud PersistentVolume.

STEP 1 | Delete the *twistlock* Project

```
$ oc delete project twistlock
```

STEP 2 | Delete the *twistlock* PersistentVolume

```
$ oc delete pv twistlock
```

Appendix: NFS PersistentVolume example

Create an NFS mount for the Prisma Cloud Console's PV on the host that serves the NFS mounts.

STEP 1 | **mkdir /opt/twistlock_console**

STEP 2 | Check selinux: **sestatus**

STEP 3 | **chcon -R -t svirt_sandbox_file_t -l s0 /opt/twistlock_console**

STEP 4 | **sudo chown nfsnobody /opt/twistlock_console**

STEP 5 | **sudo chgrp nfsnobody /opt/twistlock_console**

STEP 6 | Check perms with: **ls -lZ /opt/twistlock_console** (drwxr-xr-x. nfsnobody nfsnobody system_u:object_r:svirt_sandbox_file_t:s0)

STEP 7 | Create **/etc/exports.d/twistlock.exports**

STEP 8 | In the **/etc/exports.d/twistlock.exports** add in line **/opt/twistlock_console *(rw,root_squash)**

STEP 9 | Restart nfs mount **sudo exportfs -ra**

STEP 10 | Confirm with **showmount -e**

STEP 11 | Get the IP address of the Master node that will be used in the PV (eth0, openshift uses 172. for node to node communication). Make sure TCP 2049 (NFS) is allowed between nodes.

STEP 12 | Create a PersistentVolume for Prisma Cloud Console.

The following example uses a label for the PersistentVolume and the volume and claim pre-binding features. The PersistentVolumeClaim uses the *app-volume: twistlock-console* label to bind to the PV. The volume and claim pre-binding *claimref* ensures that the PersistentVolume is not claimed by another PersistentVolumeClaim before Prisma Cloud Console is deployed.

```
apiVersion: v1
kind: PersistentVolume
metadata:
 name: twistlock
 labels:
  app-volume: twistlock-console
storageClassName: standard
spec:
  capacity:
   storage: 100Gi
  accessModes:
  - ReadWriteOnce
  nfs:
   path: /opt/twistlock_console
   server: 172.31.4.59
persistentVolumeReclaimPolicy: Retain
claimRef:
  name: twistlock-console
  namespace: twistlock
```

Appendix: Implementing SAML federation with a Prisma Cloud Console inside an OpenShift cluster

When federating Prisma Cloud Console that is accessed through an OpenShift external route with a SAML v2.0 Identity Provider (IdP), the SAML authentication request's *AssertionConsumerServiceURL* value must be modified. Prisma Cloud automatically generates the *AssertionConsumerServiceURL* value sent in a SAML authentication request based on Console's configuration. When Console is accessed through an OpenShift external route, the URL for Console's API endpoint is most likely not the same as the automatically generated *AssertionConsumerServiceURL*. Therefore, you must configure the *AssertionConsumerServiceURL* value that Prisma Cloud sends in the SAML authentication request.

STEP 1 | Log into Prisma Cloud Console.

STEP 2 | Go to **Manage > Authentication > SAML**.

STEP 3 | In **Console URL**, define the *AssertionConsumerServiceURL*.

In this example, enter *https://twistlock-console.apps.ose.example.com*

Console on Fargate

Edit on GitHub

You can run Prisma Cloud Console in AWS Fargate.

This procedure assumes you've already created an ECS cluster.

Create a security group

Create a security group that opens ports 8083-8084 for Prisma Cloud Console and port 2049 for NFS.

STEP 1 | In the AWS console, go to **Services > Compute > EC2 > Security Groups**.

STEP 2 | Click **Create security group**.

STEP 3 | In **Security group name**, enter a name, such as **pc-security-group**.

STEP 4 | In **Description**, enter **Prisma Cloud Compute Console on Fargate**.

STEP 5 | In **VPC**, select the VPC where your ECS cluster runs.

STEP 6 | Create an inbound rule for Prisma Cloud Console ports.

1. Under **Inbound rules** , click **Add rule**.
2. Under **Type**, select **Custom TCP**.
3. Under **Port range**, enter **8083-8084**.
4. Under **Source**, select **Anywhere**.

STEP 7 | Create an inbound rule for NFS, where Console stores its data.

1. Click **Add rule**.
2. Under **Type**, select **NFS**.
3. Under **Source**, select **Anywhere**.

STEP 8 | Click **Create security group**.

STEP 9 | Write down the security group ID and save it for later.

Create an EFS file system

Create a highly available file system for Console to store its data.

STEP 1 | In the AWS console, go to **Services > Storage > EFS**.

STEP 2 | Click **Create file system**.

STEP 3 | Click **Customize** to open a more detailed dialog.

STEP 4 | Enter a value for **Name**, such as **pc-efs-console**.

STEP 5 | Set the throughput mode to **Provisioned**.

STEP 6 | Set **Provisioned Throughput (MiB/s)** to 0.1 MiB/s per Defender that will be deployed.

STEP 7 | Click **Next**.

STEP 8 | In **VPC**, select the VPC where your EC2 cluster runs and the relevant mount targets.

STEP 9 | For each mount target, change the security group to the ID of the pc-security-group.

STEP 10 | Click **Next**, accepting all defaults, until the file system is created.

STEP 11 | Write down the file system ID and save it for later.

Create target groups

Create two target groups for the load balancer, one for port 8083 and one for port 8084.

STEP 1 | In the AWS console, go to **Services** > **Compute** > **EC2** > **Load Balancing** > **Target Groups**.

STEP 2 | Click **Create target group**.

STEP 3 | In **Basic configuration**, select **IP addresses**.

STEP 4 | Enter a value for **Name**, such as **pc-tgt-8083** or **pc-tgt-8084**.

STEP 5 | Set **Protocol** to **TCP** and **Port** to **8083** or **8084** respectively.

STEP 6 | In VPC, select the VPC where your ECS cluster runs.

STEP 7 | For port 8083 only, specify the following health check configuration:

- **Health check protocol**: **HTTPS**
- **Health check path**: **/**
- **Port**: **Traffic port**
- Accept the default values for all other settings.

STEP 8 | Click **Next**, and then click **Create target group**.

STEP 9 | Repeat the process for port 8084, but accept the default values for the health check configuration.

The health check protocol for 8084 must be **TCP**.

STEP 10 | Write down the ARN for both target groups, and save them for later.

Create a load balancer

Create a network load balancer to route traffic to the Console container.

STEP 1 | In the AWS console, go to **Services** > **Compute** > **EC2** > **Load Balancers**.

STEP 2 | Click **Create Load Balancer**.

STEP 3 | Choose **Network Load Balancer** and **Create**.

STEP 4 | Enter a value for **Name**, such as **pc-ecs-lb**.

STEP 5 | Under **Network mapping**, select the VPC and subnet where the Prisma Cloud Console task will run.

STEP 6 | Under **Listeners and routing**, create a listener for port 8083.

1. Set **Protocol** to **TCP**.
2. Set **Port** to **8083**.
3. Set **Default action** to **Forward to: pc-tgt-8083**.

STEP 7 | Create a listener for port 8084.

1. Click **Add listener**.
2. Set **Protocol** to **TCP**.
3. Set **Port** to **8084**.
4. Set **Default action** to **Forward to: pc-tgt-8084**.

STEP 8 | Click **Create load balancer**.

STEP 9 | Write down the DNS name for the load balancer, and save it for later.

Create task definition

Use twistcli to generate a task definition for Console.

Each task definition's Console can support up to 1000 deployed Defenders.

The following table lists valid values for cpu-limit and memory-limit:

CPU limit	Memory limit (MiB)
1024 (1 vCPU)	2048 (2 GB), 3072 (3 GB), 4096 (4 GB), 5120 (5 GB), 6144 (6 GB), 7168 (7 GB), 8192 (8 GB)
2048 (2 vCPU)	Between 4096 (4 GB) and 16384 (16 GB) in increments of 1024 (1 GB)
4096 (4 vCPU)	Between 8192 (8 GB) and 30720 (30 GB) in increments of 1024 (1 GB)

STEP 1 | Download the Prisma Cloud Compute Edition release tarball, and unpack it.

STEP 2 | Run twistcli to create the task definition.

```
./<PLATFORM>/twistcli console export fargate \
--registry-token <registry token>  \
--cluster-ip <load balancer dns name> \
--memory-limit <memory limit number> \
--cpu-limit <cpu limit number> \
```



```
--efs-volume <efs ID>
```

For example:

```
./linux/twistcli console export fargate \
--registry-token <my_registry_token_id_string>  \
--cluster-ip my-fargate-console-dns-address.elb.us-
east-1.amazonaws.com \
--memory-limit 8192 \
--cpu-limit 2048 \
--efs-volume fs-12345678
```

STEP 3 | In the AWS console, go to **Services > Containers > Elastic Container Service > Task Definitions**.

STEP 4 | Click **Create new Task Definition**.

STEP 5 | Click **Fargate**, then **Next step**.

STEP 6 | Scroll to the bottom of the page, and click **Configure via JSON**.

STEP 7 | Clear the text box, paste the contents of **twistlock-console.json** which was generated by twistcli, and click **Save**.

STEP 8 | In **Task Role**, specify **ecsTaskExecutionRole**.

STEP 9 | Click **Create**.

STEP 10 | Click **View Task Definition**.

STEP 11 | Copy the task definition name and revision (e.g., **pc-console:1**).

Create Fargate service

Create the Fargate service.

STEP 1 | In the AWS console, go to **Services > Networking & Content Delivery > VPC > Subnets**.

STEP 2 | Filter the subnets by the VPC where your ECS cluster runs, and write down subnet IDs of the relevant availability zones.

STEP 3 | Fill out the ECS service JSON with all values you've set aside until now.

Replace the strings between the < > characters, and save the file with the name fargate-pc-console-service.json.

```
{
    "cluster": "<cluster name>",
    "serviceName": "pc-console",
    "taskDefinition": "<task definition name>:<revision>",
    "loadBalancers": [
                {
                    "targetGroupArn": "<pc-tgt-8083 ARN>",
                    "containerName": "twistlock-console",
```

```
                    "containerPort": 8083
                },
                            {
                    "targetGroupArn": "<pc-tgt-8083 ARN>",
                    "containerName": "twistlock-console",
                    "containerPort": 8084
                }
    ],
    "desiredCount": 1,
    "launchType": "FARGATE",
    "deploymentConfiguration": {
        "maximumPercent": 100,
        "minimumHealthyPercent": 0
    },
    "platformVersion": "1.4.0",
    "networkConfiguration": {
        "awsvpcConfiguration": {
            "subnets": [
                "<subnet ID>",
                "<subnet ID>"
            ],
            "securityGroups": [
                "<security group ID>"
            ],
            "assignPublicIp": "ENABLED"
        }
    },
    "enableECSManagedTags": true
}
```

STEP 4 | Create the service using awscli.

```
aws ecs create-service --cli-input-json file://path/to/fargate-pc-
console-service.json
```

If successful the service is successfully created, awscli outputs the full JSON for the service being deployed.

STEP 5 | In the AWS console, go to **Services** > **Containers** > **Elastic Container Service** > **Clusters**, click your cluster.

STEP 6 | In the **Services** tab, click the service name (**pc-console**).

You should see the details for load balancing and network access.

STEP 7 | In the **Tasks** tab, you should find details about the running container.

Log into Prisma Cloud Console

Open a web browser and go to *https://<Load balancer DNS name>:8083*. Create an initial admin account, and then enter your license to activate Console.

Amazon ECS

Edit on GitHub

This guide shows you how to deploy Prisma Cloud in an ECS cluster with a single infrastructure node and two worker nodes. Console runs on the infrastructure node. An instance of Defender runs on each node in the cluster.

Console is the Prisma Cloud management interface It runs as a service in your ECS cluster. The parameters of the service are described in a task definition, and the task definition is written in JSON format.

Defender protects your containerized environment according to the policies you set in Prisma Cloud Console It also runs a service in your ECS cluster. To automatically deploy an instance of Defender on each node in your cluster, you'll run the Defender task as a *daemon* service.

The installation described in this article is meant to be highly available. Data is persisted across nodes. If an infrastructure node were to go down, ECS can reschedule the Console service on any healthy node, and Console will continue to have access to its state. To enable this capability, you'll attach storage that's accessible from each of your infrastructure nodes, and Amazon Elastic File System (EFS) is an excellent option.

When you have multiple infrastructure nodes, ECS can schedule Console on any of them. Defenders need a reliable way to connect to Console. A load balancer automatically directs traffic to the node where Console runs, and offers a stable interface that Defenders can use to connect to Console and that operators can use to access its web interface.

We assume you are deploying Prisma Cloud to the default VPC. If you are not using the default VPC, adjust your settings accordingly.

This guide assumes you know very little about AWS ECS. As such, it is extremely prescriptive, and includes step for building your cluster. If you are already familiar with AWS ECS and do not need assistance navigating the interface, simply read the section synopsis, which summarizes all key configurations. To better understand clusters, read our cluster context topic.

Download the Prisma Cloud software

The Prisma Cloud release tarball contains all the release artifacts.

STEP 1 | Download the latest recommended release.

STEP 2 | Retrieve the release tarball.

```
$ wget <LINK_TO_CURRENT_RECOMMENDED_RELEASE_LINK>
```

STEP 3 | Unpack the Prisma Cloud release tarball.

```
$ mkdir twistlock
$ tar xvzf prisma_cloud_compute_edition_<VERSION>.tar.gz  -C
 twistlock/
```

Create a cluster

Create an empty cluster named *pc-ecs-cluster*. Later, you will create launch configurations and auto-scaling groups to start EC2 instances in the cluster.

STEP 1 | Log into the AWS Management Console.

STEP 2 | Go to **Services > Containers > Elastic Container Service**.

STEP 3 | Click **Create Cluster**.

STEP 4 | Select **Networking only**, then click **Next Step**.

STEP 5 | Enter a cluster name, such as **pc-ecs-cluster**.

STEP 6 | Click **Create**.

Create a security group

Create a new security group named *pc-security-group* that opens the following ports. This security group will be associated with resources in your cluster.

Port	Description
8083	Prisma Cloud Console's UI and API.
8084	Prisma Cloud secure websocket for Console-Defender communication.
2049	NFS for Prisma Cloud Console to access its state.
22	SSH for managing nodes.

You can harden this configuration as required. For example, you might want to limit access to port 22 to specific source IPs.

STEP 1 | Go to **Services > Compute > EC2**.

STEP 2 | In the left menu, click **NETWORK & SECURITY > Security Groups**.

STEP 3 | Click **Create Security Group**.

STEP 4 | In **Security group name**, enter a name, such as **pc-security-group**.

STEP 5 | In **Description**, enter **Prisma Cloud ports**.

STEP 6 | In **VPC**, select your default VPC.

STEP 7 | Under the **Inbound rules** section, click **Add Rule**.

1. Under **Type**, select **Custom TCP**.
2. Under **Port Range**, enter **8083-8084**.
3. Under **Source**, select **Anywhere**.

STEP 8 | Click **Add Rule**.

1. Under **Type**, select **NFS**.
2. Under **Source**, select **Anywhere**.

STEP 9 | Click **Add Rule**.

1. Under **Type**, select **SSH**.
2. Under **Source**, select **Anywhere**.

STEP 10 | Click **Create security group**.

Create an EFS file system for Console

Create the Console EFS file system, and then get the command that will be used to mount the file system on every infrastructure node.

The EFS file system and ECS cluster must be in the same VPC and security group.

Prerequisites: Prisma Cloud Console depends on an EFS file system with the following performance characteristics:

- **Performance mode:** General purpose.
- **Throughput mode:** Provisioned. Provision 0.1 MiB/s per deployed Defender. For example, if you plan to deploy 10 Defenders, provision 1 MiB/s of throughput.

STEP 1 | Log into the AWS Management Console.

STEP 2 | Go to **Services > Storage > EFS**.

STEP 3 | Click **Create File System**.

STEP 4 | Enter a value for **Name**, such as **pc-efs-console**

STEP 5 | Select a VPC.

STEP 6 | Click **Customize**.

STEP 7 | Set throughput mode to **Provisioned**, and set **Throughput** to 0.1 MiB/s per Defender to be deployed.

For example, if you plan to deploy ten Defenders, set throughput to 1 MiB/s (10 Defenders * 0.1 MiB/s = 1 MiB/s).

STEP 8 | Click **Next**.

STEP 9 | For each mount target, select the **pc-security-group**.

STEP 10 | Click **Next**.

STEP 11 | In **File System Policy**, click **Next**.

STEP 12 | Review your settings and click **Create**.

STEP 13 | Click **View file system**.

STEP 14 | Click **Attach**, copy the NFS client mount command, and set it aside for later.

You will use the mount command when setting up Console's launch configuration.

Set up a load balancer

Set up an AWS Classic Load Balancer, and capture the Load Balancer DNS name.

You'll create two load balancer listeners. One is used for Console's UI and API, which are served on port 8083. Another is used for the websocket connection between Defender and Console, which is established on port 8084.

For detailed instructions on how to create a load balancer for Console, see Configure an AWS Load Balancer for ECS.

Deploy Console

Launch an infrastructure node that runs in the cluster, then start Prisma Cloud Console as a service on that node.

Create a launch configuration for the infrastructure node

Launch configurations are templates that are used by an auto-scaling group to start EC2 instances in your cluster.

Create a launch configuration named *pc-infra-node* that:

- Creates an instance type of t2.xlarge, or higher. For more information about Console's minimum requirements, see the system requirements.
- Runs Amazon ECS-Optimized Amazon Linux 2 AMI.
- Uses the ecsInstanceRole IAM role.
- Runs a user data script that joins the *pc-ecs-cluster* and defines a custom attribute named *purpose* with a value of *infra*. Console tasks will be placed to this instance.

STEP 1 | Go to **Services > Compute > EC2**.

STEP 2 | In the left menu, click **Auto Scaling > Launch Configurations**.

STEP 3 | Click **Create launch configuration**.

STEP 4 | In **Name**, enter a name for your launch configuration, such as **pc-infra-node**.

STEP 5 | In Amazon machine image, select **Amazon ECS-Optimized Amazon Linux 2 AMI**.

You can get a complete list of per-region Amazon ECS-optimized AMIs from here.

STEP 6 | Under instance type, select **t2.xlarge**.

STEP 7 | Under **Additional Configuration**:

1. In **IAM instance profile**, select **ecsInstanceRole**.

   If this role doesn't exist, create it. For complete details, see Amazon ECS Container Instance IAM Role.

2. Under **User data**, select **Text**, and paste the following code snippet, which installs the NFS utilities and mounts the EFS file system:

```
#!/bin/bash
cat <<'EOF' >> /etc/ecs/ecs.config
ECS_CLUSTER=pc-ecs-cluster
ECS_INSTANCE_ATTRIBUTES={"purpose": "infra"}
EOF

yum install -y nfs-utils
mkdir /twistlock_console
<CONSOLE_MOUNT_COMMAND> /twistlock_console

mkdir -p /twistlock_console/var/lib/twistlock
mkdir -p /twistlock_console/var/lib/twistlock-backup
mkdir -p /twistlock_console/var/lib/twistlock-config
```

   ECS_CLUSTER must match your cluster name. If you've named your cluster something other than **pc-ecs-cluster**, then update the user data script accordingly.

   <CONSOLE_MOUNT_COMMAND> is the Console mount command you copied from the AWS Management Console after creating your console EFS file system. The mount target must be */twistlock_console*, not the *efs* mount target provided in the sample command.

3. (Optional) In **IP Address Type**, select **Assign a public IP address to every instance**.

   With this option, you can easily SSH to this instance to troubleshoot issues.

STEP 8 | Under **Security groups**:

1. Select **Select an existing security group**.
2. Select **pc-security-group**.

STEP 9 | Under **Key pair (login)**, select an existing key pair, or create a new key pair so that you can access your instances.

STEP 10 | Click **Create launch configuration**.

Create an auto scaling group for the infrastructure node

Launch a single instance of the infrastructure node into your cluster.

STEP 1 | Go to **Services > Compute > EC2**.

STEP 2 | In the left menu, click **Auto Scaling > Auto Scaling Groups**.

STEP 3 | Click **Create an Auto Scaling group**.

STEP 4 | In **Choose launch template or configuration**:

1. In **Auto Scaling group Name**, enter **pc-infra-autoscaling**.
2. In **Launch template**, click **Switch to launch configuration**.
3. Select **pc-infra-node**.
4. Click **Next**.

STEP 5 | Under **Configure settings**:

1. In **VPC**, select your default VPC.
2. In **Subnet**, select a public subnet, such as 172.31.0.0/20.
3. Click **Skip to review**.

STEP 6 | Review the configuration and click **Create Auto Scaling Group**.

After the auto scaling group spins up (it will take some time), validate that your cluster has one container instance, where a container instance is the ECS vernacular for an EC2 instance that has joined the cluster and is ready to accept container workloads:

- Go to **Services > Containers > Elastic Container Service**. The count for **Container instances** should be 1.
- Click on the cluster, then click on the **ECS Instances** tab. In the status table, there should be a single entry. Click on the link under the **EC2 Instance** column. In the details page for the EC2 instance, record the **Public DNS**.

Copy the Prisma Cloud config file into place

The Prisma Cloud API serves the version of the configuration file used to instantiate Console. Use scp to copy *twistlock.cfg* from the Prisma Cloud release tarball to */twistlock_console/var/lib/twistlock-config* on the infrastructure node.

STEP 1 | Upload *twistlock.cfg* to the infrastructure node.

1. Go to the directory where you unpacked the Prisma Cloud release tarball.
2. Copy *twistlock.cfg* to the infrastructure node.

```
$ scp -i <PATH-TO-KEY-FILE> twistlock.cfg ec2-
user@<ECS_INFRA_NODE_DNS_NAME>:~
```

STEP 2 | SSH to the infrastructure node.

```
$ ssh -i <PATH-TO-KEY-FILE> ec2-user@<ECS_INFRA_NODE_DNS_NAME>
```

STEP 3 | Copy the *twistlock.cfg* file into place.

```
$ sudo cp twistlock.cfg /twistlock_console/var/lib/twistlock-config
```

STEP 4 | Close your SSH session.

```
$ exit
```


Create a Prisma Cloud Console task definition

Prisma Cloud provides a task definition template for Console. Download the template, then update the variables specific to your environment. Finally, load the task definition in ECS.

Prerequisites:

- The task definition provisions sufficient resources for Console to operate. The template specifies reasonable defaults. For more information, see the system requirements.

STEP 1 | Download the Prisma Cloud Compute Console task definition, and open it for editing.

STEP 2 | Update the value for *image*.

Replace the following placeholder strings with the appropriate values:

- *<ACCESS-TOKEN>* — Your Prisma Cloud access token. All characters must be lowercase.
- *<VERSION>* — Version of the Console image to use. For example, for version *20.04.177*, specify *20_04_177*. The image and tag will look like *console:console_20_04_177*.

STEP 3 | Update the value for`<ECS_INFRA_NODE_IPADDR>` to the Load Balancer's DNS name.

STEP 4 | Go to **Services > Containers > Elastic Container Service**.

STEP 5 | In the left menu, click **Task Definitions**.

STEP 6 | Click **Create new Task Definition**.

STEP 7 | Select **EC2**, and then click **Next step**.

STEP 8 | In **Step 2: Configure task and container definitions**, scroll to the bottom of the page and click **Configure via JSON**.

STEP 9 | Delete the default task definition, and replace it with the Prisma Cloud Compute Console task definition.

STEP 10 | Click **Save**.

STEP 11 | (Optional) Change the name of the task definition. By default, its name is **pc-console**.

STEP 12 | Click **Create**.

Start the Prisma Cloud Console service

Create the Console service using the previously defined task definition. A single instance of Console will run on the infrastructure node.

STEP 1 | Go to **Services > Containers > Elastic Container Service**.

STEP 2 | In the left menu, click **Clusters**.

STEP 3 | Click on your cluster.

STEP 4 | In the **Services** tab, then click **Create**.

STEP 5 | In **Step 1: Configure service**:

1. For **Launch type**, select **EC2**.
2. For **Task Definition**, select **pc-console**.
3. In **Service Name**, enter **pc-console**.
4. In **Number of tasks**, enter **1**.
5. Click **Next Step**.

STEP 6 | In **Step 2: Configure network**:

1. For **Load Balancer type**, select **Classic Load Balancer**.
2. For **Service IAM role**, leave the default **ecsServiceRole**.
3. For **Load Balancer Name**, select previously created load balancer.
4. Unselect **Enable Service discovery integration**
5. click **Next Step**.

STEP 7 | In **Step 3: Set Auto Scaling**, accept the defaults, and click **Next**.

STEP 8 | In **Step 4: Review**, click **Create Service**.

STEP 9 | Wait for the service to launch, and then click **View Service**.

STEP 10 | Wait for **Last status** to change to **RUNNING** (it can take a few minutes), and then proceed to the next step.

Configure Prisma Cloud Console

Navigate to Console's web interface, create your first admin account, and enter your license.

STEP 1 | Start a browser, then navigate to https://<LB_DNS_NAME>:8083

STEP 2 | At the login page, create your first admin account. Enter a username and password.

STEP 3 | Enter your license key, then click **Register**.

Deploy Defender

Create worker nodes in your ECS cluster, create a task definition for the Prisma Cloud Defender, and then create a service of type Daemon to deploy Defender to every node in the cluster.

If you already have worker nodes in your cluster, skip directly to creating the Defender task definition.

Create a launch configuration for worker nodes

Create a launch configuration named *pc-worker-node* that:

- Runs the Amazon ECS-Optimized Amazon Linux 2 AMI.
- Uses the ecsInstanceRole IAM role.
- Runs a user data script that joins the pc-ecs-cluster and runs the commands required to install Defender.

STEP 1 | Go to **Services > Compute > EC2**.

STEP 2 | In the left menu, click **Auto Scaling > Launch Configurations**.

STEP 3 | Click **Create Launch Configuration**

STEP 4 | In **Name**, enter a name for your launch configuration, such as **pc-worker-node**.

STEP 5 | In Amazon machine image, select **Amazon ECS-Optimized Amazon Linux 2 AMI**.

You can get a complete list of per-region Amazon ECS-optimized AMIs from here.

STEP 6 | Choose an instance type, such as **t2.medium**.

STEP 7 | Under **Additional configuration**:

1. In **IAM instance profile**, select **ecsInstanceRole**.
2. Under **User data**, select **Text**, and paste the following code snippet:

```
#!/bin/bash
echo ECS_CLUSTER=pc-ecs-cluster >> /etc/ecs/ecs.config
```

Where:

- *ECS_CLUSTER* must match your cluster name. If you've named your cluster something other than *pc_ecs_cluster*, then modify your user data script accordingly.

3. (Optional) In **IP Address Type**, select **Assign a public IP address to every instance**.

With this option, you can easily SSH to this instance to troubleshoot issues.

STEP 8 | Under **Security groups**:

1. Select **Select an existing security group**.
2. Select **pc-security-group**.

STEP 9 | Under **Key pair (login)**, select an existing key pair, or create a new key pair so that you can access your instances.

STEP 10 | Click **Create launch configuration**.

Create an auto scaling group for worker nodes

Launch two worker nodes into your cluster.

STEP 1 | Go to **Services > Compute > EC2**.

STEP 2 | In the left menu, click **Auto Scaling > Auto Scaling Groups**.

STEP 3 | Click **Create an Auto Scaling group**.

STEP 4 | In **Choose launch template or configuration**:

1. In **Auto Scaling group Name**, enter **pc-worker-autoscaling**.
2. In **Launch template**, click **Switch to launch configuration**.
3. Select **pc-worker-node**.
4. Click **Next**.

STEP 5 | Under **Configure settings**:

1. In **VPC**, select your default VPC.
2. In **Subnet**, select a public subnet, such as 172.31.0.0/20.
3. Click **Next**.

STEP 6 | In **Configure advanced options**, accept the defaults, and click **Next**.

STEP 7 | In **Configure group size and scaling policies**:

1. Set **Desired capacity** to **2**.
2. Leave **Minimum capacity** at **1**.
3. Set **Maximum capacity** to **2**.
4. Click **Skip to review**.

STEP 8 | Review the configuration and click **Create Auto Scaling Group**.

After the auto scaling group spins up (it will take some time), validate that your cluster has three container instances.

1. Go to **Services > Containers > Elastic Container Service**.
2. The count for **Container instances** in your cluster should now be a total of three.

Create a Prisma Cloud Defender task definition

Generate a task definition for Defender in Prisma Cloud Console.

STEP 1 | Log into Prisma Cloud Compute Console.

STEP 2 | Go to **Manage > Defenders > Deploy > Defenders**.

STEP 3 | In **Deployment method**, select **Orchestrator**.

STEP 4 | For orchestrator type, select **ECS**.

STEP 5 | For the name that Defender uses to connect to Console, select the DNS name of the load balancer that sits in front of Console.

STEP 6 | In **Specify a cluster name**, leave the field blank.

Console will automatically retrieve the cluster name from AWS. Only enter a value if you want to override the cluster name assigned in AWS.

STEP 7 | In **Specify ECS task name**, leave the field blank.

By default, the task name is **pc-defender**.

STEP 8 | Click **Download** to download the task definition.

STEP 9 | Log into AWS.

STEP 10 | Go to **Services > Containers > Elastic Container Service**.

STEP 11 | In the left menu, click **Task Definitions**.

STEP 12 | Click **Create new Task Definition**.

STEP 13 | In **Step 1: Select launch type compatibility**, select **EC2**, then click **Next step**.

STEP 14 | In **Step 2: Configure task and container definitions**, scroll to the bottom of the page and click **Configure via JSON**.

STEP 15 | Delete the contents of the window, and replace it with the Prisma Cloud Console task definition you just generated.

STEP 16 | Click **Save**.

STEP 17 | (Optional) Change the name of the task definition before creating it. The default name is **pc-defender**.

STEP 18 | Click **Create**.

Start the Prisma Cloud Defender service

Create the Defender service using the task definition. With Daemon scheduling, ECS schedules one Defender per node.

STEP 1 | Go to **Services > Containers > Elastic Container Service**.

STEP 2 | In the left menu, click **Clusters**.

STEP 3 | Click on your cluster.

STEP 4 | In the **Services** tab, click **Create**.

STEP 5 | In **Step 1: Configure service**:

1. For **Launch type**, select **EC2**.
2. For **Task Definition**, select **pc-defender**.
3. In **Service Name**, enter **pc-defender**.
4. In **Service Type**, select **Daemon**.
5. Click **Next Step**.

STEP 6 | In **Step 2: Configure network**, accept the defaults, and click **Next step**.

STEP 7 | In **Step 3: Set Auto Scaling**, accept the defaults, and click **Next step**.

STEP 8 | In **Step 4: Review**, click **Create Service**.

STEP 9 | Click **View Service**.

STEP 10 | Verify that you have Defenders running on each node in your ECS cluster.

1. Go to your Prisma Cloud Console and view the list of Defenders in **Manage > Defenders > Manage** There should be a total of three Defenders, one for each EC2 instance in the cluster.

Using a private registry

For maximum control over your environment, you might want to store the Console container image in your own private registry, and then install Prisma Cloud from your private registry. When the Console service is started, ECS retrieves the image from your registry. This procedure shows you how to push the Console container image to Amazon's Elastic Container Registry (ECR).

Prerequisites:

- AWS CLI is installed on your machine. It is required to push the Console image to your registry.

STEP 1 | Go to the directory where you unpacked the Prisma Cloud release tarball.

```
$ cd prisma_cloud_compute_edition/
```

STEP 2 | Load the Console image.

```
$ docker load < ./twistlock_console.tar.gz
```

STEP 3 | Go to **Services > Containers > Elastic Container Service**.

STEP 4 | In the left menu, click **Repositories**.

STEP 5 | Click **Create repository**.

STEP 6 | Follow the AWS instructions for logging in to the registry, tagging the Console image, and pushing it to your repo.

Be sure to update your Console task definition so that the value for *image* points to your private registry.

Alibaba Cloud Container Service for Kubernetes (ACK)

Edit on GitHub

Alibaba Cloud Container Service for Kubernetes (ACK) is a managed Kubernetes service. Use the standard Kubernetes install procedure to deploy Prisma Cloud to Alibaba ACK, but specify an Alibaba Cloud-specific StorageClass when configuring the deployment.

This procedure shows you how to use Helm charts to install Prisma Cloud, but all other install methods are supported.

Prerequisites

- You have provisioned an ACK cluster.

STEP 1 | Go to Releases, and copy the link to current recommended release.

STEP 2 | Download the release tarball to the system where you administer your cluster (where you run your kubectl commands).

```
$ wget <LINK_TO_CURRENT_RECOMMENDED_RELEASE_LINK>
```

STEP 3 | Unpack the Prisma Cloud release tarball.

```
$ mkdir twistlock
$ tar xvzf twistlock_<VERSION>.tar.gz -C prisma_cloud/
```

STEP 4 | Create a Helm chart for Prisma Cloud Console.

```
$ <PLATFORM>/twistcli console export kubernetes \
  --storage-class alicloud-disk-available \
  --service-type LoadBalancer \
  --helm
```

STEP 5 | Install Console.

```
$ helm install twistlock-console \
  --namespace twistlock \
  ./twistlock-console-helm.tar.gz
```

STEP 6 | Change the PersistentVolumeClaim's reclaimPolicy.

```
$ kubectl get pv
$ kubectl patch pv <pvc-name> -p '{"spec":
{"persistentVolumeReclaimPolicy":"Retain"}}'
```

STEP 7 | Get the public endpoint address for Console. When the service is fully up, the LoadBalancer's IP address is shown.

```
$ kubectl get service -w -n twistlock
```


STEP 8 | Open a browser window, and navigate to Console. By default, Console is served on HTTPS on port 8083 of the *LoadBalancer*:

```
https://<LOADBALANCER_IP_ADDR>:8083
```

STEP 9 | Continue with the rest of the install here.

Azure Kubernetes Service (AKS)

Edit on GitHub

Use the following procedure to install Prisma Cloud in an AKS cluster. This setup uses dynamic PersistentVolumeClaim provisioning using Premium Azure Disk. When creating your Kubernetes cluster, be sure to specify a VM size that supports premium storage.

Prisma Cloud doesn't support Azure Files as a storage class for persistent volumes. Use Azure Disks instead.

Prerequisites

- You have deployed an Azure Container Service (AKS) cluster. Use the --node-vm-size parameter to specify a VM size that supports Premium Azure Disks.
- You have installed Azure CLI 2.0.22 or later.
- You have downloaded the Prisma Cloud software.

STEP 1 | Use *twistcli* to generate the Prisma Cloud Console YAML configuration file, where <PLATFORM> can be *linux* or *osx*. Set the storage class to Premium Azure Disk.

```
$ <PLATFORM>/twistcli console export kubernetes \
  --storage-class managed-premium \
  --service-type LoadBalancer
```

STEP 2 | Deploy the Prisma Cloud Console in the Azure Kubernetes Service cluster.

```
$ kubectl create -f ./twistlock_console.yaml
```

STEP 3 | Wait for the service to come up completely.

```
$ kubectl get service -w -n twistlock
```

STEP 4 | Change the *reclaimPolicy* of the *PersistentVolumeClaim*.

```
  $ kubectl get pv
  $ kubectl patch pv <pvc-name> -p '{"spec":
{"persistentVolumeReclaimPolicy":"Retain"}}'
```

STEP 5 | Continue with the rest of the install here.

Amazon Elastic Kubernetes Service (EKS)

Edit on GitHub

Amazon Kubernetes Service (EKS) lets you deploy Kubernetes clusters on demand. Use our standard Kubernetes install method to deploy Prisma Cloud to EKS.

```
 If using Bottlerocket OS-based nodes for your EKS Cluster,
 pass the `--cri` flag to `twistcli` (or enable the CRI
 option in the Console UI) when generating the Defender YAML
 or Helm chart.
See <<_deploying_cri_defenders,this section>> for more
 details.
```

Prerequisites

- You have deployed an Amazon EKS cluster.
- You have downloaded the Prisma Cloud software.

STEP 1 | Generate the Prisma Cloud Compute Console deployment file.

```
$ twistcli console export kubernetes \
  --service-type LoadBalancer \
  --storage-class gp2
```

STEP 2 | Deploy Console.

```
$ kubectl create -f twistlock_console.yaml
```

STEP 3 | Wait for the service to come up completely.

```
$ kubectl get service -w -n twistlock
```

STEP 4 | Continue with the rest of the install here.

Google Kubernetes Engine (GKE)

Edit on GitHub

To install Prisma Cloud on Google Kubernetes Engine (GKE), use the standard Kubernetes install flow. Before getting started, create a *ClusterRoleBinding*, which grants the permissions required to create the Defender *DaemonSet*.

For GKE Autopilot, follow the Autopilot steps.

The Google Cloud Platform (GCP) service account that you use to create the Prisma Cloud Console resources, including Deployment controller and PersistentVolumeClaim, must have at least the **Kubernetes Engine Developer** role to be successful.

The GCP service account that you use to create the Defender resources, including *DaemonSet*, must have the Kubernetes cluster-admin role. If you try to create the Defender resources from a service account without this cluster-specific role, it will fail because the GCP **Kubernetes Engine Developer** role doesn't grant the developer sufficient permissions to create a ClusterRole (one of the Defender resources). You'll need to use an account with the GCP **Kubernetes Engine Admin** role to bind the Kubernetes cluster-admin role to your Kubernetes developer's service account.

It's probably best to create the ClusterRoleBinding before turning the cluster over any user (typically DevOps) tasked with managing and maintaining Prisma Cloud.

*Run the command in the following procedure on ANY service account that attempts to apply the Defender DaemonSet YAML or Helm chart, even if that service account already has elevated permissions with the GCP **Kubernetes Engine Admin** role. Otherwise, you'll get an error.*

The following procedure uses a service account named *your-dev-user@your-org.iam.gserviceaccount.com* that has the GCP **Kubernetes Engine Developer** role. You'll also need access to a more privileged GCP account that has the **Kubernetes Engine Admin** role to create the *ClusterRoleBinding* in your cluster.

Prerequisites

- You have deployed a GKE cluster.
- You have a Google Cloud Platform (GCP) service account with the **Kubernetes Engine Developer** role.
- You have access to a GCP account with at least the **Kubernetes Engine Admin** role.

STEP 1 | With the service account that has the GCP **Kubernetes Engine Admin** role set as the active account, run:

```
$ kubectl create clusterrolebinding your-dev-user-cluster-admin-
binding \
    --clusterrole=cluster-admin \
    --user=your-dev-user@your-org.iam.gserviceaccount.com
```


STEP 2 | With the **Kubernetes Engine Developer** service account, continue with the standard installation of Kubernetes Defenders.

If you are using GKE with ARM architecture or multiple architectures you must edit the daemonset.yaml configuration file to prepare your workloads.

Troubleshooting

If you see the following error when trying to create the Defender DaemonSet, you've probably tried to create the Defender resources from a service account that has the GCP **Kubernetes Engine Developer** role. To fix the issue, grant the proper cluster role to the service account.

```
Error from server (Forbidden): error when creating "daemonset.yaml":
 clusterroles.rbac.authorization.k8s.io is forbidden: User
 "your-dev-user@your-org.iam.gserviceaccount.com" cannot create
 clusterroles.rbac.authorization.k8s.io at the cluster scope:
 Required "container.clusterRoles.create" permission.

Error from server (Forbidden): error when creating "daemonset.yaml":
 clusterrolebindings.rbac.authorization.k8s.io is forbidden: User
 "your-dev-user@your-org.iam.gserviceaccount.com" cannot create
 clusterrolebindings.rbac.authorization.k8s.io at the cluster scope:
 Required "container.clusterRoleBindings.create" permission.
```

If you see the following error when trying to create the Defender DaemonSet, you've probably tried to create the Defender resources from a service account with the **Kubernetes Engine Admin** role. To fix the issue, grant the proper cluster role to the service account.

```
Error from server (Forbidden): error when creating "daemonset.yaml":
 clusterroles.rbac.authorization.k8s.io "twistlock-view"
 is forbidden: attempt to grant extra privileges: [{[list]
 [rbac.authorization.k8s.io] [roles] [] []} {[list]
 [rbac.authorization.k8s.io] [rolebindings] [] []} {[list]
 [rbac.authorization.k8s.io] [clusterroles] [] []} {[list]
 [rbac.authorization.k8s.io] [clusterrolebindings] [] []}]
 user=&{your-admin-user@your-org.iam.gserviceaccount.com
 [system:authenticated] map[user-assertion.cloud.google.com:
[iVWgsppUtVXaN1xToHtXpQdi5jJy6jv7BlSUZSUNTMjI2N77AaL5zQwZse0rqdu0Bz/35+6CG//82j
MoqW3Cc
+VkWmuxyGUCYcW94Ttd6euy8iVWgsppUtVXaN1xToHtXpQWhRRTxlidgQdMzAbcAAbbv2C/
uMlWs4VkzII7i9l6EEg==]]} ownerrules=[{[create] [authorization.k8s.io]
 [selfsubjectaccessreviews selfsubjectrulesreviews] [] []} {[get]
 [] [] [] [/api /api/* /apis /apis/* /healthz /openapi /openapi/
* /swagger-2.0.0.pb-v1 /swagger.json /swaggerapi /swaggerapi/* /
version /version/]}] ruleResolutionErrors=[]
```


Google Kubernetes Engine (GKE) Autopilot

Edit on GitHub

You can now install the Prisma Cloud DaemonSet Defender on your GKE **Autopilot** cluster. GKE Autopilot clusters are using cos_containerd nodes, therefore the DaemonSet must be configured with **CRI runtime**.

STEP 1 | Review the prerequisites and the procedure in the **Google Kubernetes Engine (GKE)** and the **Install Prisma Cloud on a CRI (non-Docker) cluster** sections.

STEP 2 | Use the following twistcli command to generate the YAML file for the GKE Autopilot deployment.

```
$ <PLATFORM>/twistcli console export kubernetes \
 --gke-autopilot \
 --cri \
 --cluster-address <console address> \
 --address https://<console address>:8083
```

The *--gke autopilot flag adds the 'autopilot.gke.io/no-connect: "true"' annotation to the YAML file and `--cri* flag enables the CRI option for nodes that use the Container Runtime Interface (CRI), not Docker. It also removes the '/var/lib/containers' mount from the generated file as that configuration is not required for the GKE autopilot deployment.

> *If you are using the web interface, on **Manage > Defenders > Deploy > Defenders** ensure that the **orchestrator type** is **Kubernetes**, and that the **Nodes use Container Runtime Interface (CRI), not Docker** and **GKE Autopilot deployment** are set to be **On**.*

STEP 3 | Create the **twistlock** namespace on your cluster by running the following command:

```
$ kubectl create namespace twistlock
```

STEP 4 | Deploy the updated YAML or the Helm chart on your GKE Autopilot cluster.

STEP 5 | Verify that the Defenders are deployed.

After a few minutes you should observe the nodes and running containers in Console, with Prisma Cloud Compute now protecting your cluster.

IBM Kubernetes Service (IKS)

Edit on GitHub

Use the following procedure to install Prisma Cloud in an IKS cluster. IKS uses dynamic PersistentVolumeClaim provisioning (*ibmc-file-bronze* is the default StorageClass) as well as automatic LoadBalancer configuration for the Prisma Cloud Console. You can optionally specify a StorageClass for premium file or block storage options. Use a retain storage class (not default) to ensure your storage is not destroyed even if you delete the PVC.

When installing Defenders the IKS Kubernetes version you use matters. IKS Kubernetes version 1.10 uses Docker, and 1.11+ uses containerd *as the container runtime. If using containerd, pass the --cri flag to twistcli (or enable the CRI option in the Console UI) when generating the Defender YAML or Helm chart.*

STEP 1 | Use *twistcli* to generate the Prisma Cloud Console YAML configuration file, where <PLATFORM> can be linux or osx. Optionally set the storage class to premium storage class. For IKS with Kubernetes 1.10, use our standard Kubernetes instructions. Here is an example with a premium StorageClass with the retain option.

```
$ <PLATFORM>/twistcli console export kubernetes \
  --storage-class ibmc-file-retain-silver \
  --service-type LoadBalancer
```

STEP 2 | Deploy the Prisma Cloud Console in the IBM Kubernetes Service cluster.

```
$ kubectl create -f ./twistlock_console.yaml
```

STEP 3 | Wait for the service to come up completely.

```
$ kubectl get service -w -n twistlock
```

STEP 4 | Continue with the rest of the install here.

Windows

Edit on GitHub

Prisma Cloud can secure Windows containers running on Windows Server 2016 and Windows Server 2019 hosts. A single instance of Prisma Cloud Console can simultaneously protect both Windows and Linux containers on both Windows and Linux hosts. Prisma Cloud's Intelligence Stream includes vulnerability data from Microsoft, so as new CVEs are reported, Prisma Cloud can detect them in your Windows images.

The architecture for Defender on Windows is different than Defender on Linux. The Defender runs as a Docker container on Linux, and as a Windows service on Windows. On Linux, it is implemented as runtime protection in the userspace, and on Windows it is implemented using Windows drivers. This is because there is no concept of capabilities in Windows Docker containers like there is on Linux. Defender on Windows runs as service so it can acquire the permissions it needs to secure the containers on your host. When you deploy the Defender, it appears as a service. The Defender type "Container Defender - Windows" means that Defender is capable of securing your containers, not that it's deployed as a container.

To deploy Defender on Windows, you'll copy a PowerShell script from the Prisma Cloud Console and run it on the host where you want to install Defender.

Feature matrix

The following table compares Prisma Cloud's Windows Server feature support to Linux feature support:

Platform	Vulnerabi	Complianc	Runtime defense			Firewalls	
			>Processes	>Network	>Filesystem	>CNNS	>WAAS
Linux	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Windows Server 2016	Yes	Yes	No	No	No	No	Yes
Windows Server 2019 (Host Defender)	Yes	Yes	No	No	No	No	Yes
Windows Server 2019 (Container Defender) with Docker runtime	Yes	Yes	Yes	No	No	No	No
Windows Server 2019 (Container Defender) with	Yes	Yes	Yes	No	No	No	No

Platform	Vulnerabi	Complianc	Runtime defense			Firewalls	
containerd runtime[1]							

[1]Supported on AKS only.

Windows Host Defenders support Windows compliance checks for hosts. Only Windows Container Defenders for Windows based containers support custom compliance checks.

As a quick review, Prisma Cloud runtime defense builds a model of allowed activity for each container image during a learning period. After the learning period has completed, any violation of the model triggers an action as defined by your policy (alert, prevent, block).

As Prisma Cloud builds the model, any interactive tasks that are run are logged. These interactive tasks can be viewed in each model's history tab. On Windows, Prisma Cloud can't currently detect when interactive tasks are run with the *docker exec* command, although Prisma Cloud does correctly record interactive tasks run from a shell inside a container with the *docker run -it <IMAGE> sh* command. No matter how the interactive task is run, however, the model will correctly allow a process if it's in learning mode, and it will take action if the model is violated when in enforcement mode.

Windows Container Defenders scan both the containers and the hosts where they run for vulnerabilities.

Deploying Defender on Windows with Docker runtime

Prisma Cloud Console must be first installed on a Linux host. Prisma Cloud Defenders are then installed on each Windows host you want to protect. For more information about installing Console, see Getting Started. The Onebox install is the fastest way to get Console running on a stand-alone Linux machine.

Defenders are deployed with with a PowerShell 64-bit script, *defender.ps1*, which downloads the necessary files from Console. Defender is registered as a Windows service.

Run the Prisma Cloud Defender deployment PowerShell script from a Windows PowerShell 64-bit shell.

Prisma Cloud Windows container defenders are tested and supported for GKE Windows server containers.

After the install is completed, Prisma Cloud files can be found in the following locations:

- *C:\Program Files\Prisma Cloud*
- *C:\ProgramData\Prisma Cloud*

Prerequisites:

- Windows Server 2016 or Windows Server 2019. Prisma Cloud is not supported on Windows 10 or Hyper-V.
- Docker for Windows (1.12.2-cs2-ws-beta) or higher. For more information about installing Docker on Windows, see Windows Containers on Windows Server.

STEP 1 | Log into Console

STEP 2 | Go to **Manage > Defenders > Deploy**

STEP 3 | Select **Single Defender**

STEP 4 | In **Choose the Defender type**, select **Container Defender - Windows**

STEP 5 | Copy the curl script and run it on your host to install Windows Defender

If you install Windows locally on your laptop, the 'netsh' commands are not needed. They are only applicable to the GCE environment.

Deploy Container Defender on Windows with containerd runtime

You can also deploy the Windows container defender to protect your containers running on **Azure Kubernetes Service (AKS)** Windows nodes with **containerd** runtime. By installing the Defender you will be able to view the running containers and images on the Radar and leverage Prisma Cloud Runtime Defense capabilities on the running containers.

Prerequisites:

- Make sure you are using Windows Server 2019 with containerd runtime.
- The nodes are part of an Azure Kubernetes Service (AKS) Windows Server node pool
- Learn more about Using containerd with Windows Server node pools (preview)

STEP 1 | Log into Console.

STEP 2 | Go to **Manage > Defenders > Deploy**

STEP 3 | Select **Single Defender**

STEP 4 | In **Choose the Defender type**, select **Container Defender - Windows**

STEP 5 | Set the option for **Node is using containerd, not Docker** to **On**

STEP 6 | Copy the curl script and run it on your host to install Windows Defender

Twistcli can't be used on Windows machines running containerd.

Registry scanning

To scan Windows images in your registry, you must install at least one Windows Defender. Prisma Cloud automatically distributes the scan job across available Defenders. To scan registries that hold both Windows and Linux images, install at least one Linux Defender and one Windows Defender in your environment.

Registry scan settings can include a mix of both Defenders running on hosts with Docker Engine and containerd as scanners.

Uninstalling Defender

You can uninstall Defender directly from the Console UI.

You can also manually uninstall Defender from the command line by running:

```
C:\Program Files\Twistlock\scripts\defender.ps1 -uninstall
```

Since Defender runs as a Windows service, decommissioning it will stop the service. Some remnant files might need to be deleted manually.

STEP 1 | Go to **Manage > Defenders > Manage**.

This page shows a list of Defenders deployed in your environment and connected to Console.

STEP 2 | Click the **Decommission** button.

Limitations

Be aware of the following limitations:

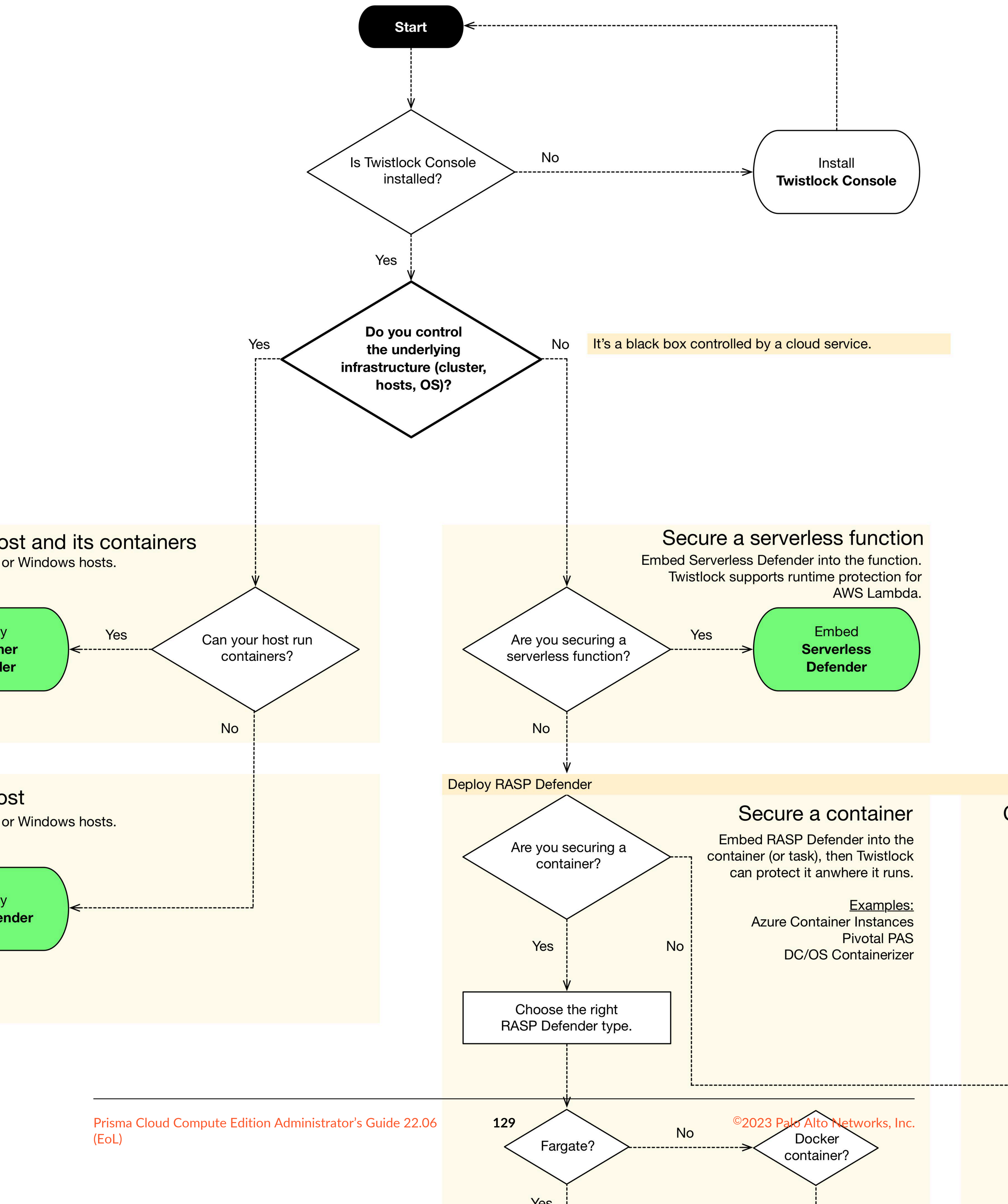
- Windows Defenders support Windows compliance checks for hosts and custom compliance checks only. Image and container compliance checks aren't supported.
- Windows requires the host OS version to match the container OS version. If you want to run a container based on a newer Windows build, make sure you have an equivalent host build. Otherwise, you can use Hyper-V isolation to run older containers on new host builds. For more information, see Windows containers version compatibility.

Defender types

Edit on GitHub

Defenders enforce the policies you set in Console. They come in a number of different flavors. Each flavor is designed for protecting specific types of cloud-native resources and for optimal deployment into the environment, with full support for automated workflows. Use the following flow chart to choose the best Defender for the job.

In general, deploy Container Defender whenever you can. It offers the most features, it can simultaneously protect both containers and host, and nothing needs to be embedded inside your containers for Defender to be able to protect them.


Start
Is Twistlock Console installed?
No
Install
Twistlock Console
Yes
Do you control the underlying infrastructure (cluster, hosts, OS)?
Yes
No
It's a black box controlled by a cloud service.
Secure a serverless function
Embed Serverless Defender into the function. Twistlock supports runtime protection for AWS Lambda.
Can your host run containers?
Yes
No
Are you securing a serverless function?
Yes
Embed
Serverless
Defender
No
or Windows hosts.
or Windows hosts.
Deploy RASP Defender
Secure a container
Embed RASP Defender into the container (or task), then Twistlock can protect it anwhere it runs.
Are you securing a container?
Examples:
Azure Container Instances
Pivotal PAS
DC/OS Containerizer
Yes
No
Choose the right RASP Defender type.
Fargate?
No
Docker container?
Yes

Container Defender (Linux and Windows)

Install Container Defender on any host that runs a container workload. Container Defender protects both your containers and the underlying host. Docker must be installed on the host because this Defender type runs as a container.

Container Defender offers the richest set of capabilities. The deployment is also the simplest. After deploying Container Defender to a host, it can immediately protect and monitor your containers and host. No additional steps are required to rebuild your containers with an agent inside. Container Defender should always be your first choice whenever possible.

There are some minimum requirements to run Container Defender. You should have full control over the host where Container Defender runs. It must be able to run alongside the other containers on the host with select kernel capabilities. And it must be able to run in the host's network and process namespace.

Deploy one Container Defender per host. Container Defender can be deployed in several ways:

- With cluster constructs. Container orchestrators often provide native capabilities for deploying agents, such as Defender, to every node in the cluster. Prisma Cloud leverages these capabilities to install Defender. Kubernetes and OpenShift, for example, offer DaemonSets As such, Container Defender is deployed as a DaemonSet on Kubernetes
- As a stand-alone entity. Stand-alone Container Defenders are installed on hosts that are not part of a cluster.

Host Defender (Linux and Windows)

Host Defender utilizes Prisma Cloud's model-based approach for protecting hosts that do not run containers. This Defender type lets you extend Prisma Cloud to protect all the hosts in your environment, regardless of their purpose. Defender runs as a systemd service on Linux and a Windows service on Windows. If Docker is deployed on your host, deploy a container Defender to protect the containers and the underlying host.

Deploy one Host Defender per host. Do not deploy Host Defender if you've already deployed Container Defender to a host. Container Defender offers the same host protection capabilities as Host Defender.

Serverless Defender

Serverless Defenders offer runtime protection for AWS Lambda functions and Azure Functions. Serverless Defender must be embedded inside your functions. Deploy one Serverless Defender per function.

App-Embedded Defender

Deploy App-Embedded Defender anywhere you can run a container, but you can't run Container Defender. Container-on-demand services are a typical use case for App-Embedded Defender. They abstract away the underlying cluster, host, operating system, and software modules (such as Docker Engine) and present them as a single black box. Hooks into the operating system that Container Defender needs to monitor and protect resources aren't available in these environments. Instead, embed App-Embedded Defender directly inside the container to establish a point of control. Prisma Cloud supports an automated workflows for embedding App-Embedded Defenders.

Deploy one App-Embedded Defender per container. For Fargate, deploy one Defender per task.

App-Embedded Defender offers three deployment mechanisms: Fargate, Dockerfile, and manual.

Fargate

If you have an AWS Fargate task, deploy App-Embedded Fargate Defender.

A key attribute of the App-Embedded Fargate Defender is that you don't need to change how the container images in the task are built. The process of embedding the App-Embedded Defender simply manipulates the task definition to inject a Prisma Cloud sidecar container, and start existing task containers with a new entry point, where the entry point binary is hosted by the Prisma Cloud sidecar container. The transformation of an unprotected task to a protected task takes place at the task definition level only. The container images in the task don't need to be manually modified. This streamlined approach means that you don't need to maintain two versions of an image (protected and unprotected). You simply maintain the unprotected version, and when you protect a task, Prisma Cloud dynamically injects App-Embedded Defender into it.

The Prisma Cloud sidecar container has a couple of jobs:

- Hosts the Defender binary that gets injected into containers in the task.
- Proxies all communication to Console. Even if you have multiple containers in a task, it appears as a single entity in Console's dashboard.
- Synchronizes policy with Console and sends alerts to Console.

Dockerfile

The Docker image format, separate from the runtime, is becoming a universal runnable artifact. If you're not using Fargate, but something else that runs a Docker image, such as Azure Container Instances, use the App-Embedded Defender with the Dockerfile method.

Provide a Dockerfile, and Prisma Cloud returns a new version of the Dockerfile in a bundle. Rebuild the new Dockerfile to embed Prisma Cloud into the container image. When the container starts, Prisma Cloud App-Embedded Defender starts as the parent process in the container, and it immediately invokes your program as its child.

There are two big differences between this approach and the Fargate approach:

- With the Fargate approach, you don't change the actual image. With the Dockerfile approach, you have the original image and a new protected image. You must modify the way your containers are built to embed App-Embedded Defender into them. You need to make sure you tag and deploy the right image.
- Each Defender binary makes it's own connection to Console. In the Console dashboard, they are each counted as unique applications.

Nothing prevents you from protecting a Fargate task using the Dockerfile approach, but it's inefficient.

Manual

Use the manual approach to protect almost any type of runtime. If you're not running a Docker image, but you still want Prisma Cloud to protect it, deploy App-Embedded Defender with the manual method. Download the App-Embedded Defender, set up the required environment variables, then start your program as an argument to the App-Embedded Defender.

If you choose the manual approach, you have to figure out how deploy, maintain, and upgrade your app on your own. While the configuration is more complicated, it's also the most universal option because you can protect almost any executable.

Tanzu Application Service Defender

Tanzu Application Service (TAS) Defenders run on your TAS infrastructure. TAS Defenders provide nearly all the same capabilities as Container Defenders, as well as the ability to scan droplets in your blobstores for vulnerabilities. For specific differences between TAS Defenders and Container Defenders, see the TAS Defender install article.

The TAS Defender is delivered as a tile that can be installed from your TAS Ops Manager Installation Dashboard.

Defender capabilities

The following table summarizes the key functional differences between Defender types.

Capabilities		Defender type			
		Container[1]	Host	Serverless	App-Embedded
Deployment methods	**Console UI**	Y	Y	Y	Y
	API	Y	Y	Y	Y
	twistcli	Y			Y
Vulnerability management		Y	Y	Y[2]	Y[3]
Compliance		Y	Y	Y[2]	Y[4]
Runtime defense	**Behavioral modeling**	Y			
	Process	Y	Y	Y	Y
	Networking	Y	Y	Y	Y
	File system	Y	Y	Y	Y
	Forensics	Y	Y		Y
Access control	**Kubernetes auditing**	Y[5]			Y[5]

Capabilities		Defender type			
	Admission control	Y			
Firewalls	**WAAS**	Y	Y	Y	Y
	CNNS	Y	Y		
Radar (visualization)	**Radar**	Y	Y	Y	

[1] Container Defender supports all Host Defender capabilities.

[2] Normally Defender scans workloads for vulnerabilities and compliance issues. For serverless functions, Console does the scanning. In the Console, create a configuration that points to your repository of functions in your cloud provider.

[3] Vulnerability management for deployed images only. Registry scanning by app-embedded Defenders is not supported.

[4] Image compliance and custom compliance checks only. The trusted images feature isn't supported.

[5] Kuberentes auditing is done by the Console, and not by the Defenders. In the Console, enable Kubernetes auditing and create a configuration that points to your cluster.

Connectivity

Defender must be able to communicate with Console over the network because it pulls policies down and sends data (alerts, events, etc) back to Console.

In simple environments, where your hosts run on the same subnet, you can connect to Console using the host's IP address or hostname. In more complex environments, where your setup runs in the cloud, it can be more difficult to determine how Defender connects to Console. When setting up Defender, use whichever address routes over your configuration and lets Defender connect to Console.

For example, Console might run in one Virtual Private Cloud (VPC) in AWS, and your containers might run in another VPC. Each VPC might have a different RFC1918 address space, and communication between VPCs might be limited to specific ports in a security group. Use whichever address lets Defender connect to Console. It might be a publicly exposed IP address, a hostname registered with a DNS, or a private address NAT'ed to the actual IP address assigned to Console. For more information about setting up name resolution in complex networks, see Best practices for for DNS and certificate management.

Deployment scenarios

Install the Defender type that best secures the resource you want to protect. Install Defender on each host that you want Prisma Cloud to protect. Container Defenders protect both the containers and the underlying host. Host Defenders are designed for legacy hosts that have no capability for running containers. Host Defenders protect the host only. For serverless technologies, embed Defender directly in the resource.

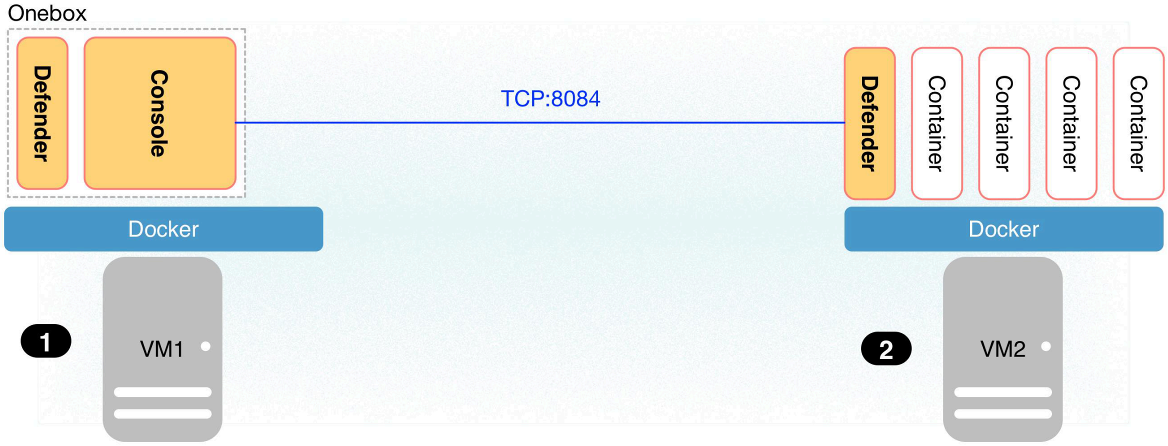
The scenarios here show examples of how the various Defender types can be deployed.

Scenario #1

Stand-alone Container Defenders are installed on hosts that are not part of a cluster. Stand-alone Container Defenders might be required in any number of situations.

For example, a very simple evaluation setup might consist of two virtual machines.

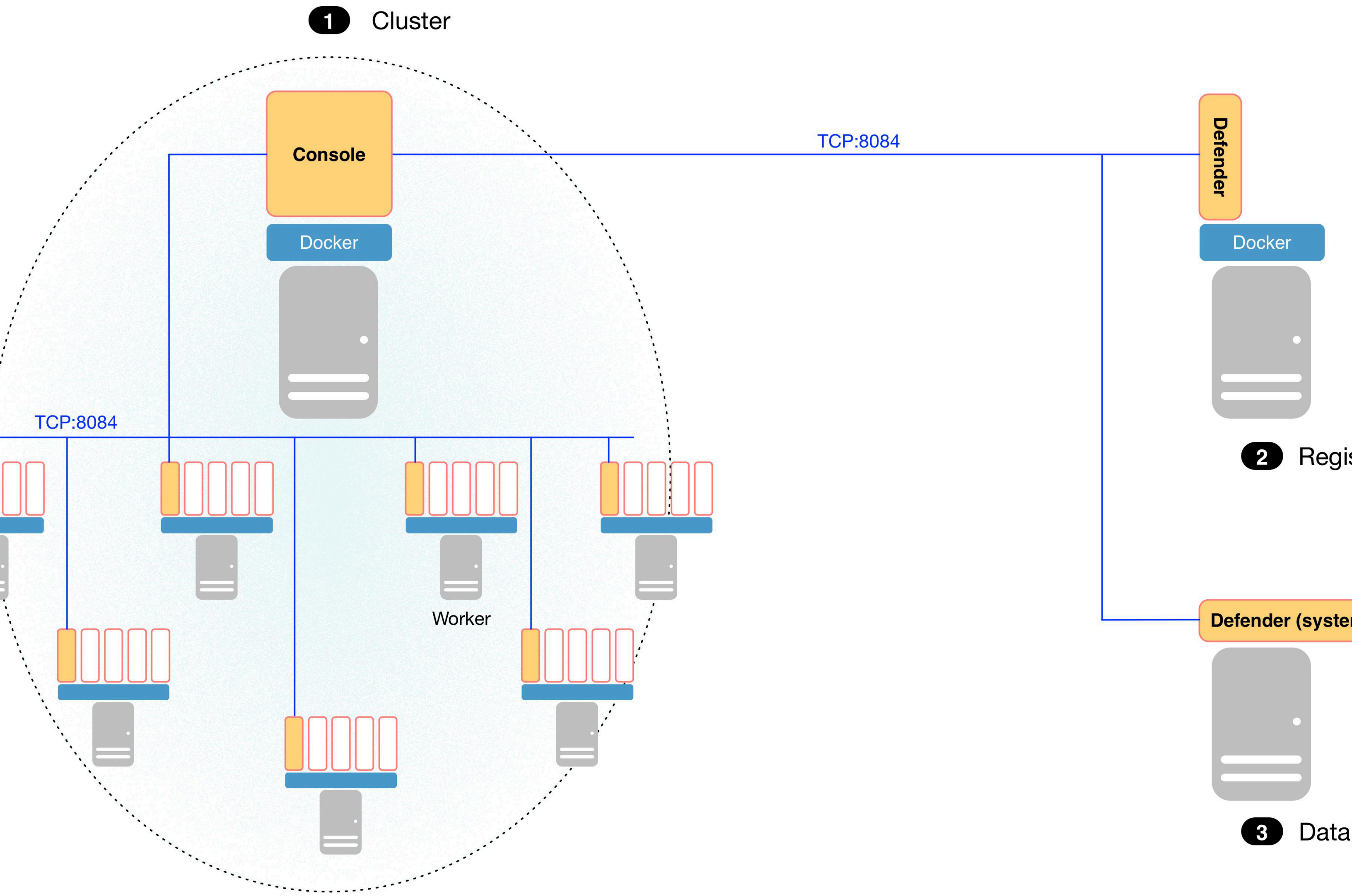
- **1** — One VM runs Onebox (Console + Container Defender).
- **2** — To protect the container workload on a second VM, install another stand-alone Container Defender.



Scenario #2

For clusters, such as Kubernetes and OpenShift, Prisma Cloud utilizes orchestrator-native constructs, such as DaemonSets, to guarantee that Defender runs on every node in the cluster. For example, the following setup has three different types of Defender deployments.

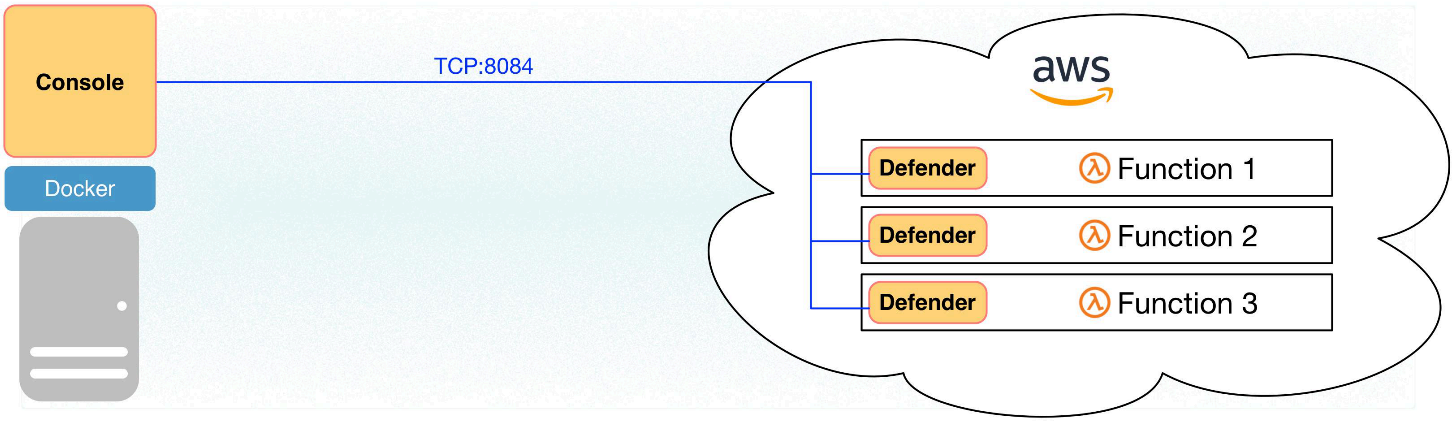
- **1** — In the cluster, Container Defenders are deployed as a DaemonSet. (Assume this is a Kubernetes cluster; it would be a similar construct, but with a different name, for AWS ECS etc).
- **2** — On the host dedicated to scanning registry images, which runs outside the cluster, a stand-alone Container Defender is deployed.
- **3** — On the legacy database server, which doesn't run containers at all, a Host Defender is deployed. Host Defenders are a type of stand-alone Defender that run on hosts that don't have Docker installed.



Scenario #3

Managed services that run functions and containers on-demand isolate the runtime from the underlying infrastructure. In these types of environments, Defender cannot access the host's operating system with elevated privileges to observe activity and enforce policies in the runtime. Instead, Defender must be built into the runtime, and control application execution and detect and prevent real-time attacks from within. App Embedded Defender can be deployed to protect any container, regardless of the platform or runtime, whether it's Docker, runC, or Diego on Tanzu Application Service.

- **1** — Serverless Defender is embedded into each AWS Lambda function.

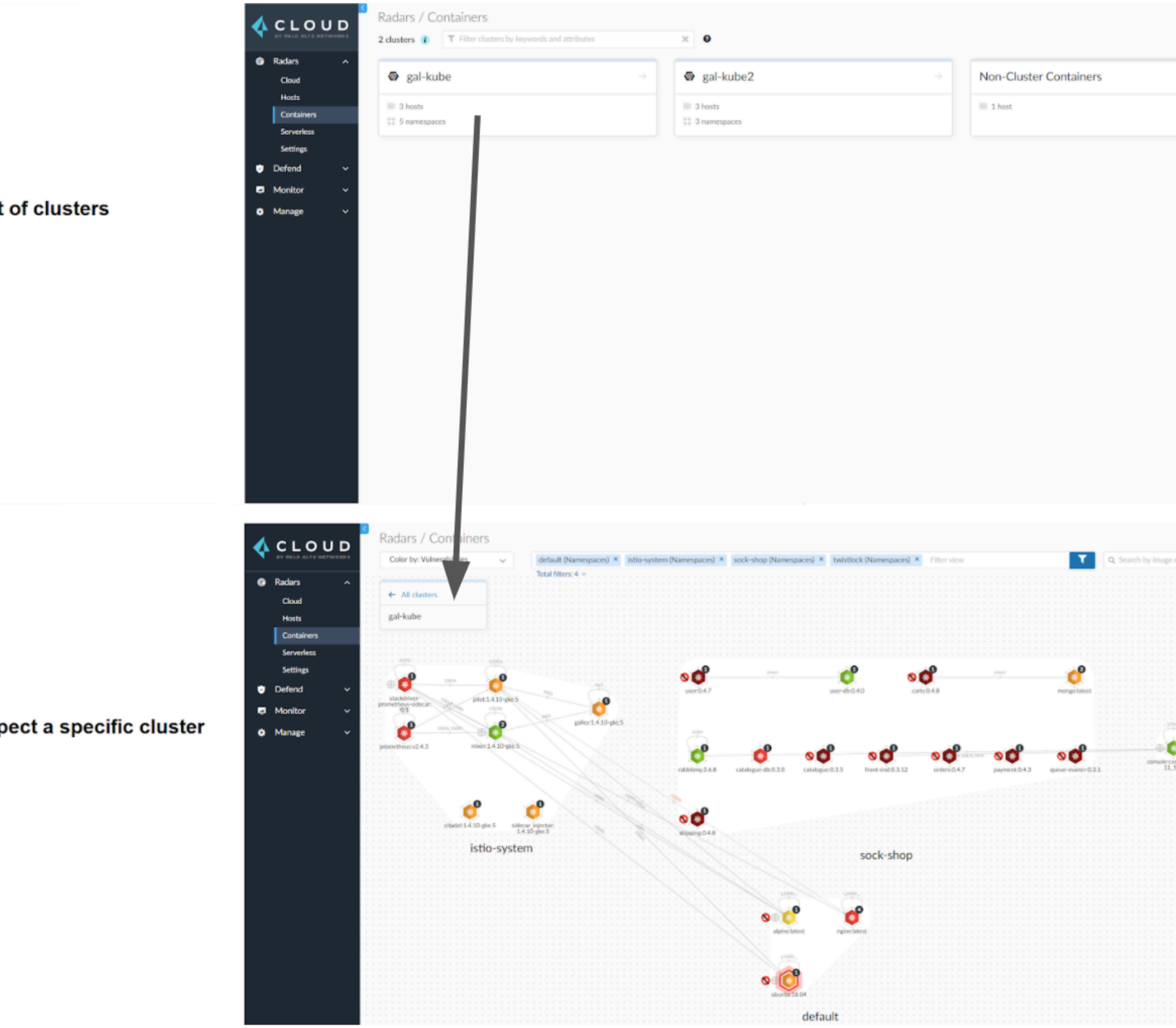


1 AWS Lambda functions

Cluster Context

Edit on GitHub

Prisma Cloud can segment your environment by cluster. For example, you might have three clusters: test, staging, and production. The cluster pivot in Prisma Cloud lets you inspect resources and administer security policy on a per-cluster basis.



Cluster awareness across the product

Radar lets you explore your environment cluster-by-cluster. Various scan reports and audits include the relevant cluster name to provide environment context. You can also create stored filters (also known as collections) based on cluster names. Finally, you can scope policy by cluster. Vulnerability and compliance rules for container images and hosts, runtime rules for container images, and trusted images rules can all be scoped by cluster name.

Determine cluster name

Defenders in each DaemonSet are responsible for reporting which resources belong to which cluster. When deploying a Defender DaemonSet, Prisma Cloud tries to determine the cluster name through introspection. First, it tries to retrieve the cluster name from the cloud provider. As a fallback, it tries to retrieve the name from the kubeconfig file (the cluster name will be taked from the *server* field). Finally, you can override these mechanisms by manually specifying a cluster name when deploying your Defender DaemonSet.

Both the Prisma Cloud UI and twistcli tool accept an option for manually specifying a cluster name. Let Prisma Cloud automatically detect the name for provider-managed clusters. Manually specify names for self-managed clusters, such as those built with kops.

There are some things to consider when manually naming clusters:

- If you specify the same name for two or more clusters, they're treated as a single cluster.
- For GCP, if you have clusters with the same name in different projects, they're treated as a single cluster. Consider manually specifying a different name for each cluster.
- Manually specifying names isn't supported in **Manage > Defenders > Manage > DaemonSet**. This page lets you deploy and manage DaemonSets directly from the Prisma Cloud UI. For this deployment flow, cluster names are retrieved from the cloud provider or the supplied kubeconfig only.

If you wish to change the cluster name determined by Prisma Cloud Compute, or the name you manually set for the cluster, you must redeploy the Defenders DaemonSet and specify the new name. Notice that after changing the name, historical records for audits and incidents, will keep the cluster name from their creation time. The new cluster name will only apply for future records. Also, if you already created collections using the old cluster name, these need to be manually updated with the new name.

Install Defender

Edit on GitHub

This section shows you how to install Defender. The type of Defender you install depends on what you're securing.

- Single Container Defender
- Cluster Container Defender
- App-Embedded Defender
- App Embedded Defender for Fargate
- VMware Tanzu Application Service (TAS) Defender
- Serverless Defender
- Serverless Defender (Lambda layer)
- Auto-defend serverless functions
- Host Defender
- Auto-defend hosts

Install a single Container Defender

Edit on GitHub

Install Container Defender on each host that you want Prisma Cloud to protect.

Single Container Defenders can be configured in the Console UI, and then deployed with a curl-bash script. Alternatively, you can use twistcli to configure and deploy Defender directly on a host.

Install a single Container Defender (Console UI)

Configure how a single Container Defender will be installed, and then install it with the resulting curl-bash script.

Prerequisites:

- Your system meets all minimum system requirements.
- You have already installed Console.
- Port 8083 is open on the host where Console runs. Port 8083 serves the API. Port 8083 is the default setting, but it is customizable when first installing Console. When deploying Defender you can configure it to communicate to Console via a proxy.
- Port 8084 is open on the host where Console runs. Console and Defender communicate with each other over a web socket on port 8084. Defender initiates the connection. Port 8084 is the default setting, but it is customizable when first installing Console. Defender can also be configured to communicate to Console via a proxy.
- Console can be accessed over the network from the host where you want to install Defender.
- You have sudo access to the host where Defender will be installed.

STEP 1 | Verify that the host machine where you install Defender can connect to Console.

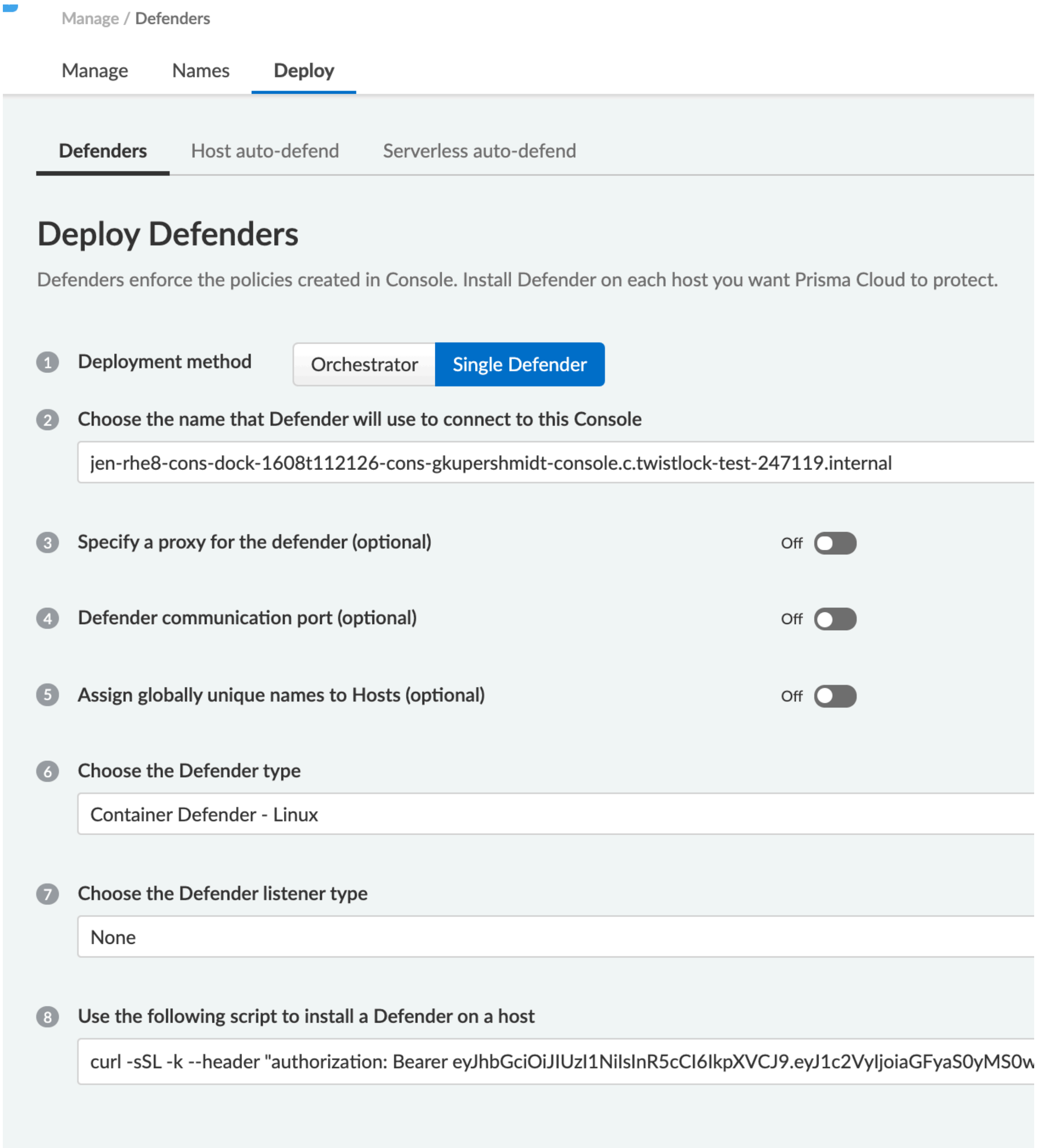
Copy the path to Console from **Manage > System > Utilities**.

```
$ curl -sk -D - https://<CONSOLE_IP_ADDRESS|HOSTNAME>:8083/api/v1/
_ping
```

If curl returns an HTTP response status code of 200, you have connectivity to Console. If you customized the setup when you installed Console, you might need to specify a different port.

STEP 2 | Log into Console.

STEP 3 | Go to **Manage > Defenders > Deploy**.



1. In the first drop-down menu (2), select the way Defender connects to Console.

   A list of IP addresses and hostnames are pre-populated in the drop-down list. If none of the items are valid, go to **Manage > Defenders > Names**, and add a new Subject

Alternative Name (SAN) to Console's certificate. After adding a SAN, your IP address or hostname will be available in the drop-down list.

> *Selecting an IP address in a evaluation setup is acceptable, but using a DNS name is more resilient. If you select Console's IP address, and Console's IP address changes, your Defenders will no longer be able to communicate with Console.*

2. (Optional) Set a proxy (3) for the Defender to use for the communication with the Console.
3. (Optional) Set a custom communication port (4) for the Defender to use.
4. (Optional) Set **Assign globally unique names to Hosts** to **ON** when you have multiple hosts that can have the same hostname.

> *After setting the toggle to **ON**, Prisma Cloud appends a unique identifier, such as ResourceId, to the host's DNS name. For example, an AWS EC2 host would have the following name: Ip-171-29-1-244.ec2internal-i-04a1dcee6bd148e2d.*

5. In the second drop-down list (5), select the Defender type. Both Linux and Windows platforms are supported.
6. In the third drop-down list (6), leave the listener type set to **None**.
7. In the final field (7), copy the install command, which is generated according to the options you selected.

STEP 4 | On the host where you want to install Defender, paste the command into a shell window, and run it.

Install a single Container Defender (twistcli)

Use twistcli to install a single Container Defender on a Linux host.

> *Anywhere <CONSOLE> is used, be sure to specify both the address and port number for Console's API. By default, the port is 8083. For example, https://<CONSOLE>:8083.*

Prerequisites:

- Your system meets all minimum system requirements.
- You have already installed Console.
- Port 8083 is open on the host where Console runs. Port 8083 serves the API. Port 8083 is the default setting, but it is customizable when first installing Console. When deploying Defender, you can configure it to communicate to Console via a proxy.
- Port 8084 is open on the host where Console runs. Console and Defender communicate with each other over a web socket on port 8084. Defender initiates the connection. Port 8084 is the default setting, but it is customizable when first installing Console. When deploying Defender, you can configure it to communicate to Console via a proxy.
- Console can be accessed over the network from the host where you want to install Defender.
- You have sudo access to the host where Defender will be installed.
- You've created a service account with the Defender Manager role. twistcl uses the service account to access Console.

STEP 1 | Verify that the host machine where you install Defender can connect to Console.

Copy the path to Console from **Manage > System > Utilities**.

```
$ curl -sk -D - https://<CONSOLE>/api/v1/_ping
```

If curl returns an HTTP response status code of 200, you have connectivity to Console.

STEP 2 | SSH to the host where you want to install Defender.

STEP 3 | Download twistcli.

```
$ curl -k \
  -u <USER> \
  -L \
  -o twistcli \
  https://<CONSOLE>/api/v1/util/twistcli
```

STEP 4 | Make the twistcli binary executable.

```
$ chmod a+x ./twistcli
```

STEP 5 | Install Defender.

```
$ sudo ./twistcli defender install standalone container-linux \
  --address https://<CONSOLE> \
  --user <USER>
```

STEP 6 | Verify Defender was installed correctly.

```
$ sudo docker ps
CONTAINER ID   IMAGE                                  COMMAND
            CREATED          STATUS         PORTS     NAMES
677c9883c4b6   twistlock/private:defender_21_04_333   "/usr/
local/bin/defe…"   11 seconds ago   Up 10 seconds
 twistlock_defender_21_04_333
```

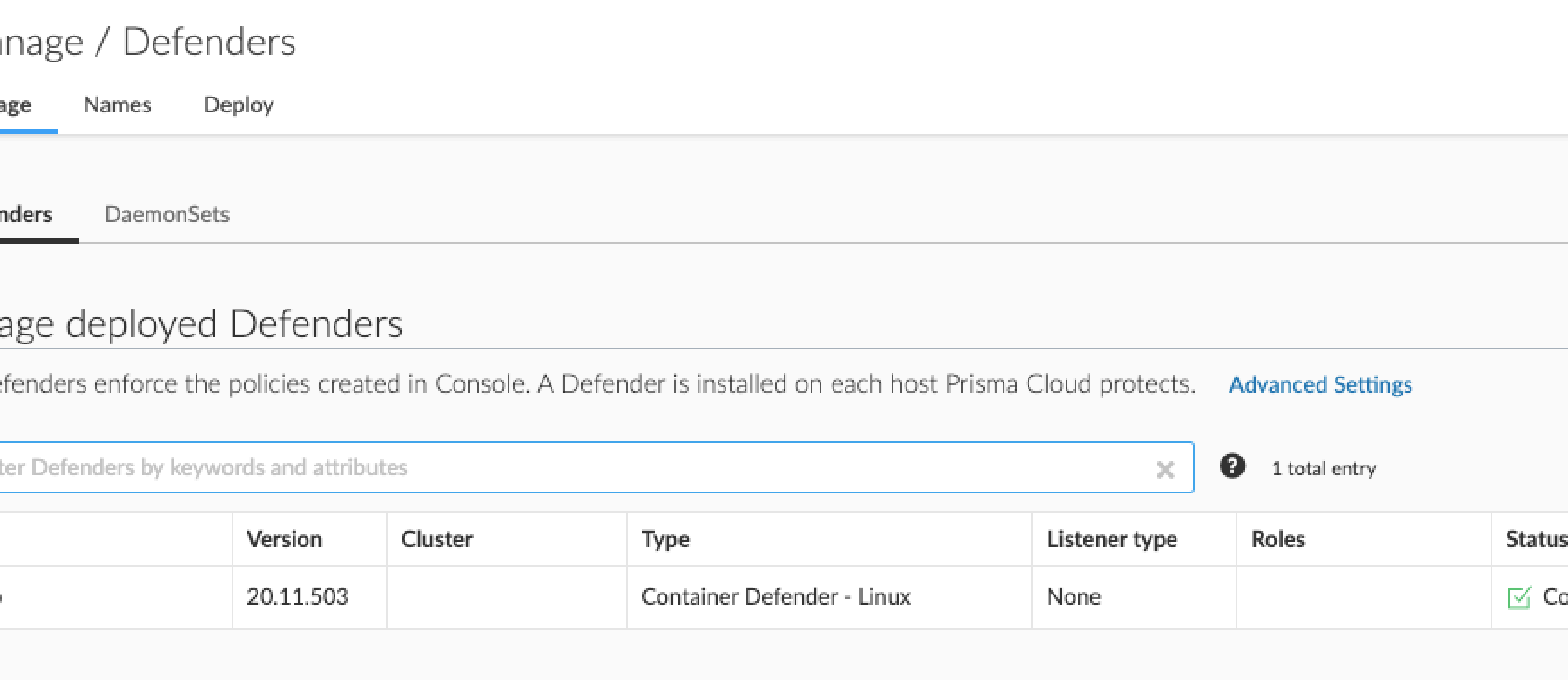
Verify the install

Verify that Defender is installed and connected to Console.

Defender can be deployed and run with full functionality when dockerd is configured with SELinux enabled (--selinux-enabled=true). All features will work normally and without any additional configuration steps required. Prisma Cloud automatically detects the SELinux configuration on a per-host basis and self-configures itself as needed. No action is needed from the user.

In Console, go to **Manage > Defenders > Manage**.

Your new Defender should be listed in the table, and the status box should be green and checked.



Automatically Install Container Defender in a Cluster

Edit on GitHub

Container orchestrators provide native capabilities for deploying agents, such as Defender, to every node in the cluster. Prisma Cloud leverages these capabilities to install Defender.

The process for deploying Container Defender to a cluster can be found in the dedicated orchestrator-specific install guides.

If you wish to automate the defenders deployment process to a cluster, or you don't have kubectl access to your cluster (or oc access for OpenShift), you can deploy Defender DaemonSets directly from the Console UI.

*This Defender install flow doesn't let you manually configure a cluster name. Cluster names let you segment your views of the environment. For most cases, this shouldn't be a problem because if you're deploying to a managed cluster, then Prisma Cloud retrieves the cluster name directly from the cloud provider. If you must manually specify a name, deploy your Defenders from **Manage > Defenders > Deploy > DaemonSet** or use twistcli.*

*If your clusters use **ARM architecture or multiple architectures** on Google Kubernetes Engine (GKE) you can't use the following procedure to automatically deploy the defenders. Instead, use the manual installation procedure for Kubernetes and edit the daemonset.yaml configuration file to prepare your workloads.*

Deploy Defender DaemonSet using kubeconfig

Prerequisites:

- You've created a kubeconfig credential for your cluster so that Prisma Cloud can access it to deploy the Defender DaemonSet.

Deployment process:

STEP 1 | Log into Prisma Cloud Console.

STEP 2 | Go to **Manage** > **Defenders** > **Manage**.

STEP 3 | Click **DaemonSets**.

STEP 4 | For each cluster in the table, click **Actions** > **Deploy**.

The table shows a count of deployed Defenders and their version number.

Deploy Defender DaemonSet for GKE

Prerequisites:

- You deployed a GKE cluster
- You created a corresponding Service Account key in JSON format. The Service Account should have the following permissions:
  - Editor
  - Compute Storage Admin
  - Kubernetes Engine Admin
  - Service Account Token Creator
- You created a GCP credential for your cluster so that Prisma Cloud can access it to deploy the Defender DaemonSet:
  1. Log into Prisma Cloud Console.
  2. Go to **Manage** > **Authentication** > **Credentials Store**
  3. Click **Add credential** button
  4. Select type **GCP** and credential level, then copy the content of the JSON Service Account key into the Service Account line (take it all including brackets).

To deploy the Defender DaemonSet, use the following procedure.

STEP 1 | Log into Prisma Cloud Console.

STEP 2 | Go to **Manage** > **Defenders** > **Manage** > **DaemonSets**.

When the page is loaded, multiple rows of K8S clusters visible with SA credentials are displayed.

For GCP organizations with hundreds of projects, using organization level credentials might affect the performance of the page and the time to load the clusters. Therefore, the best approach to reduce the time and to avoid potential timeouts, is to divide the projects within your organization into multiple GCP folders. Then, create a service account and credential for each one of them.

STEP 3 | Verify that the status is **Success** and the Defender count is 0/0 for all relevant clusters.

STEP 4 | For each cluster, click **Actions** > **Deploy**.

STEP 5 | Refresh the view and verify that for each cluster the version is the correct, the status is **Success**, and the Defender count is equal to the number of cluster nodes.

App-Embedded Defender

Edit on GitHub

App-Embedded Defenders monitor and protect your containers to ensure they execute as designed. Deploy App-Embedded Defender anywhere you can run a container, but can't deploy Container Defender. App-Embedded Defenders are typically used to protect containers that run on container-on-demand services, such as Google Cloud Run and Azure Container Instances.

To learn when to use App-Embedded Defenders, see Defender types.

To learn more about App-Embedded Defender's capabilities, see:

- Vulnerability scanning for App-Embedded
- Compliance scanning for App-Embedded
- Runtime defense for App-Embedded
- Protecting front-end containers at runtime with WAAS

App-Embedded Defender is the only supported option for securing containers at runtime when you're using nested virtualization, also known as Docker-in-Docker. Docker-in-Docker is a setup where you have a Docker container that itself has Docker installed, and from within the container you use Docker to pull images, build images, run containers, and so on. To secure the containers inside a container, use App-Embedded Defender.

Securing containers

To secure a container, embed the App-Embedded Defender into it. You can embed App-Embedded Defenders with the Console UI, twistcli, or Prisma Cloud API. App-Embedded Defender has been tested on Azure Container Instances, Google Cloud Run, and Fargate on EKS.

The steps are:

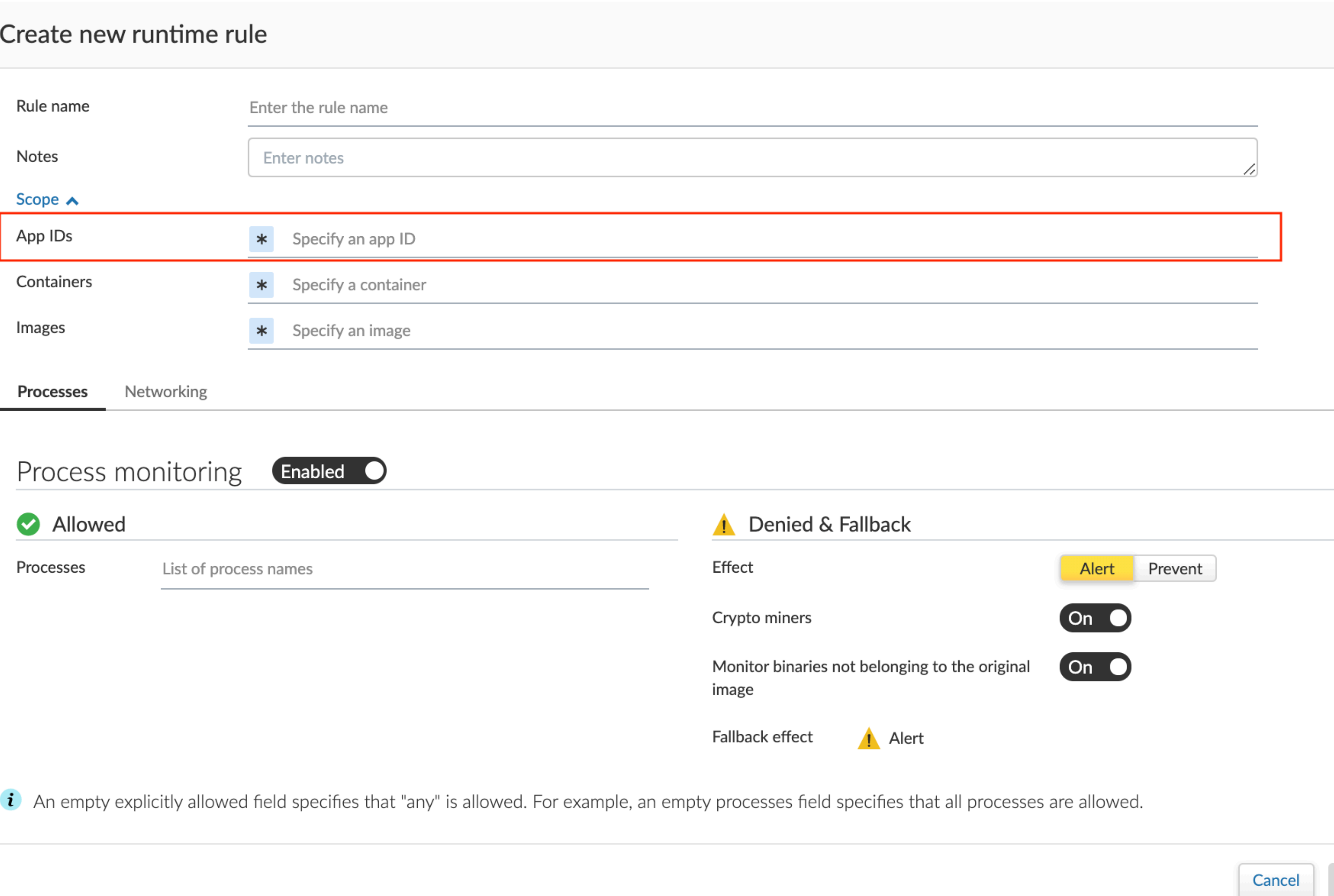
1. Define your policy in Prisma Cloud Console.

   App-Embedded Defenders dynamically retrieve rules from Console as they are updated. You can embed the App-Embedded Defender into a task with a simple initial policy, and then refine it later, as needed.
2. Embed the App-Embedded Defender into the container.
3. Start the service that runs your container.

The embed process takes a Dockerfile as input, and returns a ZIP file with an augmented Dockerfile and App-Embedded Defender binaries. Rebuild your container image with the new Dockerfile to complete the embedding process. The embed process modifies the container's entrypoint to run App-Embedded Defender. The App-Embedded Defender, in turn, runs the original entrypoint program under its control.

When embedding App-Embedded Defender, specify a unique identifier for your container image. This gives you a way to uniquely identify the App-Embedded Defender in the environment.

When securing your apps with runtime rules, target rules to apps using the App ID. (Because the App-Embedded Defender runs inside the container, it can't reliably get information such as image and container names.)



App ID

When you deploy an App-Embedded Defender, it's embedded inside the container. The embed process modifies the container's entrypoint to run App-Embedded Defender first, which in turn starts the original entrypoint program.

When App-Embedded Defender sends scan data back to Console, it must correlate it to an image. Because App-Embedded Defender runs inside the container, it can't retrieve any information about the image, specifically the image name and image ID. As such, the deployment flow sets an image name and image ID, and embeds this information alongside the App-Embedded Defender.

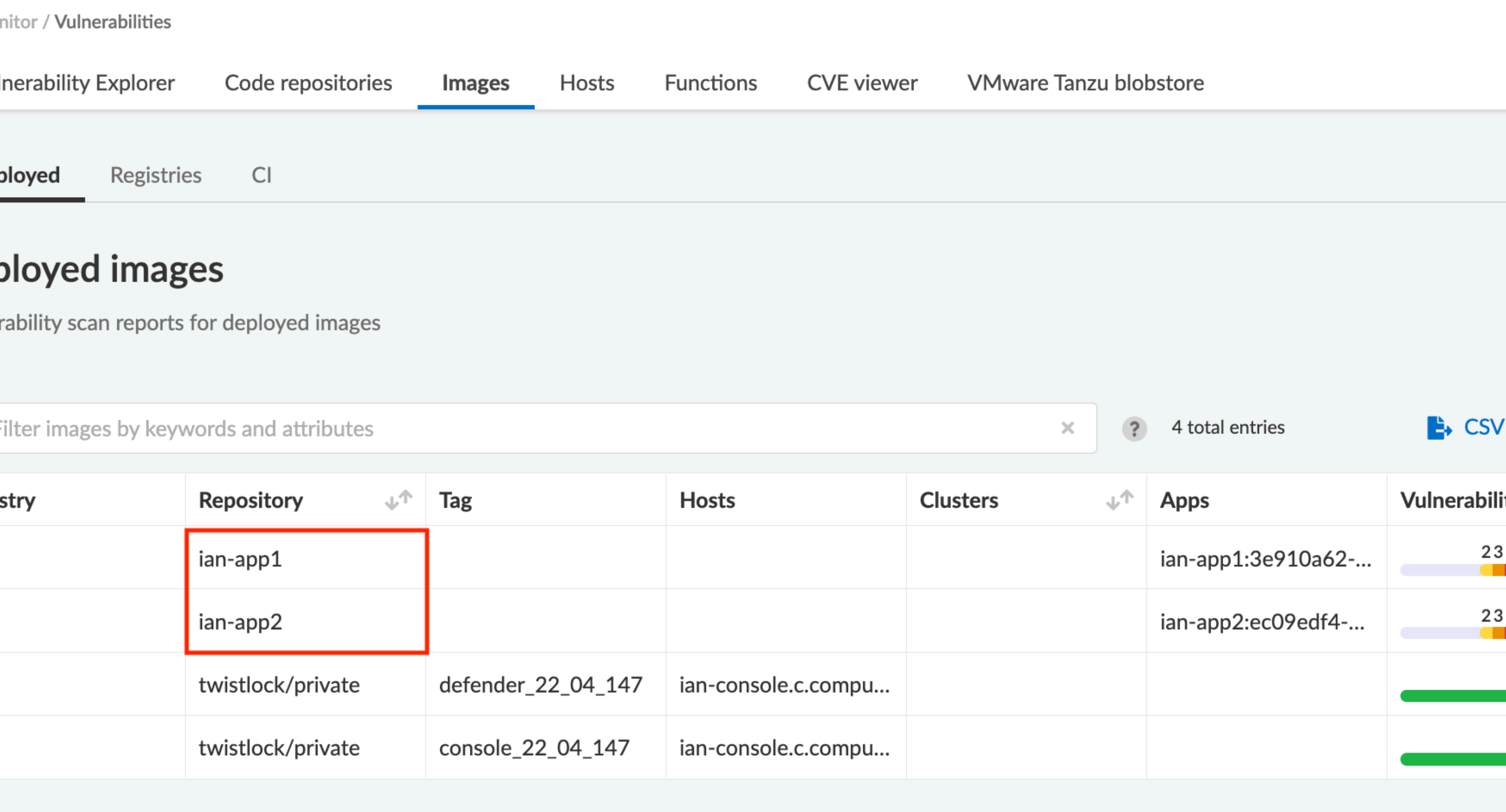
During the embed flow, you must specify a value for App ID (or more accurately, app name, which becomes part of the final App ID). In the Console, this value is presented as the image name. When specifying App ID, choose a value you can easily trace back to the image when reviewing and mitigating security findings.

As part of the embed flow, Prisma Cloud automatically generates a universally unique identifier (UUID) to represent the image ID. The image ID is a primary key in the Prisma Cloud Compute database, so it's essential that it's defined.

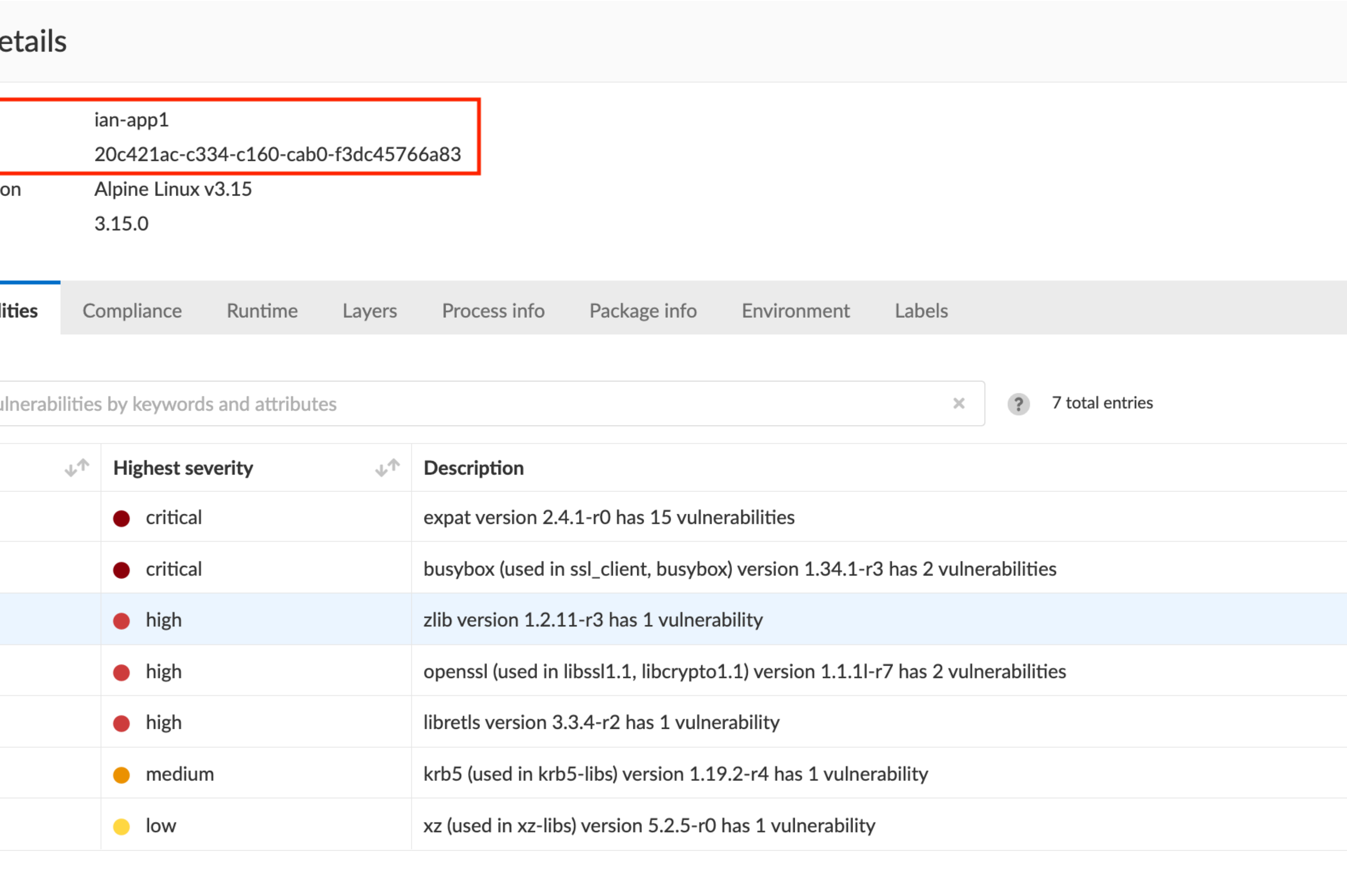
Together, the app name plus the generated UUID form the final App ID. The final App ID has the following format:

```
<app-name>:<uuid>
```

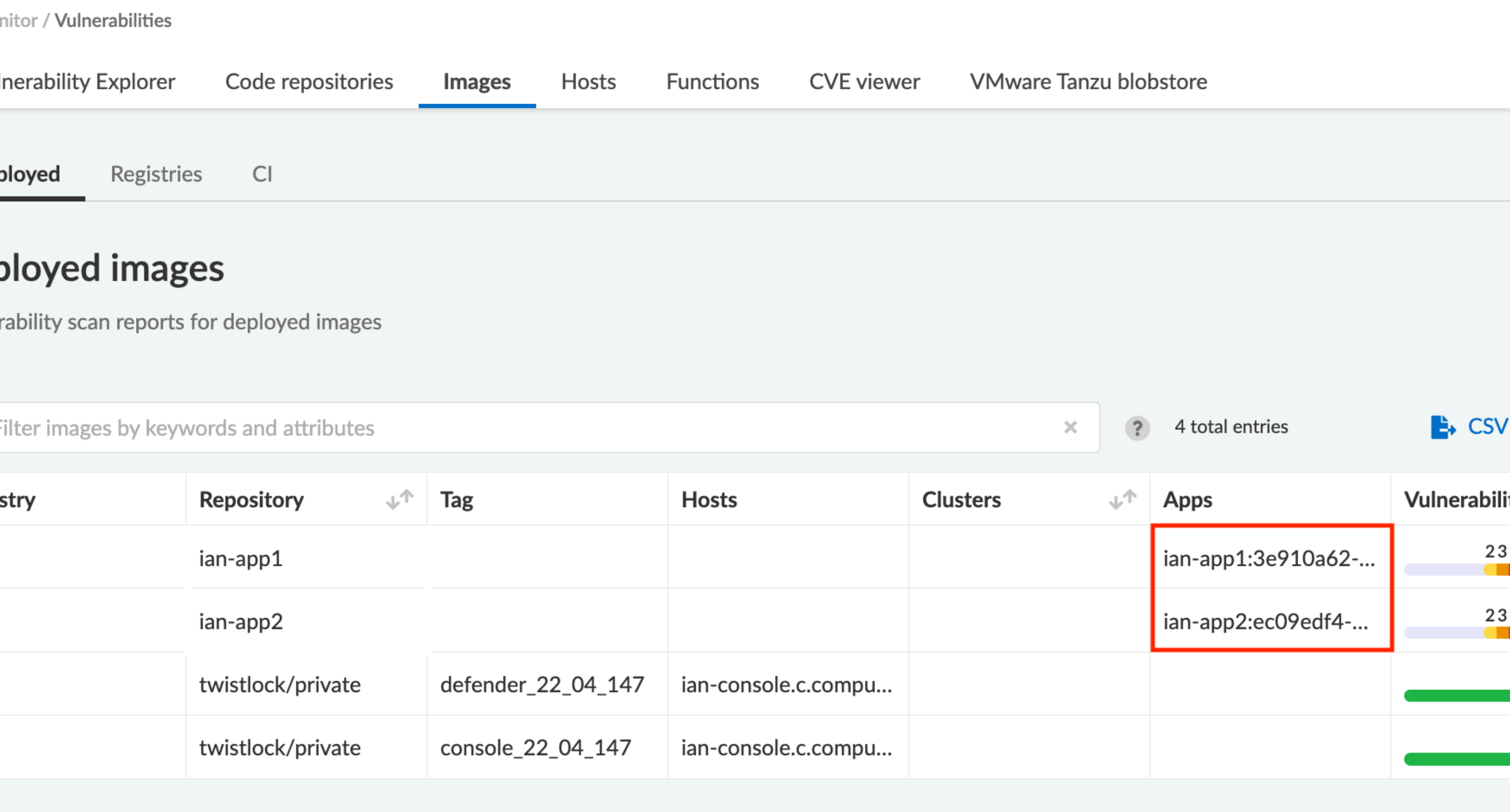
The following screenshot shows how images protected by App-Embedded Defender are listed under **Monitor > Vulnerabilities**. The **Repository** column, which represents the image name, shows two images: ian-app1 and ian-app2. Both ian-app1 and ian-app2 were specified as the App IDs when embedding Defenders into the images.



The next screenshot shows the scan report for ian-app1. Notice that **Image** is set to **ian-app1**, which was the App ID specified when embedding Defender. Also notice that the value for **Image ID** is a UUID.



Finally, back in **Monitor > Vulnerabilities**, notice that the **Apps** column shows the final App ID, which is the combination of the app name (specified as App ID in the embed flow) plus the internally generated UUID.



Embed App-Embedded Defender

Embed App-Embedded Defender into a container image from Console's UI.

Prerequisites:

- At runtime, the container where you're embedding App-Embedded Defender can reach Console over the network. For Enterprise Edition, Defender talks to Console on port 443. For Compute Edition, Defender talks to Console on port 8084.
- You have the Dockerfile for your image.

STEP 1 | Open Console, and go to **Manage > Defenders > Deploy > Defenders**.

STEP 2 | In **Deployment method**, select **Single Defender**.

STEP 3 | Select the DNS name or IP address that App-Embedded Defender uses to connect to Console.

STEP 4 | In **Choose the Defender type**, select **Container Defender - App-Embedded Defender**.

STEP 5 | In **Monitor file system events**, set the toggle to **On** if your runtime policy requires it.

If App-Embedded Defender is deployed with this setting turned on, the sensor will monitor file system events, regardless of how your runtime policy is configured, and could impact the underlying workload's performance.

If you later decide you want to disable the sensor completely, you must re-embed App-Embedded Defender with this setting turned off.

Conversely, if you deploy App-Embedded Defender with this setting disabled, and later decide you want file system protection, you'll need to re-embed App-Embedded with this setting enabled.

You can specify the default setting for this toggle so it's set the same way for all App-Embedded Defender deployments.

STEP 6 | In **Deployment type**, select **Dockerfile**.

STEP 7 | In **App ID**, enter a unique identifier for the App-Embedded Defender.

All vulnerability, compliance, and runtime findings for the container will be aggregagted under this App ID In Console, the App ID is presented as the image name. Be sure to specify an App ID that lets you easily trace findings back to the image.

STEP 8 | In **Dockerfile**, click **Choose File**, and upload the Dockerfile for your container image.

STEP 9 | Click **Create embedded ZIP**.

A file named *app_embedded_embed_help.zip* is created and downloaded to your system.

STEP 10 | Unpack app_embedded_embed_help.zip.

```
$ mkdir tmp
$ unzip app_embedded_embed_help.zip -d tmp/
```

STEP 11 | Build the modified Docker image.

```
$ cd tmp/
$ docker build .
```

STEP 12 | Tag and push the updated image to your repository.

Embed App-Embedded Defender manually

Embed App-Embedded Defender into a container image manually. Modify your Dockerfile with the supplied information, download the App-Embedded Defender binaries into the image's build context, then rebuild the image.

Prerequisites:

- At runtime, the container where you're embedding App-Embedded Defender can reach Console over the network. For Enterprise Edition, Defender talks to Console on port 443. For Compute Edition, Defender talks to Console on port 8084.
- The host where you're rebuilding your container image with App-Embedded Defender can reach Console over the network on port 8083.

- You have the Dockerfile for your image.

STEP 1 | Open Console, and go to **Manage > Defenders > Deploy > Defenders**.

STEP 2 | In **Deployment method**, select **Single Defender**.

STEP 3 | Select the DNS name or IP address that App-Embedded Defender uses to connect to Console.

STEP 4 | In **Choose the Defender type**, select **Container Defender - App-Embedded Defender**.

STEP 5 | In **Monitor file system events**, set the toggle to **On** if your runtime policy requires it.

If App-Embedded Defender is deployed with this setting turned on, the sensor will monitor file system events, regardless of how your runtime policy is configured, and could impact the underlying workload's performance.

If you later decide you want to disable the sensor completely, you must re-embed App-Embedded Defender with this setting turned off.

Conversely, if you deploy App-Embedded Defender with this setting disabled, and later decide you want file system protection, you'll need to re-embed App-Embedded with this setting enabled.

You can specify the default setting for this toggle so it's set the same way for all App-Embedded Defender deployments.

STEP 6 | In **Deployment Type**, select **Manual**.

A set of instructions for embedding App-Embedded Defender into your images is provided.

1. Using the provided curl command, download the App-Embedded Defender binary into your image's build context directory.
2. Open your Dockerfile for editing.
3. Add the App-Embedded Defender to the image.

   ```
   ADD twistlock_defender_app_embedded.tar.gz /twistlock/
   ```

4. Add the specified environment variables.

   When setting *DEFENDER_APP_ID*, specify a value that lets you easily trace findings back to the image. All vulnerability, compliance, and runtime findings for the container will be aggregagted under this App ID In Console, the App ID is presented as the image name.

5. Modify the entrypoint so that your app starts under the control of App-Embedded Defender.

   For example, to start the hello-world program under the control of App-Embedded Defender, specify the following entrypoint.

   ```
   ENTRYPOINT ["/twistlock/defender", "app-embedded", "hello-
   world"]
   ```


STEP 7 | Rebuild your image.

```
$ docker build .
```

STEP 8 | Tag and push the updated image to your repository.

Embed App-Embedded Defender with twistcli

Prisma Cloud supports automation for embedding App-Embedded Defender into container images with either twistcli or the API. This section shows you how to use twistcli. To learn how to use the API, see the API docs.

Prerequisites:

- The container where you're embedding App-Embedded Defender can reach Console's port 8084 over the network.
- You have the Dockerfile for your image.

STEP 1 | Download twistcli.

1. Log into Console, and go to **Manage > System > Utilities**.
2. Download the twistcli binary for your platform.

STEP 2 | Generate the artifacts for an updated container with twistcli.

A file named *app_embedded_embed*<app_id>.zip_ is created.

```
$ ./twistcli app-embedded embed \
  --user <USER>
  --address "https://<CONSOLE>:8083" \
  --console-host <CONSOLE> \
  --app-id "<APP-ID>"  \
  --data-folder "<DATA-FOLDER>"  \
  Dockerfile
```

- <USER> — Name of a Prisma Cloud user with a minimum role of Defender Manager.
- <CONSOLE> — DNS name or IP address for Console.
- <APP-ID> — Unique identifier. When setting *<APP-ID>*, specify a value that lets you easily trace findings back to the image. All vulnerability, compliance, and runtime findings for the container will be aggregagted under this App ID. In Console, the App ID is presented as the image name. For example, *my-app*.
- <DATA-FOLDER> — Readable and writable directory in the container's filesystem. For example, */tmp*.
- To enable file system protection, add the --*filesystem-monitoring* flag to the twistcli command.

STEP 3 | Unpack *app_embedded_embed_help.zip*.

```
$ mkdir tmp
$ unzip app_embedded_embed_help.zip -d tmp/
```

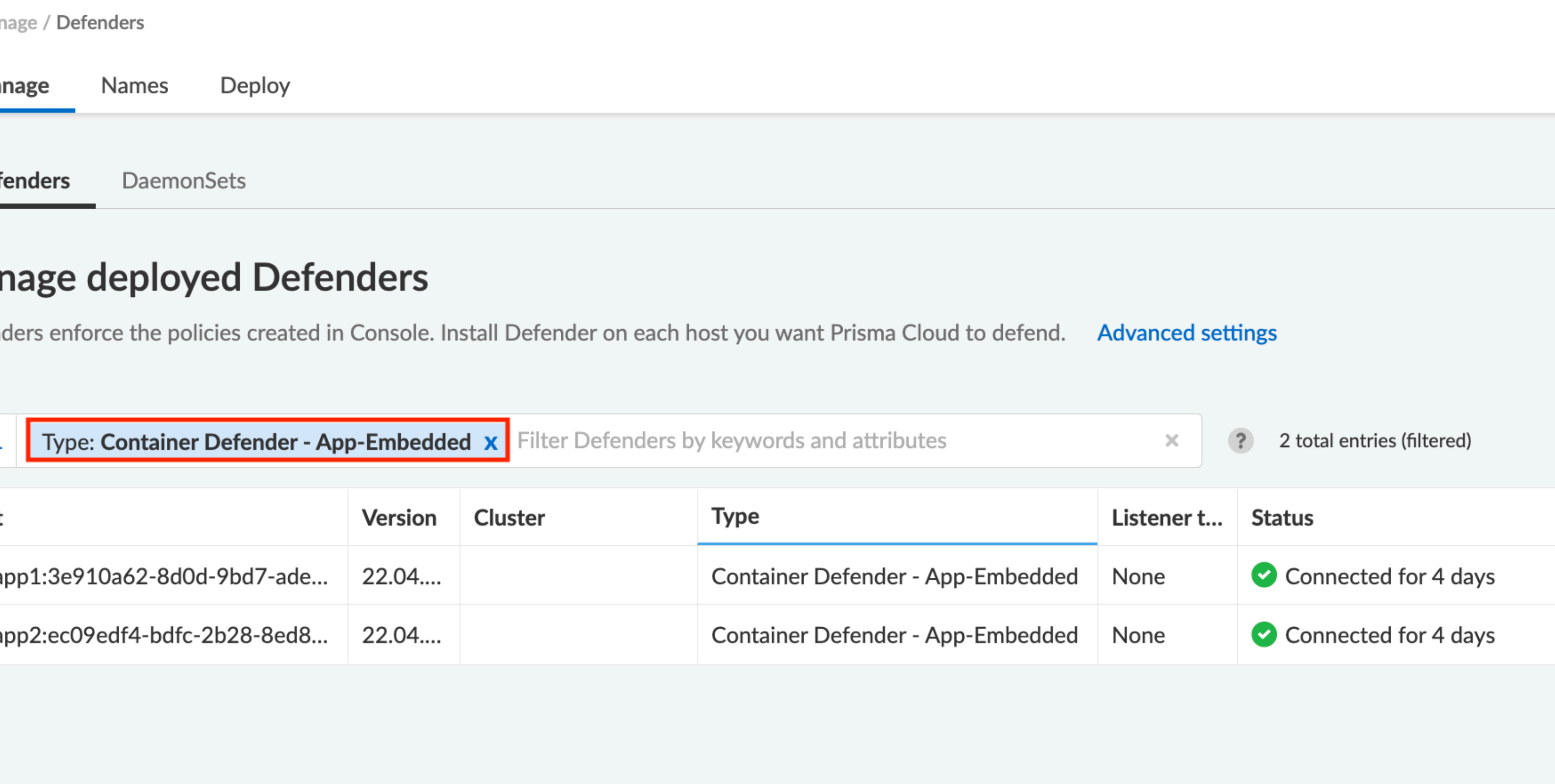

STEP 4 | Build the updated image.

```
$ cd tmp/
$ docker build .
```

STEP 5 | Tag and push the updated image to your repository.

Connected Defenders

You can review the list of all Defenders connected to Console under **Manage > Defenders > Manage > Defenders**. To see just App-Embedded Defenders, filter the table by type, *Type: Container Defender - App-Embedded*.



By default, Prisma Cloud removes disconnected App-Embedded Defenders from the list after an hour. As part of the cleanup process, data collected by the disconnected Defender is also removed from **Monitor > Runtime > App-Embedded observations**.

> *There is an advanced settings dialog under **Manage > Defenders > Manage > Defenders**, which lets you configure how long Prisma Cloud should wait before cleaning up disconnected Defenders. This setting doesn't apply to App-Embedded Defenders. Disconnected App-Embedded Defenders are always removed after one hour.*

App-Embedded Defender for Fargate

Edit on GitHub

App-Embedded Defenders for Fargate monitor and protect your Fargate tasks to ensure they execute as designed.

To learn when to use App-Embedded Defenders, see Defender types.

To learn more about App-Embedded Defender's capabilities, see:

- Vulnerability scanning for App-Embedded
- Compliance scanning for App-Embedded
- Runtime defense for App-Embedded
- Protecting front-end containers at runtime with WAAS

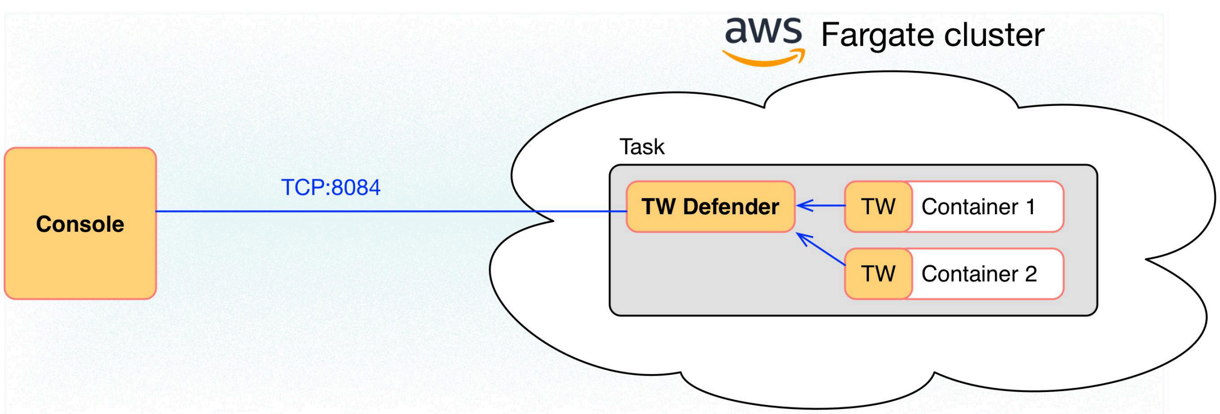
For front-end Farate tasks, deploy the WAAS application firewall for additional runtime protection.

Architecture

When you embed the App-Embedded Defender into your Fargate task, Prisma Cloud modifies the task definition. The updated task definition includes a Prisma Cloud sidecar container. The sidecar container handles all communication with Console, including retrieving policies and sending audits. It also hosts the App-Embedded Defender binaries, which are shared with the task's other containers through a shared volume. The embed process modifies each containerDefinition to:

- Mount the Prisma Cloud sidecar container's shared volume to gain access to the App-Embedded Defender binaries.
- Start the original entrypoint command under the control of App-Embedded Defender.

App-Embedded Defenders do not communicate directly with Console. All communication is proxied through the Prisma Cloud sidecar container. The following diagram illustrates the setup:



App ID

Each App-Embedded Defender deployed in an ECS Fargate task has an App ID that's automatically generated during the embed flow. For ECS Fargate tasks, the App ID is constructed from the task name and an internally generated UUID. The format is:

```
<task-name>:<UIID>
```

This App ID is used throughout the Console UI. In particular, it's listed in the **Apps** column of the vulnerability and compliance scan reports under **Monitor > Vulnerabilities > Images > Deployed** and **Monitor > Compliance > Images > Deployed**.

WAAS for Fargate

All the capabilities of standard WAAS are available for Fargate tasks. The only difference is that Fargate Defenders run as a reverse proxies to all other containers in the task. As such, when you set up WAAS for Fargate, you must specify the exposed external port where Fargate Defender can listen, and the port (not exposed to the Internet) where your web application listens. WAAS for Fargate forwards the filtered traffic to your application port - *unless an attack is detected and you chose* **Prevent** *in your WAAS for Fargate rule*.

For more information on the type of attacks that Prisma Cloud detects and prevents, see Prisma Cloud WAAS.

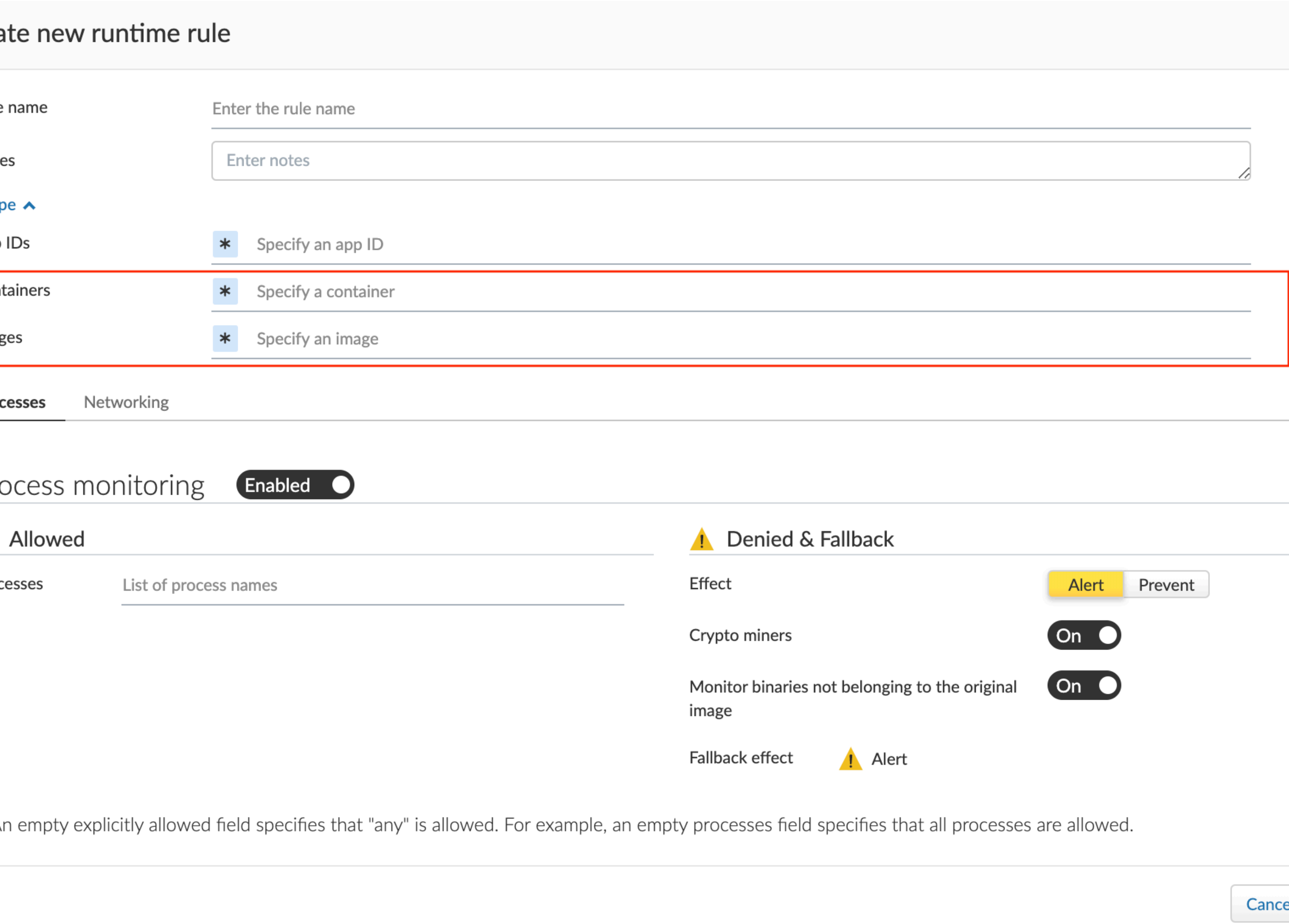
Securing Fargate tasks

To secure a Fargate task, embed the Prisma Cloud Fargate Defender into it. The steps are:

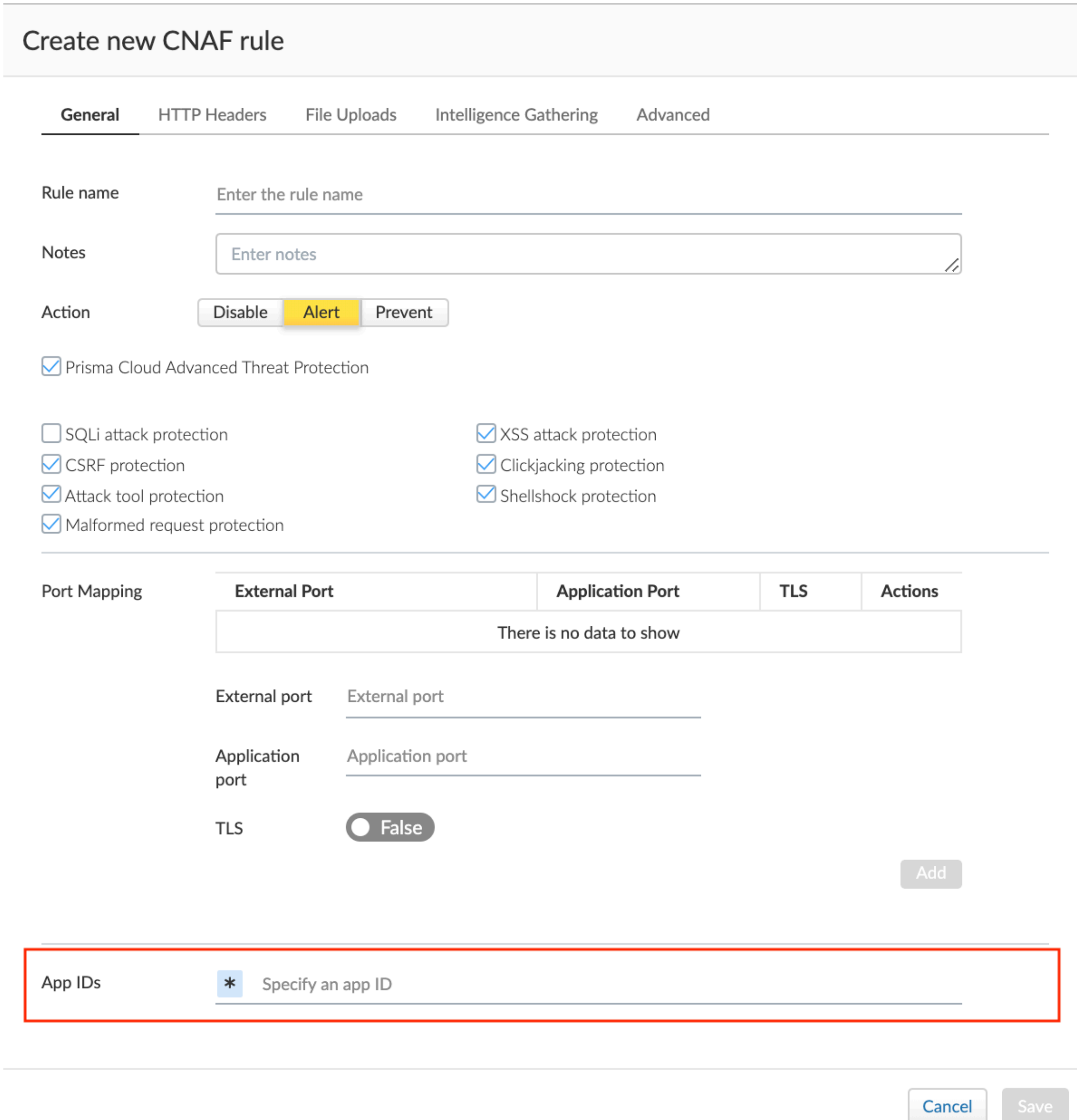
1. Define your policy in Prisma Cloud Console.

   App-Embedded Defenders dynamically retrieve rules from Console as they are updated. You can embed the App-Embedded Defender into a task with a simple initial policy, and then refine it later, as needed.
2. Embed the Fargate Defender into your task definition.
3. Start the service.

When securing Fargate tasks with runtime rules and WAAS, target rules to tasks using the **Scope** fields. For runtime, scope rules by image and container name. Policy is applied per-container in the task.



For WAAS, scope rules by App ID. Policy is applied per-task. The WAAS firewall listens on a specific port, and since all containers run in the same network namespace, it applies to the entire task.



Task entrypoint

When Prisma Cloud generates a protected task definition, it needs to know the container image's entrypoint and/or cmd instructions. We override these values to first run the App-Embedded Defender, and then run the original entrypoint/cmd under Defender's watch.

Setting the entrypoint in a task definition is optional. It's only required when you want to override the image's entrypoint as specified in its Dockerfile. As such, many task definitions don't explicitly specify it. However, Prisma Cloud needs to know what it is so it can run original app under Defender's control. To aid in embedding Defender into Fargate tasks without any manual intervention (i.e. updating task definitions to explicitly specify entrypoints), Prisma Cloud can automatically find the image's entrypoint and set it up in the protected task definition.

Prisma Cloud can find the image's entrypoint from:

- Registry scans. When Prisma Cloud scans an image from a registry, it saves the entrypoint and cmd to the database. When embedding Defender into a task, Prisma Cloud searches the database to see if it's seen the task's image before. If so, it extracts the original entrypoint, and sets it up in the new protected task definition.

- Querying the registry directly. If the image hasn't been scanned by the registry scanner, then you can point Prisma Cloud to the registry where the image lives, and Prisma Cloud can find and extract the entrypoint. Prisma Cloud supports the following registries:
  - AWS Elastic Container Registry (ECR).
  - Docker Registry v2.
  - JFrog Artifactory.

Automatically extracting the entrypoint using one of the methods described above is optional. It can be enabled or disabled when embedding Defender in a task definition.

The twistcli tool also supports entrypoint extraction when generating protected task definitions. For more information, see the help menu:

```
twistcli app-embedded generate-fargate-task --help
```

If your task definition specifies the command parameter, but no entrypoint, AND you've enabled Prisma Cloud's automatic entrypoint extraction, then Prisma Cloud will bypass automatic entrypoint extraction, and instead generate a protected task definition using the command parameter.

Embedding App-Embedded Defender into Fargate tasks

Prisma Cloud cleanly separates the code developers produce from the Fargate containers we protect. Developers don't need to change their code to accomodate Prisma Cloud. They don't need to load any special libraries, add any files, or change any manifests. When a container is ready to be deployed to test or production, run your task definition through our transform tool to automatically embed the Fargate Defender, then load the new task definition into AWS.

The method for embedding the Fargate Defender was designed to seamlessly integrate into the CI/CD pipeline. You can call the Prisma Cloud API to embed the Fargate Defender into your task definition.

Prerequisites:

- The task where you're embedding the App-Embedded Defender can reach Console over the network. For Enterprise Edition, Defender talks to Console on port 443. For Compute Edition, Defender talks to Console on port 8084.
- You have a task definition.
- You have already created an ECS cluster.
- Cluster VPC and subnets.
- Task role.
- Your image has a shell.

You can optionally run the Fargate Defender sidecar as a non-essential container. This configuration isn't recommended because Defender's goal is to ensure that tasks are always protected.

If you've configured Defender as a non-essential container and you're having issues with your setup, first validate that Defender is running as expected before contacting Palo Alto Networks customer support. By setting Defender as non-essential, there is no guarantee that Defender is running, and validating that it's running is the first step in debugging such issues.

Supported task definition formats

Prisma Cloud supports the following task definition formats:

- Standard JSON format, as described here.
- CloudFormation templates for *AWS::ECS::TaskDefinition* in JSON and YAML formats, as described here. You can use either just the task definition part of the CloudFormation template, or a full CloudFormation template.

Example of a standard JSON format task definition:

```
{
    "containerDefinitions": [
        {
            "name": "web",
            "image": "nginx",
            "entryPoint": [
                "/http_server"
            ]
        }
    ],
    "cpu": "256",
    "executionRoleArn": "arn:aws:iam::112233445566:role/
ecsTaskExecutionRole",
    "family": "webserver",
    "memory": "512",
    "networkMode": "awsvpc",
    "requiresCompatibilities": [
        "FARGATE"
    ]
}
```

Example of the equivalent task definition as a JSON CloudFormation template:

```
{
  "Type" : "AWS::ECS::TaskDefinition",
  "Properties": {
      "ContainerDefinitions": [
        {
          "Name": "web",
          "Image": "nginx",
          "EntryPoint": [
              "/http_server"
          ]
        }
```



```
      ],
      "Cpu" : 256,
      "ExecutionRoleArn": "arn:aws:iam::112233445566:role/
ecsTaskExecutionRole",
      "Family": "webserver",
      "Memory" : 512,
      "NetworkMode" : "awsvpc",
      "RequiresCompatibilities" : [
          "FARGATE"
      ]
    }
}
```

Example of a full JSON CloudFormation template that includes a Fargate task definition:

```
{
  "AWSTemplateFormatVersion": "2010-09-09",
  "Resources": {
    "fargateTaskDefinition": {
      "Type": "AWS::ECS::TaskDefinition",
      "Properties": {
        "ExecutionRoleArn": "arn:aws:iam::1234567891234:role/
ecsTaskExecutionRole",
        "ContainerDefinitions": [
          {
            "Name": "test-server",
            "Image": "1234567891234.dkr.ecr.us-east-1.amazonaws.com/
user:ubuntu-test-server",
            "MemoryReservation": "",
            "Essential": true,
            "PortMappings": [],
            "Cpu": 256,
            "Memory": 512,
            "EntryPoint": [
              "/http_server"
            ],
            "EnvironmentFiles": [],
            "LogConfiguration": {
              "LogDriver": "awslogs",
              "Options": {
                "awslogs-group": "/ecs/user-tracer-test",
                "awslogs-region": "us-east-1",
                "awslogs-stream-prefix": "ecs"
              }
            }
          }
        ],
        "Memory": "512",
        "TaskRoleArn": "arn:aws:iam::1234567891234:role/
ecsTaskExecutionRole",
        "Family": "TASK-NAME",
        "RequiresCompatibilities": [
          "FARGATE"
        ],
        "NetworkMode": "awsvpc",
        "Cpu": "256",
```

```
        "InferenceAccelerators": [],
        "Volumes": [],
        "Tags": []
      }
    },
    "HelloLambdaRole": {
      "Type": "AWS::IAM::Role",
      "Properties": {
        "RoleName": "HelloLambdaRole1",
        "AssumeRolePolicyDocument": {
          "Statement": [
            {
              "Effect": "Allow",
              "Principal": {
                "Service": "lambda.amazonaws.com"
              },
              "Action": "sts:AssumeRole"
            }
          ]
        }
      }
    }
  }
}
```

Example of a full YAML CloudFormation template that includes a Fargate task definition:

```
AWSTemplateFormatVersion: "2010-09-09"
Resources:
  fargateTaskDefinition:
    Type: 'AWS::ECS::TaskDefinition'
    Properties:
      ExecutionRoleArn: 'arn:aws:iam::1234567891234:role/
ecsTaskExecutionRole'
      ContainerDefinitions:
        - Name: test-server
          Image: >-
            1234567891234.dkr.ecr.us-east-1.amazonaws.com/
user:ubuntu-test-server
          MemoryReservation: ''
          Essential: true
          PortMappings: []
          Cpu: 256
          Memory: 512
          EntryPoint:
            - /http_server
          EnvironmentFiles: []
          LogConfiguration:
            LogDriver: awslogs
            Options:
              awslogs-group: /ecs/user-tracer-test
              awslogs-region: us-east-1
              awslogs-stream-prefix: ecs
      Memory: '512'
      TaskRoleArn: 'arn:aws:iam::1234567891234:role/
ecsTaskExecutionRole'
```

```
      Family: TASK-NAME
      RequiresCompatibilities:
        - FARGATE
      NetworkMode: awsvpc
      Cpu: '256'
      InferenceAccelerators: []
      Volumes: []
      Tags: []
  HelloLambdaRole:
    Type: 'AWS::IAM::Role'
    Properties:
      RoleName: HelloLambdaRole2
      AssumeRolePolicyDocument:
        Statement:
          - Effect: Allow
            Principal:
              Service: lambda.amazonaws.com
            Action: 'sts:AssumeRole'
```

Embed App-Embedded Defender from the Console UI

You can created a protected task definition in the Console UI.

Prerequisites:

- You've already created an ECS cluster, cluster VPC, and subnets.
- You've already created a task role.
- You have a task definition.
- At runtime, your tasks can connnect to Prisma Cloud Console over the network. Prisma Cloud Defender connects to Console to retrieve runtime policies and send audits. For Enterprise Edition, Defender talks to Console on port 443. For Compute Edition, Defender talks to Console on port 8084 (although the port number is configurable at install-time).

STEP 1 | Log into Prisma Cloud Console.

STEP 2 | Go to **Manage** > **Defenders** > **Deploy** > **Defenders**.

STEP 3 | In **Deployment method**, select **Single Defender**.

STEP 4 | Select the DNS name or IP address that App-Embedded Defender uses to connect to Console.

*A list of IP addresses and hostnames are pre-populated in the drop-down list. If none of the items are valid, select the **Names** tab and add a new subject alternative name (SAN) using **Add SAN** button. After adding a SAN, your IP address or hostname will be available in the drop-down list in the **Deploy** tab.*

Selecting an IP address in a evaluation setup is acceptable, but using a DNS name is more resilient. If you select Console's IP address, and Console's IP address changes, your Defenders will no longer be able to communicate with Console.

STEP 5 | In **Choose the Defender type**, select **Container Defender - App-Embedded Defender**.

STEP 6 | In **Monitor file system events**, set the toggle to **On** if your runtime policy requires it.

If App-Embedded Defender is deployed with this setting turned on, the sensor will monitor file system events, regardless of how your runtime policy is configured, and could impact the underlying workload's performance.

If you later decide you want to disable the sensor completely, you must re-embed App-Embedded Defender with this setting turned off.

Conversely, if you deploy App-Embedded Defender with this setting disabled, and later decide you want file system protection, you'll need to re-embed App-Embedded with this setting enabled.

You can specify the default setting for this toggle so it's set the same way for all App-Embedded Defender deployments.

STEP 7 | In **Deployment type**, select **Fargate task**.

STEP 8 | Set up the task's entrypoint.

If your task definition doesn't explicitly specify an entrypoint, Prisma Cloud can automatically determine how to set it to start the image's app under App-Embedded Defender's control.

If you don't enable any of the following options AND your task defintion doesn't specify an entrypoint, you must update your task definition to include matching entrypoint and cmd parameters from the Dockerfile(s) of the image(s) in your task. Because Prisma Cloud won't see the actual images as part of the embedding flow, it depends on having these parameter present insert the App-Embedded Defender into the task startup flow.

1. Set **Automatically extract Entrypoint** to **On**.

   Prisma Cloud finds the image and its entrypoint in the registry scan results.
2. (Optional) Tell Prisma Cloud where it can find the image.

   If Prisma Cloud hasn't scanned the image, you can point it to registry where the image resides. Prisma Cloud will find the image and extract it's entrypoint.

   Specify the registry type and pick the credential Prisma Cloud can use to access the registry.

STEP 9 | Embed the Fargate Defender into your task definition.

1. Set **Template type** according to the format used to specify your task definition.
   - **Native Fargate** — Standard JSON format, as described here.
   - **CloudFormation** — CloudFormation template for *AWS::ECS::TaskDefinition*, as described here.
2. Copy and paste your task definition into the left-hand box.
3. Click **Generate protected task**.
4. Copy the updated task definition from the right-hand box, and use it to create a new task definition in AWS.

   The newly generated task definition always uses the version of Defender that matches the Console from which you are generating the task definition. The task definition includes a complete configuration, such as volumes, startup dependencies, entrypoint, healthchecks for its successful execution. Therefore, manually changing the Defender version label in the task is not supported.

Embed App-Embedded Defender with twistcli

The twistcli command line tool lets you embed App-Embedded Defender into Fargate task definitions.

Prerequisites:

- You've already created an ECS cluster, cluster VPC, and subnets.
- You've already created a task role.
- You have a task definition.
- Running tasks can connnect to Prisma Cloud Console over the network. Prisma Cloud Defender connects to Console to retrieve runtime policies and send audits. For Enterprise Edition, Defender talks to Console on port 443. For Compute Edition, Defender talks to Console on port 8084 (although the port number is configurable at install-time).

STEP 1 | Log into Prisma Cloud Console.

STEP 2 | Go to **Manage > System > Utilities**, and download twistcli for your machine's operating system.

STEP 3 | Run twistcli to embed Defender into the task definition.

```
$ twistcli app-embedded generate-fargate-task \
  --user <USER> \
  --address "<CONSOLE_URL>" \
  --console-host "<CONSOLE_ADDR>" \
  --output-file "protected_taskdef.json" \
```



```
taskdef.json
```

If your task definition file is specified as a CloudFormation template, then add the --*cloud-formation* option to the twistcli command. You can use JSON or YAML formats in CloudFormation template.

- *<USER>* — Prisma Cloud user with a role of Defender Manager or higher.
- *<CONSOLE_URL>* — RFC 1808 scheme and netloc for Console. twistcli uses this value to connect to Console to submit the task definition for embedding Defender. Example: https://127.0.0.1:8083
- *<CONSOLE_ADDR>* — RFC 1738 host where Console runs. This value will be the fully qualified domain name of the network host, or IP address, where Console runs. This value configures how the embedded Defender connects to Console.

Creating a task definition in AWS

Create a new task definition in AWS with the output from the previous section. If you already have an existing task definition, create a new revision.

STEP 1 | Log into the AWS Management Console.

STEP 2 | Go to **Services > ECS**.

STEP 3 | Click **Task Definitions**, then click **Create new Task Definition**.

1. Select **Fargate**, then click **Next step**.
2. Scroll to the bottom of the page, and click **Configure via JSON**.
3. Delete the prepopulated JSON, then paste the JSON generated for task from the previous section.
4. Click **Save**.

STEP 4 | Validate task content.

1. Task name should be as described in the JSON.
2. Select the **Task Role**.
3. The task should include the **TwistlockDefender** container.
4. Click **Create**.
5. Click **View task definition**.

Testing the task

STEP 1 | Log into the AWS Management Console.

STEP 2 | Go to **Services > ECS**.

STEP 3 | Click **Clusters**, then select one of your Fargate cluster.

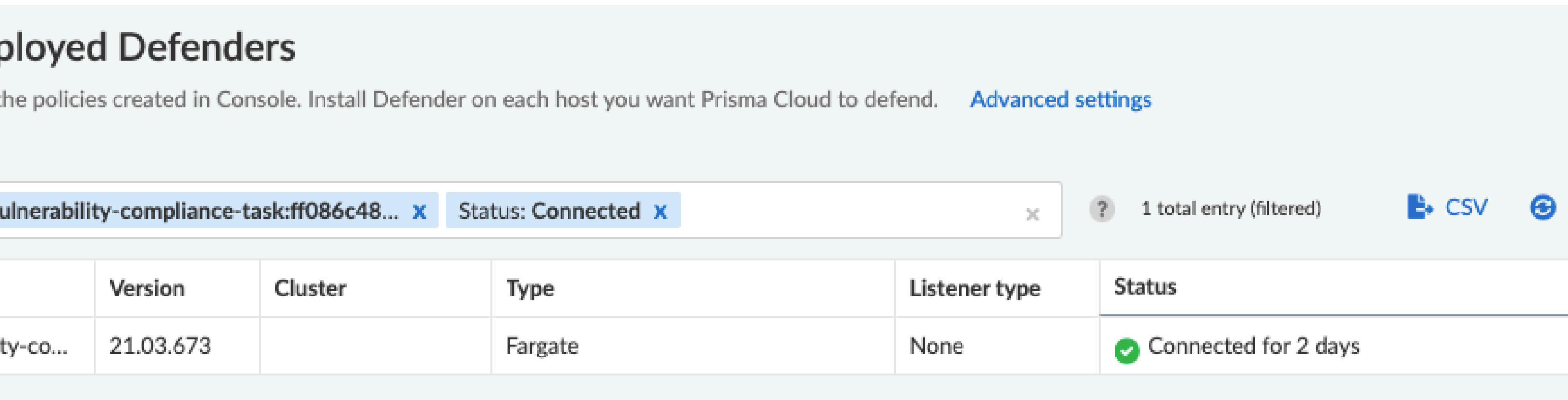
STEP 4 | Click the **Services** tab, then click **Create**.

1. For **Launch type**, select **Fargate**.
2. For **Task Definition**, select your pre-defined task.
3. Enter a **Service name.**
4. For **Number of tasks**, enter **1**.
5. Click **Next step**.
6. Select a **Cluster VPC** and **Subnets**, then click **Next step**.
7. For **Service Auto Scaling**, select **Do not adjust the service's desired count**, then click **Next step**.
8. Review your settings, then click **Create Service**.

STEP 5 | Validate the results.

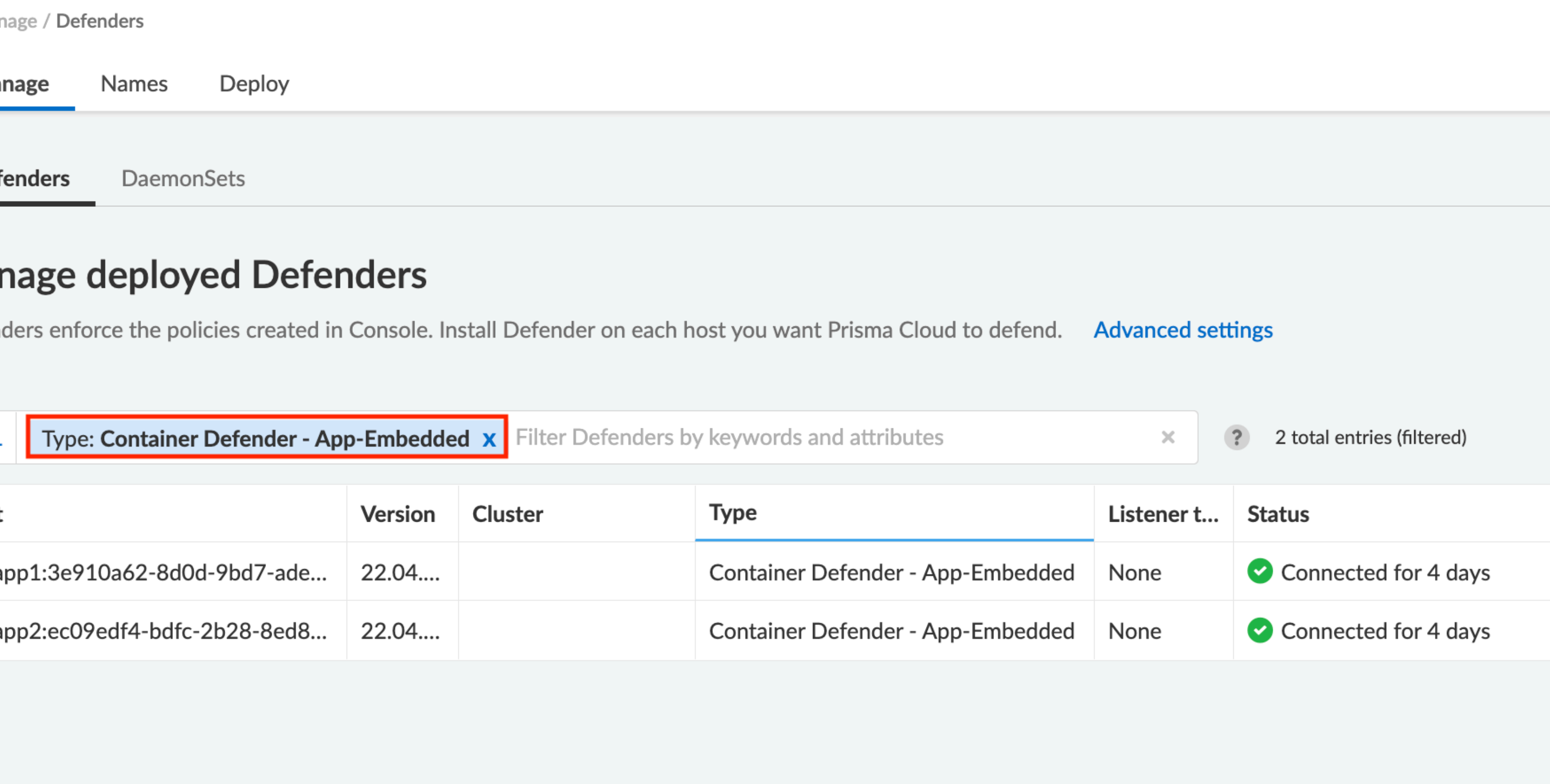
1. Click **View Service**.
2. When Last status is Running, your Fargate task is running.
3. The containers are running.

STEP 6 | View the defender in the Prisma Cloud Console: Go to **Manage > Defenders > Manage > Defenders** and search the fargate task by adding the filters **Fargate** and **Status:Connected**.



Connected Defenders

You can review the list of all Defenders connected to Console under **Manage > Defenders > Manage > Defenders**. To narrow the list to just App-Embedded Defenders, filter the table by type *Type: Container Defender - App-Embedded*. To see the list of Fargate tasks protected by App-Embedded Defender, filter the table by *Type: Fargate*.



By default, Prisma Cloud removes disconnected App-Embedded Defenders from the list after an hour. As part of the cleanup process, data collected by the disconnected Defender is also removed from **Monitor > Runtime > App-Embedded observations**.

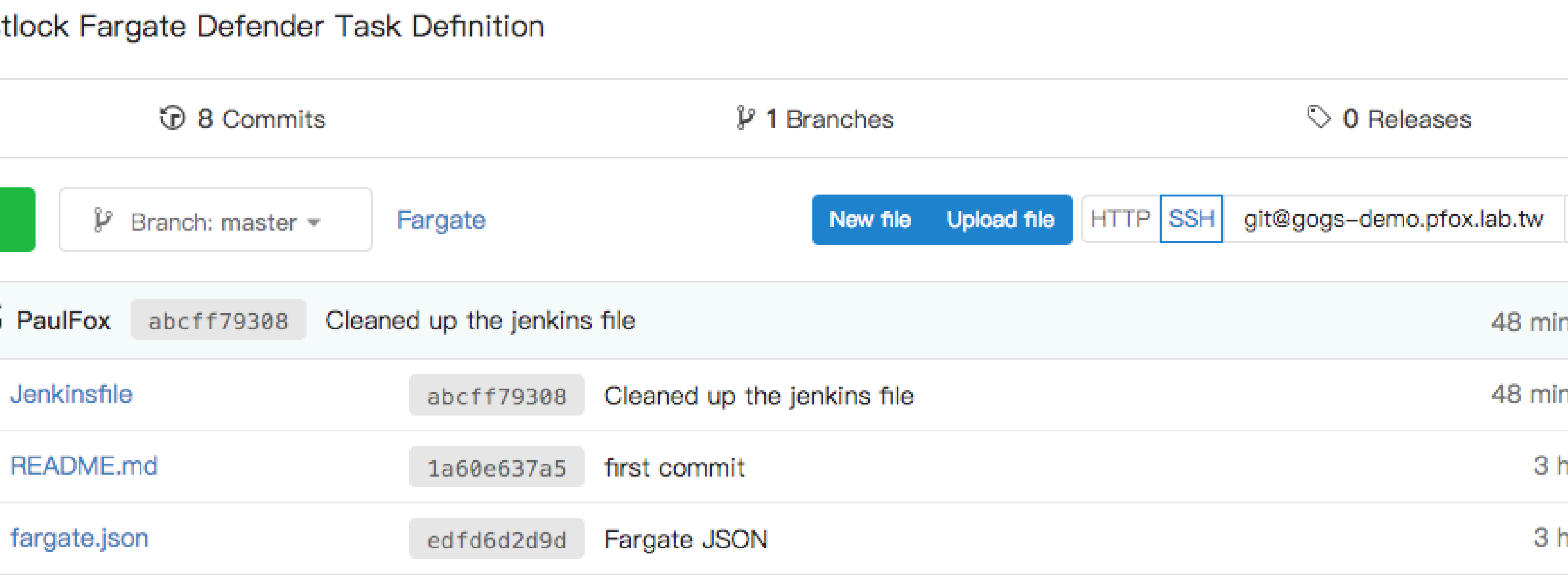
*There is an advanced settings dialog under **Manage > Defenders > Manage > Defenders**, which lets you configure how long Prisma Cloud should wait before cleaning up disconnected Defenders. This setting doesn't apply to App-Embedded Defenders. Disconnected App-Embedded Defenders are always removed after one hour.*

Jenkins Fargate example

Passing the Fargate task definition to your Prisma Cloud Console's API returns the Prisma Cloud protected Fargate task definition. Use this task definition to start Prisma Cloud protected Fargate containers. This example demonstrates using the Jenkins Pipeline build process to:

- Call the Prisma Cloud Console's API endpoint for Fargate task creation.
- Pass the Fargate task definition to the API.
- Capture the returned Prisma Cloud protected Fargate task definition.
- Save the Prisma Cloud protected Fargate task definition within the Pipeline's archive https://<jenkins>/job/<pipeline_name>/<job#>/artifact/tw_fargate.json

In this example, a simple task *fargate.json* and *Jenkinsfile* have been placed in a GitHub repository.




```
{
  node {

      stage('Clone repository') {
          checkout scm
      }

      stage('Fargate Task call') {
          withCredentials([usernamePassword(credentialsId:
 'twistlockDefenderManager', passwordVariable: 'TL_PASS',
 usernameVariable: 'TL_USER')]) {
              sh 'curl -s -k -u $TL_USER:$TL_PASS https://
$TL_CONSOLE/api/v1/defenders/fargate.json?consoleaddr=$TL_CONSOLE
 -X POST -H "Content-Type:application/json" --data-binary
 "@fargate.json" | jq . > tw_fargate.json'
              sh 'cat tw_fargate.json'
          }
      }

      stage('Publish Function') {
          archiveArtifacts artifacts: 'tw_fargate.json'}
  }
}
```

STEP 1 | Create an account in Prisma Cloud with the Defender Manager role.

STEP 2 | Create a Jenkins username/password credential for this account called **twistlockDefenderManager**.

STEP 3 | The **$TL_Console** Jenkins global variable was defined when the Prisma Cloud Jenkins plugin was installed.

STEP 4 | Create a Jenkins Pipeline.

1. Definition: **Pipeline script from SCM**.
2. SCM: **Git**.
3. Repository URL: <path to repository that contains both the Jenkinsfile and fargate.json>.
4. Credentials: <credentials for repository>.
5. Script path: **Jenkinsfile**.
6. Save.

STEP 5 | Run **Build Now**.

STEP 6 | The tw_fagate.json file will be within the archive of this build https://<jenkins>/job/<pipeline_name>/<job#>/artifact/tw_fargate.json.

```
{
  "containerDefinitions": [
    {
      "command": null,
      "entryPoint": [
        "/bin/tw/fargate/fargate_defender.sh",
        "fargate",
        "entrypoint",
        "entry.sh"
      ],
      "environment": [
        {
          "name": "TW_IMAGE_NAME",
          "value": "matthewabq/twistlock-fargate-auto"
        },
        {
          "name": "TW_CONTAINER_NAME",
          "value": "twistlock-fargate-task"
        }
      ],
      "image": "matthewabq/twistlock-fargate-auto",
      "mountPoints": [
        {
          "containerPath": "/bin/tw/fargate/policy",
          "readOnly": true,
          "sourceVolume": "tw_policy"
        }
      ],
      "name": "twistlock-fargate-task",
      "portMappings": [],
      "volumesFrom": [
        {
          "readOnly": false,
          "sourceContainer": "TwistlockDefender"
        }
      ]
    },
    {
      "entryPoint": [
        "/usr/local/bin/defender",
        "fargate",
        "sidecar"
      ],
      "environment": [
        {
          "name": "INSTALL_BUNDLE",
          "value": "eyJjYS5wZW0iOiItLS0tLUJFR0lOIENFUlRJRklDQVRFLS0tLS1cb
        },
        {
```

Default setting for App-Embedded Defender file system protection

Edit on GitHub

Because App-Embedded Defender's file system protection could affect workload performance, you can enable or disable it.

This procedure is intended for security teams that want to set a global recommendation for whether file system protection should be enabled when teams deploy App-Embedded Defenders.

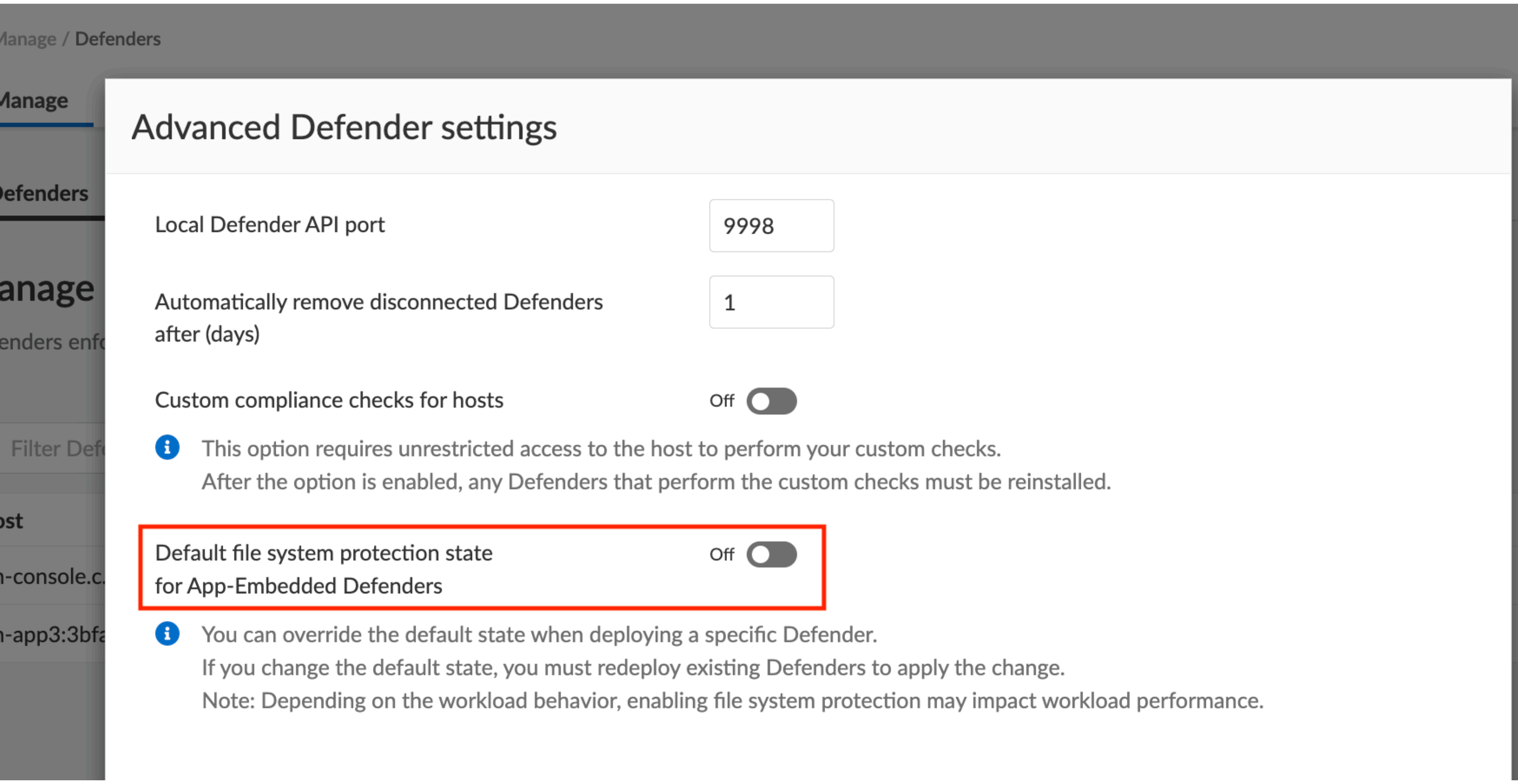
By default, file system protection is disabled in App-Embedded Defenders. Security teams can turn it on by default so that teams that build and manage apps will deploy Defender according to your organization's best practices. Individual teams can optionally override the default setting at embed-time, and they may want to do so if file system protection interferes with their workload's operation.

STEP 1 | Log into Console.

STEP 2 | Go to **Manage > Defenders > Manage > Defenders**.

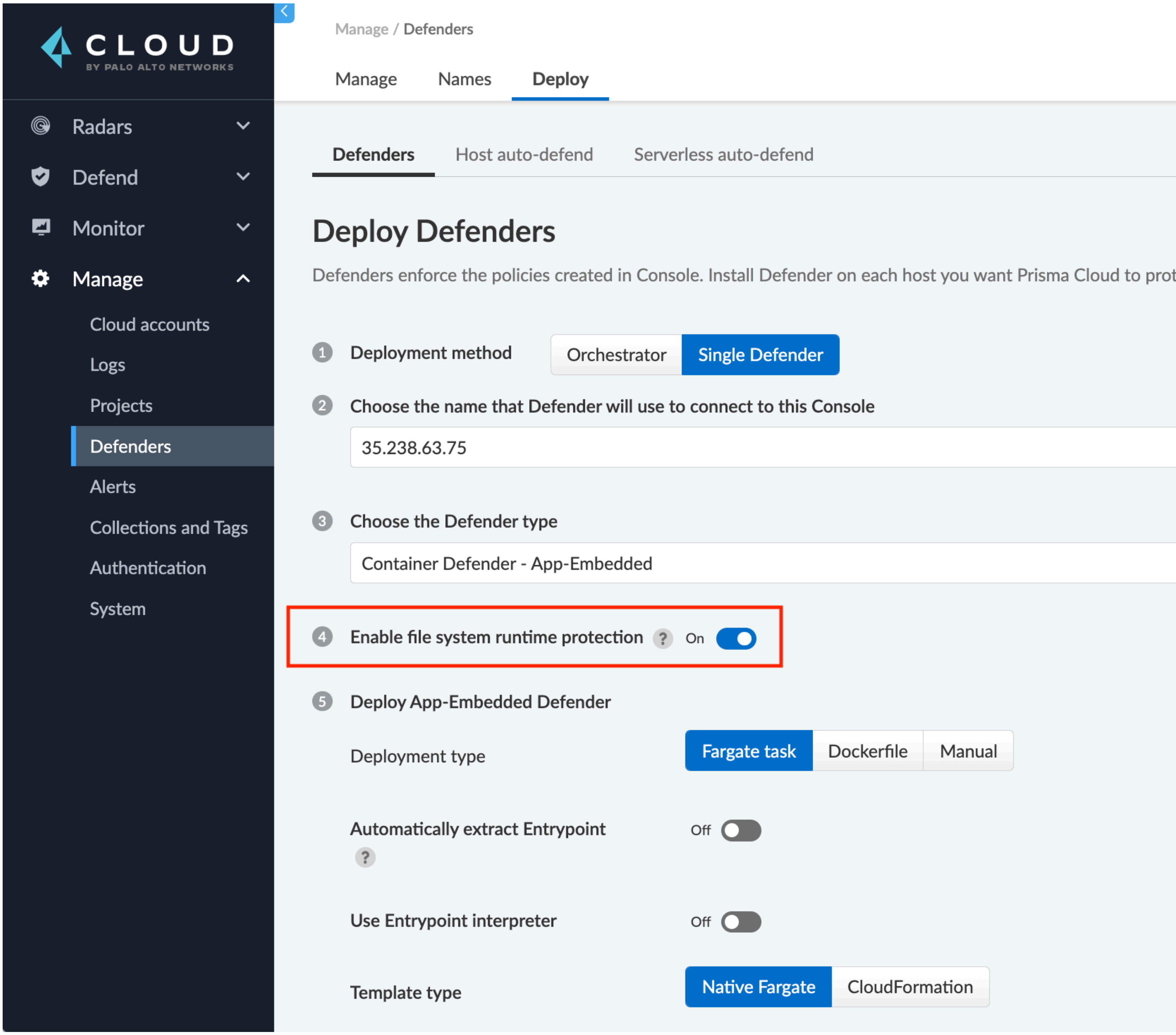
STEP 3 | Click **Advanced settings**.

STEP 4 | Set **Default file system protection statefor App-Embedded Defenders** to **On** or **Off**.



STEP 5 | Validate the global setting has been properly applied by inspecting the Defender embed flow.

1. Go to **Manage > Defenders > Deploy > Defenders**.
2. In **Deployment method**, select **Single Defender**.
3. In **Choose the Defender type**, select **Container Defender - App-Embedded**.
4. Verify that the value for **Monitor file system events** matches the value you set in **Advanced settings**.



VMware Tanzu Application Service (TAS) Defender

Edit on GitHub

You can deploy a dedicated Defender on the Diego cells (Hosts) in your environment. The Prisma Cloud tile installs a TAS Defender on every Diego cell in the TAS environment, including Diego cells that run in Isolation Segments.

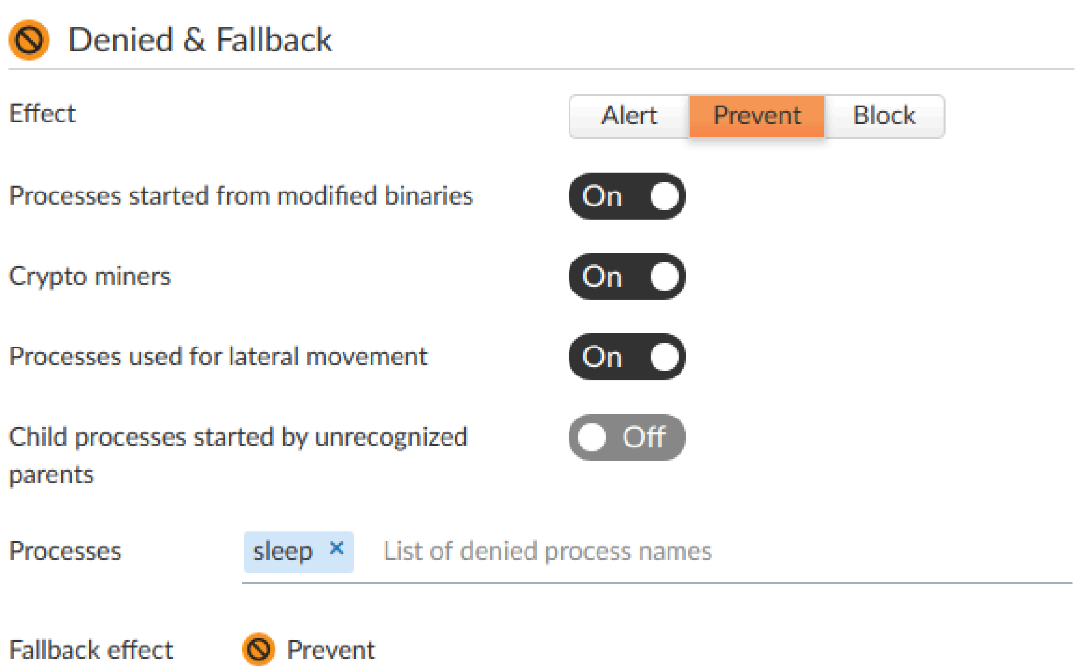
Tanzu Application Service (TAS) Defender supports the following functions:

- Vulnerability scanning for running apps.
- Vulnerability and compliance scanning for the underlying Diego cell hosts.
- Blobstore scanning for buildpack-based apps.
- Runtime protection (process, networking, and file system).

Defender is deployed as BOSH Director addon. Addons are BOSH release jobs that run on each Diego cell host. Defender runs as a service under the root user. The Defender service is monitored by the Bosh agent, with the help of Monit. Note that the Defender service isn't a Garden container.

Console lets you deploy Defender to multiple TAS environments. In Console, Defenders report which Cloud Controller they report to.

Currently, TAS Defender doesn't support blocking for runtime rules, vulnerability rules, and compliance rules. The block action stops the entire container. The app lifecycle is controlled by the Tanzu Application Service framework, so Prisma Cloud cannot effectively block running apps. TAS Defender, however, does support the prevent action.



TAS Defender currently doesn't support custom compliance checks.

Install the TAS Defender

Go to the Tanzu Ops Manager Installation Dashboard to install TAS Defender.

If you're upgrading from a release earlier than 20.09, you must first uninstall any previous versions of TAS Defender. In version 20.09, the tile has been rearchitected. The old tile created a dedicated VM in the TAS environment with a Defender installed, and supported blobstore scanning only. The new tile installs Defender on every Diego cell in the TAS environment, with expanded support for app scanning, host scanning, and runtime defense.

Prerequisites:

- Prisma Cloud Compute Console has already been installed somewhere in your environment.

STEP 1 | In Prisma Cloud Console, go to **Manage > System > Utilities**, and download the TAS tile.

STEP 2 | In the Ops Manager Installation Dashboard, click **Import a Product**, and select the tile you downloaded.

STEP 3 | Retrieve the install command from Prisma Cloud Console. It's used to configure the tile.

1. Go to **Manage > Defenders > Deploy > Single Defender**.
2. Choose the DNS name or IP address the TAS Defender will use to connect to Console. If a suitable option is not available, go to **Manage > Defenders > Names**, and add a DNS name or IP address to the SAN table.
3. Set the Defender type to **TAS Defender**.
4. Copy the install command and set it aside.

STEP 4 | Go to the Tanzu Ops Manager Installation Dashboard.

STEP 5 | Add the Prisma Cloud tile to your staging area. Click the **+** button next to the version of the tile you want to install.

STEP 6 | Click the newly added **Prisma Cloud for TAS** tile.

STEP 7 | Configure the tile.

1. In **Prisma Cloud Component Configuration**, paste the install command you copied from Prisma Cloud Console. Provide a name for this specific TAS Foundation, then click **Save**.

   By default Prisma Cloud performs strict validation of your Cloud Controller's (CC) TLS certificate. If you're using self-signed certificates, this check will fail. To add your custom certificates to trusted cert list, you need to add the custom CA's cert on the VM where the Prisma Cloud tile runs. For more information, see the article on trusted certificates.

   To skip strict validation of your Cloud Controller's (CC) TLS certificate, enable **Skip Cloud Controller TLS validation**. Strict validation verifies the name, signer, and validity date of the CC's certificate. Even with strict validation disabled, the sesssion is still encrypted. Skip strict validation when:

- You're using self-signed certificates.certificates.
- You're using certificates signed by a CA that isn't in your cert store..
- When there's a mismatch between the address you're using to connect to the CC and the common name (CN) or subject alternative name (SAN) in the CC's certificate.

2. In **Credentials**, select your preferred authentication method: Basic Authentication or Certificate-based Authentication:

   For Basic Authentication, enter your Prisma Cloud Console credentials, then click **Save**.

   For certificate-based Authentication, paste the certificate and private key used for authentication in PEM format, then click **Save**.

   Notes:

   - Your role must be Defender Manager or higher.
   - For Certificate-based Authentication, the root CA used to sign the certificate used for authentication must be entered under **Manage > Authentication > System Certificates > Certificate-based authentication to Console**.

STEP 8 | Install the Prisma Cloud tile. Return to the Ops Manager Installation Dashboard, click **Review Pending Changes**, select both **Prisma Cloud for TAS** and **Tanzu Application Service**, then click **Apply changes**.

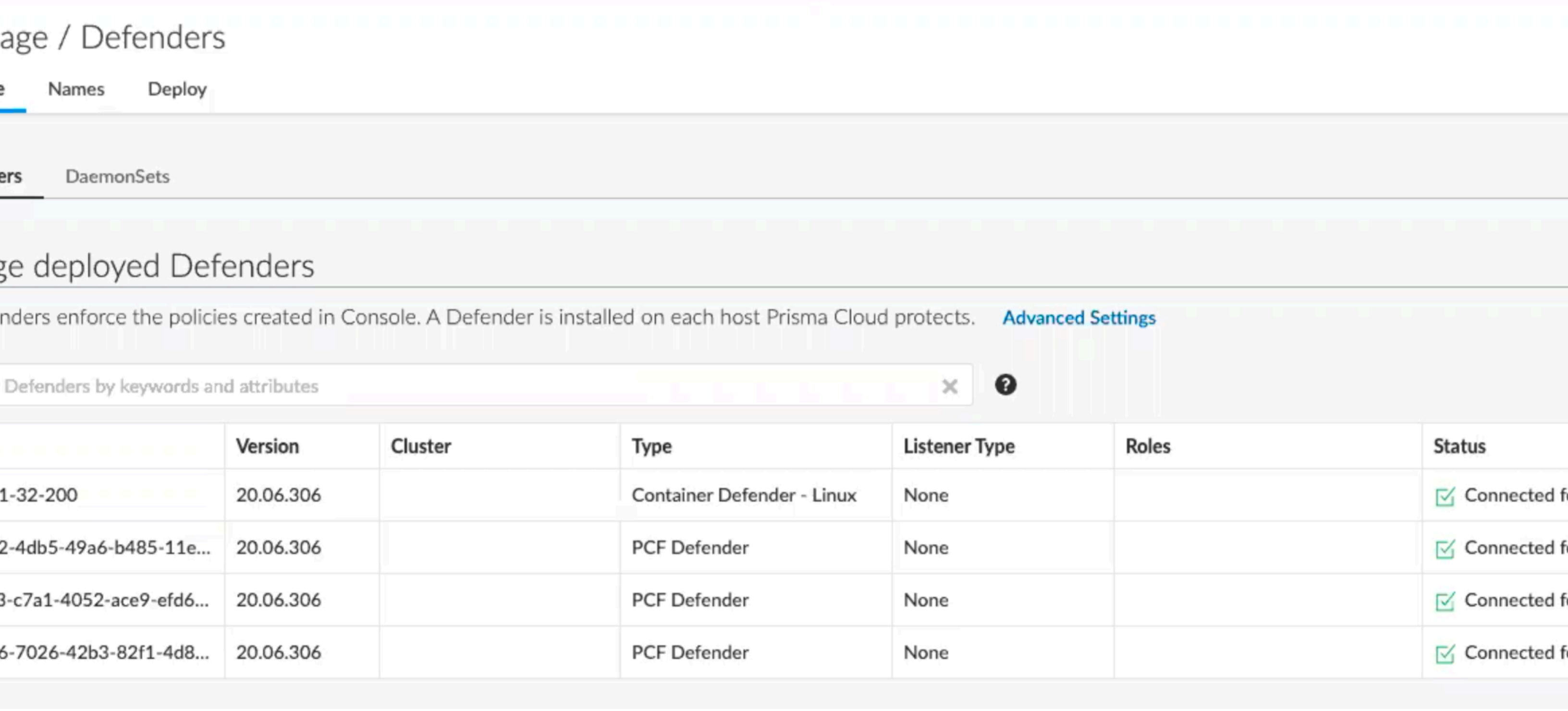
Tanzu Application Service must be staged when installing the Prisma Cloud tile.

*In order to deploy a new Defender after the Prisma Cloud tile is installed (e.g. if new Diego cells or isolation segments have been installed in the TAS environment), **Apply changes** on Ops Manager is required.*

STEP 9 | After the changes are applied, validate that Prisma Cloud Defenders are running in your environment.

1. Log into Prisma Cloud Console.
2. Go to **Manage > Defenders > Manage > Defenders**.

   In the table of deployed Defenders, you should see a Defender of type **Tanzu Application Service**, one per Diego cell, including Diego cells that run in the Isolation Segments of your TAS environment.



Prisma Cloud reports the agentID in the Host field. To correlate an agentID to a Diego cell IP address, and determine exactly which host runs a Defender, login to an Diego cell, and inspect /var/vcap/instance/dns/records.json. This file shows how the agentID maps to a host IP address.

*If a TAS Defender disconnects from Console for more than one day, all data it collected is purged from Console. The Defender is also removed from the table in **Manage > Defenders > Manage**. The period of time that data from a disconneted Defender is retained (by default, one day) can be configured in **Manage > Defenders > Manage > Defenders > Advanced Settings**.*

Utilizing collections with TAS metadata fields

Prisma Cloud automatically collects metadata fields such as Foundation, Organization Name, Application Name and ID, and Space Name and ID. To utilize these fields, you'll have to **manually create** appropriate collections that can then be used for filtering and aggregation.

Resource type	Supported Labels
Host	tas-foundation
Containers (running applications)	tas-application-id, tas-application-name, tas-space-id, tas-space-name, tas-org-id, tas-org-name, tas-foundation
Droplets	tas-application-id, tas-application-name, tas-space-id, tas-space-name, tas-org-id, tas-org-name, tas-foundation

Serverless Defender

Edit on GitHub

- Securing serverless functions
- AWS Lambda - (Optional) Download your function as a ZIP file
- AWS Lambda - Embed Serverless Defender into C# functions
- AWS Lambda - Embed Serverless Defender into Java functions
- AWS Lambda - Embed Serverless Defender into Node.js functions
- AWS Lambda - Embed Serverless Defender into Python functions
- AWS Lambda - Embed Serverless Defender into Ruby functions
- AWS Lambda - Upload the protected function
- Azure Functions - Embed Serverless Defender into C# functions
- Defining your runtime protection policy
- Defining your serverless WAAS policy

Serverless Defender protects serverless functions at runtime. It monitors your functions to ensure they execute as designed.

Per-function policies let you control:

- Process activity. Enables verification of launched subprocesses against policy.
- Network connections. Enables verification of inbound and outbound connections, and permits outbound connections to explicitly allowed domains.
- File system activity. Controls which parts of the file system functions can access.

Prisma Cloud supports AWS Lambda functions (Linux) and Azure Functions (Windows only).

See system requirements for the runtimes and architectures that are supported for Serverless Defenders.

The following runtimes are supported for AWS Lambda:

- C# (.NET Core) 3.1
- Java 8, 11
- Node.js 12.x, 14.x

- Python 3.6, 3.7, 3.8, 3.9
- Ruby 2.7

Serverless Defenders are not supported on ARM64 architecture.

The following runtimes are supported for Azure Functions (Windows only):

- v3 - C# (.NET Core) 3.1, 5.0
- v4 - C# (.NET Core) 4.8, 6.0

*Only users with the Administrator role can see the list of deployed Serverless Defenders in **Manage > Defenders > Manage**.*

Securing serverless functions

To secure a serverless function, embed the Prisma Cloud Serverless Defender into it. The steps are:

1. (Optional) If you are not using a deployment framework like SAM or Serverless Framework, download a ZIP file that contains your function source code and dependencies.
2. Embed the Serverless Defender into the function.
3. Deploy the new function or upload the updated ZIP file to the cloud provider.
4. Define a serverless protection runtime policy.
5. Define a serverless WAAS policy.

AWS Lambda - (Optional) Download your function as a ZIP file

Download your function's source code from AWS as a ZIP file.

STEP 1 | From Lambda's code editor, click **Actions > Export function**.

STEP 2 | Click **Download deployment package**.

Your function is downloaded to your host as a ZIP file.

STEP 3 | Create a working directory, and unpack the ZIP file there.

In the next step, you'll download the Serverless Defender files to this working directory.

AWS Lambda - Embed Serverless Defender into C# functions

In your function code, import the Serverless Defender library and create a new protected handler that wraps the original handler. The protected handler will be called by AWS when your function is invoked. Update the project configuration file to add Prisma Cloud dependencies and package references.

Prisma Cloud supports .NET Core 3.1.

STEP 1 | Open Compute Console, and go to **Manage > Defenders > Deploy > Defenders > Single Defender**.

STEP 2 | Choose the DNS name or IP address Serverless Defender uses to connect to Console.

STEP 3 | In **Choose Defender type**, select **Serverless Defender - AWS**.

STEP 4 | In **Runtime**, select **C#**.

STEP 5 | Download the Serverless Defender package to your workstation.

STEP 6 | Unzip the Serverless Defender bundle into your working directory.

STEP 7 | Embed the serverless Defender into the function by importing the Prisma Cloud library and wrapping the function's handler.

Function input and output can be a struct or a stream. Functions can be synchronous or asynchronous. The context parameter is optional in .NET, so it can be omitted.

```
 using Twistlock;

 public class ... {
     // Original handler
     public ApplicationLoadBalancerResponse
Handler(ApplicationLoadBalancerRequest request, ILambdaContext
context)
     {
       ...
     }

     // Application load balancer example
     // Twistlock protected handler
     public ApplicationLoadBalancerResponse
ProtectedHandler(ApplicationLoadBalancerRequest request,
ILambdaContext context)
     {
         return
Twistlock.Serverless.Handler<ApplicationLoadBalancerRequest,
ApplicationLoadBalancerResponse>(Handler, request, context);
     }
     ...
 }
```

STEP 8 | Add the Twistlock package as a dependency in your nuget.config file.

If a nuget.config file doesn't exist, create one.

```
<configuration>
  <packageSources>
    <add key="local-packages" value="./twistlock"/>
  </packageSources>
</configuration>
```

STEP 9 | Reference the Twistlock package in your csproj file.

```
<Project>
  <ItemGroup>
    <PackageReference Include="Twistlock" Version="19.11.462"/>
    <TwistlockFiles Include="twistlock/*" Exclude="twistlock/
twistlock.19.11.462.nupkg"/>
  </ItemGroup>
  <Target Name="CopyCustomContentOnPublish" AfterTargets="Publish">
```



```
    <Copy SourceFiles="@(TwistlockFiles)"
 DestinationFolder="$(PublishDir)/twistlock"/>
  </Target>
   .
   .
   .
</Project>
```

STEP 10 | Generate the value for the TW_POLICY environment variable by specifying your function's name and region.

> *If **Any** is selected for region, only policies that contain * in the region label will be matched.*

Serverless Defender uses TW_POLICY to determine how to connect to Compute Console to retrieve policy and send audits.

Copy the value generated for TW_POLICY, and set it aside.

STEP 11 | Upload the protected function to AWS, and set the TW_POLICY environment variable.

AWS Lambda - Embed Serverless Defender into Java functions

To embed Serverless Defender, import the Twistlock package and update your code to start Serverless Defender as soon as the function is invoked. Prisma Cloud supports both Maven and Gradle projects. You'll also need to update your project metadata to include Serverless Defender dependencies.

Prisma Cloud supports both predefined interfaces in the AWS Lambda Java core library: RequestStreamHandler (where input must be serialized JSON) and RequestHandler.

AWS lets you specify handlers as functions or classes. In both cases, Twistlock.Handler(), the entry point to Serverless Defender, assumes the entry point to your code is named handleRequest. After embedding Serverless Defender, update the name of the handler registered with AWS to be the wrapper method that calls Twistlock.Handler() (for example, protectedHandler).

Prisma Cloud supports both service struct and stream input (serialized struct). Even though the Context parameter is optional for unprotected functions, it's manadatory when embedding Serverless Defender.

Prisma Cloud supports Java 8 and Java 11.

STEP 1 | Open Compute Console, and go to **Manage > Defenders > Deploy > Defenders > Single Defender**.

STEP 2 | Choose the DNS name or IP address Serverless Defender uses to connect to Console.

STEP 3 | In **Choose Defender type**, select **Serverless Defender - AWS**.

STEP 4 | In **Runtime**, select **Java**.

STEP 5 | In **Package**, select **Maven** or **Gradle**.

The steps for embedding Serverless Defender differ depending on the build tool.

STEP 6 | Download the Serverless Defender package to your workstation.

STEP 7 | Unzip the Serverless Defender bundle into your working directory.

STEP 8 | Inside the *twistlock* directory (the root directory in the zip), create a new sub-directory with the following structure: *com/twistlock/serverless/defender/<version>/*. For example, for version 22.06.286:

```
mkdir -p com/twistlock/serverless/defender/22.06.286
```

STEP 9 | Move the *twistlock*.jar* file into this new subdirectory.

STEP 10 | Rename the *.jar file to the convention: defender-<version>.jar (e.g. defender-22.06.286.jar).

STEP 11 | Create a file called defender-<version>-pom.xml in the same location of the jar (change the version tag based on your version):

1. Enter the package details and artifact id in the **pom.xml* file:

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.twistlock.serverless</groupId>
  <artifactId>defender</artifactId>
  <version>22.06.286</version>
  <description>twistlock serverless defender pom</description>
</project>
```

STEP 12 | Embed Serverless Defender into your function by importing the Prisma Cloud package and wrapping the function's handler.

```
import com.twistlock.serverless.Twistlock;

public class ... implements
 RequestHandler<APIGatewayProxyRequestEvent,
 APIGatewayProxyResponseEvent> {

  // Original handler
  @Override
  public APIGatewayProxyResponseEvent
 handleRequest(APIGatewayProxyRequestEvent request, Context
 context) {
  {
    ...
  }

  // RequestHandler example
  // Twistlock protected handler
  public APIGatewayProxyResponseEvent
 protectedHandler(APIGatewayProxyRequestEvent request, Context
 context) {
    return Twistlock.Handler(this, request, context);
  }
  ...
}
```

```
...
```

STEP 13 | Update your project configuration file.

1. **Maven**

   Update your **pom.xml* file. Don't create new sections for the Prisma Cloud configurations. Just update existing sections. For example, don't create a new <plugins> section if one exists already. Just append a <plugin> section to it.

   Add the assembly plugin to include the Twistlock package in the final function JAR. Usually the shade plugin is used in AWS to include packages to standalone JARs, but it doesn't let you include local system packages.

```
<project>
  <build>
    <!-- Add assembly plugin to create a standalone jar that
 contains Twistlock library -->
    <plugins>
      <plugin>
        <artifactId>maven-assembly-plugin</artifactId>
        <configuration>
          <appendAssemblyId>false</appendAssemblyId>
          <descriptors>
            <descriptor>assembly.xml</descriptor>
          </descriptors>
        </configuration>
        <executions>
          <execution>
           <id>make-assembly</id>
           <phase>package</phase>
           <goals>
            <goal>attached</goal>
           </goals>
          </execution>
        </executions>
      </plugin>
      ...
    </plugins>
```

```
  <!-- Add Twistlock resources -->
  <resources>
    <resource>
      <directory>${project.basedir}</directory>
      <includes>
        <include>twistlock/*</include>
        </includes>
      <excludes>
        <exclude>twistlock/com/**</exclude>
      </excludes>
    </resource>
    ...
  </resources>
  ...
```

```
</build>
```

```
<!-- Define the internal (local) repository in the `*pom.xml`
 file: -->
<project>
  <repositories>
    <repository>
      <id>twistlock-internal</id>
      <name>twistlock</name>
      <url>file://${project.basedir}/twistlock</url>
    </repository>
  ...
</project>
```

```
  <!-- Add Twistlock package reference -->
  <dependencies>
    <dependency>
      <groupId>com.twistlock.serverless</groupId>
      <artifactId>defender</artifactId>
      <version>22.06.286</version>
    </dependency>
    ...
  </dependencies>
  ...
</project>
```

2. Create an assembly.xml file, which packs all dependencies in a standalone JAR.

```
<assembly>
  <id>twistlock-protected</id>
  <formats>
    <format>jar</format>
  </formats>
  <includeBaseDirectory>false</includeBaseDirectory>
  <dependencySets>
    <!-- Unpack runtime dependencies into runtime jar -->
    <dependencySet>
      <unpack>true</unpack>
      <scope>runtime</scope>
    </dependencySet>
    <!-- Unpack local system dependencies into runtime jar -->
    <dependencySet>
      <unpack>true</unpack>
      <scope>system</scope>
    </dependencySet>
  </dependencySets>
</assembly>
```


STEP 14 | Gradle

Gradle supports Maven repositories and can fetch artifacts directly from any kind of Maven repository.

Update your *build.gradle* file.

1. Add the Maven repository for this project.
2. Set the *.jar* file as an "implementation" dependency from the filesystem.
3. Update the zip resources.

```
repositories {
    maven {
        url "file://$projectDir/twistlock"
    }
}

dependencies {
    implementation
 'com.twistlock.serverless:defender:22.06.286'
}

task buildZip(type: Zip) {
    from compileJava
    from processResources
    into('lib') {
        from configurations.runtimeClasspath
    }
    // Include Twistlock resources
    into ('twistlock') {
        from 'twistlock'
        exclude "com/**"
    }
}

build.dependsOn buildZip
```

STEP 15 | In AWS, set the name of the Lambda handler for your function to protectedHandler.

STEP 16 | Generate the value for the TW_POLICY environment variable by specifying your function's name and region.

*If **Any** is selected for region, only policies that contain * in the region label will be matched.*

Serverless Defender uses TW_POLICY to determine how to connect to Compute Console to retrieve policy and send audits.

Copy the value generated for TW_POLICY, and set it aside.

STEP 17 | Upload the protected function to AWS, and set the TW_POLICY environment variable.

AWS Lambda - Embed Serverless Defender into Node.js functions

Import the Serverless Defender module, and configure your function to start it. Prisma Cloud supports Node.js 12.x, and 14.x.

STEP 1 | Open Compute Console, and go to **Manage > Defenders > Deploy > Single Defender**.

STEP 2 | Choose the DNS name or IP address Serverless Defender uses to connect to Console.

STEP 3 | In **Choose Defender type**, select **Serverless**.

STEP 4 | In **Runtime**, select **Node.js**.

STEP 5 | Download the Serverless Defender package to your workstation.

STEP 6 | Unzip the Serverless Defender bundle into your working directory.

STEP 7 | Embed the serverless Defender into the function by importing the Prisma Cloud library and wrapping the function's handler.

1. For asynchronous handlers:

```
// Async handler
var twistlock = require('./twistlock');
exports.handler = async (event, context) => {
.
.
.
};
exports.handler = twistlock.asyncHandler(exports.handler);
```

2. For synchronous handlers:

```
// Non-async handler
var twistlock = require('./twistlock');
exports.handler = (event, context, callback) => {
.
.
.
};
exports.handler = twistlock.handler(exports.handler);
```

STEP 8 | Generate the value for the TW_POLICY environment variable by specifying your function's name and region.

*If **Any** is selected for region, only policies that contain * in the region label will be matched.*

Serverless Defender uses TW_POLICY to determine how to connect to Compute Console to retrieve policy and send audits.

Copy the value generated for TW_POLICY, and set it aside.

STEP 9 | Upload the protected function to AWS, and set the TW_POLICY environment variable.

- Prisma Cloud Serverless Defender includes native node.js libraries. If you are using webpack, please refer to tools such as native-addon-loader to make sure these libraries are included in the function ZIP file.

AWS Lambda - Embed Serverless Defender into Python functions

Import the Serverless Defender module, and configure your function to invoke it. Prisma Cloud supports Python 3.6, 3.7, and 3.8.

STEP 1 | Open Compute Console, and go to **Manage** > **Defenders** > **Deploy** > **Single Defender**.

STEP 2 | Choose the DNS name or IP address Serverless Defender uses to connect to Console.

STEP 3 | In **Choose Defender type**, select **Serverless**.

STEP 4 | In **Runtime**, select **Python**.

STEP 5 | Download the Serverless Defender package to your workstation.

STEP 6 | Unzip the Serverless Defender bundle into your working directory.

STEP 7 | Embed the serverless Defender into the function by importing the Prisma Cloud library and wrapping the function's handler.

```
import twistlock.serverless
@twistlock.serverless.handler
def handler(event, context):
.
.
.
```

STEP 8 | Generate the value for the TW_POLICY environment variable by specifying your function's name and region.

*If **Any** is selected for region, only policies that contain * in the region label will be matched.*

Serverless Defender uses TW_POLICY to determine how to connect to Compute Console to retrieve policy and send audits.

Copy the value generated for TW_POLICY, and set it aside.

STEP 9 | Upload the protected function to AWS, and set the TW_POLICY environment variable.

AWS Lambda - Embed Serverless Defender into Ruby functions

Import the Serverless Defender module, and configure your function to invoke it. Prisma Cloud supports Ruby 2.7.

STEP 1 | Open Compute Console, and go to **Manage** > **Defenders** > **Deploy** > **Single Defender**.

STEP 2 | Choose the DNS name or IP address Serverless Defender uses to connect to Console.

STEP 3 | In **Choose Defender type**, select **Serverless**.

STEP 4 | In **Runtime**, select **Ruby**.

STEP 5 | Download the Serverless Defender package to your workstation.

STEP 6 | Unzip the Serverless Defender bundle into your working directory.

STEP 7 | Embed the serverless Defender into the function by importing the Prisma Cloud library and wrapping the function's handler.

1. Option 1:

```
require_relative './twistlock/twistlock'
def handler(event:, context:)
    Twistlock.handler(event: event, context: context) { |
event:, context:|
        # Original handler
        ...
    }
end
.
.
.
```

2. Option 2:

```
require_relative './twistlock/twistlock'
# Handler as a class method
module Module1
    class Class1
        def self.original_handler(event:, context:)
            ...
        end
        def self.protected_handler(event:, context:)
            return Twistlock.handler(event: event, context:
 context, &method(:original_handler))
        end
    end
end
.
.
.
```

STEP 8 | Generate the value for the TW_POLICY environment variable by specifying your function's name and region.

*If **Any** is selected for region, only policies that contain * in the region label will be matched.*

Serverless Defender uses TW_POLICY to determine how to connect to Compute Console to retrieve policy and send audits.

Copy the value generated for TW_POLICY, and set it aside.

STEP 9 | Upload the protected function to AWS, and set the TW_POLICY environment variable.

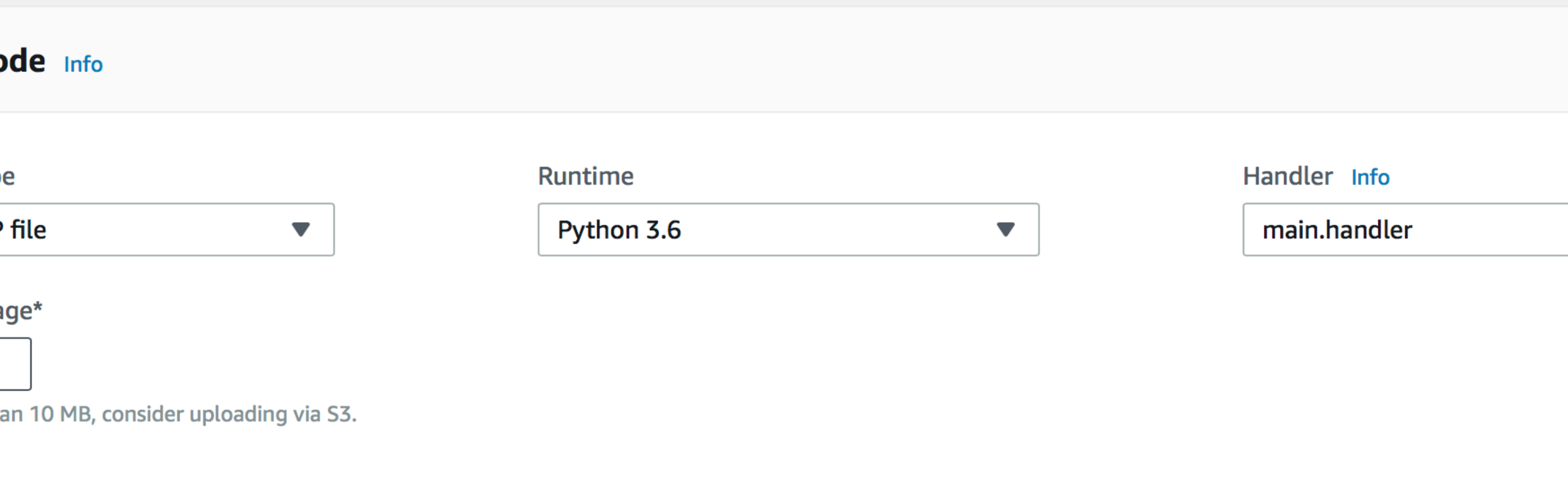
AWS Lambda - Upload the protected function

After embedding Serverless Defender into your function, upload it to AWS. If you are using a deployment framework such as SAM or Serverless Framework just deploy the function with your standard deployment procedure. If you are using AWS directly, follow the steps below:

STEP 1 | Upload the new ZIP file to AWS.

1. In **Designer**, select your function so that you can view the function code.
2. Under **Code entry type**, select **Upload a .ZIP file**.
3. Specify a runtime and the handler.

   Validate that **Runtime** is a supported runtime, and that **Handler** points to the function's entry point.
4. Click **Upload**.



5. Click **Save**.

STEP 2 | Set the TW_POLICY environment variable.

1. In Designer, open the environment variables panel.
2. For Key, enter TW_POLICY.
3. For Value, paste the rule you copied from Compute Console.
4. Click Save.

Azure Functions - Embed Serverless Defender into C# functions

In your function code, import the Serverless Defender library and create a new protected handler that wraps the original handler. The protected handler will be called by Azure when your function is invoked. Update the project configuration file to add Prisma Cloud dependencies and package references.

Prisma Cloud supports .NET Core 3.1 on Windows.

STEP 1 | Open Compute Console, and go to **Manage > Defenders > Deploy > Single Defender**.

STEP 2 | Choose the DNS name or IP address Serverless Defender uses to connect to Console.

STEP 3 | In **Choose Defender type**, select **Serverless Defender - Azure**.

STEP 4 | In **Runtime**, select **C#**.

STEP 5 | Download the Serverless Defender package to your workstation.

STEP 6 | Unzip the Serverless Defender bundle into your working directory.

STEP 7 | Embed the serverless Defender into the function by importing the Prisma Cloud library and wrapping the function's handler.

Function input and output can be a struct or a stream. Functions can be synchronous or asynchronous. The context parameter is optional in .NET, so it can be omitted.

```
using Twistlock;

public class ... {
// Original handler
public static async Task<IActionResult> Run(
      [HttpTrigger(AuthorizationLevel.Function, "get", "post",
 Route = null)] HttpRequest req,
      ILogger log, ExecutionContext context)
      {
       Twistlock.Serverless.Init(log, context);
       ...
      }
}
```

STEP 8 | Add the Twistlock package as a dependency in your nuget.config file.

If a nuget.config file doesn't exist, create one.

```
<configuration>
  <packageSources>
    <add key="local-packages" value="./twistlock"/>
  </packageSources>
</configuration>
```

STEP 9 | Reference the Twistlock package in your project configuration file.

```
<Project>
  <ItemGroup>
    <PackageReference Include="Twistlock" Version="22.04.147" />
    <TwistlockFiles Include="twistlock\*" Exclude="twistlock
\twistlock.22.04.147.nupkg"/>
  </ItemGroup>
  <ItemGroup>
    <None Include="@(TwistlockFiles)"
 CopyToOutputDirectory="Always" LinkBase="twistlock\" />
  </ItemGroup>
  ...
</Project>
```


STEP 10 | Generate the value for the TW_POLICY environment variable by specifying your function's name and region.

*If **Any** is selected for region, only policies that contain a wildcard in the region label will be matched.*

Serverless Defender uses TW_POLICY to determine how to connect to Compute Console to retrieve policy and send audits.

Copy the value generated for TW_POLICY, and set it aside.

STEP 11 | Upload the protected function to Azure, and set the TW_POLICY environment variable.

Defining your runtime protection policy

By default, Prisma Cloud ships with an empty serverless runtime policy. An empty policy disables runtime defense entirely.

You can enable runtime defense by creating a rule. By default, new rules:

- Apply to all functions (*), but you can target them to specific functions by function name.
- Block all processes from running except the main process. This protects against command injection attacks.

When functions are invoked, they connect to Compute Console and retrieve the latest policy. To ensure that functions start executing at time=0 with your custom policy, predefine the policy. Predefined policy is embedded into your function along with the Serverless Defender by way of the *TW_POLICY* environment variable.

STEP 1 | Log into Prisma Cloud Console.

STEP 2 | Go to **Defend** > **Runtime** > **Serverless Policy**.

STEP 3 | Click **Add rule**.

STEP 4 | In the **General** tab, enter a rule name.

STEP 5 | (Optional) Target the rule to specific functions.

Use collections to scope functions by name or region (label). Pattern matching is supported. For Azure Funtions only, you can additionally scope rules by account ID.

STEP 6 | Set the rule parameters in the **Processes**, **Networking**, and **File System** tabs.

STEP 7 | Click **Save**.

Defining your serverless WAAS policy

Prisma Cloud lets you protect your serverless functions against application layer attacks by utlizing the serverless Web Application and API Security (WAAS).

By default, the serverless WAAS is disabled. To enable it, add a new serverless WAAS rule.

STEP 1 | Log into Prisma Cloud Console.

STEP 2 | Go to **Defend** > **WAAS** > **Serverless**.

STEP 3 | Click **Add rule**.

STEP 4 | In the **General** tab, enter a rule name.

STEP 5 | (Optional) Target the rule to specific functions.

Use collections to scope functions by name or region (label). Pattern matching is supported. For Azure Funtions only, you can additionally scope rules by account ID.

STEP 6 | Set the protections you want to apply (**SQLi**, **CMDi**, **Code injection**, **XSS**, **LFI**).

STEP 7 | Click **Save**.

Serverless Defender as a Lambda layer

Edit on GitHub

Prisma Cloud Serverless Defenders protect serverless functions at runtime. Currently, Prisma Cloud supports AWS Lambda functions.

Lambda layers are ZIP archives that contain libraries, custom runtimes, or other dependencies. Layers let you add reusable components to your functions, and focus deployment packages on business logic. They are extracted to the */opt* directory in the function execution environment. For more information, see the AWS Lambda layers documentation.

Prisma Cloud delivers Serverless Defender as a Lambda layer. Deploy Serverless Defender to your function by wrapping the handler and setting an environment variable. See system requirements for the runtimes that are supported for Serverless Defender as a Lambda layer.

Securing serverless functions

To secure an AWS Lambda function with the Serverless Defender layer:

1. Download the Serverless Defender Lambda layer ZIP file.
2. Upload the layer to AWS.
3. Define a serverless protection runtime policy.
4. Define a serverless WAAS policy.
5. Add the layer to your function, update the handler, and set an environment variable. After completing this integration, Serverless Defender runs when your function is invoked.

Download the Serverless Defender layer

Download the Serverless Defender layer from Compute Console.

STEP 1 | Open Console, then go to **Manage > Defenders > Deploy> Defenders > Single Defender**.

STEP 2 | Choose the DNS name or IP address that Serverless Defender uses to connect to Console.

STEP 3 | Set the Defender type to **Serverless Defender**.

STEP 4 | Select a runtime.

Prisma Cloud supports Lambda layers for **Node.js**, **Python**, and **Ruby** only.

STEP 5 | For **Deployment Type**, select **Layer**.

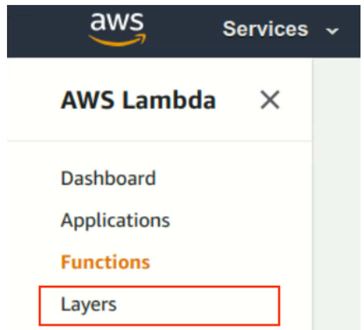
STEP 6 | Download the Serverless Defender layer. A ZIP file is downloaded to your host.

Upload the Serverless Defender layer to AWS

Add the layer to the AWS Lambda service as a resource available to all functions.

STEP 1 | In the AWS Management Console, go to the Lambda service.

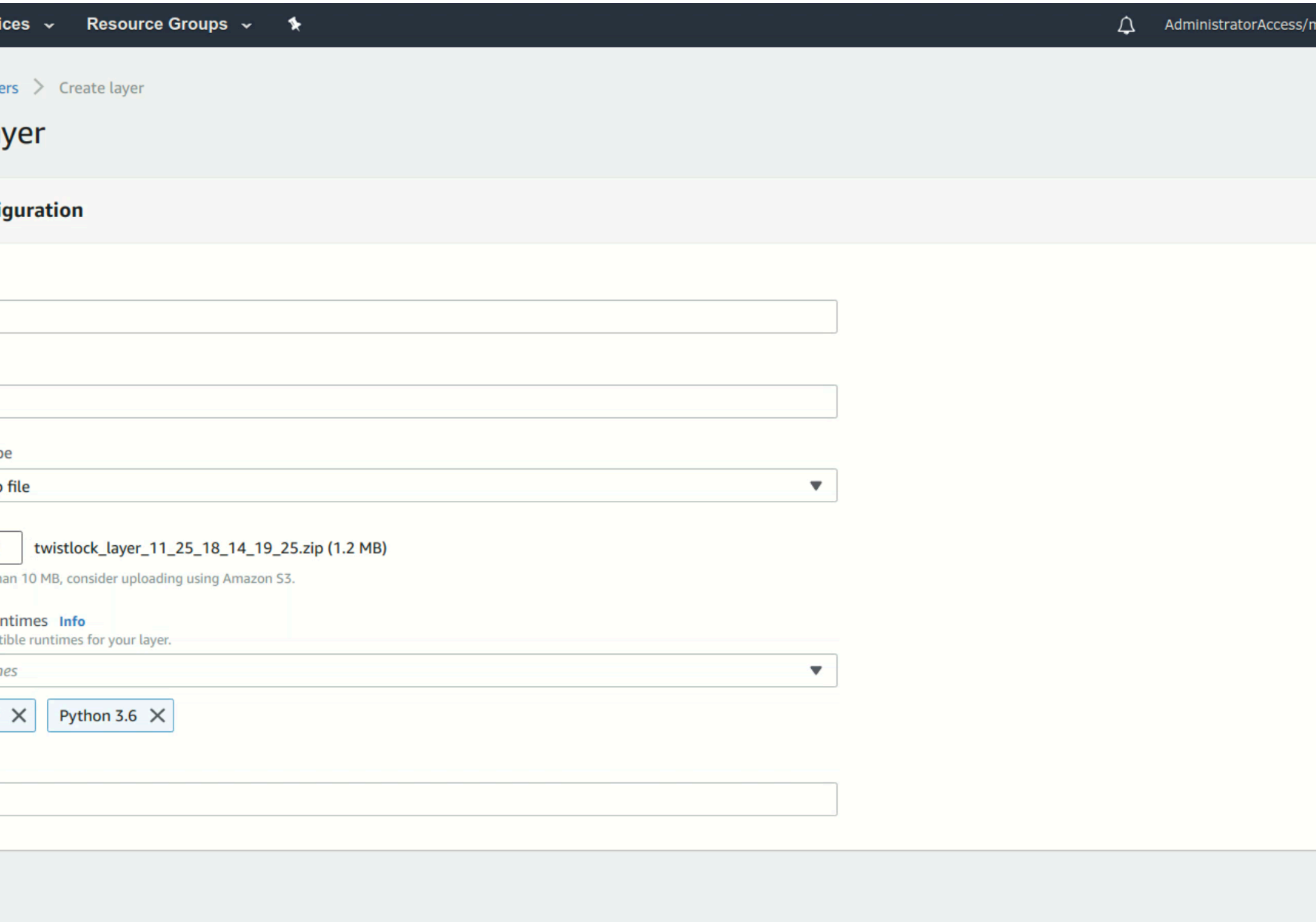
STEP 2 | Select **Layers > Create Layer**.



STEP 3 | In **Name**, enter **twistlock**.

STEP 4 | Click **Upload**, and select the file you just downloaded, *twistlock_defender_layer.zip*

1. Select the compatible runtimes: **Python**, **Node.js**, or **Ruby**.
2. Click **Create**.



Defining your runtime protection policy

By default, Prisma Cloud ships with an empty serverless runtime policy. An empty policy disables runtime defense entirely.

You can enable runtime defense by creating a rule. By default, new rules:

- Apply to all functions (*), but you can target them to specific functions by function name.
- Block all processes from running except the main process. This protects against command injection attacks.

When functions are invoked, they connect to Compute Console and retrieve the latest policy. To ensure that functions start executing at time=0 with your custom policy, you must predefine the policy. Predefined policy is embedded into your function along with the Serverless Defender by way of the *TW_POLICY* environment variable.

STEP 1 | Log into Prisma Cloud Console.

STEP 2 | Go to **Defend > Runtime > Serverless Policy**.

STEP 3 | Click **Add rule**.

STEP 4 | In the **General** tab, enter a rule name.

STEP 5 | (Optional) Target the rule to specific functions.

STEP 6 | Set the rule parameters in the **Processes**, **Networking**, and **File System** tabs.

STEP 7 | Click **Save**.

Defining your serverless WAAS policy

Prisma Cloud lets you protect your serverless functions against application layer attacks by utlizing the serverless Web Application and API Security (WAAS).

By default, the serverless WAAS is disabled. To enable it, add a new serverless WAAS rule.

STEP 1 | Log into Prisma Cloud Console.

STEP 2 | Go to **Defend > WAAS > Serverless**.

STEP 3 | Click **Add rule**.

STEP 4 | In the **General** tab, enter a rule name.

STEP 5 | (Optional) Target the rule to specific functions.

STEP 6 | Set the protections you want to apply (**SQLi**, **CMDi**, **Code injection**, **XSS**, **LFI**).

STEP 7 | Click **Save**.

Embed the Serverless Defender

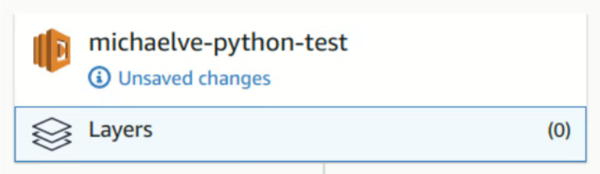
Embed the Serverless Defender as a layer, and run it when your function is invoked. If you are using a deployment framework such as SAM or Serverless Framework you can reference the layer from within the configuration file.

Prerequisites:

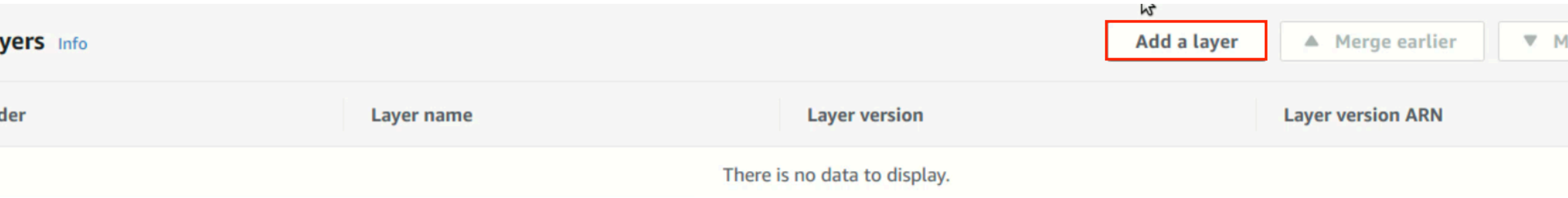
- You already have a Lambda function.
- Your Lambda function is written for Node.js, Python, or Ruby.
- Your function's execution role grants it permission to write to CloudWatch Logs. Note that the **AWSLambdaBasicExecutionRole** grants permission to write to CloudWatch Logs.

STEP 1 | Go to the function designer in the AWS Management Console.

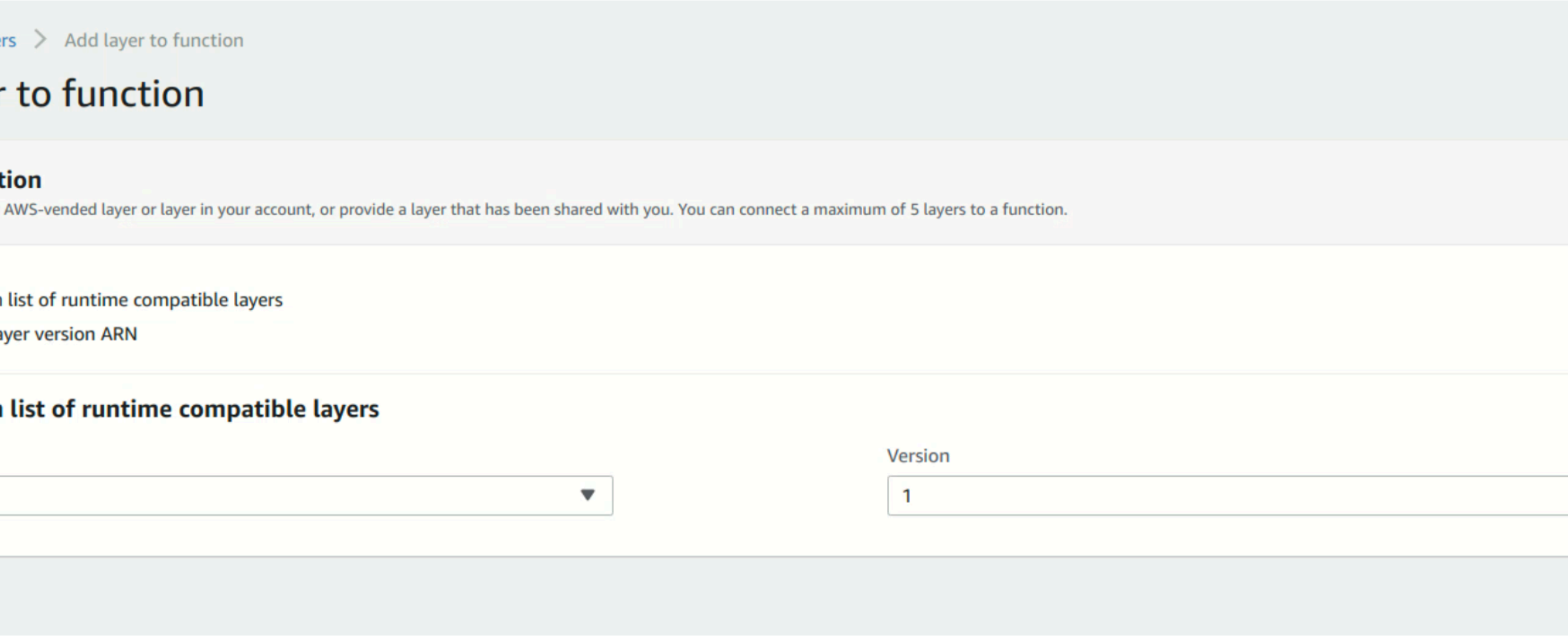
STEP 2 | Click on the **Layers** icon.



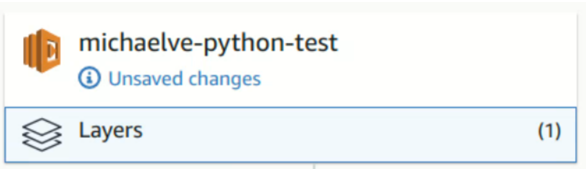
STEP 3 | In the **Referenced Layers** panel, click **Add a layer**.



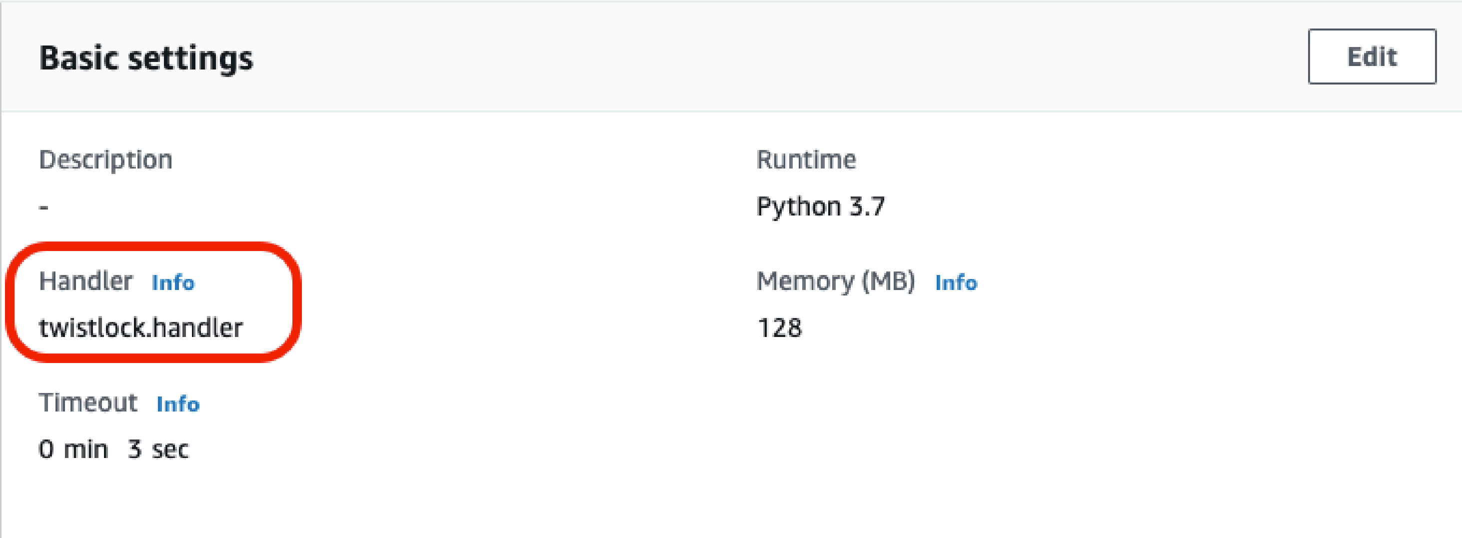
1. In the **Select from list of runtime compatible layers**, select **twistlock**.
2. In the **Version** drop-down list, select **1**.
3. Click **Add**.



When you return to the function designer, you'll see that your function now uses one layer.



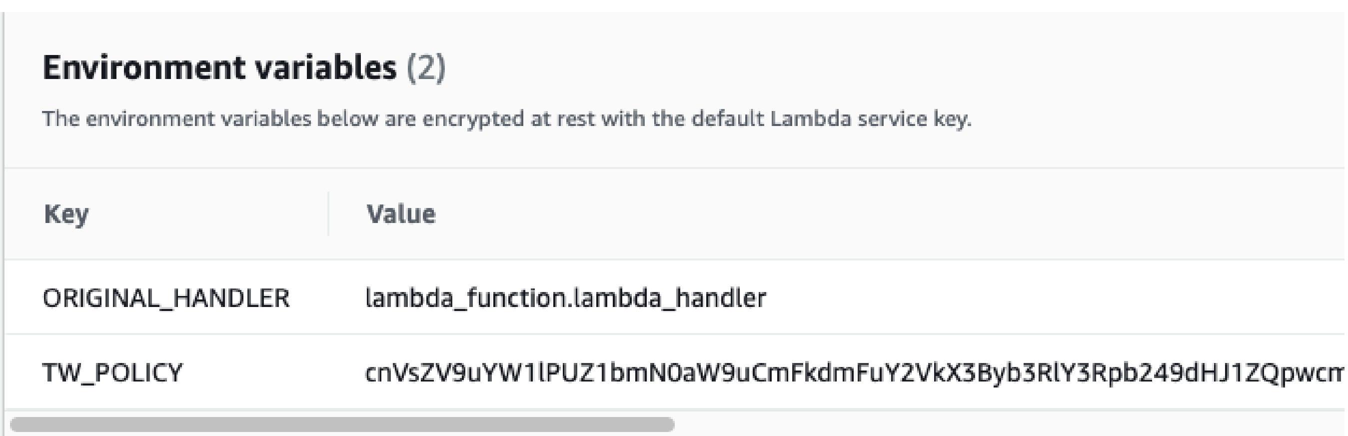
STEP 4 | Update the handler for your function to be *twistlock.handler*.



STEP 5 | Set the *TW_POLICY* and *ORIGINAL_HANDLER* environment variable, which specifies how your function connects to Compute Console to retrieve policy and send audits.

1. In Compute Console, go to **Manage > Defenders > Deploy > Single Defender**.
2. For **Defender type**, select **Serverless**.
3. In **Set the Twistlock environment variable**, enter the function name and region.
4. Copy the generated **Value**.
5. In AWS Console, open your function in the designer, and scroll down to the **Environment variables** panel.
6. For **Key**, enter TW_POLICY.
7. For **Value**, paste the rule you copied from Compute Console.
8. For *ORIGINAL_HANDLER*, this is the original value of handelr for your function before your modification.

STEP 6 | Click **Save** to preserve all your changes.



Auto-defend serverless functions

Edit on GitHub

Serverless auto-defend lets you automatically add the Serverless Defender to the AWS Lambda functions deployed in your account. Prisma Cloud uses the AWS API to deploy the Serverless Defender as a Lambda layer based on the auto-defend rules.

It is an additional option for deploying the Serverless Defender, on top of manually adding it as a dependency or adding it as a Lambda layer.

Serverless auto-defend supports the following runtimes:

- Node.js 12.x, 14.x
- Python 3.6, 3.7, 3.8, 3.9
- Ruby 2.7

Limitations

Auto-protect is implemented with a layer. AWS Lambda has a limit of five layers per function. If your functions have mutiple layers, and they might exceed the layer limit with auto-defend, consider protecting them with the embedded option.

Required permissions

Prisma Cloud needs the following permissions to automatically protect Lambda functions in your AWS account. Add the following policy to an IAM user or role:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "PrismaCloudComputeServerlessAutoProtect",
            "Effect": "Allow",
            "Action": [
                "lambda:PublishLayerVersion",
                "lambda:UpdateFunctionConfiguration",
                "lambda:GetLayerVersion",
                "lambda:GetFunctionConfiguration",
                "iam:SimulatePrincipalPolicy",
                "lambda:GetFunction",
                "lambda:ListFunctions",
                "iam:GetPolicyVersion",
                "iam:GetRole",
                "iam:ListRolePolicies",
                "iam:ListAttachedRolePolicies",
                "iam:GetRolePolicy",
                "iam:GetPolicy",
                "lambda:ListLayerVersions",
                "lambda:ListLayers",
                "lambda:DeleteLayerVersion",
                "kms:Decrypt",
                "kms:Encrypt",
                "kms:CreateGrant"
            ],
```



```
                "Resource": "*"
            }
        ]
}
```

Serverless auto-defend rules

To secure one or multiple AWS Lambda functions using serverless auto-defend:

1. Define a serverless protection runtime policy.
2. Define a serverless WAAS policy.
3. Add a serverless auto-defend rule.

Defining your runtime protection policy

By default, Prisma Cloud ships with an empty serverless runtime policy. An empty policy disables runtime defense entirely.

You can enable runtime defense by creating a rule. By default, new rules:

- Apply to all functions (*), but you can target them to specific functions by function name.
- Block all processes from running except the main process. This protects against command injection attacks.

When functions are invoked, they connect to Compute Console and retrieve the latest policy. To ensure that functions start executing at time=0 with your custom policy, you must predefine the policy. Predefined policy is embedded into your function along with the Serverless Defender by way of the *TW_POLICY* environment variable.

STEP 1 | Log into Prisma Cloud Console.

STEP 2 | Go to **Defend** > **Runtime** > **Serverless Policy**.

STEP 3 | Click **Add rule**.

STEP 4 | In the **General** tab, enter a rule name.

STEP 5 | (Optional) In **Scope**, target the rule to specific functions.

Create a new collection.

STEP 6 | Set the rule parameters in the **Processes**, **Networking**, and **File System** tabs.

STEP 7 | Click **Save**.

Defining your serverless WAAS policy

Prisma Cloud lets you protect your serverless functions against application layer attacks by utlizing the serverless Web Application and API Security (WAAS).

By default, the serverless WAAS is disabled. To enable it, add a new serverless WAAS rule.

STEP 1 | Log into Prisma Cloud Console.

STEP 2 | Go to **Defend** > **WAAS** > **Serverless**.

STEP 3 | Click **Add rule**.

STEP 4 | In the **General** tab, enter a rule name.

STEP 5 | (Optional) In **Scope**, target the rule to specific functions.

Create a new collection. In the **Functions** field, enter a function name. Use pattern matching to refine how it's applied.

STEP 6 | Set the protections you want to apply (**SQLi**, **CMDi**, **Code injection**, **XSS**, **LFI**).

STEP 7 | Click **Save**.

Add a serverless auto-defend rule

The serverless auto-defend rules let you specify which functions you want to protect. When defining a specific rule you can reference the relevant credential, regions, tags, function names and runtimes. Each auto-defend rule is evaluated separately.

STEP 1 | Open Compute Console, and go to **Manage > Defenders > Deploy > Serverless auto-defend**.

STEP 2 | Click on **Add rule**.

STEP 3 | In the dialog, enter the following settings:

1. Enter a rule name.
2. In **Provider** - only AWS is supported.
3. Specify the scope.

   The available resources for scope are:

   - **Functions** - either specific names or prefix.
   - **Labels** - allows specifying either regions (format - region:REGION_NAME) or AWS tags (format - KEY:VALUE).
4. Specify the Console name.
5. Specify the runtimes.
6. Select or create credentials so that Prisma Cloud can access your account.
7. (Optional) Specify a proxy for the Defenders to use when communicating with the Console.
8. Click **Add**.

STEP 4 | The new rule appears in the table of rules.

STEP 5 | Click **Apply Defense**.

By default, the serverless auto-defend rules are evaluated every 24 hours.

When a rule is deleted, the new set of rules is evaluated and applied ***immediately****.*

Install a single Host Defender

Edit on GitHub

Install Host Defender on each host that you want Prisma Cloud to protect.

Single Host Defenders can be configured in the Console UI, and then deployed with a curl-bash script. Alternatively, you can use twistcli to configure and deploy Defender directly on a host.

Install a Host Defender (Console UI)

Host Defenders are installed with a curl-bash script. Install Host Defender on each host that you want Prisma Cloud to protect.

Anywhere <CONSOLE> is used, be sure to specify both the address and port number for Console's API. By default, the port is 8083. For example, https://<CONSOLE>:8083.

Prerequisites:

- Your system meets all minimum system requirements.
- You have sudo access to the host where Defender will be installed.
- You have already installed Console
- Port 8084 is open on the host where Defender runs. Console and Defender communicate with each other over a web socket on port 8084 (by default the communication port is set to 8084 - however, you can specify your own custom port when deploying a Defender).
- Console can be accessed over the network from the host where you will install Defender.

STEP 1 | Verify that the host machine where you install Defender can connect to Console.

```
$ curl -sk -D - https://<CONSOLE>/api/v1/_ping
```

If curl returns an HTTP response status code of 200, you have connectivity to Console. If you customized the setup when you installed Console, you might need to specify a different port.

STEP 2 | Log into Console.

STEP 3 | Go to **Manage > Defenders > Deploy**.

1. In the first drop-down menu (2), select the way Defender connects to Console.

   A list of IP addresses and hostnames are pre-populated in the drop-down list. If none of the items are valid, go to **Manage > Defenders > Names**, and add a new Subject

Alternative Name (SAN) to Console's certificate. After adding a SAN, your IP address or hostname will be available in the drop-down list.

Selecting an IP address in a evaluation setup is acceptable, but using a DNS name is more resilient. If you select Console's IP address, and Console's IP address changes, your Defenders will no longer be able to communicate with Console.

2. (Optional) Set a proxy (3) for the Defender to use for the communication with the Console.
3. (Optional) Set a custom communication port (4) for the Defender to use.
4. (Optional) Set **Assign globally unique names to Hosts** to **ON** when you have multiple hosts that can have the same hostname (e.g., autoscale groups, overlapping IP addresses, etc).

   *After setting the toggle to **ON**, Prisma Cloud appends a unique identifier, such as ResourceId, to the host's DNS name. For example, an AWS EC2 host would have the following name: Ip-171-29-1-244.ec2internal-i-04a1dcee6bd148e2d.*
5. In the second drop-down list (5), select **Host Defender - Linux** or **Host Defender - Windows**.
6. In the final field, copy the install command, which is generated according to the options you selected.

STEP 4 | On the host where you want to install Defender, paste the command into a shell window, and run it.

Install a single Host Defender (twistcli)

Use twistcli to install a single Host Defender on a Linux host.

Anywhere <CONSOLE> is used, be sure to specify both the address and port number for Console's API. By default, the port is 8083. For example, https://<CONSOLE>:8083.

Prerequisites:

- Your system meets all minimum system requirements.
- You have already installed Console.
- Port 8083 is open on the host where Console runs. Port 8083 serves the API. Port 8083 is the default setting, but it is customizable when first installing Console. When deploying Defender, you can configure it to communicate to Console via a proxy.
- Port 8084 is open on the host where Console runs. Console and Defender communicate with each other over a web socket on port 8084. Defender initiates the connection. Port 8084 is the default setting, but it is customizable when first installing Console. When deploying Defender, you can configure it to communicate to Console via a proxy.
- You've created a service account with the Defender Manager role. twistcli uses the service account to access Console.
- Console can be accessed over the network from the host where you want to install Defender.
- You have sudo access to the host where Defender will be installed.

STEP 1 | Verify that the host machine where you install Defender can connect to Console.

```
$ curl -sk -D - https://<CONSOLE>/api/v1/_ping
```

If curl returns an HTTP response status code of 200, you have connectivity to Console. If you customized the setup when you installed Console, you might need to specify a different port.

STEP 2 | SSH to the host where you want to install Defender.

STEP 3 | Download twistcli.

```
$ curl -k \
  -u <USER> \
  -L \
  -o twistcli \
  https://<CONSOLE>/api/v1/util/twistcli
```

STEP 4 | Make the twistcli binary executable.

```
$ chmod a+x ./twistcli
```

STEP 5 | Install Defender.

```
$ sudo ./twistcli defender install standalone host-linux \
  --address https://<CONSOLE> \
  --user <USER>
```

Verify the install

Verify that Defender is installed and connected to Console.

In Console, go to **Manage > Defenders > Manage**. Your new Defender should be listed in the table, and the status box should be green and checked.

Auto-defend hosts

Edit on GitHub

Host auto-defend lets you automatically deploy Host Defenders on virtual machines/instances in your AWS, Azure and Google Cloud accounts. This covers AWS EC2 instances, Azure Virtual Machines, and GCP Compute Engine instances.

- Deployment process
- AWS EC2 instances
- Azure virtual machines
- GCP Compute Engine instances
- Instance types
- Add a host auto-defend rule

Deployment process

After setting up auto-defend for hosts, Prisma Cloud discovers and protects unsecured hosts as follows:

1. Discover - Prisma Cloud uses cloud provider APIs to get a list of all VM instances.
2. Identify - Prisma Cloud identifies unprotected instances.
3. Verify - Ensure unprotected resources meet auto-defend prerequisites.
4. Install - Prisma Cloud installs Host Defender on unprotected instances using cloud provider APIs.

Regardless of the underlying container runtime, the host deployment process skips your worker nodes.

AWS EC2 instances

Prisma Cloud uses AWS Systems Manager (formerly known as SSM) to deploy Defenders to EC2 instances.

Minimum requirements

The following sections describe the minimum requires to auto-defend to hosts in AWS.

AWS Systems Manager

Prisma Cloud uses AWS Systems Manager (formerly known as SSM) to deploy Defenders to instances. This means that:

- The SSM Agent must be installed on every instance.
- AWS Systems Manager must have permission to perform actions on each instance.

To view all SSM managed instances, go to the AWS console here.

SSM Agent

Prisma Cloud uses the SSM Agent to deploy Host Defender on an instance. The SSM Agent must be installed prior to deploying the Host Defenders. The SSM Agent is installed by default on the following distros.

- Amazon Linux
- Amazon Linux 2
- Amazon Linux 2 ECS-Optimized AMIs
- Ubuntu Server 16.04, 18.04, and 20.04

The SSM Agent doesn’t come installed out of the box but supported on the following distributions. Ensure its installed ahead of time before proceeeding. :

- CentOS
- Debian Server
- Oracle Linux
- Red Hat Enterprise Linux
- SUSE Linux Enterprise Server

IAM instance profile for Systems Manager

By default, AWS Systems Manager doesn't have permission to perform actions on your instances. You must grant it access with an IAM instance profile.

If you've used System Manager's Quick Setup feature, assign the **AmazonSSMRoleForInstancesQuickSetup** role to your instances.

Required permissions

Prisma Cloud needs a service account with the following permissions to automatically protect EC2 instances in your AWS account. Add the following policy to an IAM user or role:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeImages",
                "ec2:DescribeInstances",
                "ssm:SendCommand",
                "ssm:DescribeInstanceInformation",
                "ssm:ListCommandInvocations",
                "ssm:CancelCommand",
                "ec2:DescribeRegions",  //You can ignore if you
 already have these permissions as apart of the discovery feature
                "ec2:DescribeTags",//You can ignore if you already
 have these permissions as apart of the discovery feature
                "ssm:SendCommand"
            ],
            "Resource": "*"
        }
    ]
}
```

Azure virtual machines

Prisma Cloud uses the Azure VM agent Run Command option to invoke the script to deploy Host defenders. You are required to configure the permissions below in your subscription and create host deploy rules to begin installing Defenders.

Minimum requirements

The following sections describe the minimum requires to auto-defend to hosts on Azure.

Azure Linux VM agent & Run command

Prisma Cloud uses the *run command* action on the Azure Linux VM agent to deploy Defenders on instances.

The VM Agent must be on every instance. By default, the VM agent is available on most Linux OS machines. Refer to the documentation for more information.

Currently cancelling running operation is not supported. Dangling command will automatically timeout after 90 minutes. Also, run command is only supported on Linux VMs.

Required permissions

In addition to the Reader role to get the list and details of the virtual machines, the Azure credential user needs permissions to invoke the runcommand.

```
Microsoft.Compute/virtualMachines/runCommand/action
```

Typically, the Virtual Machine Contributor role and higher levels have this permission. You can either directly use the role or create a custom role with the above permission.

GCP Compute Engine instances

The installation uses OS Patch Management service. Prisma Cloud creates an OS patch job with the information of the installation script stored in the temporarily created storage bucket and the list of instances to deploy the Host defender on the instances.

Minimum requirements

The following sections describe the minimum requires to auto-defend hosts on GCP.

Storage Buckets

Prisma cloud auto creates a temporary storage bucket in the region you selected for the auto-defend rule. The bucket is named 'prisma-defender-bucket-<hash>' where <hash> is a randomly generated string, e.g., 'prisma-defender-bucket-346a7e425d344c8a7dd9ce75da674970'. The Prisma defender installation script 'prisma-defender-script.sh' is stored in the bucket.

The service account user needs permissions to be able to create and delete the bucket.

OS Patch Management

VM Manager is a suite of tools that can be used to manage operating systems for large virtual machine (VM) fleets running Windows and Linux on Compute Engine. Prisma cloud uses OS Patch Management service which is a part of a broader VM Manager service to deploy the host defenders.

- Setup VM Manager for OS patch management. Users can do auto enablement of VM Manager from the Google cloud console as shown here
- VM is supported on most of the active OS versions for Linux. For more information, refer to Operating system for details.
- In Google Cloud project, OS Config API should be enabled. This needs to be done via the google cloud console.

Required permissions

Prisma Cloud needs a service account with the following permissions to automatically protect GCP compute instances in your Google project. Add the following permissions:

```
Compute.instances.list
Compute.zones.list
Compute.projects.get
osconfig.patchJobs.exec
osconfig.patchJobs.get
osconfig.patchJobs.list
storage.buckets.create
storage.buckets.delete
```



```
storage.objects.create
storage.objects.delete
storage.objects.get
storage.objects.list
compute.disks.get
```

Instance types

Host auto-defend is supported on Linux hosts only. Hosts must have either *wget* or *curl* installed. Hosts must be able to communicate to Console on port 8083.

Auto-defend is supported for stand-alone hosts only, not hosts that are part of clusters. For hosts that are part of clusters, use one of the cluster-native install options (e.g., DaemonSets on Kubernetes).

When configuring the scope of hosts that should be auto-defended, ensure that the scope doesn't include any hosts that are part of a cluster or that run containers. Auto-defend doesn't currently check if a host is part of cluster. If you mistakenly include nodes that are part of a cluster in an auto-defend rule, and the cluster is not already protected, the auto-defend rule will deploy Host Defenders to the cluster nodes.

Add a host auto-defend rule

Host auto-defend rules let you specify which hosts you want to protect. You can define a specific account by referencing the relevant credential or collection. Each auto-defend rule is evaluated separately.

STEP 1 | Open Compute Console, and go to **Manage > Defenders > Deploy > Host auto-defend**.

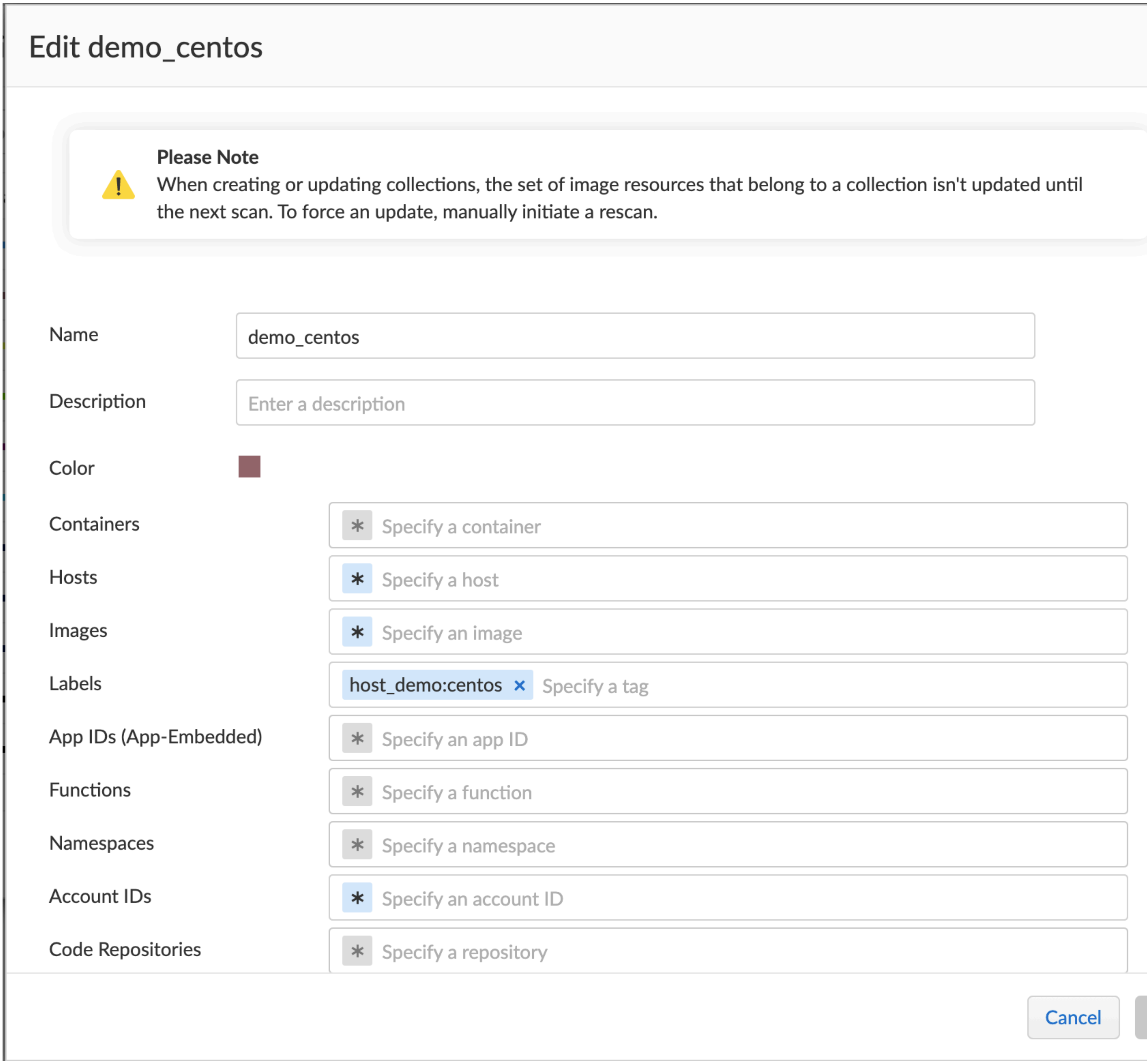
STEP 2 | Click on **Add rule**.

STEP 3 | In the dialog, enter the following settings:

1. Enter a rule name.
2. In **Provider** - AWS, Azure and GCP are currently supported.
3. In **Console**, specify a DNS name or IP address that the installed Defender can use to connect back to Console after it's installed.
4. (Optional) In **Scope**, target the rule to specific hosts.

   Create a new collection. Supported attributes are hosts, images, labels, account IDs.

   The following example shows a collection that is based on hosts labels, in this case a label of host_demo with the value centos.



5. Set up these options for specific Cloud Service Providers.
   - (For AWS only) Specify the Scanning scope for the AWS region- Commercial or regular, Government, or China.
   - (For GCP only) Specify the Bucket region. Prisma cloud auto creates a temporary storage bucket named 'prisma-bucket' in the region and deletes it after the process of creating the rule is completed.

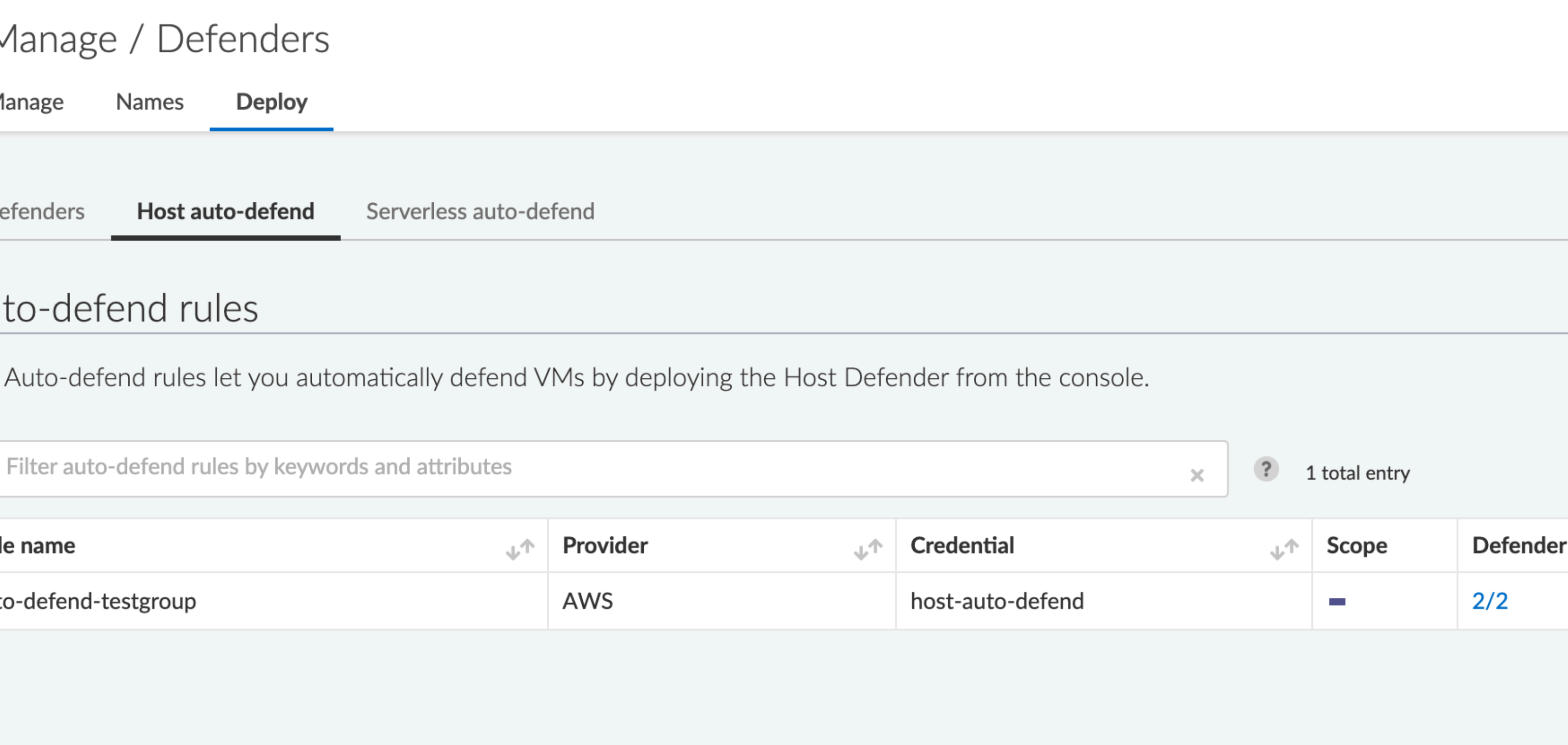
6. Select or create credentials so Prisma Cloud can access your account. The service account must have the minimum permissions specified here.
7. Click **Add**.

   The new rule appears in the table of rules.

STEP 4 | Click **Apply Defense**.

Select the rule to start the scan. By default, host auto-protect rules are evaluated every 24 hours. Click the **Apply Defense** button to force a new scan.

The following screenshot shows that the *auto-defend-testgroup* discovered two EC2 instances and deployed two Defenders (2/2).

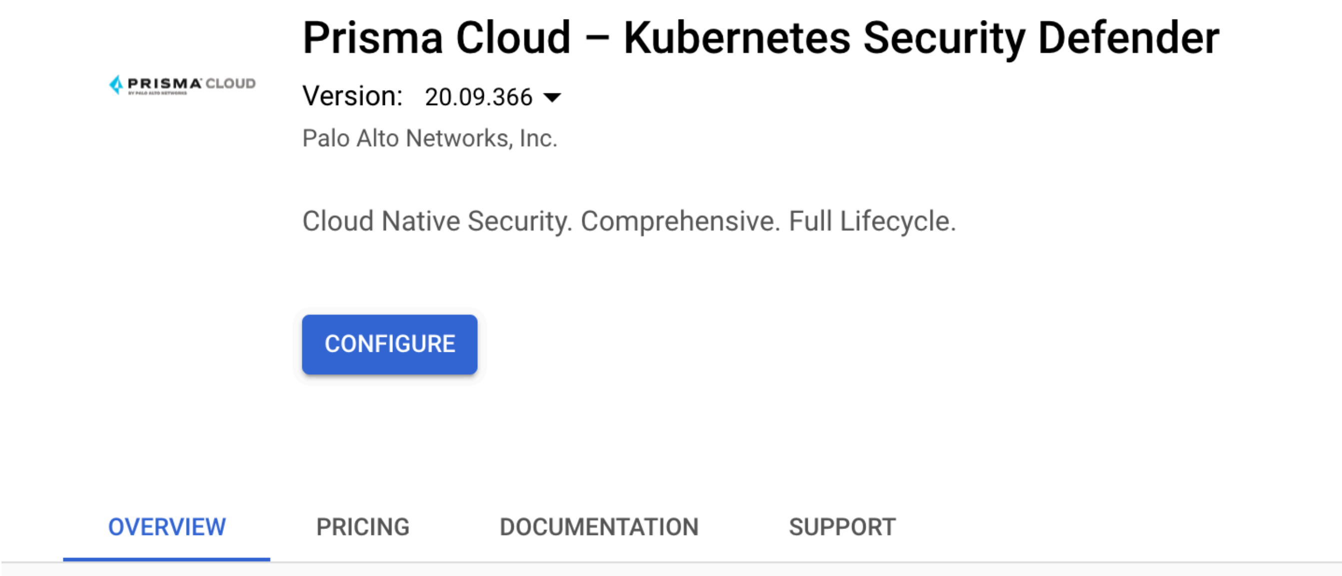


Deploy Prisma Cloud Defender from the GCP Marketplace

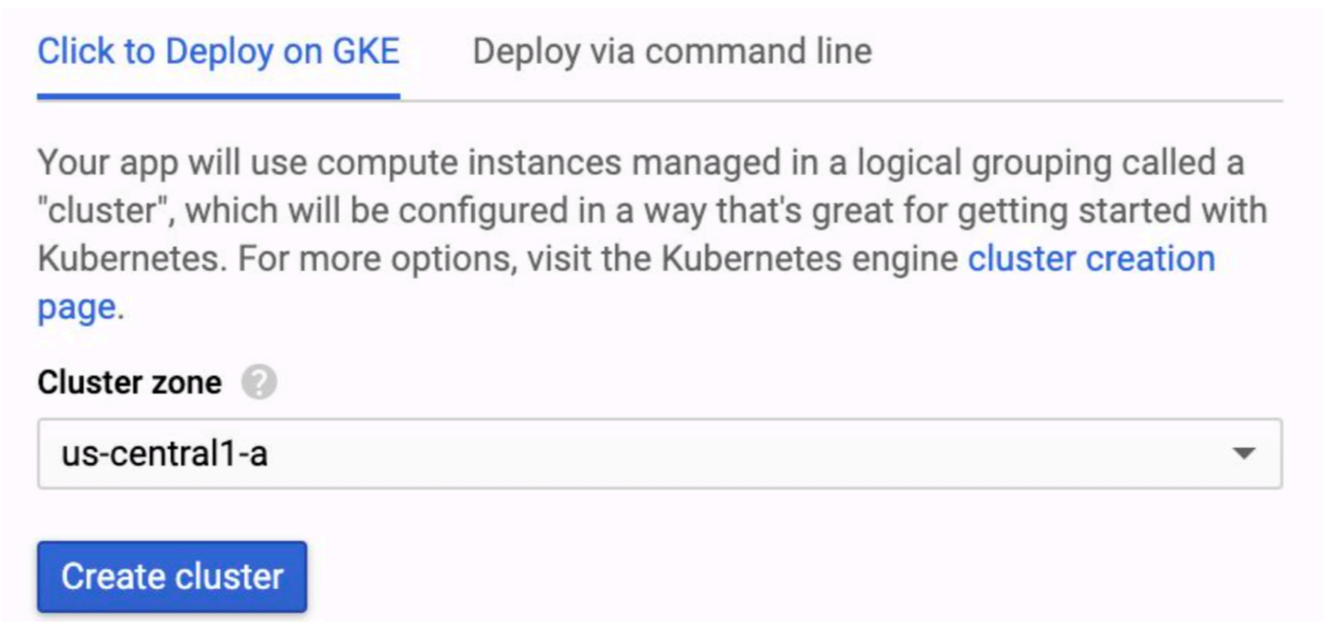
Edit on GitHub

Prerequisites: You need access to a Prisma Cloud SaaS Console. You can sign up for a free trial of Prisma Cloud on the Google Cloud Marketplace.

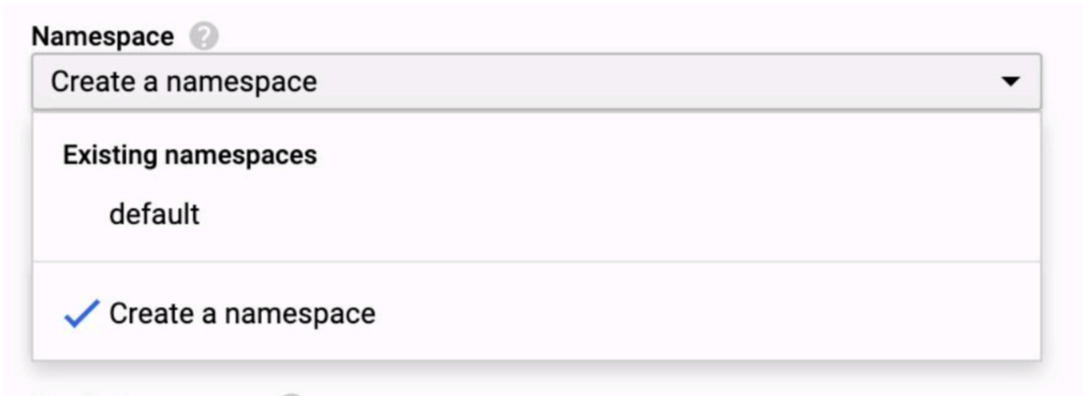
STEP 1 | Find Prisma Cloud - Kubernetes Security Defender in the GCP Marketplace. Click Configure.

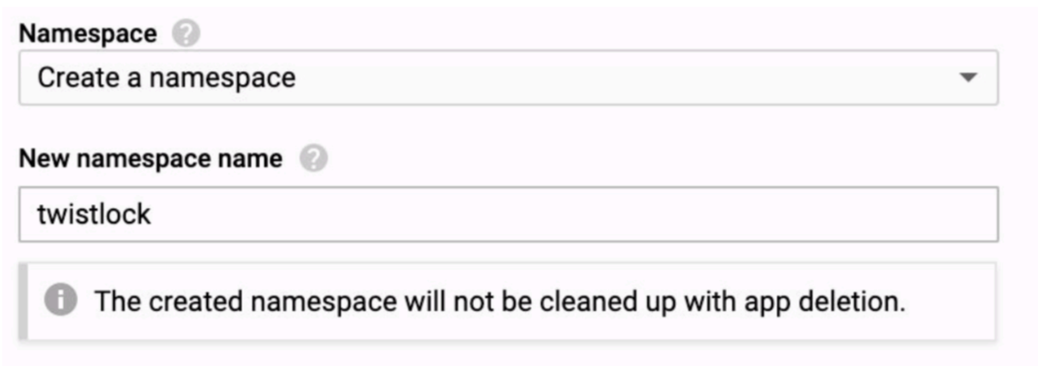


STEP 2 | Create Cluster, if you don't have an existing Kubernetes cluster. Otherwise, continue to the next step.



STEP 3 | Select an existing namespace to install Defender, or Create a namespace (recommended). The default new namespace is "twistlock".





STEP 4 | Enter the App instance name for the Defender the installation. This name displays on the Application section of the GKE portal:

STEP 5 | Specify the following information about your Prisma Cloud SaaS Console (go through steps 6-8 to get these info):

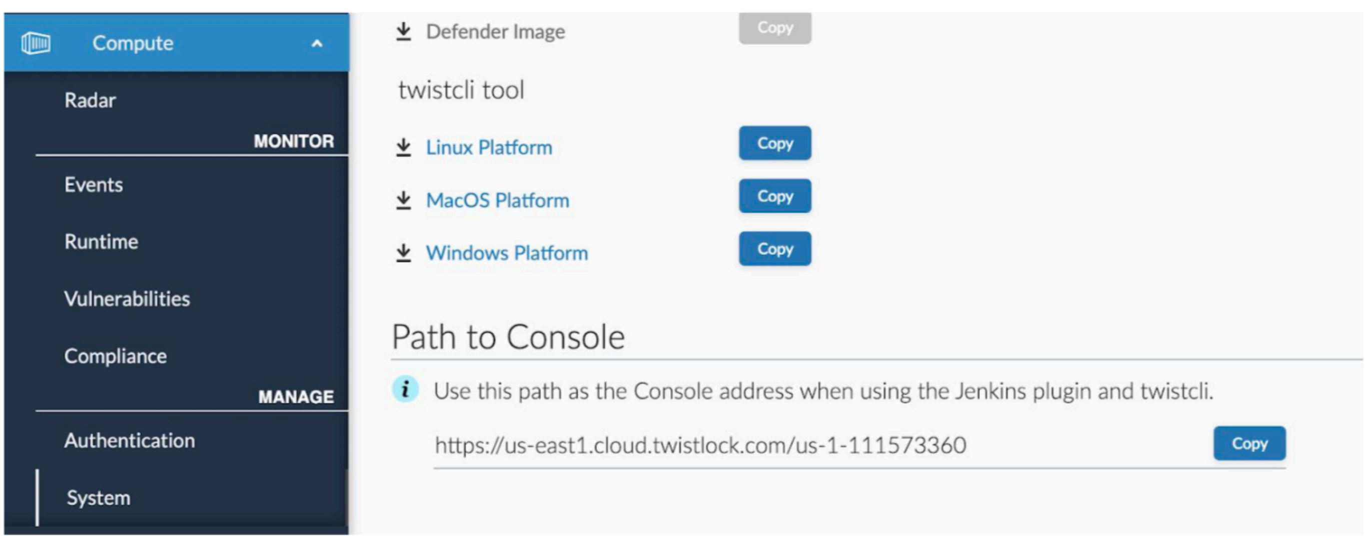
The URL to access Prisma Cloud Console. For example:<https://domain_name>

Provide the API token in Prisma Cloud Console.

Specify the IP Address or Domain Name of the Prisma Cloud Console. For example:us-west1.cloud.twistlock.com

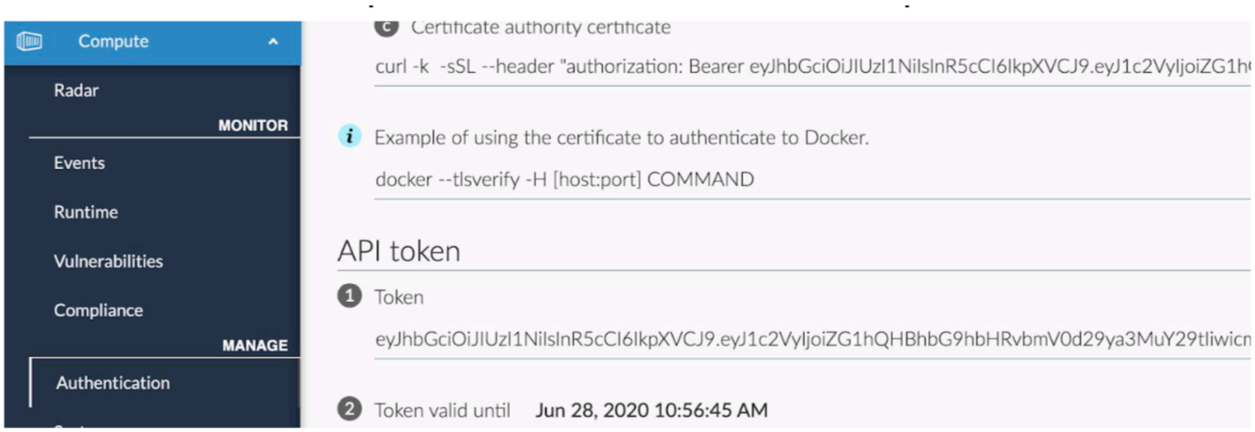
STEP 6 | To get the URL for your Prisma Cloud Console:

1. Log into your Prisma Cloud portal (e.g., https://app.prismacloud.io/).
2. Navigate to **Compute > System**.
3. Copy the URL in Path to Console. GCP uses this URL to get all the setup artifacts from your Prisma Cloud Console. In this example, it's https://us-east1.cloud.twistlock.com/us-1-111573360.



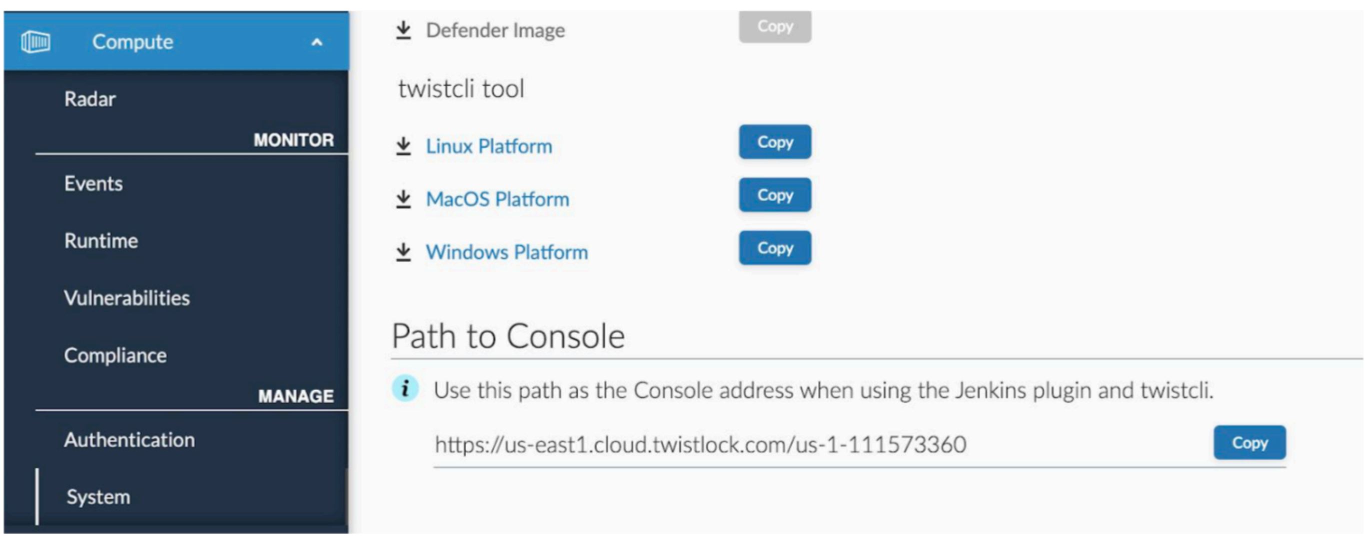
STEP 7 | To get a token for your Prisma Cloud Compute Console.

1. Go to Compute > Authentication.
2. Copy the API token. and paste it into the GCP Marketplace form.

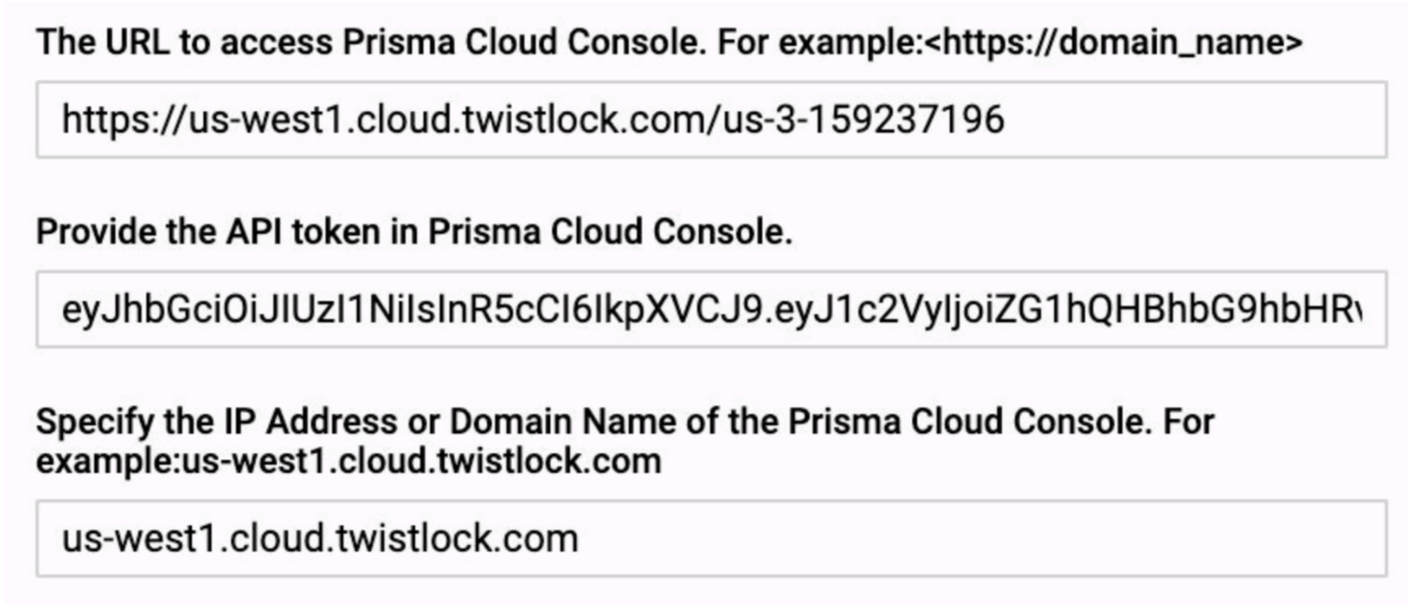


STEP 8 | Specify the IP address or domain name of your Prisma Cloud Compute Console.

The Defenders that you are deploying will use this IP address to communicate with Prisma Cloud. It’s almost the same as the URL, but remove the protocol (https://) and the path (everything trailing the first "/"). In this example, us-east1.cloud.twistlock.com.



STEP 9 | When the form is filled out, click Deploy.



STEP 10 | Go to Prisma Cloud SaaS Console to confirm the deployment is successful.

1. In the GKE console, review the status of your deployment:

2. In Prisma Cloud Console, go to Compute > Defender to review the status of your deployment:

Decommission Defenders

Edit on GitHub

Regularly decommissioning stale Defenders keeps your view of the environment clean and conserves licenses. Defenders can be decommissioned from the Console UI or the Prisma Cloud API.

Prisma Cloud automatically decommissions stale Defenders for you. In large scale environments, manually decommissioning Defenders could be onerous. If left undone, however, it can lead to lots of Defenders being left in a permanently offline state, cluttering your view of environment. To keep your view clean, Console automatically decommissions Defenders that haven't been connected to Console for more than one day. This keeps the list of connected Defenders valid to a 24-hour window. The refresh period can be configured up to a maximum of 365 days under

Manage > Defenders > Manage > Advanced Settings > Automatically remove disconnected Defenders after (days).

We recommend letting Prisma Cloud automatically decommission stale Defenders rather than using the UI or API.

Decommission Defenders manually

Decommissioning Defenders can be done manually from Console.

Go to **Manage > Defenders > Manage**, where you will find a list of all Defenders connected to Console. Click **Actions > Decommission** for each respective Defender.

Decommission Defenders with the API

The following endpoint can be used to decommission a Defender.

Path

```
DELETE /api/v1/defenders/[hostname]
```

Description

Deletes a Defender from the database. This endpoint does not actually uninstall Defender. Use the fully qualified domain name (FQDN) of the host. You can find the FQDN of the host in **Manage > Defenders > Actions > Manage**.

Example request

```
$ curl -X DELETE \
  -u <USERNAME>:<PASSWORD>
  'https://<CONSOLE>:8083/api/v1/defenders/aqsa-cto.sandbox'
```

Force uninstall Defender

The preferred method for uninstalling Defenders is via the Console UI. However, if a Defender instance is not connected to Console, or is otherwise not manageable through the Console UI, it can be manually removed.

On the Linux host where Container Defender runs, use the following command:

```
$ sudo /var/lib/twistlock/scripts/twistlock.sh -u
```

If you run this command on the same Linux host where the Prisma Cloud Console is installed, it also uninstalls Prisma Cloud Console.

On the Linux host where Host Defender runs, use the following command:

```
$ sudo /var/lib/twistlock/scripts/twistlock.sh -u defender-server
```

On the Windows host where Defender runs, use the following command:

```
C:\Program Files\Prisma Cloud\scripts\defender.ps1 -uninstall
```


Redeploy Defenders

Edit on GitHub

When you redeploy the Prisma Cloud Console, the client and server certificates change. That certificate change requires that you redeploy your defenders. Once redeployed, the defenders can connect to the new console without certificate issues.

STEP 1 | To redeploy the defenders, generate a new `DaemonSet`configuration file:

```
$ <PLATFORM>/twistcli defender export kubernetes \
    --address https://yourconsole.example.com:8083 \
    --user <ADMIN_USER> \
    --cluster-address twistlock-console
```

STEP 2 | Apply the in-place updates to your *Defender* resources.

```
$ kubectl apply -f defender.yaml
```

Uninstall Defenders

Edit on GitHub

To uninstall Prisma Cloud, delete the *twistlock* namespace. Deleting this namespace deletes every resource within the namespace.

When you delete the *twistlock* namespace, you also delete the persistent volume (PV) in the namespace. By default, the Prisma Cloud Console stores its data in that PV. When the PV is deleted, all data is lost, and you can't restore the Prisma Cloud Console.

Delete the *twistlock* namespace.

```
$ kubectl delete namespaces twistlock
```


Upgrade

Edit on GitHub

Console notifies you when new versions of Prisma Cloud are available. You can upgrade Prisma Cloud without losing any of your data or configurations. After upgrading Console, upgrade all of your deployed Defenders.

- Support lifecycle for connected components
- Prisma Cloud’s backward compatibility and upgrade process
- Upgrade Onebox
- Kubernetes
- OpenShift
- Helm charts
- Amazon ECS
- Upgrade the Single Container Defenders
- Upgrade Defender DaemonSets
- Upgrade Defender DaemonSets (Helm)

Support lifecycle for connected components

Edit on GitHub

To simplify upgrades, older versions of Defenders, Jenkins plugins, and twistcli can interoperate with newer versions of Console. With this capability, you have a larger window to plan upgrades for connected components.

Window of support

Any supported version of Defender, twistcli, and the Jenkins plugin can connect to Console. Prisma Cloud supports the latest release and the previous two releases (n, n-1, and n-2).

There are some exceptions to this policy as we roll out this new capability.

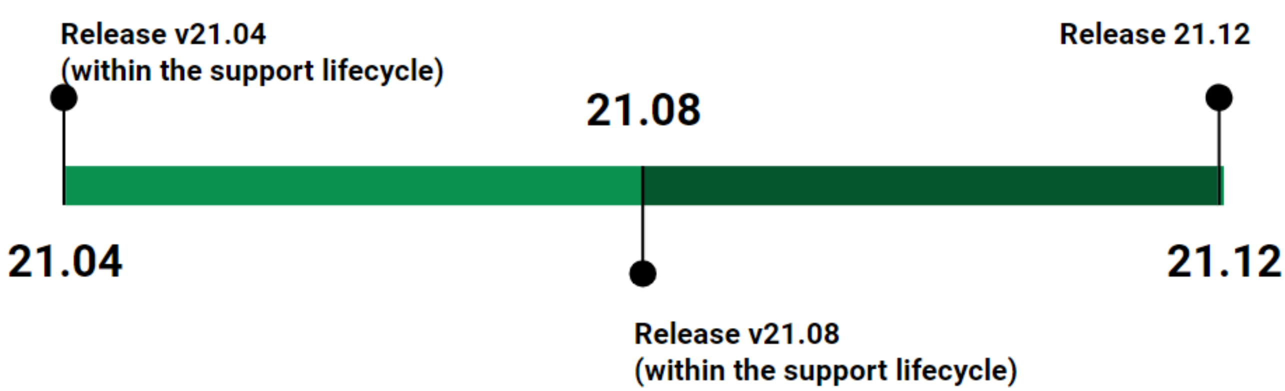
For Defenders:

- 21.08 supports n and n-1 (21.04) only.
- Starting with the next release (Joule), there will be full support for n, n-1, and n-2.

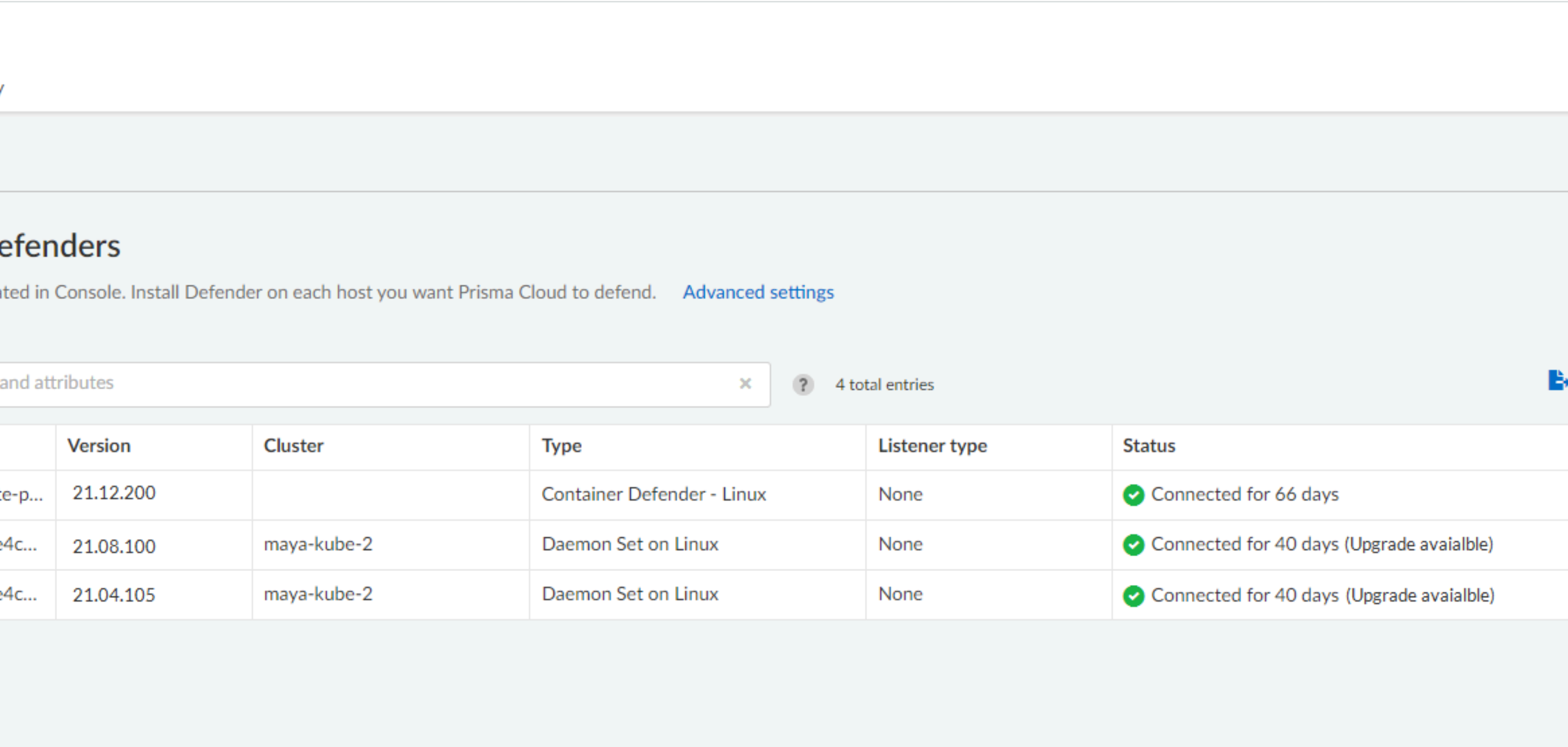
For twistcli and the Jenkins plugin:

- 21.08 supports itself (n) only.
- In the next release (Joule), Console will support n and n-1.
- In release after Joule (Kepler), Console will support n, n-1, n-2.

For example, if Console runs version 21.12, it will support Defenders, twistcli, and the Jenkins plugin running either version 21.08 or 21.04:



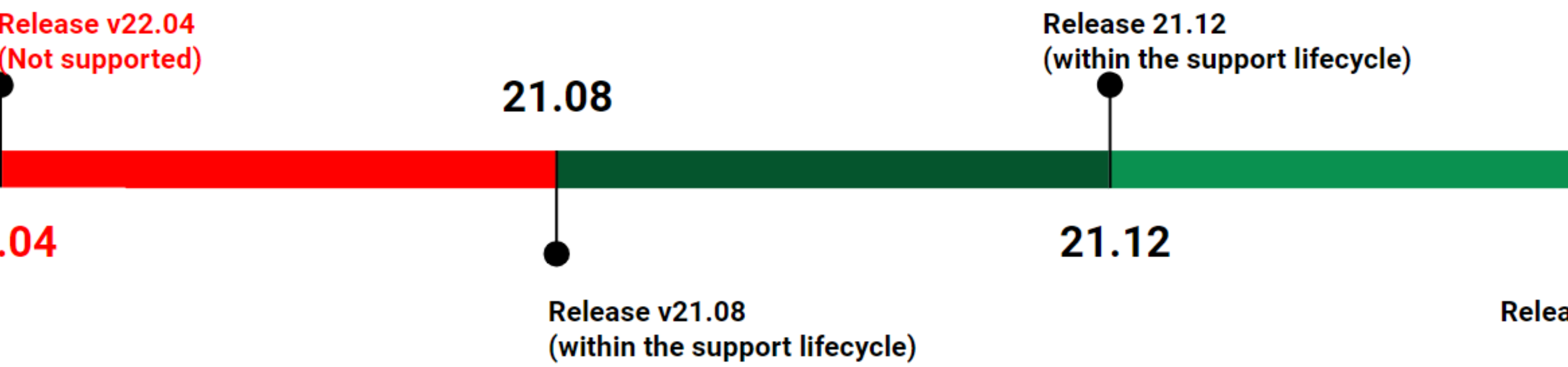
Defender's connection status on the Defender management page indicates how it interoperates with Console. Defenders that match Console's version show the status of Connected. Defenders still supported, but running a previous version, show the connected status with a message that upgrade is available (but not mandatory).



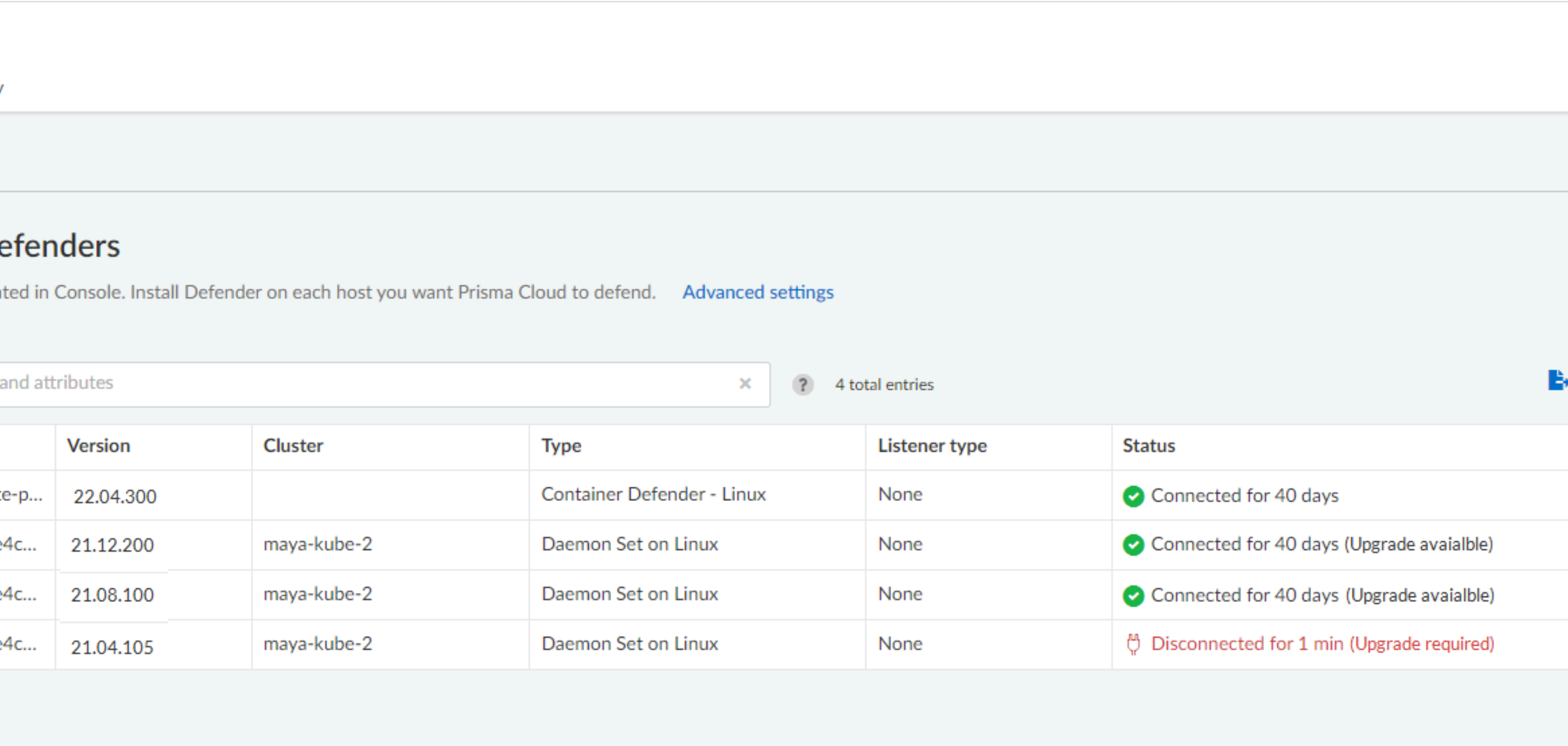
Twistcli and the Jenkins plugin function as normal, with an indictor that an upgrade is available shown in the scan reports in the Console web UI.

End of support

Once a version is no longer supported, any Defenders based on that version must be upgraded (mandatory). For example, if Console runs 22.04, it will support Defenders running either 21.12 or 21.08, but will no longer support Defenders running on 21.04.



Defenders which are no longer within the support lifecycle will not be able to connect to the Console. That state will be reflected on the Defender management page, with a status of **Disconnected** and an associated message that upgrade is required:



Versions of twistcli and Jenkins plugin outside of the support lifecycle fail open. Their requests to Console will be refused, but builds will pass. Console returns a status of 400 Bad Request, which indicates an error due to the fact that the plugin version is no longer supported.

Prisma Cloud's backward compatibility and upgrade process

Edit on GitHub

Prisma Cloud console is backward compatible up to two major releases back (including all minor versions) with the following:

- All types of Defenders.
- Twistcli/Jenkins plugin.

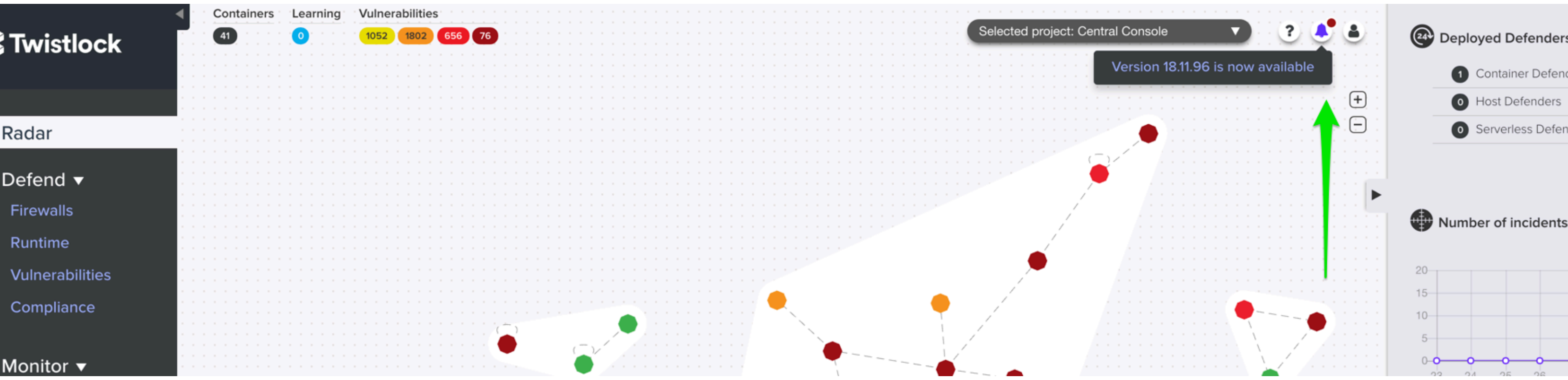
When projects are used, the exact same version is required for master Console and tenant Consoles.

Upgrade and notifications

You can upgrade Prisma Cloud without losing any of your data or configurations. First, upgrade Console. Then, upgrade any of the Defenders that have reached end of their support lifecycle.

You can upgrade from up to two release back **directly** to the current major version.

Console notifies you when new versions of Prisma Cloud are available. Notifications are displayed in the top right corner of the dashboard.



When you upgrade Console, the old Console container is completely replaced with a new container. Because Prisma Cloud stores state information outside of the container, all your rules and settings are immediately available to the upgraded Prisma Cloud containers.

Prisma Cloud state information is stored in a database in the location specified by DATA_FOLDER, which is defined in *twistlock.cfg*. By default, the database is located in */var/lib/twistlock*.

Overview of the upgrade process

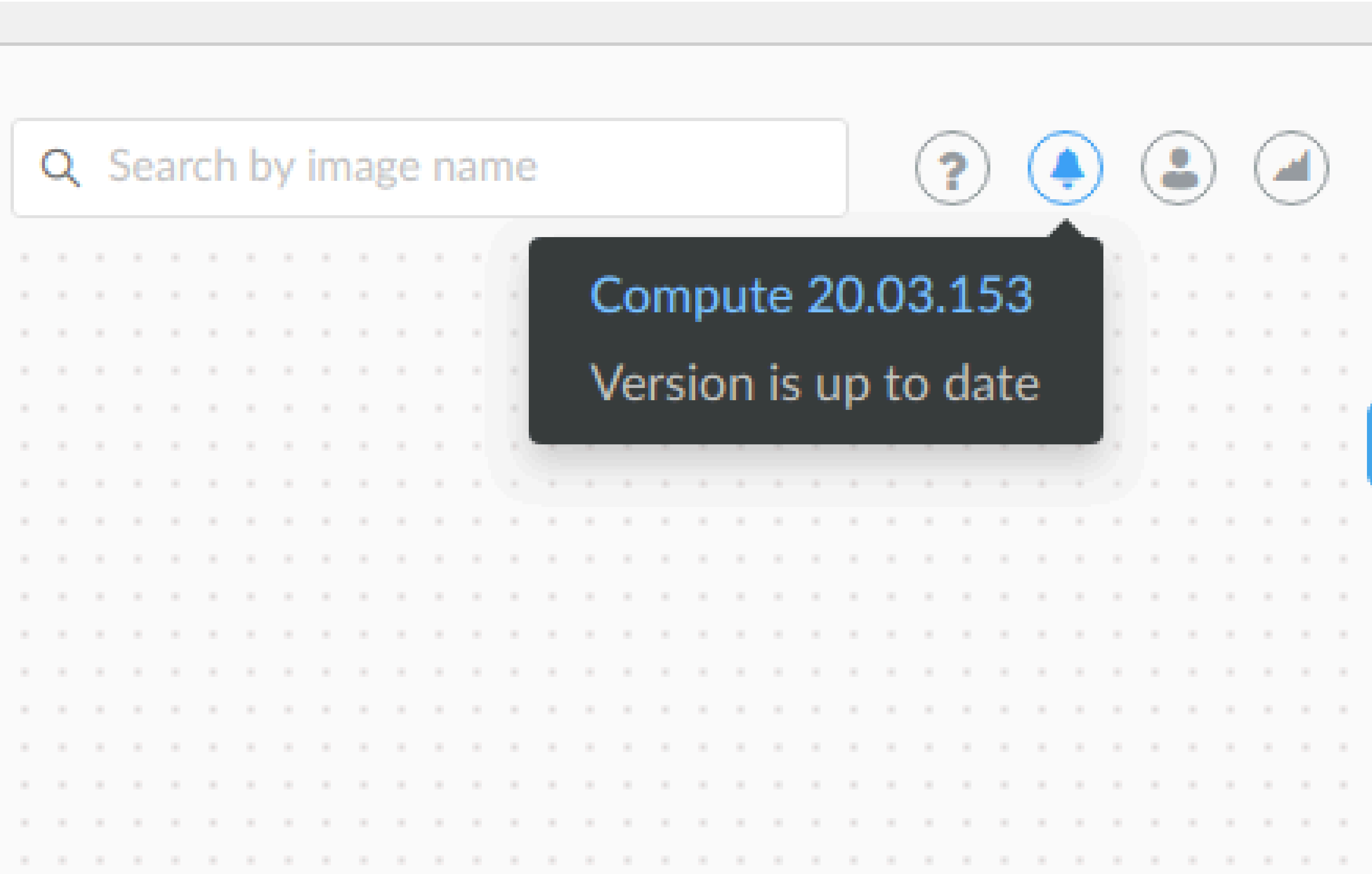
First, upgrade Console. Then, upgrade any of the Defenders that have reached the end of the support lifecycle. Because the release images for Console and Defender are built from the UBI8-minimal base image, the upgrade is a full container image upgrade and the old container is replaced with a new container. Finally, upgrade all other Prisma Cloud components, such as the Jenkins plugin.

The steps in the upgrade process are:

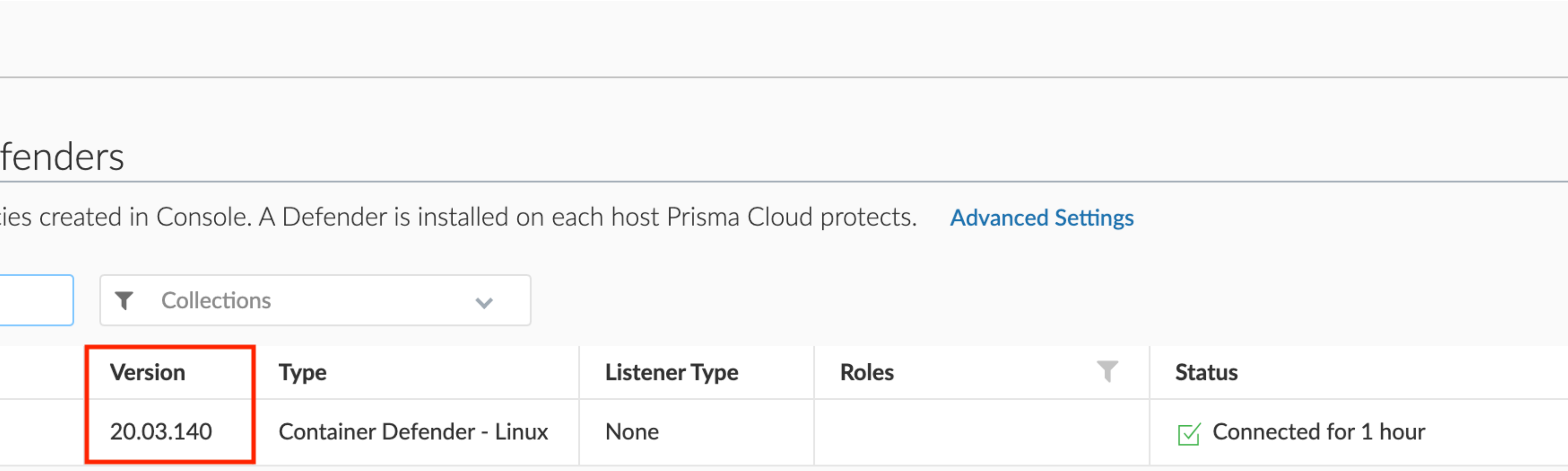
1. Upgrade Console.
2. Go to **Manage > Defenders > Manage**, filter the the **Status** column by **Upgrade Required**, and upgrade all the listed Defenders.
3. Validate that all deployed Defenders have been upgraded.
4. To download the latest version of all other Prisma Cloud Compute components (such as the Jenkins plugin), either go to **Manage > System > Utilities** to download the latest versions or retrieve them using the API.

Version numbers of installed components

The currently installed version of Console is displayed in the bell menu.



The versions of your deployed Defenders are listed under **Manage > Defenders > Manage**:



Upgrading Console when using projects

When you have one or more tenant projects, upgrade all Supervisor Consoles before upgrading the Central Console. During the upgrade process, there may be periods where the supervisors appear disconnected. This is normal, because supervisors are disconnected while the upgrade occurs and Central Console will try to reestablish connectivity every 10 minutes. Within 10 minutes of upgrading all supervisors and the Central Console, all supervisors should appear healthy.

Except during the upgrade process, the Central Console and all Supervisor Consoles must run the same product version. Having different product versions is not supported and may lead to instability and connectivity problems.

Upgrade each Supervisor and then the Central Console using the appropriate procedure:

- Console - Onebox
- Console - Kubernetes
- Console - Open Shift
- Console - Helm
- Console - Amazon ECS

Upgrade Onebox

Edit on GitHub

Upgrade Prisma Cloud Onebox. First upgrade Console. Console will then automatically upgrade all deployed Defenders for you.

If Console fails to upgrade one or more Defenders, manually upgrade your Defenders.

You must manually upgrade App-Embedded Defenders.

Upgrading Console

To upgrade Console, rerun the install script for the latest version of Prisma Cloud. Use this method for any Console that was originally installed with the *twistlock.sh* script.

STEP 1 | Download the latest recommended release.

STEP 2 | Unpack the downloaded tarball.

Optional: you may wish to unpack the tarball to a different folder than any previous tarballs.

```
$ mkdir twistlock_<VERSION>
$ tar -xzf prisma_cloud_compute_edition_<VERSION>.tar.gz -C
 twistlock_<VERSION>/
```

The setup package contains updated versions of *twistlock.sh* and *twistlock.cfg*.

STEP 3 | Check the version of Prisma Cloud that will be installed:

```
$ grep DOCKER_TWISTLOCK_TAG twistlock.cfg
```

STEP 4 | Upgrade Prisma Cloud while retaining your current data and configs by using the *-j* option. The *-j* option merges your current configuration with any new configuration settings in the new version of the software.

You must use the same install target in your upgrade as your original installation. There are two install targets: *onebox* and *console*, where *onebox* installs both Console and Defender onto a host and *console* just installs Console.

To upgrade your *onebox* install, run:

```
$ sudo ./twistlock.sh -syj onebox
```

To upgrade your *console* install, run:

```
$ sudo ./twistlock.sh -syj console
```

STEP 5 | Go to **Manage > Defenders > Manage** and validate that Console has upgraded your Defenders.

Kubernetes

Edit on GitHub

Upgrading Prisma Cloud running in your Kubernetes cluster requires the following steps.

1. Upgrade the Prisma Cloud Console. **Only required for the Prisma Cloud Compute Edition (self-hosted).**
2. Upgrade your Defenders deployed in your cluster.

Upgrading Console

Since Prisma Cloud objects can be specified with configuration files, we recommend declarative object management for both install and upgrade.

You should have kept good notes when initially installing Prisma Cloud. The configuration options set in *twistlock.cfg* and the parameters passed to *twistcli* in the initial install are used to generate working configurations for the upgrade.

Prerequisites: You know how you initially installed Prisma Cloud, including all options set in *twistcli.cfg* and parameters passed to *twistcli*.

STEP 1 | Download the latest recommended release to the host where you manage your cluster with *kubectl*.

STEP 2 | If you customized *twistlock.cfg*, port those changes forward to *twistlock.cfg* in the latest release. Otherwise, proceed to the next step.

STEP 3 | Generate new YAML configuration file for the latest version of Prisma Cloud. Pass the same options to *twistcli* as you did in the original install. The following example command generates a YAML configuration file for the default basic install.

```
$ <PLATFORM>/twistcli console export kubernetes --service-type
 LoadBalancer
```

STEP 4 | Update the Prisma Cloud objects.

```
$ kubectl apply -f twistlock_console.yaml
```

STEP 5 | Go to **Manage > Defenders > Manage** and validate that Console has upgraded your Defenders.

OpenShift

Edit on GitHub

Upgrade Prisma Cloud running in your OpenShift cluster.

First upgrade Console. Console will then automatically upgrade all deployed Defenders for you.

If you've disabled Defender auto-upgrade or if Console fails to upgrade one or more Defenders, manually upgrade your Defenders.

You must manaully upgrade App-Embedded Defenders.

Upgrading Console

STEP 1 | Download the latest recommended release to the host where you manage your cluster with *oc*.

STEP 2 | If you customized *twistlock.cfg*, port those changes forward to *twistlock.cfg* in the latest release. Otherwise, proceed to the next step.

STEP 3 | (Optional) If you're storing Twistlock images in the cluster's internal registry, pull the latest images from Twistlock's cloud registry and push them there. >>>>>>> master:upgrade/upgrade_openshift.adoc Otherwise, proceed to the next step.

1. Pull the latest Prisma Cloud images using URL auth.

```
$ sudo docker pull registry-auth.twistlock.com/
tw_<ACCESS_TOKEN>/twistlock/defender:defender_<VERSION>
$ sudo docker pull registry-auth.twistlock.com/
tw_<ACCESS_TOKEN>/twistlock/console:console_<VERSION>
```

2. Retag the images so that they can be pushed to your

```
$ sudo docker tag \
  registry-auth.twistlock.com/tw_<ACCESS_TOKEN>/twistlock/
defender:defender_<VERSION> \
  docker-registry.default.svc:5000/twistlock/
private:defender_<VERSION>
$ sudo docker tag \
  registry-auth.twistlock.com/tw_<ACCESS_TOKEN>/twistlock/
console:console_<VERSION> \
  docker-registry.default.svc:5000/twistlock/
private:console_<VERSION>
```

3. Push the Prisma Cloud images to your cluster's internal registry.

```
$ sudo docker push docker-registry.default.svc:5000/twistlock/
private:defender_<VERSION>
$ sudo docker push docker-registry.default.svc:5000/twistlock/
private:console_<VERSION>
```


STEP 4 | Generate new YAML configuration file for the latest version of Twistlock. Pass the same options to *twistcli* as you did in the original install. The following example command generates a YAML configuration file for the default basic install.

```
$ <PLATFORM>/twistcli console export openshift \
  --persistent-volume-labels "app-volume=twistlock-console" \
  --service-type "ClusterIP"
```

If you want to pull the image from the internal registry:

```
$ <PLATFORM>/twistcli console export openshift \
  --persistent-volume-labels "app-volume=twistlock-console" \
  --image-name "docker-registry.default.svc:5000/twistlock/
private:console_<VERSION>" \
  --service-type "ClusterIP"
```

For other command variations, see the OpenShift 4 deployment guide.

STEP 5 | Update the Twistlock objects.

```
$ oc apply -f twistlock_console.yaml
```

STEP 6 | Go to **Manage > Defenders > Manage** and validate that Console has upgraded your Defenders.

Helm charts

Edit on GitHub

If you installed Prisma Cloud into your Kubernetes or OpenShift cluster with Helm charts, you can upgrade with the *helm upgrade* command.

First upgrade Console. Console will then automatically upgrade all deployed Defenders for you.

If you've disabled Defender auto-upgrade or if Console fails to upgrade one or more Defenders, manually upgrade your Defenders.

You must manually upgrade App-Embedded Defenders.

Upgrading Console

Generate an updated Helm chart for Console, and then upgrade to it.

STEP 1 | Download the latest recommended release.

STEP 2 | Create an updated Console Helm chart.

```
$ <PLATFORM>/twistcli console export kubernetes \
  --service-type LoadBalancer \
  --helm
```

STEP 3 | Install the updated chart.

```
$ helm upgrade twistlock-console \
  --namespace twistlock \
  --recreate-pods \
 ./twistlock-console-helm.tar.gz
```

STEP 4 | Go to **Manage > Defenders > Manage** and validate that Console has upgraded your Defenders.

Amazon ECS

Edit on GitHub

Upgrade Prisma Cloud running on Amazon ECS.

First upgrade Console. Then, upgrade your Defenders.

When you upgrade Defenders, for any unsuccessful upgrades you can review the error messages in **Manage > Defenders > Manage** . And, if you've created an alert for Defender health events, you also receive a notification to the configured alert provider.

Upgrade Console

To upgrade Console, update the service with a new task definition that points to the latest image.

This procedure assumes you're using images from Prisma Cloud's registry. If you're using your own private registry, push the latest Console image there first.

Copy the Prisma Cloud config file into place

STEP 1 | Download the latest recommended release to your local machine.

```
$ wget <LINK_TO_CURRENT_RECOMMENDED_RELEASE_LINK>
```

STEP 2 | Unpack the Prisma Cloud release tarball.

```
$ mkdir twistlock
$ tar xvzf twistlock_<VERSION>.tar.gz  -C twistlock/
```

STEP 3 | Upload the *twistlock.cfg* files to the host that runs Console.

```
$ scp twistlock.cfg <ECS_INFRA_NODE>:/twistlock_console/var/lib/
twistlock-config
```

Create a new revision of the task definition

Create a new revision of the task definition.

STEP 1 | Log into the Amazon ECS console.

STEP 2 | In the left menu, click **Task Definitions**.

STEP 3 | Check the box for the Prisma Cloud Console task definition, and click **Create new revision**.

STEP 4 | Scroll to the bottom of the page and click **Configure via JSON**.

1. Update the *image* field to point to the latest Console image.

   For example, if you were upgrading from Prisma Cloud version 2.4.88 to 2.4.95, simply change the version string in the image tag.

   ```
   "image": "registry-auth.twistlock.com/tw_<accesstoken>/
   twistlock/console:console_2_4_95"
   ```

2. Click **Save**.

STEP 5 | Click **Create**.

Update the Console service

Update the Console service.

STEP 1 | In the left menu of the Amazon ECS console, click **Clusters**.

STEP 2 | Click on your cluster.

STEP 3 | Select the **Services** tab.

STEP 4 | Check the box next the Console service, and click **Update**.

STEP 5 | In **Task Definition**, select the version of the task definition that points to the latest Console image.

STEP 6 | Validate that **Cluster**, **Service name**, and **Number of tasks** are correct. These values are set based on the values for the currently running task, so the defaults should be correct. The number of tasks must be 1.

STEP 7 | Set **Minimum healthy percent** to **0**.

This lets ECS safely stop the single Console container so that it can start an updated Console container.

STEP 8 | Set **Maximum percent** to **100**.

STEP 9 | Click **Next**.

STEP 10 | In the **Configure network** page, accept the defaults, and click **Next**.

STEP 11 | In the **Set Auto Scaling** page, accept the defaults, and click **Next**.

STEP 12 | Click **Update Service**.

It takes a few moments for the old Console service to be stopped, and for the new service to be started. Open Console, and validate that the UI shows new version number in the bottom left corner.

STEP 13 | Go to **Manage > Defenders > Manage** and validate that Console has upgraded your Defenders.

If Console fails to upgrade any Defender, upgrade it manually.

Upgrade the Single Container Defenders

Edit on GitHub

The Console user interface lets you upgrade all Defenders in a single shot. This method minimizes the effort required to upgrade all your deployed Defenders.

Alternatively, you can select which Defenders to upgrade. Use this method when you have different maintenance windows for different deployments. For example, you might have an open window on Tuesday to upgrade thirty Defenders in your development environment, but no available window until Saturday to upgrade the remaining twenty Defenders in your production environment. In order to give you sufficient time to upgrade your environment, older versions of Defender can coexist with the latest version of Defender and the latest version of Console.

Prerequisites: You have already upgraded Console.

STEP 1 | Open Console.

STEP 2 | On the left menu bar, go to **Manage > Defender > Manage** and click **Defenders** to see a list of all your deployed stand-alone Container Defenders.

STEP 3 | Upgrade your stand-alone Defenders. You can either:

- Upgrade all Defenders at the same time by clicking **Upgrade all**.
- Upgrade a subset of your Defenders by clicking the individual **Actions > Upgrade** button in the row that corresponds to the Defender you want to upgrade.

*The **Restart** and **Decommission** buttons are not available for DaemonSet Defenders. They are only available for stand-alone Defenders.*

Upgrade Defender DaemonSets

Edit on GitHub

Upgrade the Defender DaemonSets in your environment.

Upgrade the Defender DaemonSets with twistcli (Kubernetes)

Delete the Defender DaemonSet, then rerun the original install procedure.

Prerequisites: You know all the parameters passed to *twistcli* when you initially deployed the Defender DaemonSet. You'll need them to recreate a working configuration file for your environment.

STEP 1 | Delete the Defender DaemonSet.

```
$ kubectl -n twistlock delete ds twistlock-defender-ds
$ kubectl -n twistlock delete sa twistlock-service
$ kubectl -n twistlock delete secret twistlock-secrets
```

STEP 2 | Determine the Console service's external IP address.

```
$ kubectl get service -o wide -n twistlock
```

STEP 3 | Generate a *defender.yaml* file. Pass the same options to *twistcli* as you did in the original install. The following example command generates a YAML configuration file for the default install.

The following command connects to Console's API (specified in *--address*) as user <ADMIN> (specified in *--user*), and retrieves a Defender DaemonSet YAML config file according to the configuration options passed to *twistcli*. In this command, there is just a single mandatory configuration option. The *--cluster_address* option specifies the address Defender uses to connect to Console, and the value is encoded in the DaemonSet YAML file.

```
$ <PLATFORM>/twistcli defender export kubernetes \
  --address https://yourconsole.example.com:8083 \
  --user <ADMIN_USER> \
  --cluster-address twistlock-console
```

- <PLATFORM> can be linux or osx.
- <ADMIN_USER> is the name of an admin user.

STEP 4 | Deploy the Defender DaemonSet.

```
   $ kubectl create -f defender.yaml
```

STEP 5 | Open a browser, navigate to Console, then go to **Manage > Defenders > Manage** to see a list of deployed Defenders.

Upgrade the Defender DaemonSets with twistcli (OpenShift)

Delete the Defender DaemonSet, then rerun the original install procedure.

Prerequisites: You know all the parameters passed to *twistcli* when you initially deployed the Defender DaemonSet. You'll need them to recreate a working configuration file for your environment.

STEP 1 | Delete the Defender DaemonSet.

```
$ oc -n twistlock delete ds twistlock-defender-ds
$ oc -n twistlock delete sa twistlock-service
$ oc -n twistlock delete secret twistlock-secrets
```

STEP 2 | Determine the Console service's external IP address.

```
$ oc get service -o wide -n twistlock
```

STEP 3 | Generate a *defender.yaml* file. Pass the same options to *twistcli* as you did in the original install. The following example command generates a YAML configuration file for the default install.

The following command connects to Console's API (specified in --*address*) as user <ADMIN> (specified in --*user*), and retrieves a Defender DaemonSet YAML config file according to the configuration options passed to *twistcli*. In this command, there is just a single mandatory configuration option. The --*cluster_address* option specifies the address Defender uses to connect to Console, and the value is encoded in the DaemonSet YAML file.

```
$ <PLATFORM>/twistcli defender export openshift \
  --address https://yourconsole.example.com:8083 \
  --user <ADMIN_USER> \
  --cluster-address twistlock-console \
  --selinux-enabled
```

- <PLATFORM> can be linux or osx.
- <ADMIN_USER> is the name of an admin user.

STEP 4 | Deploy the Defender DaemonSet.

```
   $ oc create -f defender.yaml
```

STEP 5 | Open a browser, navigate to Console, then go to **Manage > Defenders > Manage** to see a list of deployed Defenders.

Upgrade the Defender DaemonSets from Console

Upgrade the Defender DaemonSets directly from the Console UI.

If you can't access your cluster with kubectl or oc, then you can upgrade Defender DaemonSets directly from the Console UI.

Prerequisites: You've created a kubeconfig credential for your cluster so that Prisma Cloud can access it to upgrade the Defender DaemonSet.

STEP 1 | Log into Prisma Cloud Console.

STEP 2 | Go to **Manage > Defenders > Manage**.

STEP 3 | Click **DaemonSets**.

STEP 4 | For each cluster in the table, click **Actions > Upgrade.**

The table shows a count of deployed Defenders and their new version number.

Upgrade Defender DaemonSets (Helm)

Edit on GitHub

Generate an updated Helm chart for the Defender DaemonSet, and then upgrade to it.

STEP 1 | Create an updated Defender DaemonSet Helm chart.

```
$ <PLATFORM>/twistcli defender export kubernetes \
  --address https://yourconsole.example.com:8083 \
  --user <ADMIN_USER> \
  --cluster-address twistlock-console \
  --helm
```

STEP 2 | Install the updated chart.

```
$ helm upgrade twistlock-defender-ds \
  --namespace twistlock \
  --recreate-pods
  ./twistlock-console-helm.tar.gz
```


Technology overviews

Edit on GitHub

This section describes how key Prisma Cloud components work.

- Intelligence Stream
- Prisma Cloud Advanced Threat Protection
- App-specific network intelligence
- Container Runtimes
- Radar
- Serverless Radar
- Prisma Cloud Rules Guide - Docker
- Defender architecture
- Host Defender architecture
- TLS v1.2 cipher suites
- Telemetry

Intelligence Stream

Edit on GitHub

The Prisma Cloud Intelligence Stream (IS) is a real-time feed that contains vulnerability data and threat intelligence from a variety of certified upstream sources. Prisma Cloud continuously pulls data from known vulnerability databases, official vendor feeds and commercial providers to provide the most accurate vulnerability detection results.

The console automatically connects to the intelligence server and downloads updates without any special configuration required. The IS is updated several times per day, and consoles continuously check for updates.

You can update Console vulnerability and threat data even if it runs in an offline environment. For more information, see Update Intelligence Stream in offline environments.

In addition to the information collected from official feeds, Prisma Cloud feeds are enriched with vulnerability data curated by a dedicated research team. Our security researchers monitor cloud and open source projects to identify security issues through automated and manual means. As a result, Prisma Cloud can detect new vulnerabilities that were only recently disclosed, and even vulnerabilities that were quietly patched.

Prisma Cloud Advanced Threat Protection

Edit on GitHub

Prisma Cloud Advanced Threat Protection (ATP) is a collection of malware signatures and IP reputation lists aggregated from commercial threat feeds, open source threat feeds, and Prisma Cloud Labs. It is delivered to your installation via the Prisma Cloud Intelligence Stream.

The data in ATP is used by Prisma Cloud's runtime defense system to detect suspicious activities, such as a container communicating with a botnet herder or Tor entry node. You can augment ATP by importing custom malware data and importing IP reputation lists. ATP is the combination of both the Prisma Cloud-provided data set and your own custom data set.

The following hypothetical scenario illustrates how ATP protects your cloud workloads:

1. An attacker exploits a vulnerability in an app running in a container.
2. The attacker attempts to download malware into a workload from a distribution point.
3. Based on the ATP feed, Prisma Cloud runtime defense detects both the connection to the malware server and the write of the malicious file to the workload file system.
4. Alerts/prevention is applied based on the runtime configuration.

Enabling ATP

ATP is enabled in the default rules that ship with the product, with the effect set to alert. You can impose more stringent control by setting effect to prevent or block. Runtime defense for file systems lets you actively stop (block) any container that tries to download malware. To disable ATP, create or modify a runtime rule, select the **General** tab, and set **Enable Prisma Cloud Advanced Threat Protection** to **Off**. When ATP is disabled, container interaction with malicious files or IP endpoints does not trigger a runtime event.

Serverless ATP

In Serverless, Prisma Cloud Advanced Threat Protection (ATP) has a slightly different functionality. It's a collection of paths (researched by Prisma Cloud Labs) that define which file system and process activity is allowed within the function. Activities that do not match these paths will raise a security audit. When enabled, it creates an automatic hardening for the function in runtime, without the need to manually configure the runtime policy.

Serverless ATP is enabled by default when creating a new runtime rule. It's effect is similiar to the effects configured under the Processes/File System tabs. To disable ATP, create or modify a runtime rule, select the **General** tab, and set **Prisma Cloud advanced threat protection** to **Off**.

App-specific network intelligence

Edit on GitHub

Prisma Cloud can learn about the settings for your apps from their configuration files, and use this knowledge to detect runtime anomalies. No special configuration is required to enable this feature.

In addition to identifying ports that are exposed via the EXPOSE directive in a Dockerfile, or the -p argument passed to docker run, Prisma Cloud can identify port settings from an app's configuration file. This enables Prisma Cloud to detect, for example, if the app has been commandeered to listen on an unexpected port, or if a malicious process has managed to listen on the app's port to steal data.

Consider the following scenario:

1. You create an Apache image. The default port for httpd, specified in */etc/apache2/apache2.conf*, is 80. In your *Dockerfile*, you use *EXPOSE* to bind port 80 in the container to port 80 on the host.
2. A user runs your Apache image with the *-P* option, mapping port 80 in the container to a random ephemeral port on the host.
3. The running container is compromised. An attacker kills the Apache process, spawns a new process that listens on that port, and harvests data from other containers on the same subnet.
4. Prisma Cloud detects the runtime anomaly, and either alerts you or blocks the container.

Prisma Cloud protects your containers by combining static analysis of the image with runtime analysis of the container. The Prisma Cloud Intelligence Stream delivers app-specific knowledge so that Defender can inspect an image and:

- Identify processes that the container will execute.
- Correlate the processes with their configuration files.
- Parse the configuration files to extract information such as port assignments.

Runtime analysis completes the picture. Some information can only be determined at runtime. For example, MongoDB might be deployed to a container without a configuration file. At runtime, MongoDB is launched with the *--port* parameter, dynamically specifying the port it will listen on. Static analysis tells us that MongoDB is part of the container image, but in this case, only dynamic analysis tells us which port it listens on.

Additional apps will be added periodically, and your installation will be automatically updated via the Prisma Cloud Intelligence Stream.

Supported Apps

Prisma Cloud Intelligence Stream currently delivers app-specific knowledge for:

- Apache
- Elasticsearch
- HAProxy
- Kibana

- MariaDB
- MongoDB
- MySQL
- Nginx
- PostgreSQL
- RabbitMQ
- Redis
- Tomcat
- WordPress
- BusyBox

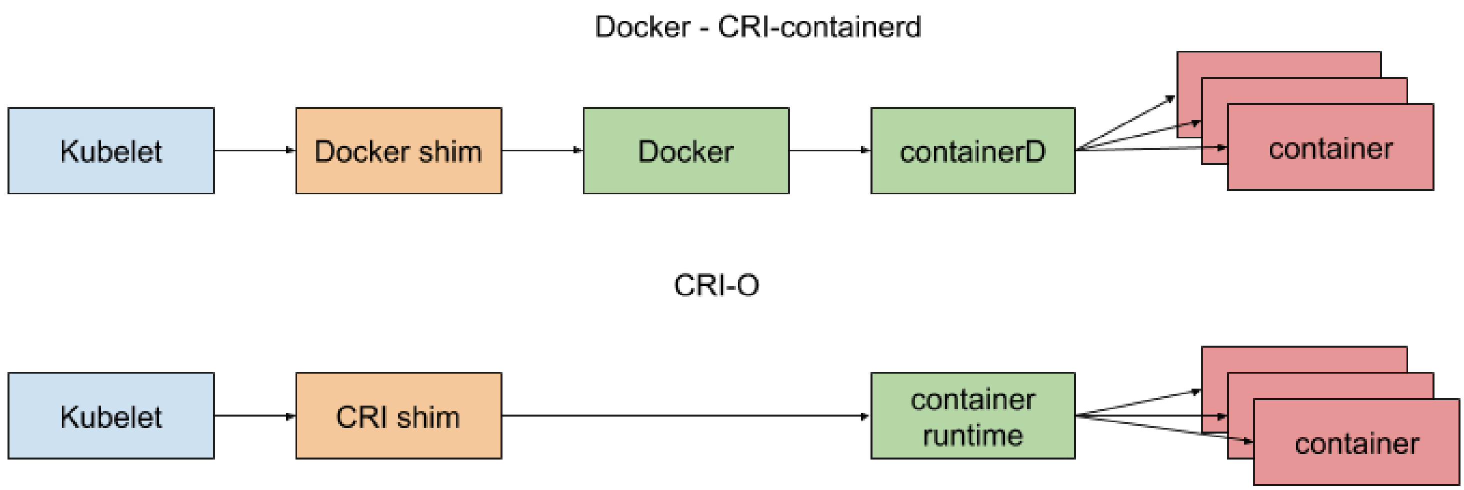
If you would like to see coverage for a specific app, open a support ticket and make a request.

Container Runtimes

Edit on GitHub

Docker Engine is a general purpose container runtime. Docker can run containers from images, but it can also build images from Dockerfiles. Docker supports multiple different environmens and orchestrators, including Kubernetes.

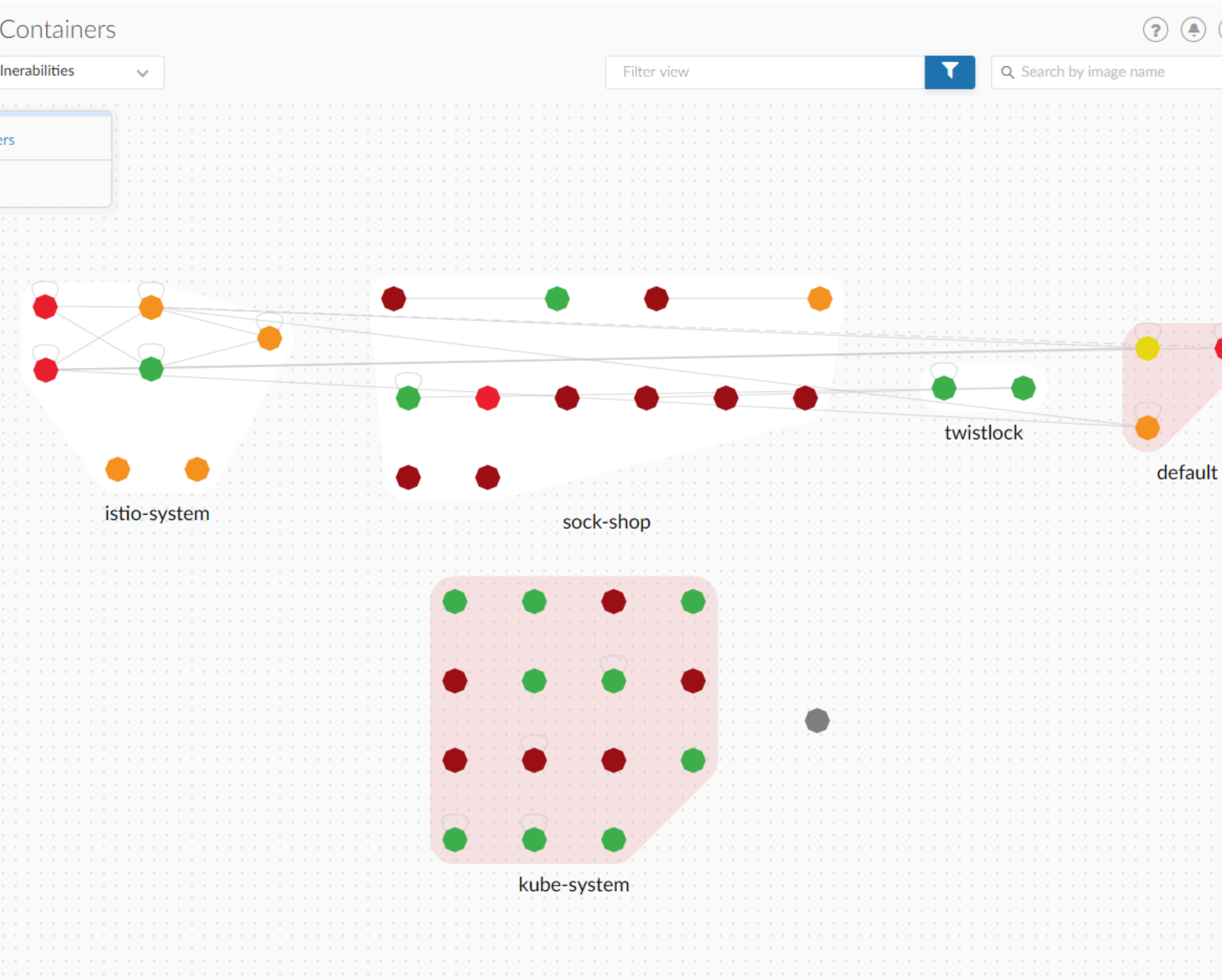
Container Runtime Interface (CRI) is a plugin interface that lets Kubernetes use a wide variety of container runtimes, including Docker Engine. The interface implements only the features needed to run containers from images. Its goal is to be as simple as possible to complete its given task. Since its range of capabilities is tightly scoped, it can be more easily secured.



Radar

Edit on GitHub

Radar is the primary interface for monitoring and understanding your environment. It is the default view when you first log into Console. It is designed to let you visualize and navigate through all of Prisma Cloud's data. For example, you can visualize connectivity between microservices, then instantly drill into the per-layer vulnerability analysis tool, assess compliance, and investigate incidents, all without leaving the Radar canvas.



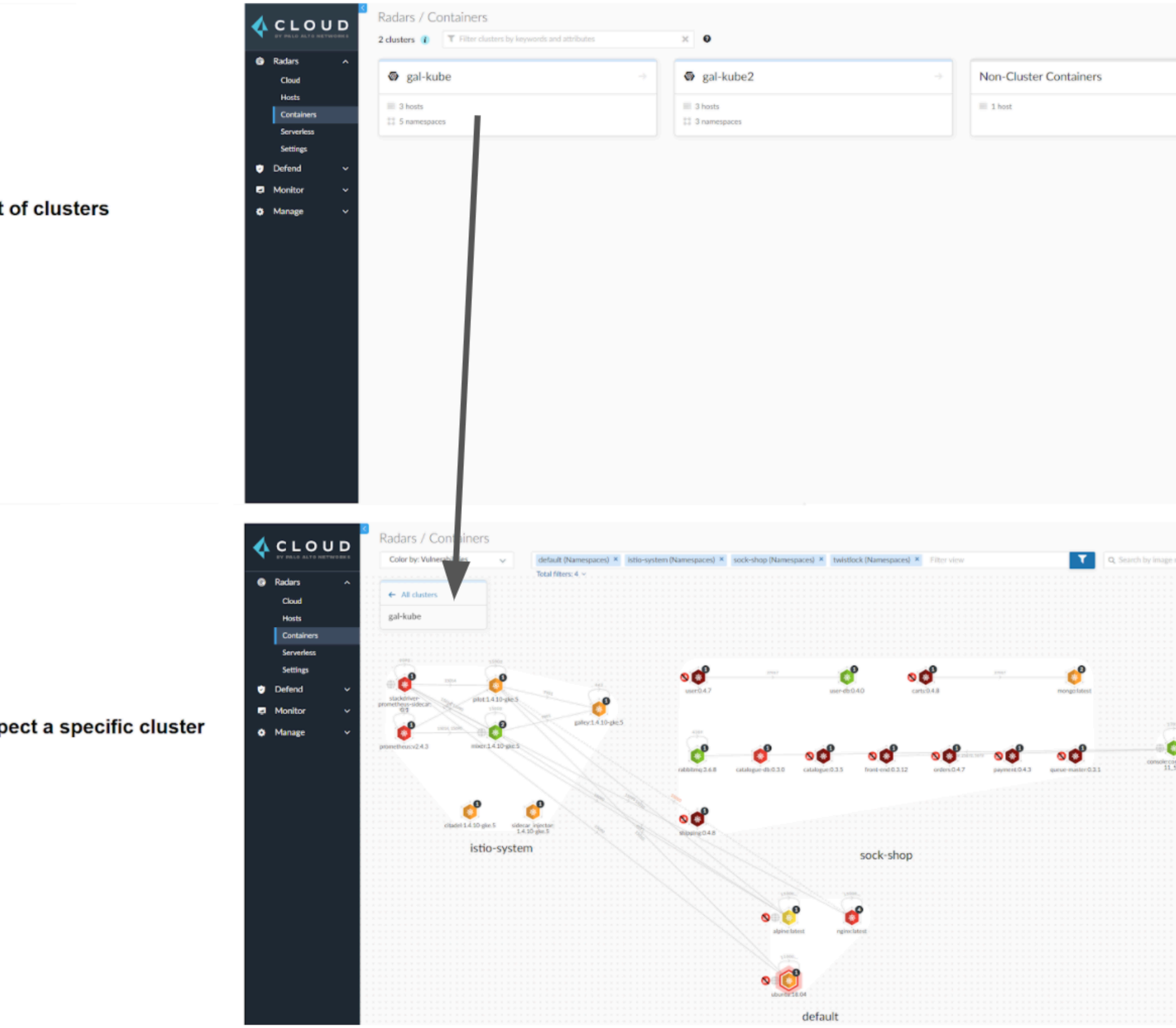
Radar makes it easy to conceptualize the architecture and connectivity of large environments, identify risks, and zoom in on incidents that require response. Radar provides a visual depiction of inter- and intra-network connections between containers, apps, and cluster services across your environment. It shows the ports associated with each connection, the direction of traffic flow, and internet accessibility. When Cloud Native Network Firewall is enabled, Prisma Cloud automatically generates the mesh shown in Radar based on what it has learned about your environment.

Radar's principal pivot is the container view and host view. In the container view, each image with running containers is depicted as a node in the graph. In the host view, each host machine is depicted as a node in the graph. Clicking on a node pops up an overlay that shows vulnerability, compliance, and runtime issues.

Radar refreshes its view every 24 hours. The Refresh button has a red marker when new data is available to be displayed. In order to get full visibility into your environment, Defender should be installed on every host in your environment.

Cluster pivot

Radar segments your environment by cluster. The main view lists all clusters in your environment. You can view information about each cluster such as its cloud provider, number of namespaces, and number of hosts in the cluster. Clicking a card open the image pivot, which shows you all the namespaces and containers in the cluster.



Defenders report which resources belong to which cluster. For managed clusters, Prisma Cloud automatically retrieves the name from the cloud provider. As a fallback, Prisma Cloud can retrieve the name from your kubeconfig file. Finally, you can manually specify the cluster name.

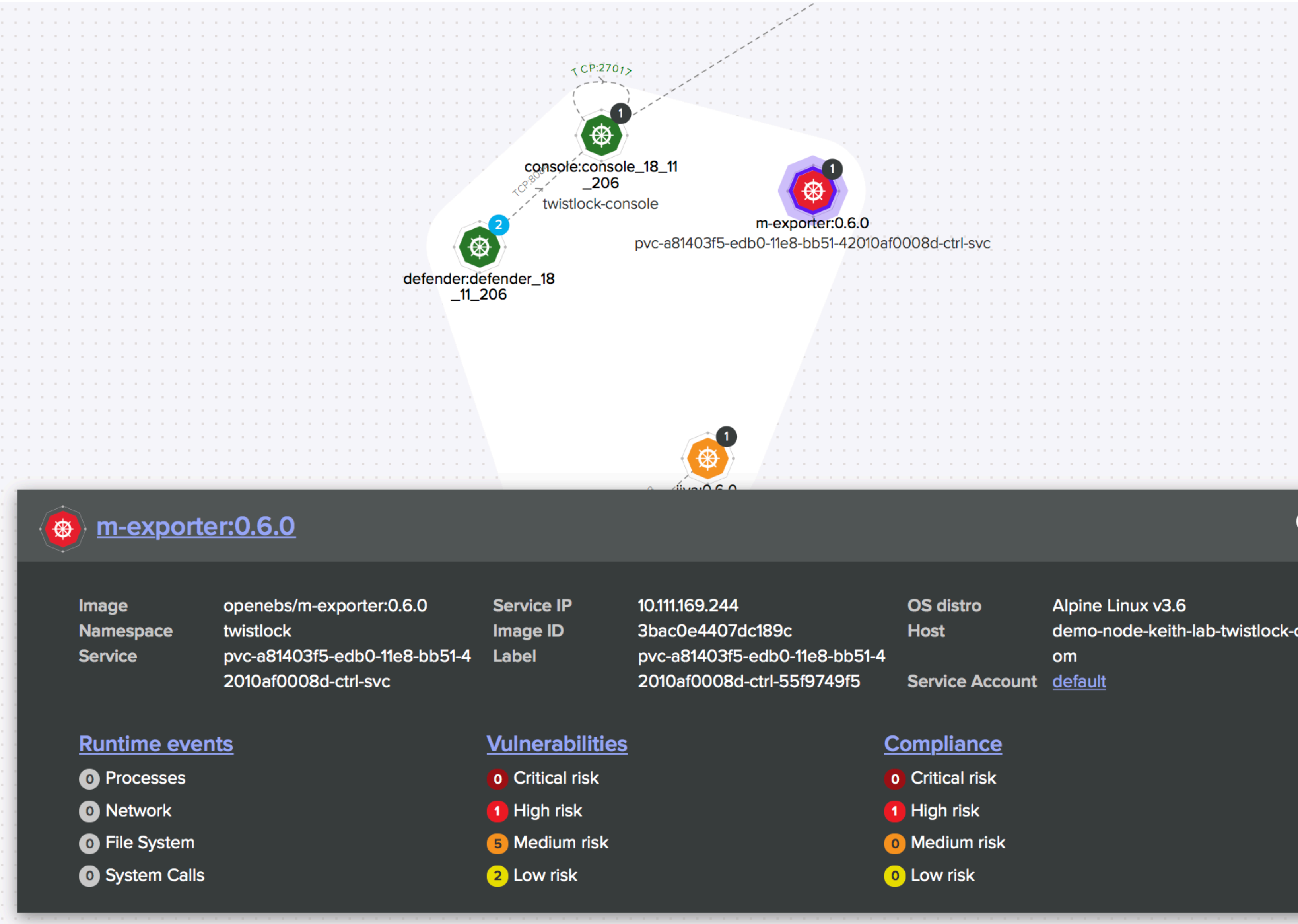
The cluster pivot is currently supported for Kubernetes, OpenShift, and ECS clusters only. All other running containers in your environment are collected in the **Non-Cluster Containers** view.

Image pivot

Radar lays out nodes on the canvas to promote easy analysis of your containerized apps. Interconnected nodes are laid out so network traffic flows from left to right. Traffic sources are weighted to the left, while destinations are weighted to the right. Single, unconnected nodes are arranged in rows at the bottom of the canvas.

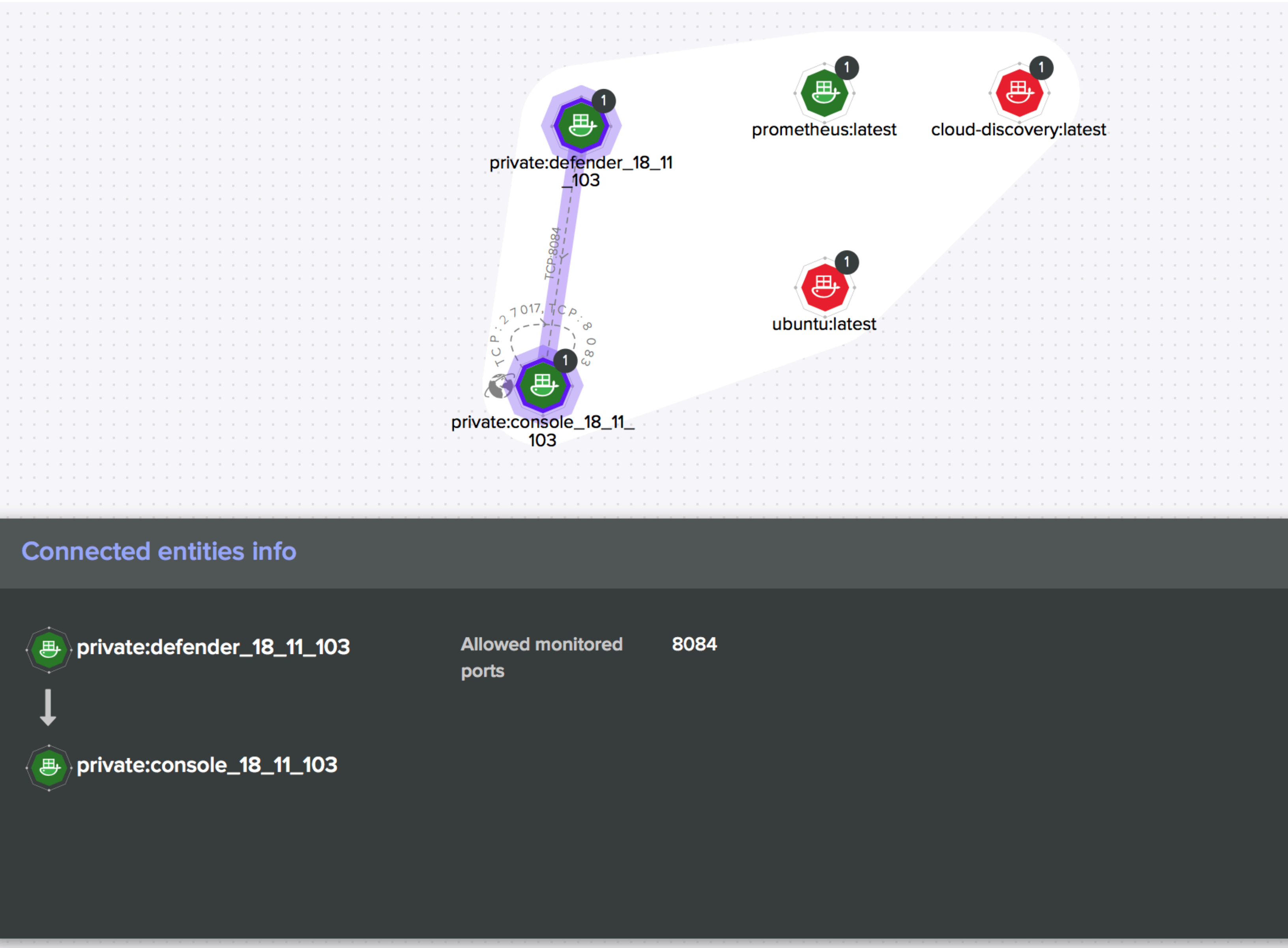
Nodes are color-coded based on the highest severity vulnerability or compliance issue they contain, and reflect the currently defined vulnerability and compliance policies. Color coding lets you quickly spot trouble areas in your deployment.

- Dark Red — High risk. One or more critical severity vulnerabilities detected.
- Red — High severity vulnerabilities detected.
- Orange — Medium vulnerabilities detected.
- Green — Denotes no vulnerabilities detected.

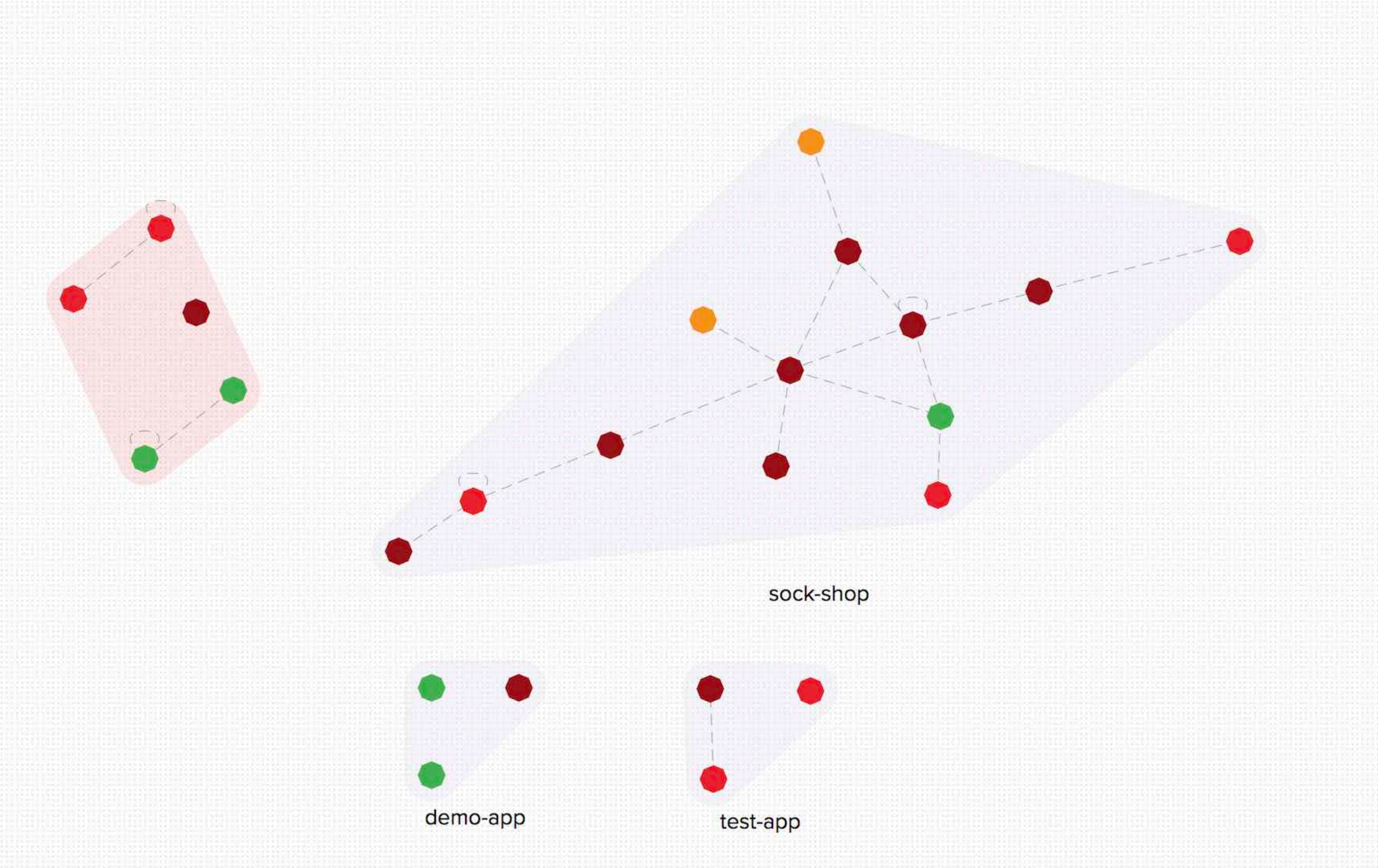


The numeral encased by the circle indicates the number of containers represented by the node. For example, a single Kubernetes DNS node may represent five services. The color of the circle specifies the state of the container's runtime model. A blue circle means the container's model is still in learning mode. A black circle means the container's model is activated. A globe symbol indicates that a container can access the Internet.

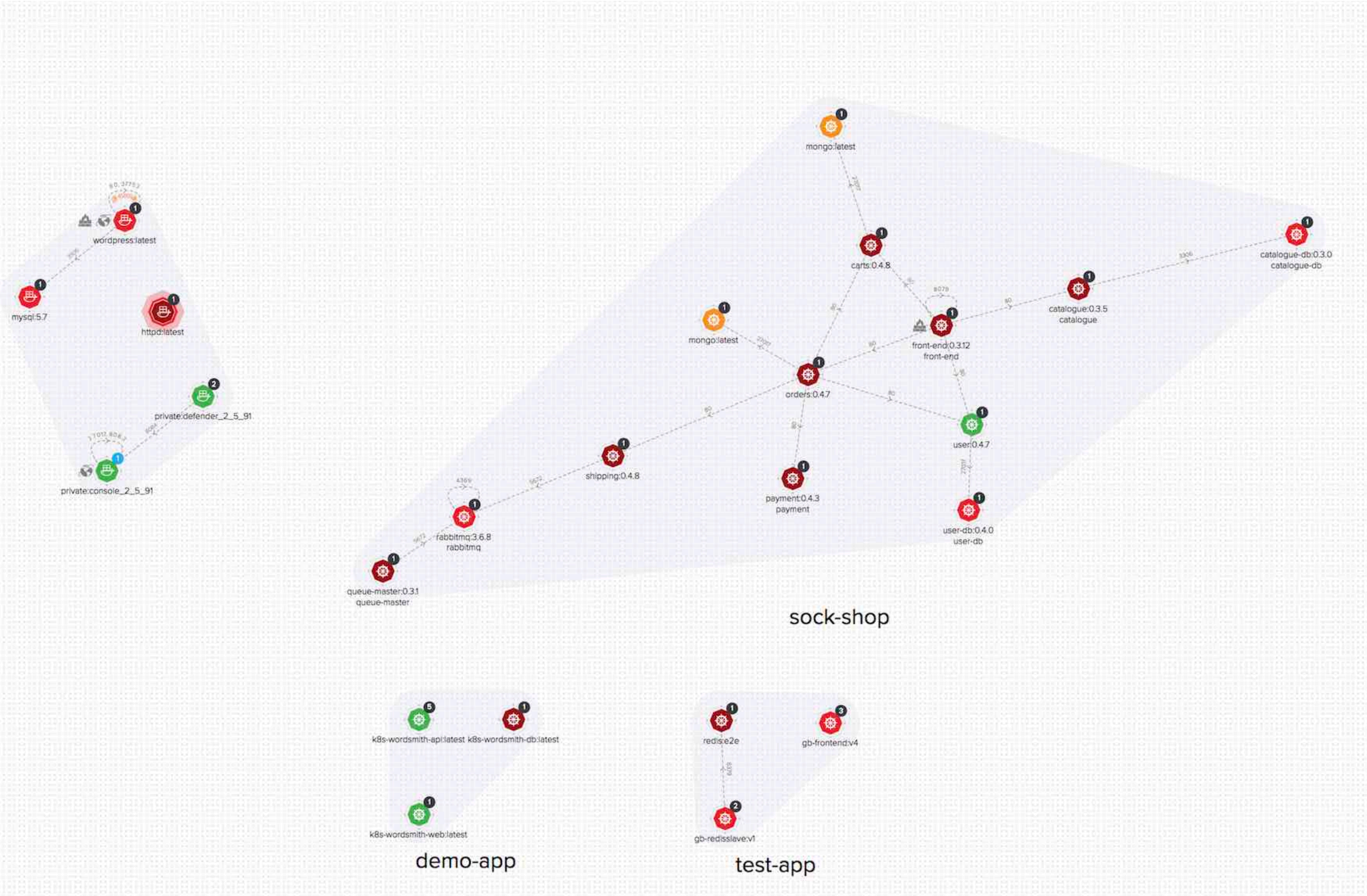
Connections between running containers are depicted as arrows in Radar. Click on an arrow to get more information about the direction of the connection and the port.



The initial zoomed out view gives you a bird's-eye view of your deployments. Deployments are grouped by namespace. A red pool around a namespace indicates an incident occurred in a resource associated with that namespace.



Zooming in provides more detail about each running container. Click on an individual pod to drill down into its vulnerability report, compliance report, and runtime anomalies.



Host pivot

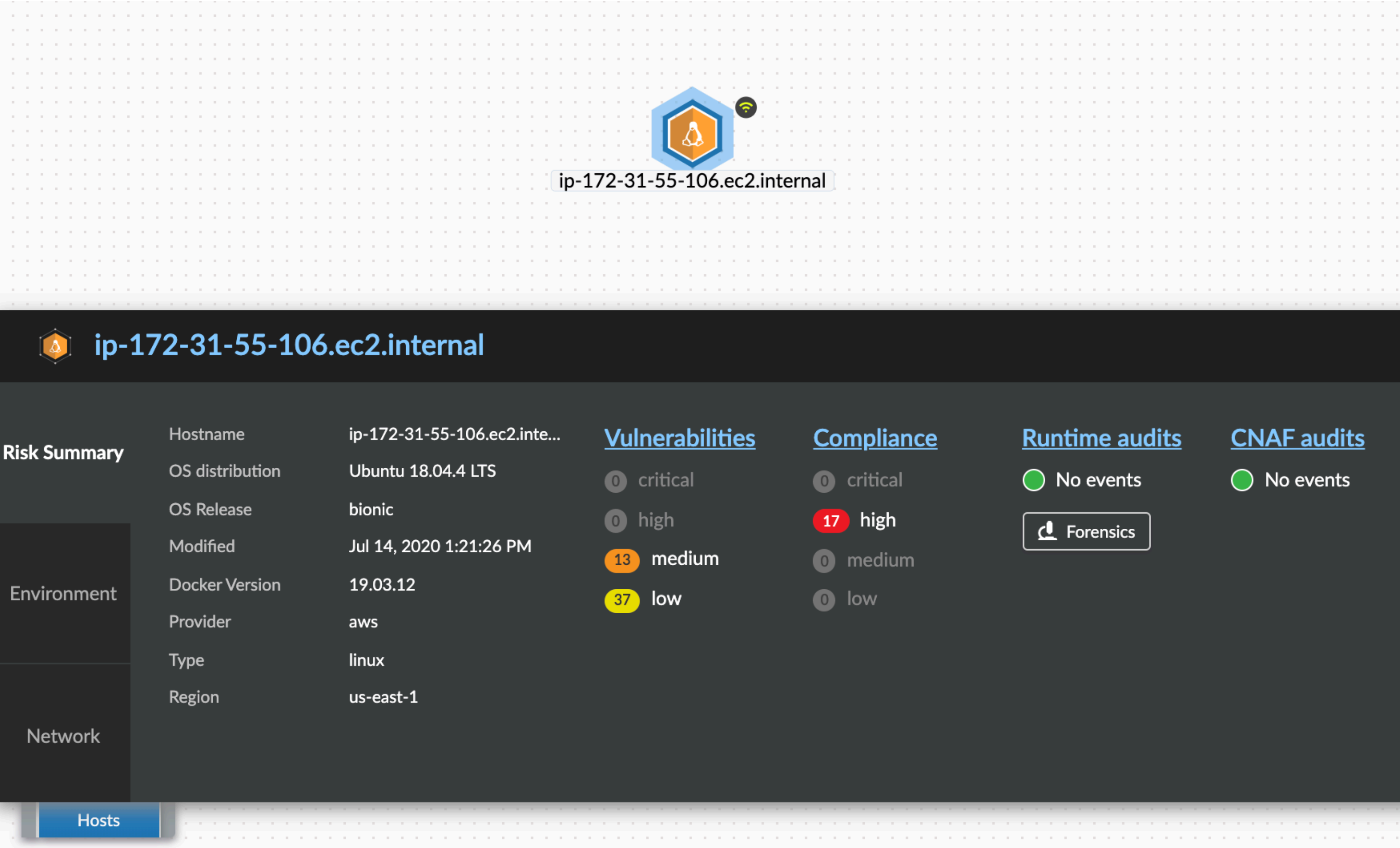
Radar shows the hosts in your environment, how they communicate with each other over the network, and their security posture.

Each node in the host pivot represents a host machine. The mesh shows host-to-host communication.

The color of a node represents the most severe issue detected.

- Dark Red — High risk. One or more critical severity issues detected.
- Red — High severity issues detected.
- Orange — Medium issues detected.
- Green — No issues detected.

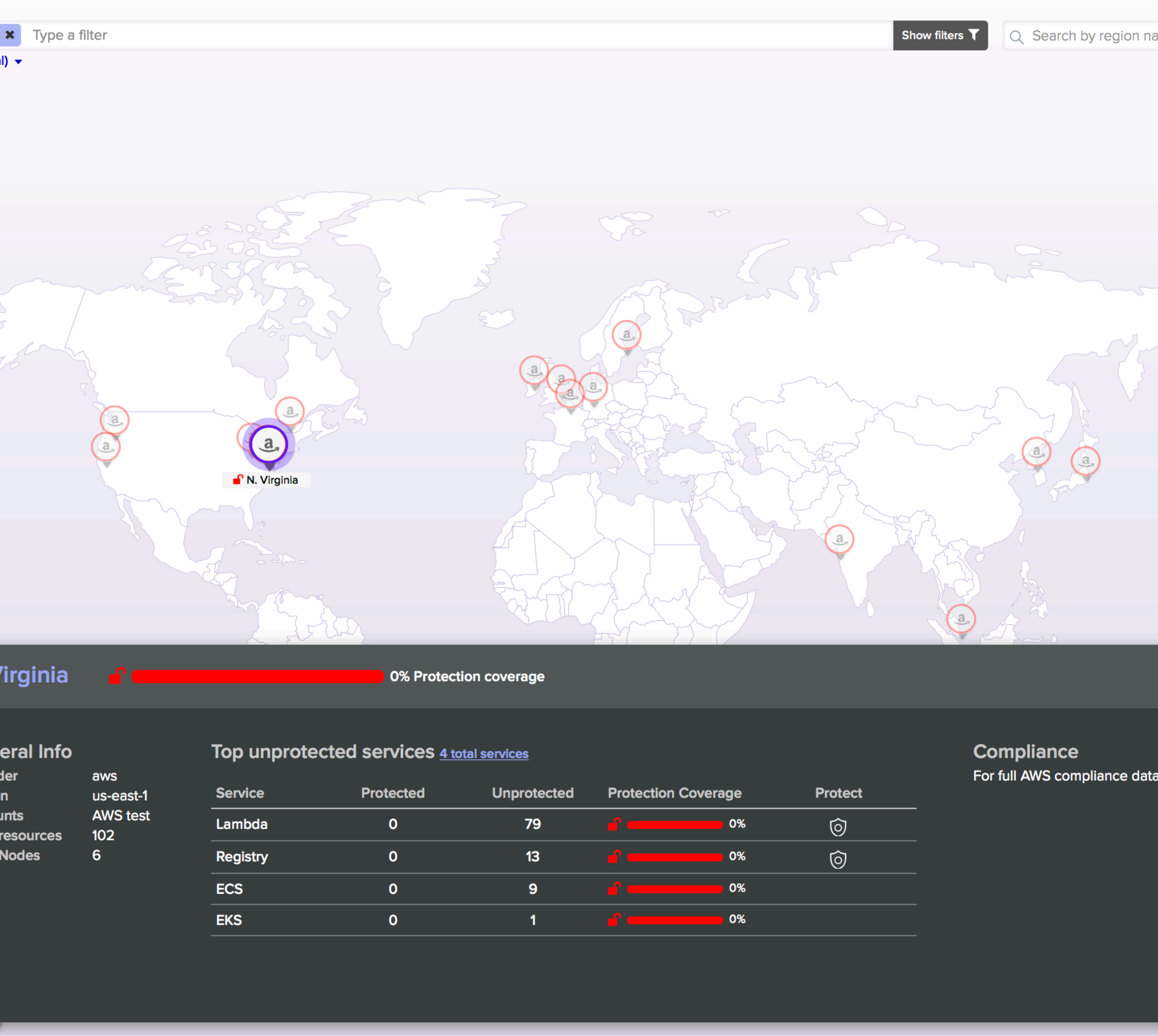
When you click on an node, an overlay shows a summary of all information Prisma Cloud knows about the host. Use the links to drill down into scan reports, audits, and other data.



Cloud pivot

You can't secure what you don't know about. Prisma Cloud cloud discovery finds all cloud-native services deployed in AWS, Azure, and Google Cloud. Cloud Radar helps you visualize what you've deployed across different cloud providers and accounts using a map interface. The map tells you what services are running in which data centers, which services are protected by Prisma Cloud, and their security posture.

Clicking on a marker on the map shows more details about the services deployed in the account/region. Both registries and serverless functions can be secured directly from the info pop-up by clicking **Protect**.



Filtering and search lets you narrow your focus to the data of interest. For example, filters can narrow your view to just the serverless functions in your Azure development team accounts.

By default, there's no data in Cloud Radar.

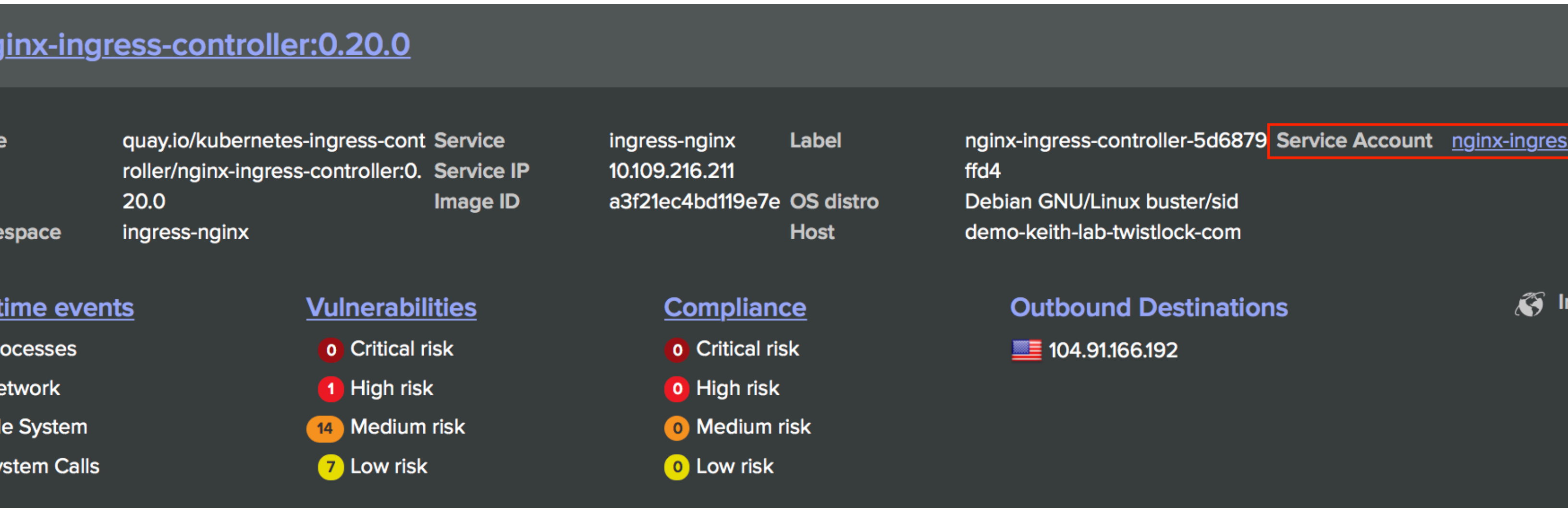
To populate Cloud Radar, configure cloud discovery scans.

Service account monitoring

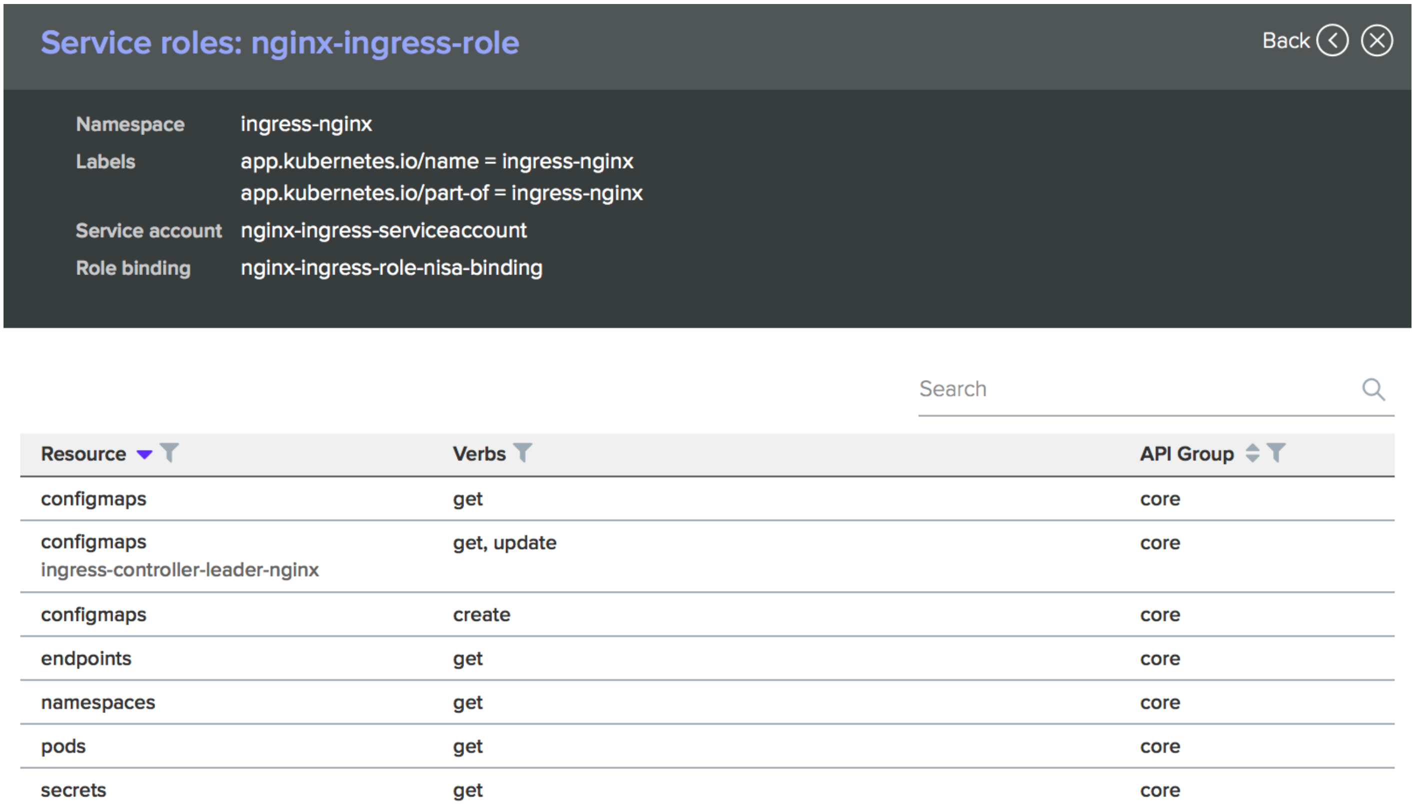
Kubernetes has a rich RBAC model based around the notion of service and cluster roles. This model is fundamental to the secure operation of the entire cluster because these roles control access to resources and services within namespaces and across the cluster. While these service accounts can be manually inspected with kubectl, it's difficult to visualize and understand their scope at scale.

Radar provides a discovery and monitoring tool for service accounts. Every service account associated with a resource in a cluster can easily be inspected. For each account, Prisma Cloud shows detailed metadata describing the resources it has access to and the level of access it has to each of them. This visualization makes it easy for security staff to understand role configuration, assess the level of access provided to each service account, and mitigate risks associated with overly broad permissions.

Clicking on a node opens an overlay, and reveals the service accounts associated with the resource.



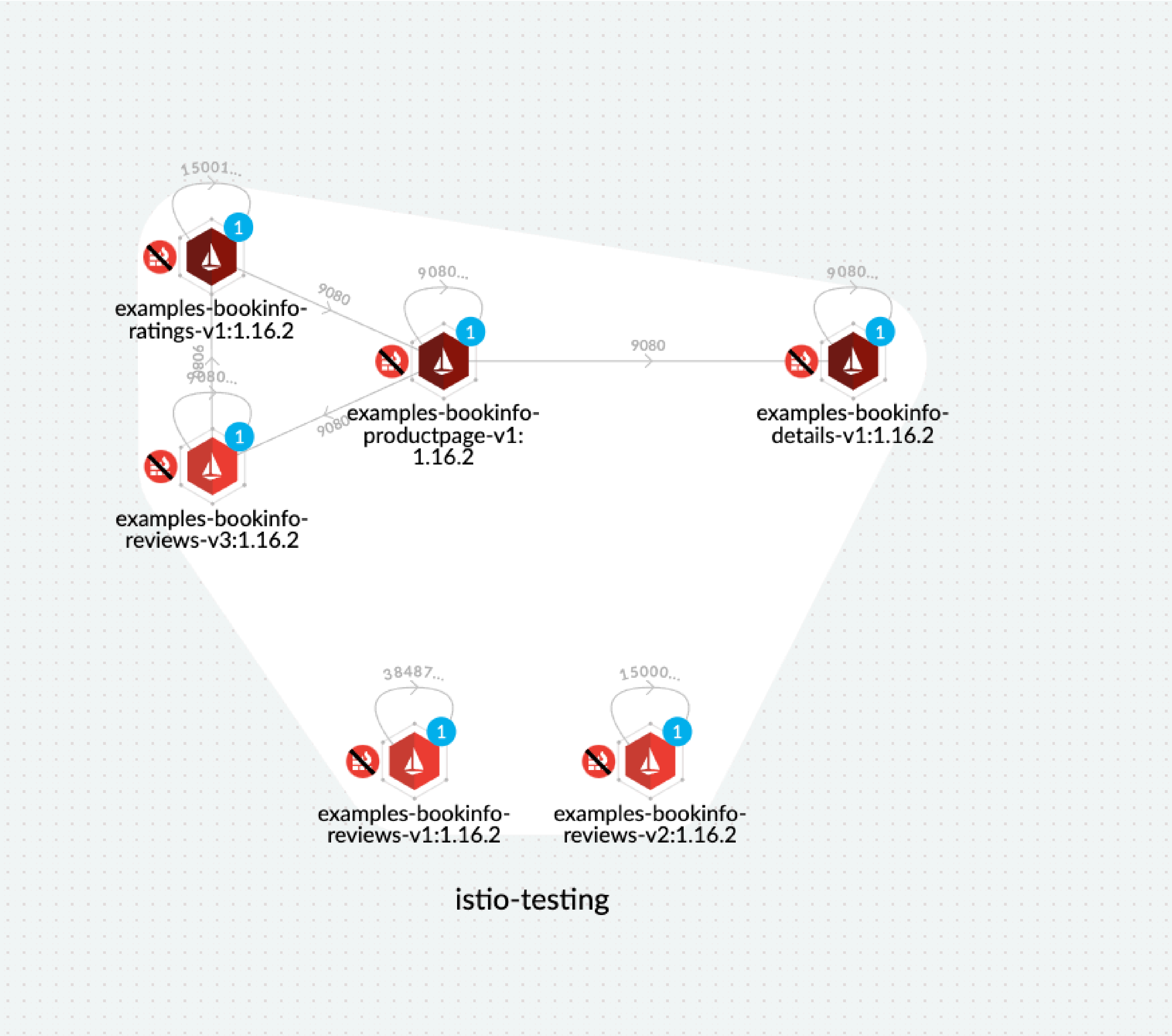
Clicking on the service accounts lists the service roles and cluster roles.



Service account monitoring is available for Kubernetes and OpenShift clusters. When you install the Defender DaemonSet, enable the 'Monitor service accounts' option.

Istio monitoring

When Defender DaemonSets are deployed with Istio monitoring enabled, Prisma Cloud can discover the service mesh and show you the connections for each service. Services integrated with Istio display the Istio logo.



Istio monitoring is available for Kubernetes and OpenShift clusters. When you install the Defender DaemonSet, enable the 'Monitor Istio' option.

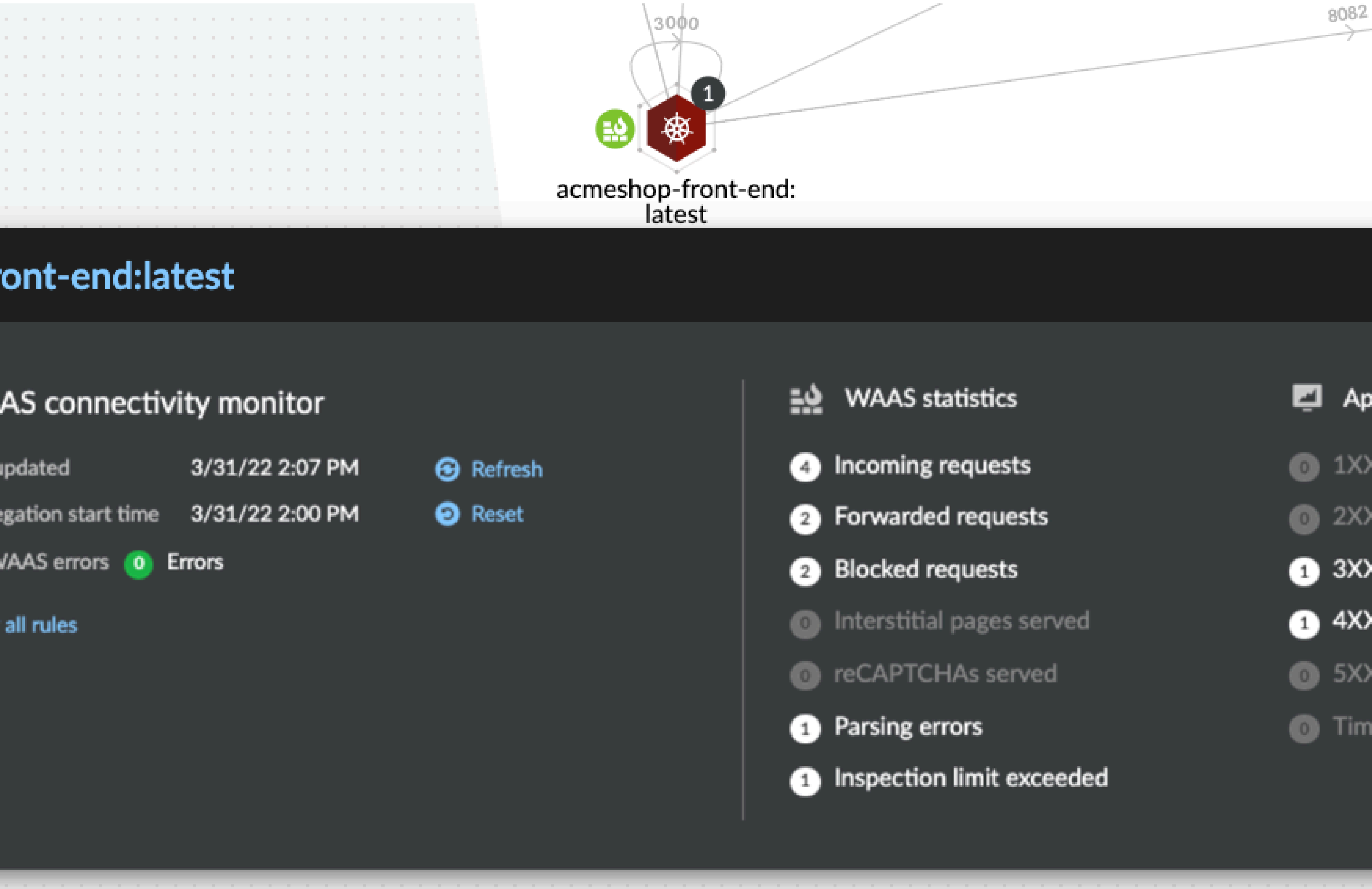
WAAS connectivity monitor

WAAS connectivity monitor monitors the connection between WAAS and the protected application.

WAAS connectivity monitor aggregates data on pages served by WAAS and the application responses.

In addition, it provides easy access to WAAS related errors registered in the Defender logs (Defenders sends logs to the Console every hour).

The monitor tab becomes available when you click on an image or host protected by WAAS.



- **Last updated** - Most recent time when WAAS monitoring data was sent from the Defenders to the Console (Defender logs are sent to the Console on an hourly basis). By clicking on the **refresh** button users can initiate sending of newer data.
- **Aggregation start time** - Time when data aggregation began. By clicking on the **reset** button users can reset all counters.
- **WAAS errors** - To view recent errors related to a monitored image or host, click the **View recent errors** link.

- **WAAS statistics:**
  - *Incoming requests* - Count of HTTP requests inspected by WAAS since the start of aggregation.
  - *Forwarded requests* - Count of HTTP requests forwarded by WAAS to the protected application.
  - *Interstitial pages served* - Count of interstitial pages served by WAAS (interstitial pages are served once Prisma Sessions Cookies are enabled).
  - *reCAPTCHAs served* - Count of reCAPTCHA challenges served by WAAS (when enabled as part of bot protection).
  - *Blocked requests* - Count of HTTP requests blocked by WAAS since the start of aggregation.
  - *Inspection limit exceeded* - Count of HTTP requests since the start of aggregation, in which the body content length exceeded the inspection limit set in the advanced settings.
  - *Parsing errors* - Count of HTTP requests since the start of aggregation, where WAAS encountered an error when trying to parse the message body according to the *Content-Type* HTTP request header.
- **Application statistics**
  - Count of server responses returned from the protected application to WAAS grouped by HTTP response code prefix
  - Count of timeouts (a timeout is counted when a request is forwarded by WAAS to the protected application with no response received within the set timeout period).

Existing WAAS and application statistics counts will be lost once users reset the aggregation start time. ***Reset*** *will* ***not*** *affect WAAS errors and will not cause recent errors to be lost.*

For further details on WAAS deployment, monitoring and troubleshooting please refer to the WAAS deployment page

Serverless Radar

Edit on GitHub

Serverless Radar helps you to visualize and inspect the attack surface of the serverless functions in your environment. Although Prisma Cloud supports multiple serverless environments, currently serverless radar supports AWS Lambda only.

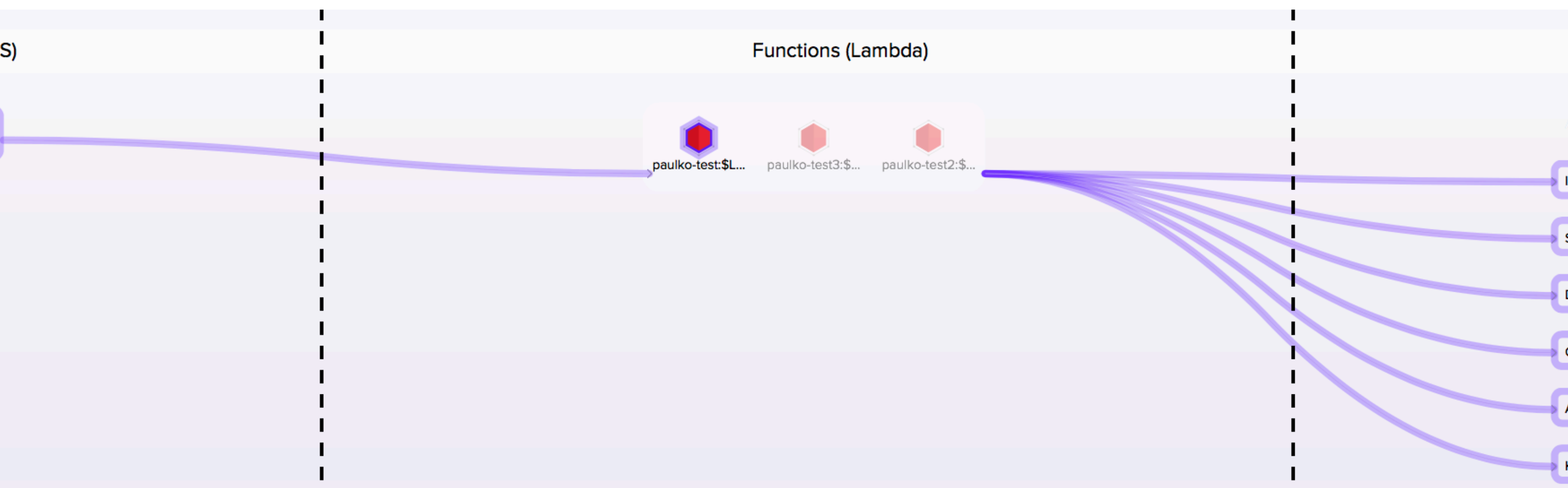
Serverless functions use different interconnect patterns than containers. Serverless apps are highly decomposed and interact with services using cloud provider-specific gateways, rather than directly with each other or through service meshes. Security teams can have difficulty conceptualizing these interactions, identifying which functions interface with which high value assets, and pinpointing unaccpetable exposure.

Even though cloud providers secure the underlying infrastructure that enables Functions as a Service (including isolating functions from each other), it's still easy to deploy functions with vulnerabilities, insecure configurations, and overly permissive roles. The underlying platform might be secure, but sensitive data can still be lost when an insecure function with read access to an S3 bucket is compromised.

Prisma Cloud offers a serverless-specific view in Radar. Serverless Radar uses a three panel view to show the invocation methods for each function, the services they use, and the permissions granted to access those services.

Layout

Serverless Radar shows you how functions interface with other services in their environment.



The left-most column shows how functions are invoked. This is known as the *trigger* or *event source*. Triggers publish events, and Lambda functions are the custom code that process those events.

The middle column shows all the functions in your environment. Functions are colored maroon, red, orange, yellow, or green to let you quickly assess their security posture. By default, functions are colored by their most severe vulnerabilities, but you can view functions by highest severity compliance issue or runtime events. For vulnerability results, you must configure Prisma Cloud to scan your functions. For runtime issues, you must embed Serverless Defender into your functions.

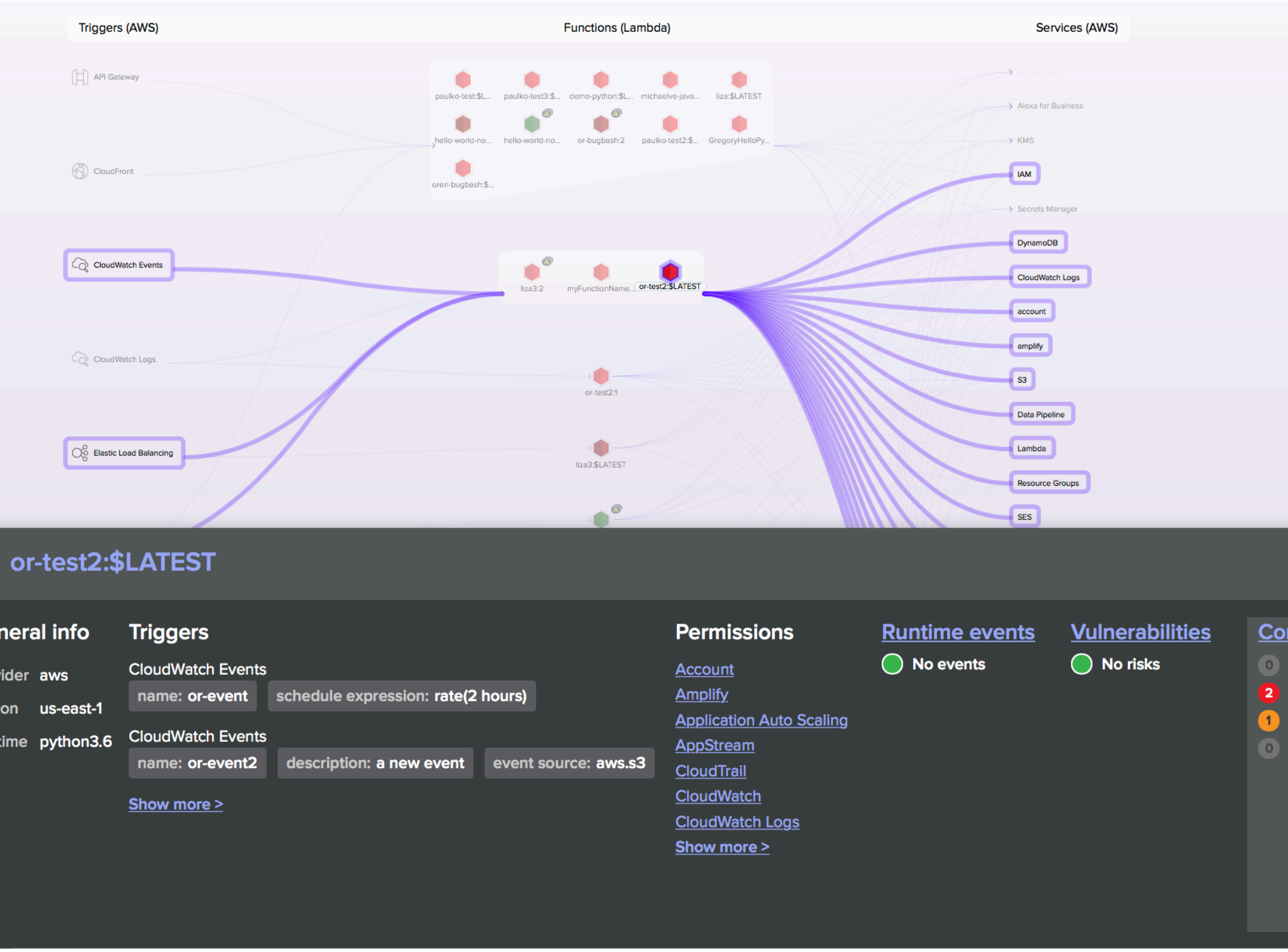
The right-most column shows the services with which each function interfaces. Drilling into the function data reveals the permissions each function has been granted to access those services.

Lines connect triggers to functions to services, letting security teams to visualize the entire connectivity flow and access rights. Clicking on individual functions highlights their interconnects in the radar, and opens a pop-up that lets you drill into the details.

Exploring the data

Prisma Cloud finds, scans, and displays the $LATEST version and all published versions of your functions. Clicking a node in Serverless Radar lets you inspect a function's configuration and explore all the security-related data that Prisma Cloud has indexed about it.

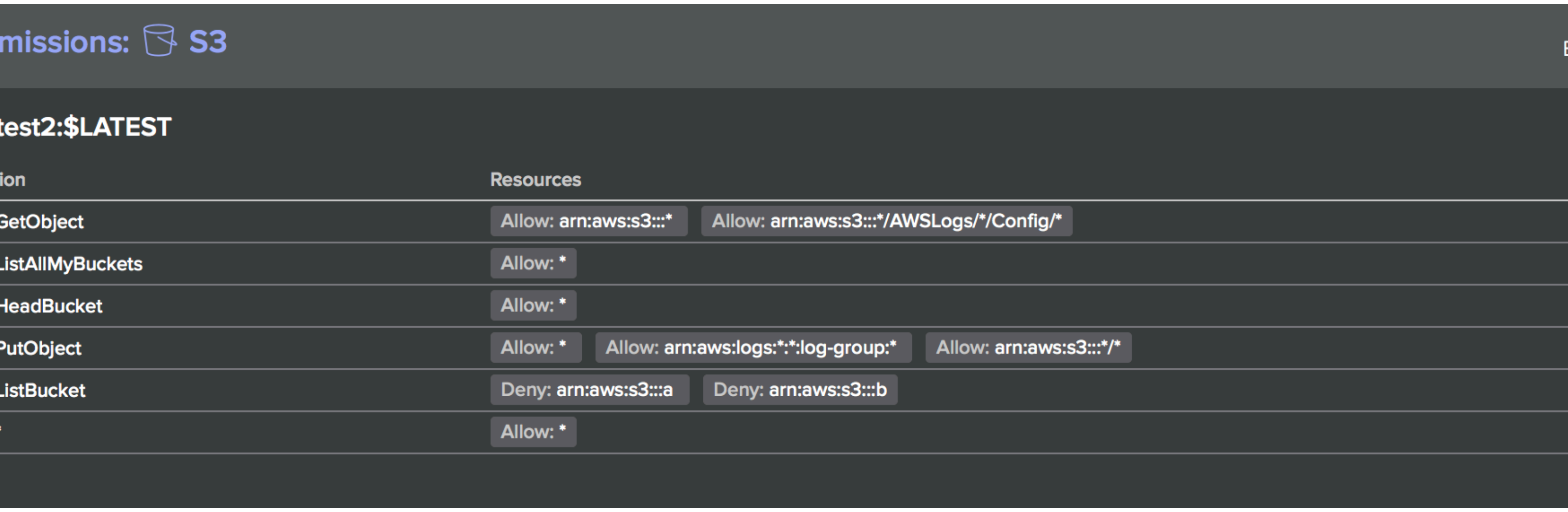
For example, clicking on the or-test2:$LATEST function opens a popup with summary findings. This particular function has two high risk compliance issues. Clicking on the compliance link takes you to a list of compliance issues for the function.



Compliance issue 437 indicates overly permissive access to one or more services. Expanding the issue reveals the reason why this compliance issue was raised, with a list of non-compliant service access configurations. One of the misconfigured access policy is for S3.

Id	Type	Severity	Description
437		high	Overly permissive service access Hide details
	Full description		Function has permission to all actions of one or more services.
	Cause		6 overly permissive services, including: AppStream, CloudWatch Logs, DAX, DynamoDB, S3
439		high	Suspicious function actions Show details
438		medium	Broad resource access Show details

Returning to the first pop-up window, and clicking into the S3 service, you can see that all the actions for the function's execution role are tightly scoped, except for the last one. It allows all actions on all resources, and could easily be an erroneous configuration overlooked when it was pushed into production.

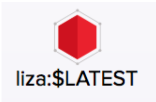


Icons and colors

Nodes are color coded based on the highest severity vulnerability or compliance issue they contain, and reflect the currently defined vulnerability and compliance policies. Color coding lets you quickly spot trouble areas in your deployment. Use the drop-down list at the top of the view to choose how you want nodes colored.

- Maroon -- High risk. One or more critical severity issues detected.

- Red -- High severity issues detected.



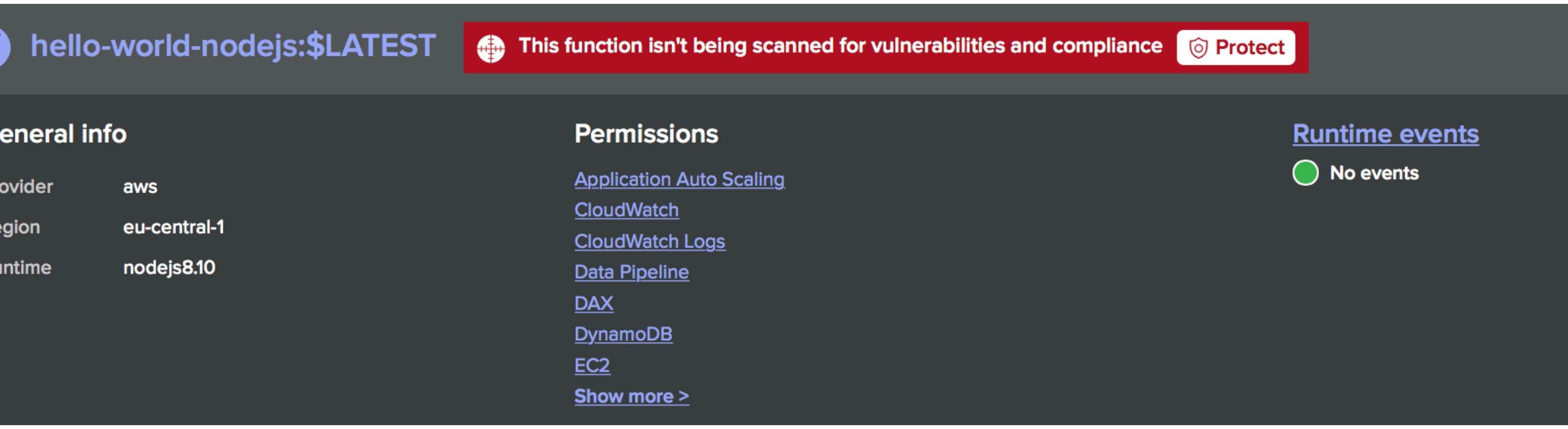
- Orange -- Medium severity issues detected.

- Yellow — Low severity issues detected.
- Green -- Denotes no issues detected.

- Gray — Prisma Cloud hasn't been configured to scan this function for vulnerability and compliance issues.

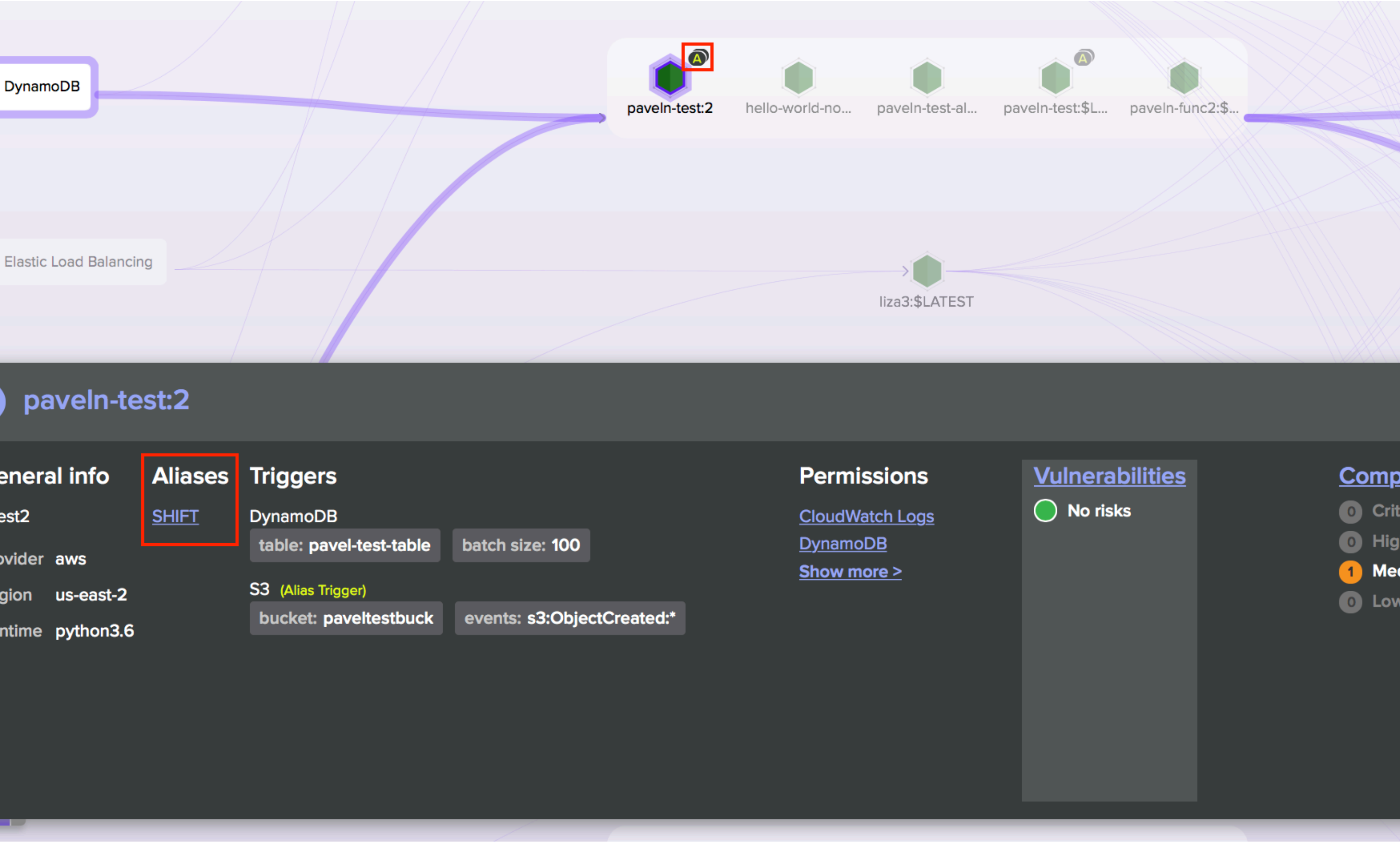
  

  To configure Prisma Cloud to scan the function, click on the node, and then click **Protect** in the pop-up.



- Alias annotation — AWS lets you create aliases to manage the process of promoting new function versions into production. They're conceptually similar to symbolic links in the UNIX file system. Prisma Cloud uses a marker to indicate that an alias points to a specific version of a function.

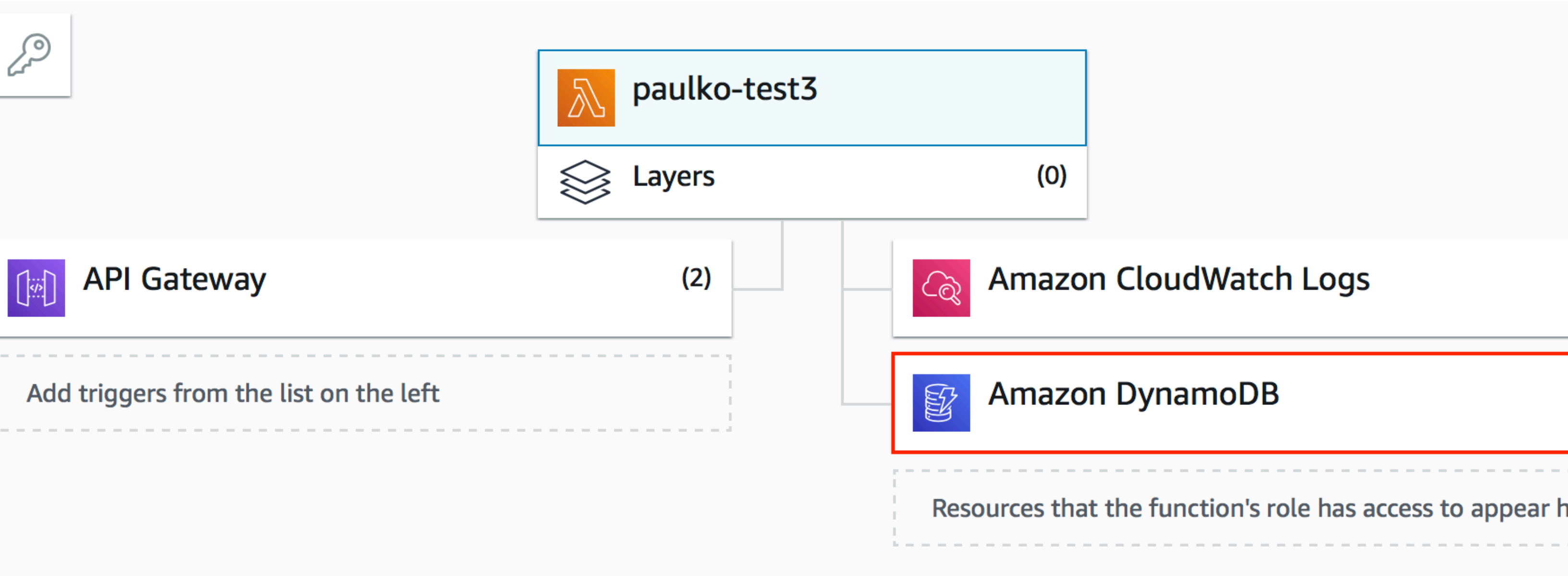
Clicking on the node reveals the aliases that point to the function.



Notes

There can be a discrepancy between what the AWS Lambda designer shows your function can do and its effective permissions when IAM permission boundaries are considered.

For example, if a role is set with permission boundary for DynamoDB, then even though the function's execution role has permission to access DynamoDB, it still might be blocked by the permission boundary. The function designer in AWS's console shows that the function has permission to DyanmoDB, but it might not be accurate.



Setting up Serverless Radar

Serverless Radar uses the AWS APIs to discover and inspect the functions in your environment. Create an IAM user or role for Prisma Cloud, provide the credentials to Console, and then enable Serverless Radar. With this basic setup, Prisma Cloud will show the triggers, services, and permissions for each function.

Prerequisites:

- Prisma Cloud needs an AWS service account to scan your serverless functions. In AWS, you've created an IAM user or role with the following permission policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "PrismaCloudComputeServerlessRadar",
            "Effect": "Allow",
            "Action": [
                "apigateway:GET",
                "cloudfront:ListDistributions",
                "cloudwatch:GetMetricData",
                "cloudwatch:DescribeAlarms",
                "elasticloadbalancing:DescribeListeners",
                "elasticloadbalancing:DescribeRules",
                "elasticloadbalancing:DescribeTargetGroups",

 "elasticloadbalancing:DescribeListenerCertificates",
                "events:ListRules",
                "iam:GetPolicy",
                "iam:GetPolicyVersion",
                "iam:GetRole",
                "iam:GetRolePolicy",
                "iam:ListAttachedRolePolicies",
                "iam:ListRolePolicies",
```



```
                "lambda:GetFunction",
                "lambda:GetPolicy",
                "lambda:ListAliases",
                "lambda:ListEventSourceMappings",
                "lambda:ListFunctions",
                "logs:DescribeSubscriptionFilters",
                "s3:GetBucketNotification",
                "kms:Decrypt"
            ],
            "Resource": "*"
        }
    ]
}
```

STEP 1 | Open Console.

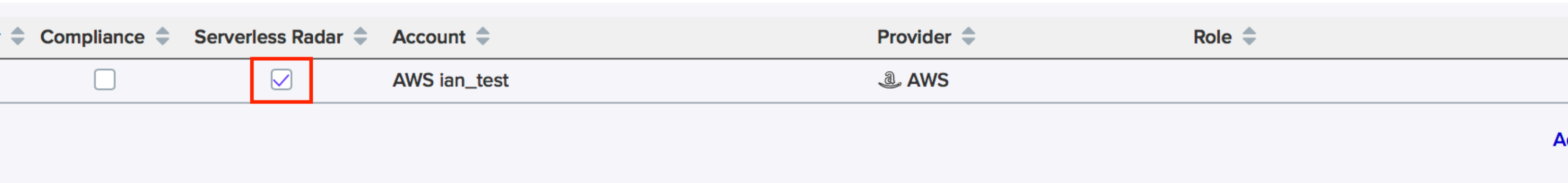
STEP 2 | Go to **Defend > Compliance > Cloud Platforms**.

STEP 3 | Click **Add account**, and configure an AWS account.

STEP 4 | Select the checkbox for the credential.

STEP 5 | Click **Add**.

STEP 6 | For the account just added, select the **Serverless Radar** checkbox.



STEP 7 | Click the yellow save button.

After Prisma Cloud finishes scanning your environment, you should see your functions in Serverless Radar.

What's next?

To see vulnerability and compliance information in Serverless Radar, configure Prisma Cloud to scan the contents of each function.

Prisma Cloud Rules Guide - Docker

Edit on GitHub

This article provides a list of all rules and their intended behavior in Prisma Cloud Console UI. The purpose of this article is to help users better understand the intention of each rule in the Console and it's corresponding effect on the host environment.

Running Docker commands through Defender

To access Docker daemon through Defender, you must explicitly specify Defender's host and port. For example:

```
$ docker -H <DEFENDER_HOST_ADDRESS>:9998 run alpine
```

It is possible to make the management traffic between the Docker client and the Docker daemon flow through Defender by default via two environment variables. Those can be configured on a remote machine that accesses Docker daemon on some host (such as dev laptop), or the host itself for users who do not have root privileges (which should be the majority of users).

```
$ export DOCKER_HOST=tcp://<defender host address>:9998
```

```
$ export DOCKER_TLS_VERIFY=1
```

Once set, default calls to Docker flow through Defender (e.g., docker ps, docker run alpine). Throughout this guide however, in this guide, we have followed the default command without setting environment variables.

About this reference environment

This guide is designed as a reference document for all access rule policies enlisted in Prisma Cloud Console and their intended affect on host environment. These commands are run from a Docker client to a Prisma Cloud Defender using the access control feature. Access control rules can be configured at **Defend > Access > Docker**.

We have organized this document using the same structure as the Prisma Cloud product UI, which follows the structure in the Docker Remote API documentation. Note that there may be minor differences in the structure as the Docker Remote API evolves; this document is currently aligned with the documentation for API v 1.24 and will be updated periodically with new releases.

For understanding purposes all rules are set to deny and their corresponding influence on host environment is recorded.

Defend access rules

Navigate to **Defend > Access > Docker**.

Containers

For more information about the Docker API for containers, see https://docs.docker.com/engine/api/v1.30/#tag/Container.

container_list - List containers

Affects docker ps command on host which is used to list all running containers.

Command:

```
docker -H 10.0.0.1:9998 --tlsverify ps
```

Response:

```
[Prisma Cloud] The command container_list denied for user admin by
 rule Deny
```

container_create - Create a container

Affects docker create command used to create a new container.

Command:

```
docker -H 10.0.0.1:9998 --tlsverify create morello/docker-whale
```

Response:

```
[Prisma Cloud] The command container_create denied for user admin by
 rule Deny
```

container_inspect - Inspect a container

Affects docker inspect command used for returning information about the container.

Command:

```
docker -H 10.0.0.1 --tlsverify  inspect ubuntu_bash2
```

Response:

```
[Prisma Cloud] The command container_inspect denied for user admin by
 rule inspect
```

container_top - List processes running inside a container

Affects docker top command used to display the running processes of a container

Command:

```
docker -H 10.0.0.1:9998 --tlsverify top ubuntu_bash
```

Response:

```
[Prisma Cloud] The command container_top denied for user admin by
 rule Deny
```

container_logs - Get container logs

Affects docker logs command used for returning logs from the container present at the time of execution.

Command:

```
docker -H 10.0.0.1 --tlsverify logs ubuntu_bash2
```

Response:

```
[Prisma Cloud] The command container_logs denied for user admin by
 rule logs
```

container_changes - Inspect changes on a container's filesystem

Affect docker commit command and restricts any changes to the container.

Command:

```
docker -H 10.0.0.1 --tlsverify  commit --change "ENV DEBUG true"
 cc2d57988b aqsa/testimage:version3
```

Response:

```
[Prisma Cloud] The command container_commit denied for user admin by
 rule commit
```

container_export - Export a container

Affects docker export command that exports a container's filesystem as a tar archive

Command:

```
docker -H 10.0.0.1:9998 --tlsverify  export  twistlock_console -o
 saved.tar
```

Response:

```
[Prisma Cloud] The command container_export denied for user admin by
 rule export
```

container_stats - Get container stats based on resource usage

Affects docker stats command on host which returns live data stream for running containers.

Command:

```
docker -H 10.0.0.1 --tlsverify stats  silly_stallman
```

Response:

```
[Prisma Cloud] The command container_stats denied for user admin by
 rule status
```

container_resize - Resize a container

Affects docker logs command used for returning logs from the container present at the time of execution. This related to the size of the window of how output is returned from the container. It is called TTY.

Command:

Response:

container_start - Start a container

Affects docker start command used to start one or more stopped containers

Command:

```
docker -H 10.0.0.1:9998 --tlsverify start ubuntu_bash
```

Response:

```
[Prisma Cloud] The command container_start denied for user admin by
 rule Deny all
```

container_stop - Stop a container

Affects docker stop command used to stop running container

Command:

```
docker -H 10.0.0.1:9998 --tlsverify stop ubuntu_bash
```

Response:

```
[Prisma Cloud] The command container_stop denied for user admin by
 rule Deny
```

container_restart - Restart a container

Affects docker restart command on host, used to restart a container.

Command:

```
docker -H 10.0.0.1:9998 --tlsverify restart ubuntu_bash
```

Response:

```
[Prisma Cloud] The command container_restart denied for user admin by
 rule Deny
```


container_kill - Kill a container

Affects docker kill command used to kill a running container.

Command:

```
docker -H 10.0.0.1:9998 --tlsverify kill ubuntu_bash
```

Response:

```
[Prisma Cloud] The command container_kill denied for user admin by
 rule Deny
```

container_rename - Rename a container

Affects docker rename command on host that is used to rename a container.

Command:

```
docker -H 10.0.0.1:9998 --tlsverify rename ubuntu_bash unbuntu
```

Response:

```
[Prisma Cloud] The command container_rename denied for user admin by
 rule Deny
Error: failed to rename container named ubuntu_bash
```

container_pause - Pause a container

Affects docker pause command on host which is used to pause all processes within one or more containers.

Command:

```
docker -H 10.0.0.1 --tlsverify pause  focused_cori
```

Response:

```
[Prisma Cloud] The command container_pause denied for user admin by
 rule Deny
```

container_unpause - Unpause a container

Affects docker unpause command on host which is used to un-suspend all processes in a container.

Command:

```
docker -H 10.0.0.1 --tlsverify unpause  silly_stallman
```

Response:

```
[Prisma Cloud] The command container_unpause denied for user admin by
 rule unpause
```

container_attach - Attach to a container

Affects docker attach command on host where defender is deployed.

Command:

```
docker -H 10.0.0.1 --tlsverify attach  mycontainer
```

Response:

```
[Prisma Cloud] The command container_attach denied for user admin by
 rule attach persistent connection closed
```

container_attachws - Attach to a container (websocket)

Affects docker attach command on host where defender is deployed. Attach to the container id via websocket. Implements websocket protocol handshake according to RFC 6455

Command:

```
docker -H 10.0.0.1 --tlsverify attach  mycontainer
```

Response:

```
[Prisma Cloud] The command container_attach denied for user admin by
 rule attach persistent connection closed
```

container_wait - Wait a container

Affects docker wait command used to block until a container stops, then print its exit code.

Command:

```
docker -H 10.0.0.1:9998 --tlsverify wait ubuntu_bash
```

Response:

```
[Prisma Cloud] The command container_wait denied for user admin by
 rule Deny
```

container_delete - Remove a container

Affects docker rm command used for deleting a container.

Command:

```
docker -H 10.0.0.1:9998 --tlsverify rm  <container>
```

Response:

```
[Prisma Cloud] The command container_delete denied for user admin by
 rule delete
```


container_archive - Gets an archive of filesystem resource in a container

Get a tar archive of a resource in the filesystem of container id. Affects docker cp command

Command:

```
docker -H 10.0.0.1:9998 --tlsverify cp <container> > latest.tar
```

Response:

```
[Prisma Cloud] The command container_copy denied for user admin by
 rule delete
```

container_extract - Extract an archive of files or folders to a directory in a container

Affects docker export command. Uploads a tar archive to be extracted to a path in the filesystem of container id

Command:

```
docker -H 10.0.0.1:9998 --tlsverify cp <container> > latest.tar
```

Response:

```
[Prisma Cloud] The command container_exec_start denied for user admin
 by rule exec
```

Images

For more information about the Docker API for images, see https://docs.docker.com/engine/api/v1.30/#tag/Image.

image_list - List images

Affects docker images command used to list all images

Command:

```
docker -H 10.0.0.1:9998 --tlsverify images
```

Response:

```
[Prisma Cloud] The command image_list denied for user admin by rule
 Deny
```

image_build - Build image from a Dockerfile

Affects docker build command that is used to build an image from a Dockerfile.

Command:

```
docker -H 172.18.0.1:9998 --tlsverify build -t aqsa/testimage:v2 .
```


Response:

```
[Prisma Cloud] The command image_build denied for user admin by rule
 Default - deny all
```

image_create - Create an image

Affects docker pull command which is used to pull an image

Command:

```
docker -H 10.0.0.1:9998 --tlsverify pull ubuntu:latest
```

Response:

```
[Prisma Cloud] The command image_create denied for user admin by rule
 Deny
```

image_inspect - Inspect an image

Description

Affects docker inspect command used for returning information about the container.

Command:

```
docker -H 10.0.0.1:9998 --tlsverify inspect 28e7d49f8e6d
```

Response:

```
[Prisma Cloud] The command image_inspect denied for user admin by
 rule images
```

image_history - Get the history of an image

Affects docker history <image> command.

Command:

```
docker -H 172.18.0.1:9998 --tlsverify history twistlock
```

Response:

```
[Prisma Cloud] The command image_history denied for user admin by
 rule Default - deny all
```

image_push - Push an image on the registry

Affects command docker push for pushing an image to repository

Command:

```
docker -H 10.0.0.1:9998 --tlsverify push ubuntu:latest
```

Response:

```
[Prisma Cloud] The command image_push denied for user admin by rule
 Deny
```

image_tag - Tag an image into a repository

Affects docker tag command used to tag an image in the repository

Command:

```
docker -H 10.0.0.1:9998 --tlsverify tag ubuntu:latest aqsa:tag
```

Response:

```
[Prisma Cloud] The command image_tag denied for user admin by rule
 Deny
```

image_delete - Remove an image

Affects docker rmi command used to delete an image

Command:

```
docker -H 10.0.0.1:9998 --tlsverify  rmi aqsa/testimage:version3
```

Response:

```
[Prisma Cloud] The command image_delete denied for user admin by rule
 Deny
```

images_search - Search images

Affects docker search command which gives a list of available images matching the search item.

Command:

```
docker -H 10.0.0.1:9998 --tlsverify search twistlock
```

Response:

```
[Prisma Cloud] The command images_search denied for user admin by
 rule deny
```

MISC

Misc other docker commands.

docker_check_auth - Check auth configuration

Validates credentials for a registry and get identity token, if available, for accessing the registry without password. Affects docker login on the host.

Command:

```
docker -H 172.18.0.1:9998 --tlsverify login
```

Response:

```
[Prisma Cloud] The command docker_info denied for user admin by rule
 Default - deny all
```

docker_info - Display system-wide information

Affects docker info command used to display system-wide information

Command:

```
docker -H 10.0.0.1:9998 --tlsverify info
```

Response:

```
[Prisma Cloud] The command docker_info denied for user admin by rule
 Deny
```

docker_version - Show the docker version information

Affects docker version command on host which is used to find docker version.

Command:

```
docker -H 10.0.0.1 --tlsverify version
```

Response:

```
[Prisma Cloud] The command docker_version denied for user admin by
 rule version
```

docker_ping - Ping the docker server

The goal of this api is to ping the Docker server and make sure it is up and running.

Command:

It is intended to be called by an external monitoring system. It does not have a direct docker CLI command.

container_commit - Create a new image from a container's changes

Affects docker commit command used for committing container's file changes etc into a new image.

Command:

```
docker -H 10.0.0.1 --tlsverify  commit --change "ENV DEBUG true"
 cc2d57988b aqsa/testimage:version3
```

Response:

```
[Prisma Cloud] The command container_commit denied for user admin by
 rule commit
```

docker_events - Monitor docker's events

Affects docker events command on host which is used to return real time events from the server.

Command:

```
docker -H 10.0.0.1 --tlsverify events
```

Response:

```
[Prisma Cloud] The command docker_events denied for user admin by
 rule events
```

images_archive - Get a tarball containing all images

Affects docker save command to save images to a tar archive

Command:

```
docker -H 172.17.0.1:9998 --tlsverify save $(docker images -q) -o
 home/aqsa/mydockersimages.tar
```

Response:

```
[Prisma Cloud] The command images_archive denied for user admin by
 rule Default - deny all
```

images_load - Load a tarball with a set of images and tags into docker

Affects docker load command to load an image from a tar archive or STDIN

Command:

```
docker -H 172.17.0.1:9998 --tlsverify load -i /home/aqsa/
twistlock_1_6_81.tar.gz
```

Response: [Prisma Cloud] The command images_load denied for user admin by rule Default - deny all

container_exec_create - Exec Create

Affects docker_exec command to create any new container.

Command:

```
docker -H 10.0.0.1 --tlsverify   exec -d ubuntu_bash2 touch /tmp/
execWorks
```


Response:

```
[Prisma Cloud] The command container_exec_start denied for user admin
 by rule exec
```

container_exec_start - Exec Start

Affects docker exec command used for running a command in a running container.

Command:

```
docker -H 10.0.0.1 --tlsverify   exec -d ubuntu_bash2 touch /tmp/
execWorks
```

Response:

```
[Prisma Cloud] The command container_exec_start denied for user admin
 by rule exec
```

container_exec_inspect - Exec Inspect

Affects docker exec command used for running a command in a running container.

Command:

```
docker -H 10.0.0.1 --tlsverify   exec -d ubuntu_bash2 touch /tmp/
execWorks
```

Response:

```
[Prisma Cloud] The command container_exec_start denied for user admin
 by rule exec
```

container_archive_head

Command:

```
docker -H 10.0.0.1 --tlsverify unpause  silly_stallman
```

Response:

```
[Prisma Cloud] The command container_unpause denied for user admin by
 rule unpause
```

container_copyfiles

Affects docker cp command used to copy files from and to containers and local file system on host.

Command:

```
docker -H 10.0.0.1 --tlsverify cp file  mycontainer:~
```

Response:

```
[Prisma Cloud] The command container_copyfiles denied for user admin
 by rule unpause
```

Volumes

For more information about the Docker API for volumes, see https://docs.docker.com/engine/api/v1.30/#tag/Volume.

volume_list - List volumes

Affects docker volume ls command to list all volumes

Command:

```
docker -H 10.0.0.1:9998 --tlsverify volume ls
```

Response:

```
[Prisma Cloud] The command volume_list denied for user admin by rule
 Deny
```

volume_create - Create a volume

Affects docker volume create command to create a volume

Command:

```
docker -H 10.0.0.1:9998 --tlsverify volume create
```

Response:

```
[Prisma Cloud] The command volume_create denied for user admin by
 rule Deny
```

volume_inspect - Inspect a volume

Affects docker volume inspect command to display detailed information on one or more volumes

Command:

```
docker -H 10.0.0.1:9998 --tlsverify volume inspect f1c7
```

Response:

```
[Prisma Cloud] The command volume_inspect denied for user admin by
 rule Deny
```

volume_remove - Remove a volume

Affects docker volume rm command to remove one or more volumes

Command:

```
docker -H 10.0.0.1:9998 --tlsverify volume rm f671
```

Response:

```
[Prisma Cloud] The command volume_remove denied for user admin by
 rule Deny
```

Networks

For information about the Docker API for networks, see https://docs.docker.com/engine/api/v1.30/#tag/Network.

network_list - list networks

Affects docker network ls to list networks

Command:

```
docker -H 172.17.0.1:9998 --tlsverify network ls
```

Response:

```
[Prisma Cloud] The command network_list denied for user admin by rule
 Default - deny all
```

network_inspect - Inspect network

Affects docker network inspect to display detailed information on one or more networks

Command:

```
docker -H 172.17.0.1:9998 --tlsverify network inspect 82b1c
```

Response:

```
[Prisma Cloud] The command network_inspect denied for user admin by
 rule Default - deny all
```

network_create - Create a network

Affects docker network create to create a network

Command:

```
docker -H 172.17.0.1:9998 --tlsverify network create new-network
```

Response:

```
[Prisma Cloud] The command network_create denied for user admin by
 rule Default - deny all
```


network_connect - Connect a container to a network

Affects docker network connect to connect a container to a network

Command:

```
docker -H 172.17.0.1:9998 --tlsverify network connect new-network
 container1
```

Response:

```
[Prisma Cloud] The command network_connect denied for user admin by
 rule Default - deny all
```

network_disconnect - Disconnect a container from a network

Affects docker network disconnect to disconnect a container from a network

Command:

```
docker -H 172.17.0.1:9998 --tlsverify network disconnect new-network
 container1
```

Response:

```
[Prisma Cloud] The command network_disconnect denied for user admin
 by rule Default - deny all
```

network_remove - Remove a network

Affects docker network rm to remove one or more networks

Command:

```
docker -H 172.17.0.1:9998 --tlsverify network rm new-network
```

Response:

```
[Prisma Cloud] The command network_remove denied for user admin by
 rule Default - deny all
```

Secrets

Secrets are added in Prisma Cloud 2.0 in accordance with Docker Engine API v1.26.

For more information about the Docker API for secrets, see https://docs.docker.com/engine/api/v1.30/#tag/Secret.

secret_list - List secrets

Affects docker secret ls command used to list secrets.

Command:

```
docker -H 10.0.0.1:9998 --tlsverify secret ls
```


Response:

```
[Prisma Cloud] The command secret_ls denied for user admin by rule
 Default - deny all
```

secret_create - Create secrets

Affects docker secret create command used to create secrets.

Command:

```
docker -H 10.0.0.1:9998 --tlsverify secret create my-secret ./
aqsa.json
```

Response:

```
[Prisma Cloud] The command secret_create denied for user admin by
 rule Default - deny all
```

secret_inspect - Inspect secrets

Affects docker secret inspect command used to inspect secrets.

Command:

```
docker -H 10.0.0.1:9998 --tlsverify secret inspect <id>
```

Response:

```
[Prisma Cloud] The command secret_inspect denied for user admin by
 rule Default - deny all
```

secret_remove - Delete secrets

Affects docker secret rm command used to remove one or more secrets.

Command:

```
docker -H 10.0.0.1:9998 --tlsverify secret rm aqsa.json
```

Response:

```
[Prisma Cloud] The command secret_rm denied for user admin by rule
 Default - deny all
```

secret_update - Update a secret

Affects POST /secrets/{id}/update command used to remove one or more secrets.

Command:

Response:

Defender architecture

Edit on GitHub

Customers often ask how Prisma Cloud Defender really works under the covers. Prisma Cloud leverages Docker's ability to grant advanced kernel capabilities to enable Defender to protect your whole stack, while being completely containerized and utilizing a least privilege security design.

Defender design

Because we've built Prisma Cloud expressly for cloud native stacks, the architecture of our agent (what we call Defender) is quite different. Rather than having to install a kernel module, or modify the host OS at all, Defender instead runs as a Docker container and takes only those specific system privileges required for it to perform its job. It does not run as --privileged and instead takes the specific system capabilities of net_admin, sys_admin, sys_ptrace, mknod, and setfcap that it needs to run in the host namespace and interact with both it and other containers running on the system. You can see this clearly by inspecting the Defender container:

```
# docker inspect twistlock_defender_<VERSION> | grep -e CapAdd -A 7 -
e Priv
            "CapAdd": [
                "NET_ADMIN",
                "SYS_ADMIN",
                "SYS_PTRACE",
                "MKNOD",
                "SETFCAP"
            ],
--
            "Privileged": false,
```

This architecture allows Defender to have a near real time view of the activity occurring at the kernel level. Because we also have detailed knowledge of the operations of each container, we can correlate the kernel data with the container data to get a comprehensive view of process, file system, network, and system call activity from the kernel and all the containers running on it. This access also allows us to take preventative actions like stopping compromised containers and blocking anomalous processes and file system writes.

Critically, though, Defender runs as a user mode process. If Defender were to fail (and if that were to happen, it would be restarted immediately), there would be no impact on the containers on the host, nor the host kernel itself. Additionally, we can and do apply *cgroups* to set hard limits on CPU and memory consumption, guaranteeing it will be a 'good neighbor' on the host and not interfere with host performance or stability.

In the event of a communications failure with Console, Defender continues running and enforcing the active policy that was last pushed by the management point. Events that would be pushed back to Console are cached locally until it is once again reachable.

Why not a kernel module?

Given the broad range of security protection Prisma Cloud provides, not just for containers, but also for the hosts they run on, you might assume that we use a kernel module - with all the associated baggage that goes along with that. However, that's not actually how Prisma Cloud works.

Kernel modules are compiled software components that can be inserted into the kernel at runtime and typically provide enhanced capabilities for low level functionality like process scheduling or file monitoring. Because they run as part of the kernel, these components are very powerful and privileged. This allows them to perform a wide range of functions but also greatly increases the operational and security risks on a given system. The kernel itself is extensively tested across broad use cases, while these modules are often created by individual companies with far fewer resources and far more narrow test coverage.

Because kernel modules have unrestricted system access, a security flaw in them is a system wide exposure. A single unchecked buffer or other error in such a low level component can lead to the complete compromise of an otherwise well designed and hardened system. Further, kernel modules can introduce significant stability risks to a system. Again, because of their wide access, a poorly performing kernel module that's frequently called can drag down performance of the entire host, consume excessive resources, and lead to kernel panics. For these reasons, many modern operating systems designed for cloud native apps, like Google Container-Optimized OS, explicitly prevent the usage of kernel modules.

Defender-Console communication

By default, Defender connects to Console with a websocket on TCP port 8084. This port number can be customized to meet the needs of your environment. All traffic between Defender and Console is TLS encrypted.

Defender has no privileged access to Console or the underlying host where Console is installed. By design, Console and Defender don't trust each other and Defender mutual certificate-based authentication is required to connect. For more information about the Console-Defender communication certificates, see the certificates article. Pre-auth, connections are blocked. Post-auth, Defender's capabilities are limited to getting policies from Console and sending event data to Console.

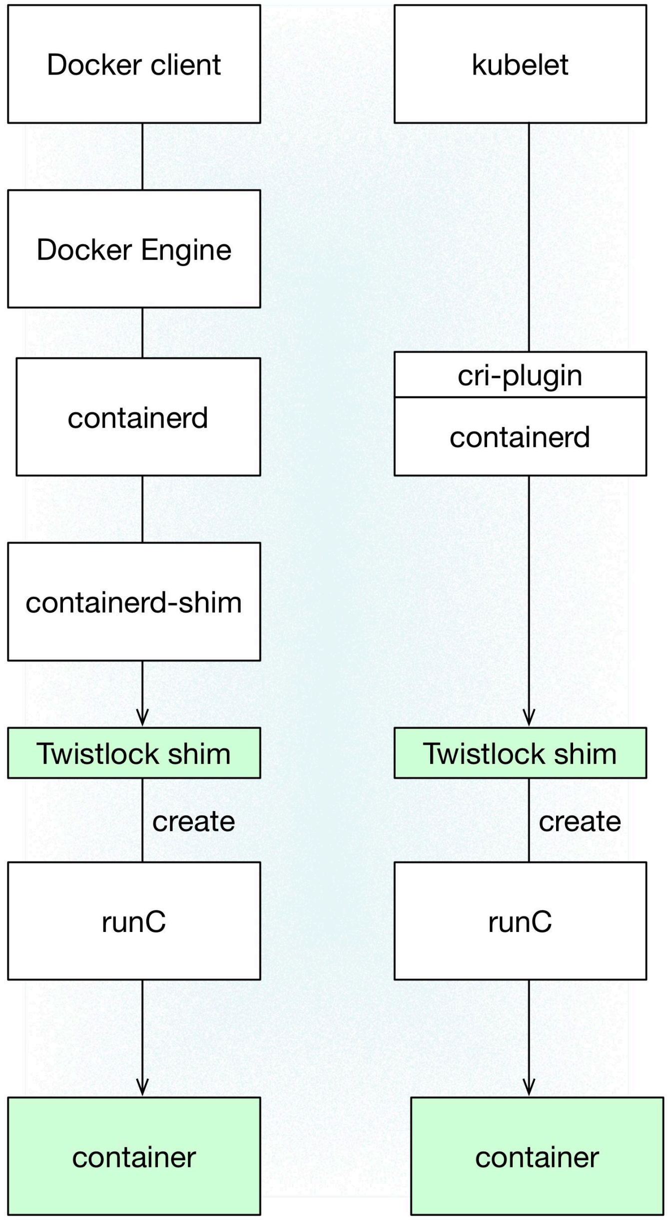
If Defender were to be compromised, the risk would be local to the system where it is deployed, the privilege it has on the local system, and the possibility of it sending garbage data to Console. Console communication channels are separated, with no ability to jump channels.

Defender has no ability to interact with Console beyond the websocket. Both Console's API and web interfaces, served on port 8083 (HTTPS), require authentication over a different channel with different credentials (e.g. username and password, access key, and so on), none of which Defender holds.

Blocking rules

Defender is responsible for enforcing vulnerability and compliance blocking rules. When a blocking rule is created, Defender moves the original runC binary to a new path and inserts a Prisma Cloud runC shim binary in its place.

When a command to create a container is issued, it propagates down the layers of the container orchestration stack, eventually terminating at runC. Regardless of your environment (Docker, Kubernetes, or OpenShift, etc) and underlying CRI provider, runC does the actual work of instantiating a container.



When starting a container in a Prisma Cloud-protected environment:

1. The Prisma Cloud runC shim binary intercepts calls to the runC binary.
2. The shim binary calls the Defender container to determine whether the new container should be created based on the installed policy.
    - If Defender replies affirmatively, the shim calls the original runC binary to create the container, and then exits.
    - If Defender replies negatively, the shim terminates the request.
    - If Defender does not reply within 60 seconds, the shim calls the original runC binary to create the container and then exits.

The last step guarantees that Defender always fails open, which is important for the resiliency of your environment. Even if the Defender process terminates, becomes unresponsive, or cannot be restarted, a failed Defender will not hinder deployments or the normal operation of a node.

Firewalls

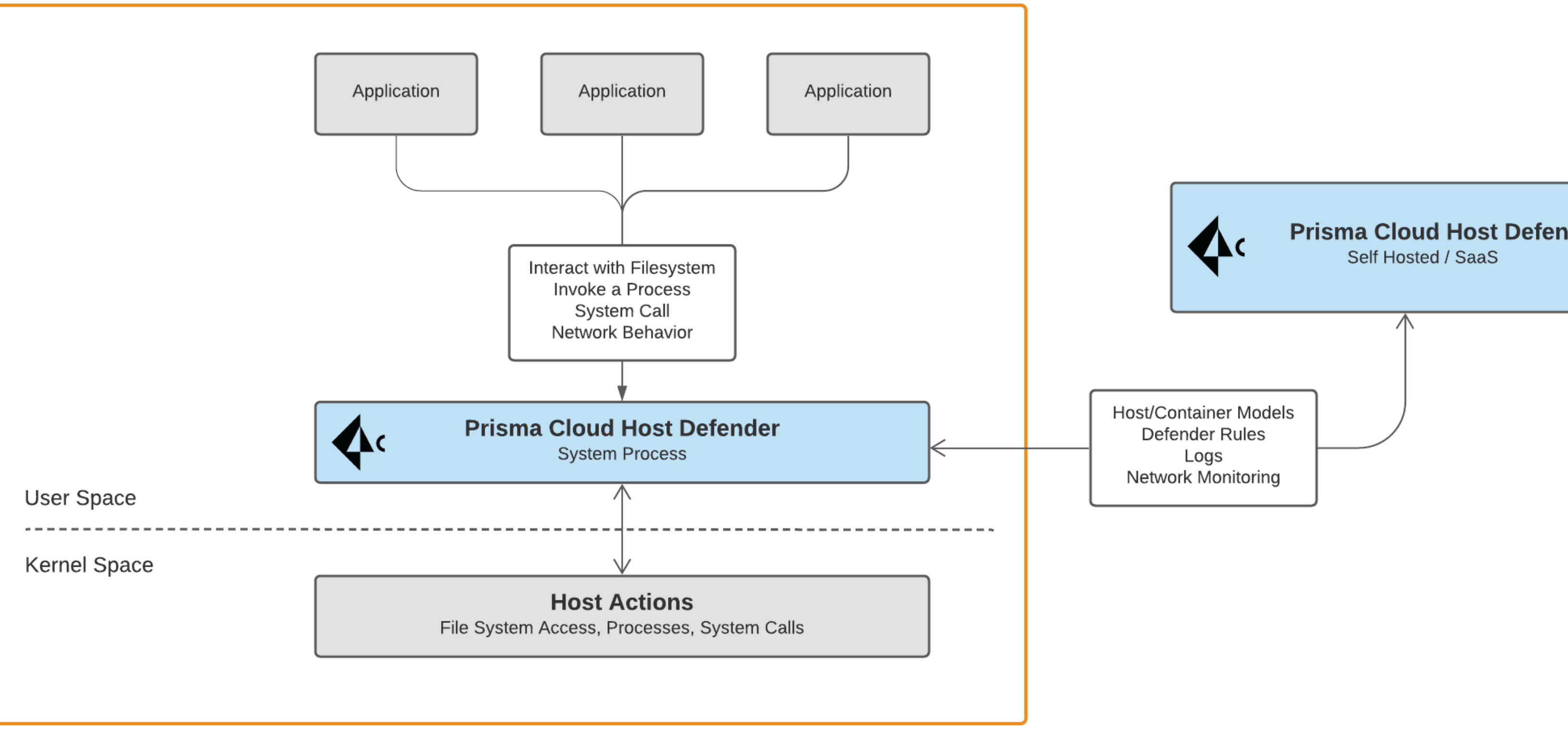
Defender enforces WAF policies (WAAS), and monitors and enforces layer 4 traffic (CNNS). In both cases, Defender creates iptables rules on the host so it can observe network traffic. For more information, see CNNS architecture and WAAS architecture.

Host Defender architecture

Edit on GitHub

Because we've built Prisma Cloud expressly for cloud native stacks, the architecture of our agent (what we call Defender) is quite different. Rather than having to install a kernel module, or modify the host OS at all, Defender instead runs as a systemd or system module (not a kernel module) in user space, intercepting every syscall interaction and file changes, and actively blocking or alerting according to the rules defined in Console.

Host Defender Architecture Diagram



TLS v1.2 cipher suites

Edit on GitHub

Prisma Cloud Compute uses the Go programming language cryptographic libraries to protect all network communications via the Transport Layer Security (TLS) v1.2 protocol.

Prisma Cloud Compute Self-Hosted

The User Interface (UI) and API access is protected using server side TLS v1.2 authentication. The cipher suites offered by the Console adhere to NIST SP800-52r2 guidance.

- TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
- TLS_RSA_WITH_AES_256_GCM_SHA384
- TLS_RSA_WITH_AES_256_CBC_SHA
- TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA

The Console enforces HTTP Strict Transport Security (HSTS).

Validating Console's UI and API TLS cipher suites

Use nmap to confirm the cipher suites supported by the Console.

STEP 1 | Install nmap

STEP 2 | Call the Console's UI/API endpoint (default TCP port 8083) to enumerate the ciphers suites supported by the Console.

```
$ nmap -sV --script ssl-enum-ciphers -p 8083 172.17.0.2
```

The following is an abbreviated return from the nmap command:

```
Starting Nmap 7.60 ( https://nmap.org ) at 2022-02-16 21:32 UTC
Nmap scan report for 172.17.0.2
Host is up (0.000046s latency).

PORT     STATE SERVICE     VERSION
8083/tcp open  ssl/us-srv?
| fingerprint-strings:

..
..
..

| ssl-enum-ciphers:
|   TLSv1.2:
|     ciphers:
|       TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (secp256r1) - A
|       TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (secp256r1) - A
|       TLS_RSA_WITH_AES_256_GCM_SHA384 (rsa 2048) - A
```

```
|       TLS_RSA_WITH_AES_256_CBC_SHA (rsa 2048) - A
|     compressors:
|       NULL
|     cipher preference: server
|_  least strength: A
```

Console to Defender communication

The Console and Defenders communication using a mutual TLS v1.2 web-socket session (default TCP 8084). The allowed cypher suites used for this communication adhere to NIST SP800-52r2 guidance.

- TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
- TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
- TLS_RSA_WITH_AES_256_GCM_SHA384
- TLS_RSA_WITH_AES_256_CBC_SHA

The certificate/keys used for the Console to Defender mutual TLS v1.2 web socket session are generated during the Console's initiation process. The reason for this is to ensure the Console is only communicating with the Defenders it has deployed and the Defenders only communicate with its controlling Console.

Validating Console to Defender communication

Use nmap to confirm the cipher suites supported by the Console.

STEP 1 | Install nmap

STEP 2 | Call the Console's Defender communications endpoint (default TCP port 8084) to enumerate the ciphers suites supported by the Console for Defender communications.

```
$ nmap -sV --script ssl-enum-ciphers -p 8084 172.17.0.2
```

Following is a return from the nmap command

```
Starting Nmap 7.60 ( https://nmap.org ) at 2022-02-16 22:30 UTC
Nmap scan report for 172.17.0.2
Host is up (0.000048s latency).

PORT     STATE SERVICE     VERSION
8084/tcp open  ssl/unknown
| ssl-enum-ciphers:
|   TLSv1.2:
|     ciphers:
|       TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (secp256r1) - A
|       TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (secp256r1) - A
|       TLS_RSA_WITH_AES_256_GCM_SHA384 (rsa 2048) - A
|       TLS_RSA_WITH_AES_256_CBC_SHA (rsa 2048) - A
|     compressors:
|       NULL
|     cipher preference: server
|_  least strength: A
```


Prisma Cloud Compute Enterprise (SaaS)

The Compute Console runs within a Google Cloud Platform's Google Kubernetes Engine (GKE). The GKE Console containers are fronted by GCP Load Balancers that have been configured to use the Modern pre-configured profile. The Modern profile supports a wide set of SSL features, allowing modern clients to negotiate SSL.

Validating Console's UI and API TLS cipher suites (SaaS)

Use nmap to confirm the cipher suites supported by the Console.

STEP 1 | Install nmap

STEP 2 | Call the Console's UI/API endpoint (default TCP port 443) to enumerate the ciphers suites supported by the Console. The Console's URL can be found within the Compute Module's **Manage > System > Utilities > Path to Console**.

```
$ nmap -sV --script ssl-enum-ciphers -p 443 us-
west1.cloud.twistlock.com
```

The following is an abbreviated return from the nmap command:

```
Starting Nmap 7.70 ( https://nmap.org ) at 2022-02-28 22:44 UTC
Nmap scan report for us-west1.cloud.twistlock.com (34.82.51.12)
Host is up (0.073s latency).

PORT    STATE SERVICE  VERSION
443/tcp open  ssl/http nginx 1.17.10
|_http-server-header: nginx/1.17.10
| ssl-enum-ciphers:
|   TLSv1.2:
|     ciphers:
|       TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (ecdh_x25519) - A
|       TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (ecdh_x25519) -
 A
|       TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (ecdh_x25519) - A
|     compressors:
|       NULL
|     cipher preference: server
|_  least strength: A
```

Console to Defender communication (SaaS)

The SaaS Console and Defender communication uses the same TCP port (i.e. 443) and cipher suites as the UI/API endpoint.

Credential store's secrets storage

Local username/password accounts within a local database table using HMAC256. This is in compliance with NIST SP800-107r1. Currently, there are seven approved hash algorithms specified in FIPS 180-4: SHA-1, SHA-224, **SHA-256**, SHA-384 SHA-512, SHA-512/224 and SHA-512/256.

Industrial guidance

NIST SP800-52r2

NIST Special Pulication 800-52r2 provides guidance to the selection and configuration of TLS protocol implementations while making effective use of Federal Information Processing Standards (FIPS) and NIST-recommended cryptographic algorithms. Prisma Cloud Compute's cipher suites adhere to SP800-52r2 guidance.

NIST SP800-52r2 approved suites	Compute cipher suites
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
TLS_RSA_WITH_AES_256_GCM_SHA384	TLS_RSA_WITH_AES_256_GCM_SHA384
TLS_RSA_WITH_AES_256_CBC_SHA	TLS_RSA_WITH_AES_256_CBC_SHA
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA

NSA approved

The NSA's Commercial National Security Algorithm Suite provides cryptographic guidance for replacement of Suite B algorithms prior to the availability of quantum resistant algorithms. "For those customers who are looking for mitigations to perform while the new algorithm suite is developed and implemented into products, there are several things they can do." These recommendations have been implemented within Prisma Cloud's cryptographic settings.

Function	NSA recommendation	Compute cipher	Guidance
Key establishment	RSA - 3072 key ECDHE - Curve P-384	**TLS_ECDHE_RSA**_WITH_AES_256_GCM_SHA384 **TLS_ECDHE_RSA**_WITH_AES_256_CBC_SHA **TLS_RSA**_WITH_AES_256_GCM_SHA384 **TLS_RSA**_WITH_AES_256_CBC_SHA **TLS_ECDHE_ECDSA**_WITH_AES_256_GCM_SHA384 **TLS_ECDHE_ECDSA**_WITH_AES_256_CBC_SHA	Generate the appropriate certificate and key size for Console TLS UI and API endpoint
Symmetric block cipher used for information protection	AES 256	TLS_ECDHE_RSA_WITH_**AES_256_GCM_SHA384** TLS_ECDHE_RSA_WITH_**AES_256_CBC_SHA** TLS_RSA_WITH_**AES_256_GCM_SHA384** TLS_RSA_WITH_**AES_256_CBC_SHA** TLS_ECDHE_ECDSA_WITH_**AES_256_GCM_SHA384** TLS_ECDHE_ECDSA_WITH_**AES_256_CBC_SHA**	All cipher suites enforce the AES 256 block cipher. Both GCM and CBC modes are supported.

Telemetry

Edit on GitHub

To drive product improvements, Prisma Cloud captures anonymous data about how our product is used. By default, telemetry is enabled.

No information that is collected can be used to uniquely identify you or your deployment.

Telemetry can be disabled any time.

We never collect IP addresses, host names, user names, container labels, or image tags.

Our telemetry is designed to help us understand how customers use our product at an aggregate level. For example, we detect which features are enabled, the number of containers and images in an environment, and so on.

The legal terms governing telemetry can be found in the Prisma Cloud License Agreement. The Prisma Cloud License Agreement is included with the installation package.

Disabling telemetry

Disable telemetry in Console's settings page.

STEP 1 | Open Console.

STEP 2 | Go to **Manage** > **System** > **General**.

STEP 3 | Under **Telemetry**, set **Share telemetry on product usage with Palo Alto Networks** to **Off**.

Configure

Edit on GitHub

After installing Prisma Cloud, configure it to meet your operational and security requirements.

- Rule ordering and pattern matching
- Backup and restore
- Custom feeds
- Configuring Prisma Cloud proxy settings
- Prisma Cloud Compute certificates
- Configure Agentless Scanning
- Agentless Scanning Modes
- Configure scanning
- User certificate validity period
- Enable HTTP access to Console
- Set different paths for Defender and Console (with DaemonSets)
- Authenticate to Console with certificates
- Configure custom certs from a predefined directory
- Customize terminal output
- Collections
- Tags
- Logon settings
- Reconfigure Prisma Cloud
- Subject Alternative Names
- WildFire Settings
- Log Scrubbing
- Clustered-DB
- Permissions by feature

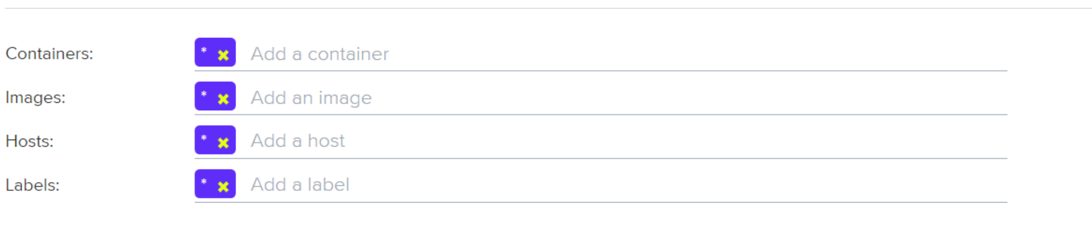
Rule ordering and pattern matching

Edit on GitHub

Prisma Cloud supports pattern matching so that rules can be applied granularly. For example, you could apply a rule to any image with the name *ubuntu**. Or you could apply a rule to all hosts, except those named **test**. This article describes how filtering and pattern matching works in Prisma Cloud.

Pattern matching

All rules have resource filters that let you precisely target specific parts of your environment. This is known as a rule's *scope*. Scope is specified using collections. Rules reference collections to set their scope.

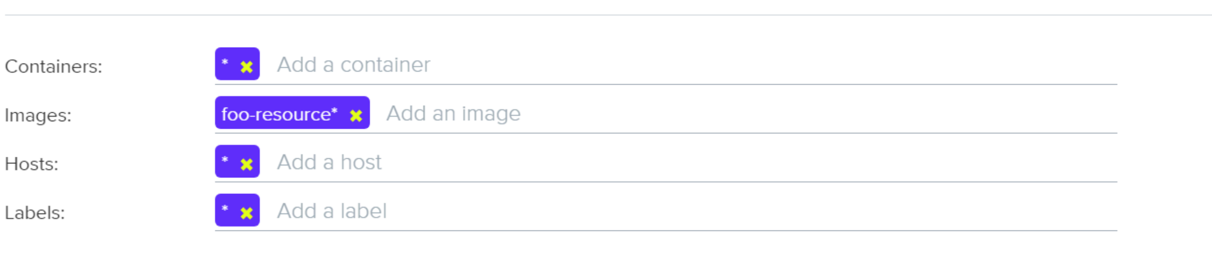


The fields in a collection let you capture a segment of the resources in your environment based on container name, image name, host name, etc. By default, each field is populated with a wildcard. Wildcards capture all objects of a given type. Constrain the scope of a collection by specifying filters in one or more field.

You can customize how a field is evaluated with string matching. When Prisma Cloud encounters a wildcard in a resource name, it evaluates the resource name according to the position of the wildcard.

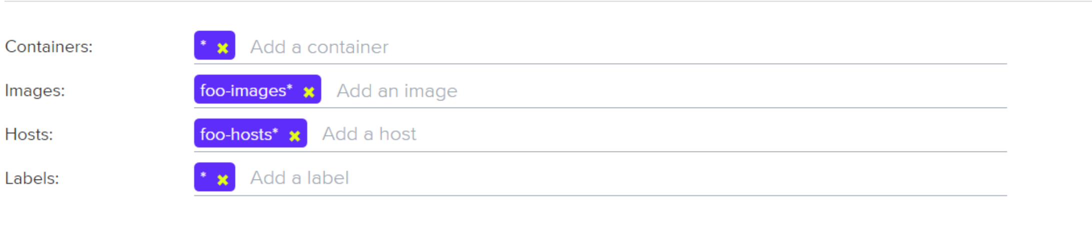
- If the string starts with a wildcard, it's evaluated as *string-ends-with*.
- If the string terminates with a wildcard, it's evaluated as *string-starts-with*.
- If a string is starts and terminates with a wildcard, it's evaluated as *string-contains*.

For example, if you specify a resource filter of **foo-resource**, Prisma Cloud matches that resource to any value that contains the string, such as *example-foo-resource* and *foo-resource-1*. Matching logic is case insensitive.



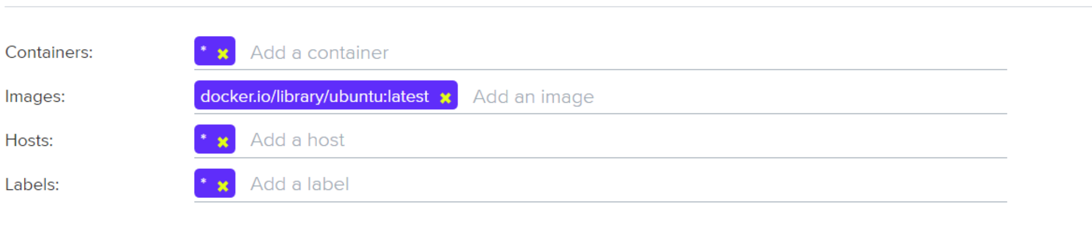
Individual fields are combined using **AND** logic. In the following example, there are filters for hosts named *foo-hosts** and images named *foo-images**. There are no filters for containers or labels (they're wildcards). If this collection were used to scope a rule, the effective result would be to

apply this rule anytime the host name starts with foo-hosts and image name starts with foo-images, regardless of the container name or label.



If strings have no wildcards, Prisma Cloud exactly matches the value you enter against the resource string. This gives you precise control over which values match. For example:

- **/ubuntu:latest* matches */library/ubuntu:latest* or *docker.io/library/ubuntu:latest*.
- **:latest* matches *ubuntu:latest* or *debian:latest*.
- If you want to explicitly target just *ubuntu:latest* from Docker Hub, use *docker.io/library/ubuntu:latest*. Because the value you provide is the complete name of the resource, Prisma Cloud matches it exactly.
- **_test* matches *host_sandbox_test* and *host_preprod_test* but doesn't match *host_test_server*.

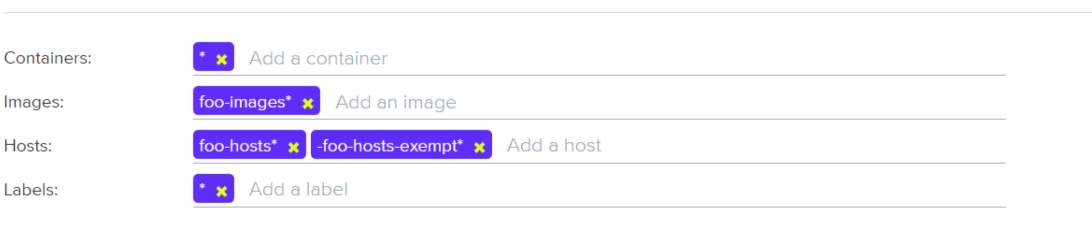


For DNS filtering, Prisma Cloud doesn't prevent you from entering multiple wildcards per string, but it's treated the same as if you simply entered the right-most wildcard. The following patterns are equivalent:

```
*.*.b.a == *.b.a
```

Exemptions

While basic string matching makes it easy to manage rules for most scenarios, you sometimes need more sophisticated logic. Prisma Cloud lets you exempt objects from a rule with the minus (-) sign (the NOT operator). From example, if you want a rule to apply to all hosts starting with *foo-hosts**, except those starting with *foo-hosts-exempt**, then you could create the following rule:



When Prisma Cloud evaluates an object against a rule with a NOT operator, it first skips any object for which there is a match with the exempted object. So, from our example:

1. If the host name starts with *foo-hosts-exempt*, skip the rule.
2. If the host name starts with *foo-hosts* AND the image name starts with *foo-images*, apply the rule.

All scope fields, in both policy rules and collection specs, support the NOT operator.

When using the NOT operator, remember that what's being excluded can't be broader than what's included. For example, the following expression for scoping images is illogical:

```
-ngnix*, ngnix:latest
```

The following expression, however, is valid. It sets the scope to all NGINX images, and then excludes *nginx:latest* from the set.

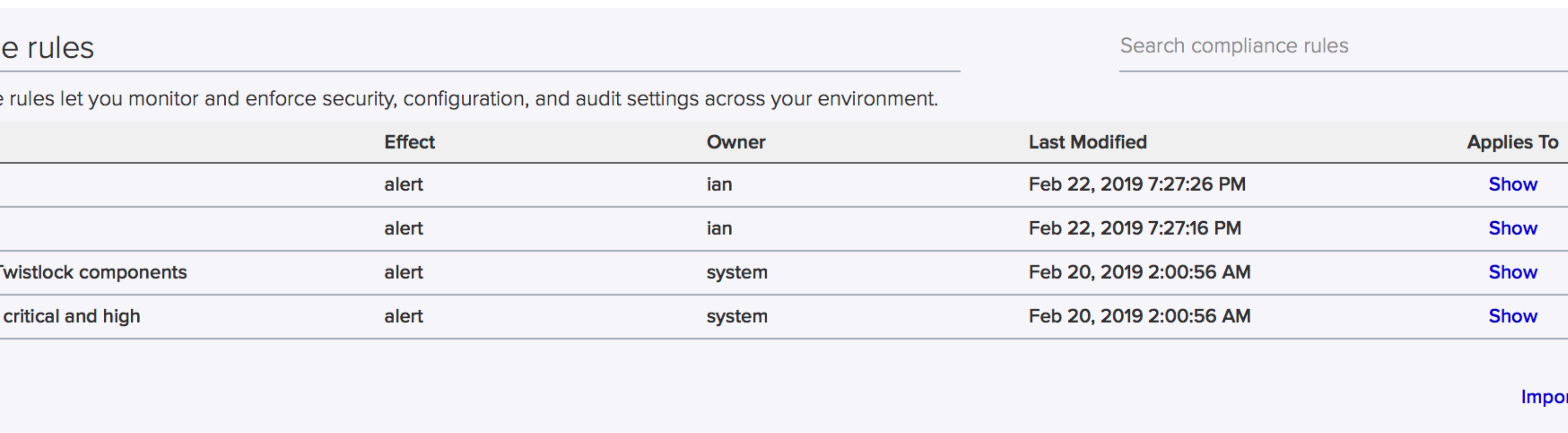
```
ngnix*, -ngnix:latest
```

To exclude just a single image from the universe, set the include scope with a wildcard, and then use the NOT operator to omit the image.

```
*, -mongo:latest
```

Rule ordering

For any given feature area, such as vulnerability management or compliance, you might have multiple rules, such as *test 1* and *test 2*.



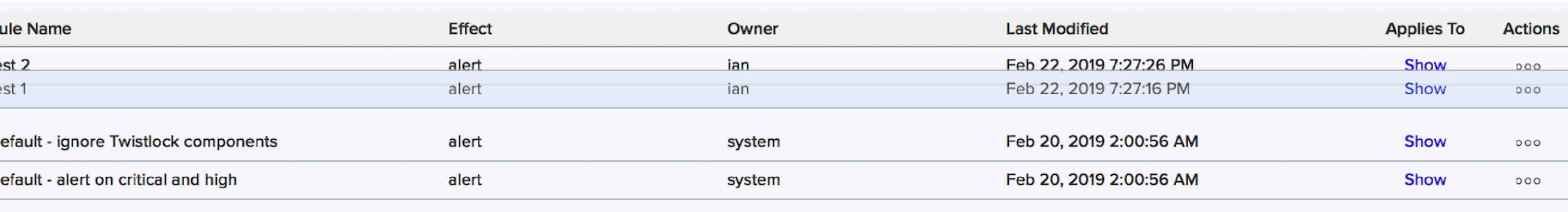
The entire set of rules in a given feature area is called the policy. The rules in the policy are evaluated from top to bottom, making it easy to understand how policy is applied. When evaluating whether to apply a rule to a given object, Prisma Cloud uses the following logic:

1. Does rule 1 apply to object? If yes, apply action(s) defined in rule and stop. If no, go to 2.
2. Does rule 2 apply to object? If yes, apply action(s) defined in rule and stop. If no, go to 3.
3. ...
4. Apply the built-in Default rule (unless it was removed or modified).

Prisma Cloud evaluates the rule list from top to bottom until it finds a match based on the object filters. When a match is found, it applies the actions in the rule and stops processing further rules.

If no match is found, then no action is applied. Sometimes this could mean that an attempted action is blocked (e.g. if no access control rule is matched that allows a user to run a container).

To reorder rules, click on a rule's hamburger button and drag it to a new position in the list.



Disabling rules

If you want to test how the system behaves without a particular rule, you can temporarily disable it. Disabling a rule gives you a way to preserve the rule and its configuration, but take it out of service, so that it's ignored when Prisma Cloud evaluates events against your policy.

To disable a rule, click **Actions > Disable**.

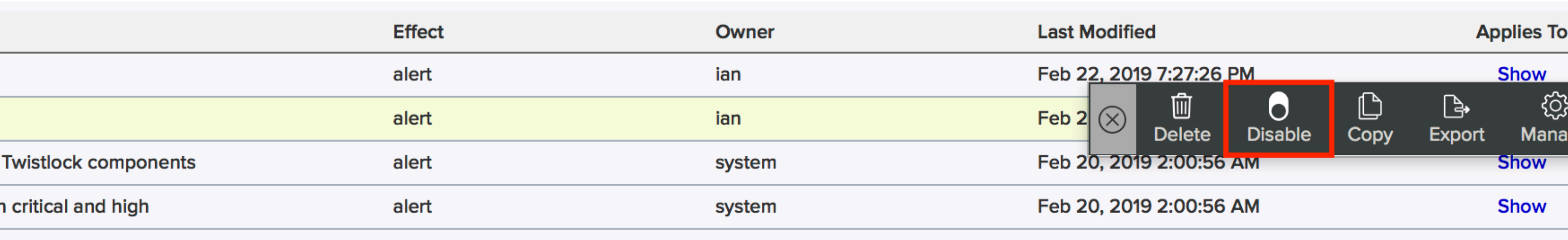


Image names

The canonical name of an image is it's **full name** in a format like registry/repo/image-name. For example: *1234.dkr.ecr.us-east-1.amazonaws.com/morello:foo-images*. Within Docker itself, these canonical names can be seen by inspecting any given image, like this:

```
$ sudo docker inspect morello/foo-images | grep Repo -A 3
      "RepoTags": [
          "1234.dkr.ecr.us-east-1.amazonaws.com/morello:foo-images",
```

However, there's a special case to be aware of with images sourced from Docker Hub. For those images, the Docker Engine and client do not show the full path in the canonical name; instead it only shows the 'short name' that can be used with Docker Hub and the full name is implied. For example, compare the previous example of an image on AWS ECR, with this image on Docker Hub:

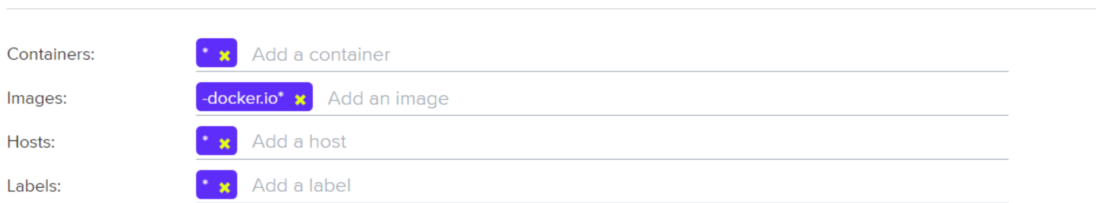
```
$ sudo docker inspect morello/docker-whale | grep Repo -A 3
      "RepoTags": [
          "morello/docker-whale:latest",
```

Note that when the image is from Hub, the canonical name is listed as just the short name (the same name you could use with the Docker client to issue a command like 'docker run morello/docker-whale'). For images like this, Prisma Cloud automatically prepends the actual address of the Docker Hub registry (docker.io) and, if necessary, the library repo name as well, even though these values are not shown by Docker itself.

For example, you can run the Alpine image from Docker Hub simply by issuing a Docker client command like ‘docker run -ti alpine /bin/sh’. The Docker client automatically knows that this means to pull and run the image that has a canonical name of docker.io/library/alpine:latest. However, this full canonical name is not exposed by the Docker client when inspecting the image:

```
$ sudo docker inspect alpine | grep Repo -A 2
      "RepoTags": [
          "alpine:latest"
      ],
      "RepoDigests": [

 "alpine@sha256:1354db23ff5478120c980eca1611a51c9f2b88b61f24283ee8200bf9a54f2e5
      ],
```

But because Prisma Cloud automatically prepends the proper values to compose the canonical name, a rule like this blocks images from Hub from running:




```
$ docker -H :9998 --tls run -ti alpine /bin/sh
docker: Error response from daemon: [Prisma Cloud] The command
 container_create denied for user admin by rule Deny - deny all
 docker.io images.
```

Backup and restore

Edit on GitHub

Prisma Cloud automatically backs up all data and configuration files periodically. You can view all backups, make new backups, and restore specific backups from the Console UI. You can also restore specific backups using the twistcli command line utility.

Prisma Cloud is implemented with containers that cleanly separate the application from its state and configuration data. To back up a Prisma Cloud installation, only the files in the data directory need to be archived. Because Prisma Cloud containers read their state from the files in the data directory, Prisma Cloud containers do not need to be backed up, and they can be installed and restarted from scratch.

When data recovery is enabled (default), Prisma Cloud archives its data files periodically and copies the backup file to a location you specify. The default path to the data directory is */var/lib/twistlock*. You can specify a different path to the data directory in *twistlock.cfg* when you install Console.

Configuring automated backups

By default, automated backups are enabled. With automated backups enabled, Prisma Cloud takes a daily, weekly, and monthly snapshots. These are known as system backups.

To specify a different backup directory or to disable automated backups, modify *twistlock.cfg* and install (or reinstall) Prisma Cloud Console. The following configuration options are available:

Configuration option	Description
DATA_RECOVERY_ENABLED	Enables or disables automated backups. • *true* — Enables automated backups (default). • *false* — Disables automated backups.
DATA_RECOVERY_VOLUME	Specifies the directory where backups are saved. For example, archives could be saved on durable persistent storage, such as a volume from Amazon Elastic Block Storage (EBS). The default value is */var/lib/twistlock-backup*.

STEP 1 | Open *twistlock.cfg* for editing.

STEP 2 | Scroll down to the Data recovery section.

STEP 3 | Enable (or disable) automated back up by setting DATA_RECOVERY_ENABLED to true (or false).

```
DATA_RECOVERY_ENABLED=true
```


STEP 4 | Specify the location where backups should be stored.

```
DATA_RECOVERY_VOLUME=</PATH/TO/BACKUP/VOLUME>
```

STEP 5 | Load your new configuration settings.

If you have not installed Prisma Cloud Console yet, follow the regular installation procedure. For more information, see Install Prisma Cloud.

If Prisma Cloud has already been installed on your host, load your new *twistlock.cfg* file by re-running *twistlock.sh*. The following command assumes that *twistlock.sh* and your updated *twistlock.cfg* reside in the same directory.

```
$ sudo ./twistlock.sh console
```

Making manual backups

Prisma Cloud automatically creates and maintains daily, weekly, and monthly backups. These are known as system backups. You can also make your own backups at any point in time. These are known as manual backups.

STEP 1 | Open Console.

STEP 2 | Go to **Manage > System > Backup & Restore**.

STEP 3 | Under **Manual backups**, click **Create backup**.

STEP 4 | Give your backup a name, then click **Create**.

Your backup file is stored in */var/lib/twistlock-backup* in the storage volume allocated to Prisma Cloud Console. For a onebox installation, this would simply be the local file system of the host where Console runs. For a cluster, such as Kubernetes, this would be the persistent volume allocated to the Console service.

Restoring backups from the Console UI

You can restore Console from a backup file directly from within the Console UI. The Console UI lists all available backups.

You can only restore Console from a backup file whose version exactly matches the current running version of Console. Therefore, if the current running version of Console is 19.11.512, you cannot restore a backup whose version is 19.11.506. To restore a different version of Console, install the Prisma Cloud version that matches your backup version, then follow the procedure here to restore that backup. As long as the specified backup directory (by default, /var/lib/twistlock-backup) contains your backup file, you'll be able to restore it.

STEP 1 | Open Console.

STEP 2 | Go to **Manage > System > Backup & Restore**.

STEP 3 | Click **Restore** on one of the system or manual backups.

STEP 4 | After the database is reloaded from the backup file, restart Console.

For a onebox installation, ssh to the host where Console runs, then run the following command:

```
$ docker restart twistlock_console
```

For a Kubernetes installation, delete the Console pod, and the replication controller will automatically restart it:

```
// Get the name of Prisma Cloud Console pod:
$ kubectl get po -n twistlock | grep console

// Delete the Prisma Cloud Console pod:
$ kubectl delete po <TWISTLOCK_CONSOLE> -n twistlock
```

If any new Defenders were installed since the backup was created, restart those Defenders. Otherwise, they might not function properly.

If a Defender created any new runtime models since the backup was created, restart those Defenders. Otherwise, those models might not be visible.

Restoring backups from twistcli

You can restore Console from a backup using *twistcli*. Use this restore flow when Console is unresponsive and you cannot access the UI to force a restore to a known good state.

You can only restore Console from a backup file whose version exactly matches the current running version of Console. Therefore, if the current running version of Console is 2.5.88, you cannot restore a backup whose version is 2.5.50. To restore a different version of Console, install the Prisma Cloud version that matches your backup version, then follow the procedure here to restore that backup. As long as the specified backup directory (by default, /var/lib/twistlock-backup) contains your backup file, you'll be able to restore it.

Prerequisites:

- Your host can access the volume where the Prisma Cloud backups are stored. By default, backups are stored in */var/lib/twistlock-backup*, although this path might have been customized at install time.
- Your host can access the Prisma Cloud's data volume. By default, the data volume is located in */var/lib/twistlock*, although this path might have been customized at install time.
- Your version of *twistcli* matches the version of the backup you want to restore.

STEP 1 | Go to the directory where you unpacked the Prisma Cloud release.

STEP 2 | Run the *twistcli restore* command. Run *twistcli restore --help* to see all arguments.

1. List all available backups. To list all files in the default backup folder (/var/lib/twistlock-backup), run *twistcli restore* without any arguments:

```
$ ./twistcli restore
```

To list all backup files in a specific location, run:

```
$ ./twistcli restore <PATH/TO/FOLDER>
```

2. Choose a file to restore by entering the number that corresponds with the backup file.

For example:

```
aqsa@aqsa-faith: ./twistcli restore --data-recovery-folder /
var/lib/twistlock-backup/
Please select from the following:
0: backup1      2.5.91  2018-08-07 15:10:10 +0000 UTC
1: daily        2.5.91  2018-08-06 16:10:48 +0000 UTC
2: monthly      2.5.91  2018-08-06 16:10:48 +0000 UTC
3: weekly       2.5.91  2018-08-06 16:10:48 +0000 UTC
Please enter your selection:
0
```

STEP 3 | After the database is reloaded from the backup file, re-install/restart Console.

For a onebox installation, ssh to the host where Console runs, then rerun the installer:

```
$ sudo ./twistlock.sh -ys onebox
```

For a Kubernetes installation, delete the Console pod, and the replication controller will automatically restart it:

```
// Get the name of Prisma Cloud Console pod:
$ kubectl get po -n twistlock | grep console

// Delete the Prisma Cloud Console pod:
$ kubectl delete po <TWISTLOCK_CONSOLE> -n twistlock
```

If any new Defenders were installed since the backup was created, restart those Defenders. Otherwise, they might not function properly.

If a Defender created any new runtime models since the backup was created, restart those Defenders. Otherwise, those models might not be visible.

Restoring Fargate Console

When restoring a Console running on Fargate perform the following steps:

STEP 1 | Create a new Console Fargate task.

STEP 2 | Create Console's first administrative account and enter your license.

STEP 3 | Restoring backups from the Console UI.

STEP 4 | Restart the Console by stopping the task and allowing the scheduler to create a new Console task.

Downloading backup files

Prisma Cloud Compute lets you download backup files so that they can be copied to another location. Backup files can be downloaded from the Console. Go to **Manage > System > Backup & Restore**, and click **Actions > Export** to download a backup.

Custom feeds

Edit on GitHub

You can supplement the Prisma Cloud Intelligence Stream with your own custom data, including:

- Suspicious or high-risk IP addresses
- Malware signatures
- Trusted executables
- Custom vulnerabilities for proprietary software components..
- Allowed CVEs

For each data type, you can add individual entries to a table from the Console web interface, bulk upload a list from a CSV file, or submit a JSON object using the Prisma Cloud API.

Supplementing the IP reputation list

You can supplement the Prisma Cloud Intelligence Stream with your own list of suspicious or high-risk IP addresses that you want to ban on your network.

STEP 1 | Open Console.

STEP 2 | Go to **Manage > System > Custom Feeds**.

STEP 3 | Click **IP Reputation Lists**, and either click **Add IP** or **Import CSV**.

Your list of banned IP addresses is immediately enforced when your data is imported. A default runtime defense rule, **Default - detect suspicious runtime behavior**, logs an alert when a container tries to connect to a banned IP address.

You can manually add one entry at a time, or do a bulk upload from a CSV file. The maximum file size for a csv is 20MB.

The first line in your CSV file must be a header record that contains the field names. Specify one IP address per line. For example:

```
ip
99.104.125.48
101.200.81.187
103.19.89.118
```

STEP 4 | Review the default rule

Go to **Defend > Runtime > {Container Policy | Host Policy}**, then click manage for the **Default - detect suspicious runtime behavior** rule. You should see that **Prisma Cloud Advanced Threat Protection** is set to **On**.

Create a list of malware signatures and trusted executables

You can supplement the Prisma Cloud Intelligence Stream with your own custom malware signatures and trusted executable list. The trusted executable list is a mechanism to address potential misidentification of legitimate files as malware.

You can add MD5 hashes of custom malicious executables to the malware signatures list, it enables you to monitor malware that you want to alert or block in runtime rules.

Malware scanning and detection is supported for Linux container images and hosts only. Windows containers and hosts are not supported.

When you add MD5 hashes (signatures of binaries) to the trusted executables list, it enables you to ensure that a legitimate binary is not potentially identified as malicious by file system based defense capability in runtime rules.

The trusted executable list does not apply to other runtime defense capabilities, such as process runtime protection. To exclude files from other runtime detection capabilities use the allowed list in the runtime defense section.

STEP 1 | Open Console.

STEP 2 | Go to **Manage > System > Custom Feeds**.

STEP 3 | Select **Malware signatures** or **Trusted executables**.

STEP 4 | Choose how to add the MD5 hashes for malware signatures or trusted executables.

The MD5 hashes you add to either list of custom feed, are used in all subsequent image scans. It is also used immediately by the runtime defense file system sensor, which assesses all writes to the host and container file system.

1. For **Add MD5**, you can manually add one entry at a time.
2. For **Import CSV**, you can bulk upload from a CSV file.

   The maximum file size is 20MB.

   The first line in your CSV file must be a header record that contains the field names.

   Specify one entry per line. Each entry must include the MD5, followed by a good description to identify why the MD5 is trusted (in the case of trusted executables) or is known to be malicious (in the case of malware signatures).

   For example:

```
md5,name
194836fbe0f121a25b145e55e80cef22,legitimate binary built in-
house
0aeb0cac186a81a6ac45776d6b56dd70,test file
33cc273ae3aa8bce6a22c92e7d11f63a,benign file
```

STEP 5 | Review the default rule.

A default runtime defense rule, **Default - detect suspicious runtime behavior**, logs an alert when malware is detected using signatures from the Prisma Cloud data set or your custom data set.

To review the default rule, go to **Defend > Runtime > {Container Policy | Host Policy}**, then click manage for the **Default - detect suspicious runtime behavior** rule. You should see that **Prisma Cloud Advanced Threat Protection** is set to **On**.

Allowing CVEs globally

Some organizations have have very sophisticated CI pipelines that encompass many teams and products. When your security team concludes that a CVE doesn't impact the organization, they want to dismiss it globally without having to manage individual rules or exceptions.

The CVE Allow List lets you allow CVEs system-wide. Any entry in the CVE Allow List affects all flows in the product, including twistcli, the Jenkins plugin, registry scanning, deployment blocking, Vulnerability Explorer, and so on. Adding a CVE to this list effectively filters it out from the data in the Prisma Cloud Intelligence Stream before it's used by the scanner.

The CVE Allow List takes precedence over any rule that's been created under **Defend > Vulnerabilities**. It is a feature designed to complement rules. Rules also let you allow a CVE, but more granularly, by scoping them to specific resources or parts of your environment.

STEP 1 | Open Console.

STEP 2 | Go to **Manage > System > Custom Feeds**.

STEP 3 | Click **CVE Allow List**, and either click **Add CVE** or **Import CSV**.

You can set an expiration date for the CVE, if you want to set a time restriction for when it should no longer be allowed.

Test Prisma Cloud malware detection capabilities

Safely simulate malware in your environment to test the malware detection capabilities on Prisma Cloud.

Configure a custom malware feed

Set up a custom feed by uploading the provided CSV file to Prisma Cloud Console. This file specifies the MD5 signature for a file that will be considered malware for the purposes of this demo.

STEP 1 | Download *malware.csv*.

STEP 2 | In Console, go to **Manage > System > Custom Feeds > Malware Signatures**.

STEP 3 | Click **Import CSV**, and upload *malware.csv*.

Detect malware at runtime

Test how Prisma Cloud detects malware being downloaded into a container at runtime.

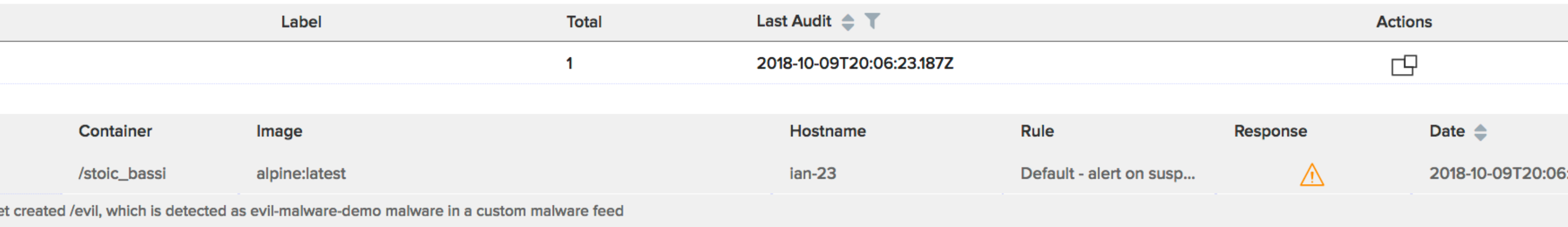
Prerequisites: The default runtime rule, **Default - alert on suspicious runtime behavior** under **Defend > Runtime > Container Policy** is in place. If you have deleted or changed the default rule, create a new one.

1. Go to **Defend > Runtime > Container Policy**, and click **Add rule**.
2. Enter a name for the rule.
3. In the **General** tab, verify **Prisma Cloud Advanced Threat Protection** is **On**.
4. In each of the **Process**, **Networking**, **File System**, and **System Calls** tabs, set **Effect** to **Alert**.

STEP 1 | Run a container and download malware into it.

```
$ docker run -ti alpine sh
/ # wget https://cdn.twistlock.com/docs/attachments/evil
```

STEP 2 | Look at resulting audit. Open Console and browse to **Monitor > Events > Container Audits**. You will see a file system audit that says malware was detected.



Configuring Prisma Cloud proxy settings

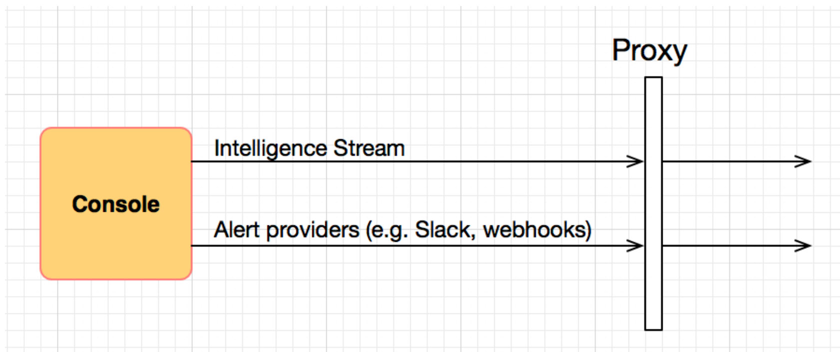
Edit on GitHub

In some environments, access to the internet must go through a proxy and you can configure Prisma Cloud to route requests through your proxy. Proxy settings can either be applied to both Console and Defender containers or separately for each Defender deployment.

The global proxy settings are configured in the UI after Console is installed. Console starts using these settings after you apply it. Any Defenders deployed after you configure the proxy settings will use it unless you explicitly choose a different proxy when deploying the Defenders. Any Defenders that were deployed before you saved your proxy settings must be redeployed.

Console

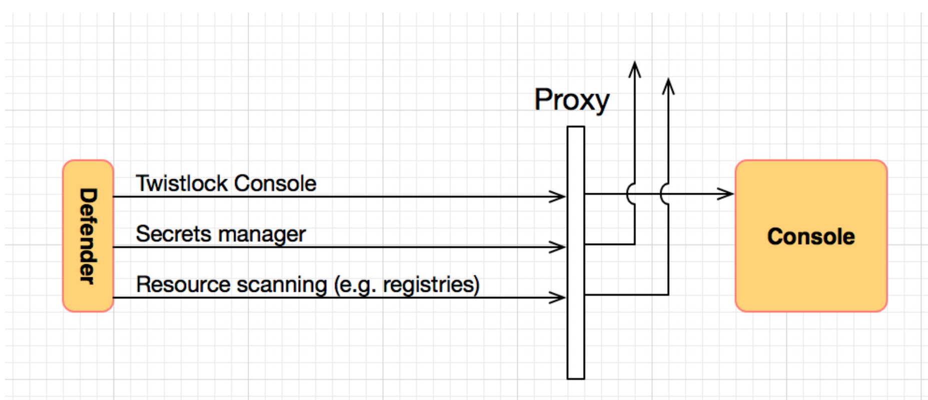
Console has a number of connections that might traverse a proxy.



- Retrieving Intelligence Stream updates.
- Connecting to services, such as Slack and JIRA, to push alerts.

Defenders

Defender has a number of connections that might traverse a proxy.



- Connecting to Console.

  If you have a proxy or a load balancer between Defender and Console, make sure that TLS interception is not enabled. The certificate and keys used for the Console to Defender mutual TLS v1.2 web socket session cannot be intercepted. This ensures that the Console is only communicating with the Defenders it has deployed and the Defenders only communicate with the Console that manages them.

- Connecting to external systems, such as Docker Hub or Google Container Registry, for scanning.
- Connecting to your secrets store to retrieve secrets for injection into your containers.

Global proxy settings

A number of settings let you specify how Prisma Cloud interfaces with your proxy.

Proxy bypass

You can provide a list of addresses—DNS names, IP addresses, or a combination of both—that Prisma Cloud can contact directly without connecting through the proxy. Specifying IP addresses in CIDR notation is supported. Specifying DNS names using wildcards is supported.

CA certificate

Console verifies server certificates for all TLS connections. With TLS intercept proxies, the connection from Console to the Internet passes through a proxy, which may be transparent. To facilitate traffic inspection, the proxy terminates the TLS connection and establishes a new connection to the final destination.

If you have a TLS intercept proxy, it will break the Console's ability to connect to external services, because Console won't be able to verify the proxy's certificate. To get Console to trust the proxy, provide the CA certificates for Console to trust.

Proxy authentication

If egress connections through your proxy require authentication, you can provide the credentials in Prisma Cloud's proxy settings. Prisma Cloud supports Basic authentication for the Proxy-Authenticate challenge-response framework defined in RFC 7235. When you provide a username and password, Prisma Cloud submits the credentials in the request's Proxy-Authorization header.

Configuring global proxy settings

Configure your proxy settings in Console.

STEP 1 | Open Console, and go to **Manage > System > Proxy**.

STEP 2 | In **HTTP Proxy**, enter the address of the web proxy. Specify the address in the following format: <PROTOCOL>://<IP_ADDR|DNS_NAME>:<PORT>, such as http://proxyserver.company.com:8080.

STEP 3 | (Optional) In **No Proxy**, enter addresses that Prisma Cloud can access directly without connecting to the proxy. Enter a list of IP addresses and domain names. Specifying IP addresses in CIDR notation is supported. Specifying DNS names using wildcards is supported.

STEP 4 | (Optional) For TLS intercept proxies, enter the root trusted authority certificate, in PEM format, that Console should trust.

STEP 5 | (Optional) If your proxy requires authentication, enter a username and password.

STEP 6 | Click **Save**.

STEP 7 | Redeploy your Defenders to propagate updated proxy settings to them.

Console does not need to be restarted. After proxy settings are saved, Console automatically uses the settings the next time it establishes a connection.

Any newly deployed Defenders will use your proxy settings.

Any already deployed Defenders must be redeployed. For single Container Defenders, uninstall then reinstall. For Defender DaemonSets, regenerate the DaemonSet YAML, then redeploy.

```
$ kubectl apply -f defender.yaml
```

Configuring per-deployment proxy settings

Prisma Cloud supports setting custom proxy settings for each Defender deployment. This way you can set multiple proxies for Defenders which are deployed in different environments.

STEP 1 | Open Console, and go to **Manage > Defenders > Deploy**.

STEP 2 | Choose your preferred deployment method.

STEP 3 | Click on **Specify a proxy for the defender (optional)** and enter your proxy details.

Prisma Cloud Compute certificates

Edit on GitHub

This article summarizes all the certificates used by Prisma Cloud Compute. Learn more about the certificates used on Prisma Cloud Compute including details on what it is used for, signing CA, and your customization options.

Customizing certificates is only allowed for Prisma Cloud Compute edition.

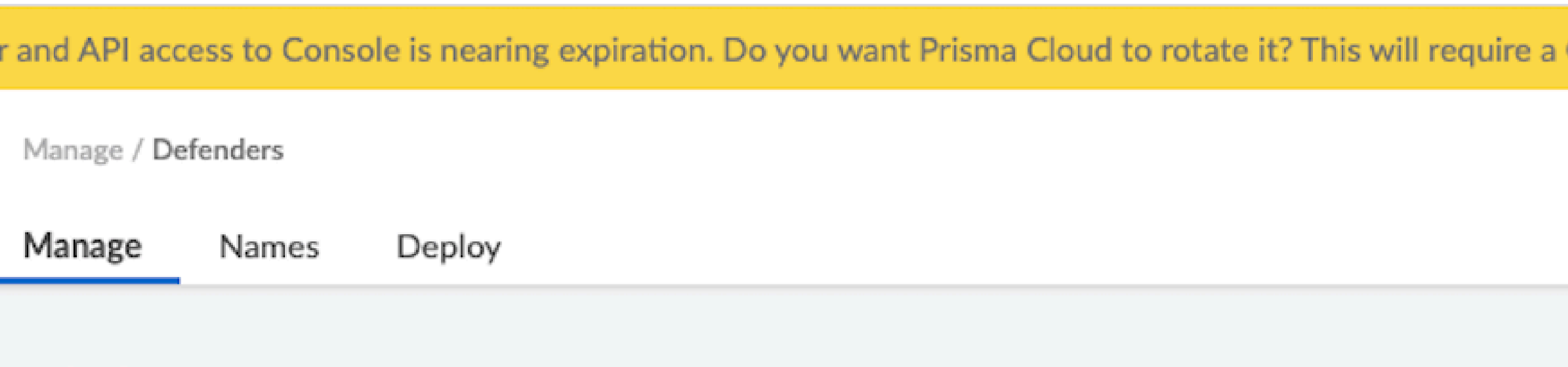
Category	Certificate	Commun	Certificate customization	Default CA	CA customization	Prisma Cloud edition
Console TLS communication	Console Web and API certificate	Web browser, API, and Twistcli access to Console	Customize under **Manage > Authentication > System certificates > TLS certificate for Console > Concatenate public cert and private key**	Console CA	Your organization CA	Compute edition, Enterprise edition
Docker access control	Client certificates To enforce Docker access control, client certs should be installed on any host where the docker client can be run.	Clients (users) access to remote Docker Engine instances	Customize your own certificates for your clients Explicit list of trusted certificates can be defined under **Manage > Authentication > System certificates > Client certificates > Explicit certificate trust list**	Console CA	Customize under **Manage > Authentication > System certificates > Client certificates > CA certificate**	Compute edition, Enterprise edition
Certificate-based authentication to Console		Clients access the Console		No CA by default	Enable Console verification of the client's CA certificate	Compute edition

Category	Certificate	Communication	Certificate customization	Default CA	CA customization	Prisma Cloud edition
					when accessing the Console. Define CA under **Manage > Authentication > System certificates > Certificate-based authentication to Console > CA certificate**	
Console-Defender communication	Defender server certificate (Console side)	Console-Defender communication	Yes, for Compute Edition only See here	Defender CA (defender-ca.pem)	Yes, for Compute Edition only. See here	Compute edition only. Not relevant for Enterprise edition (uses API token)
Console-Defender communication	Defender client certificate (Defender side)	Console-Defender communication	No	Defender CA (defender-ca.pem)	No	Compute edition, not relevant for Enterprise edition (uses API token)
Admission control	Admission certificate (admission-cert.pem)	Admission webhook authentication with Prisma Cloud Defender	No	Defender CA (defender-ca.pem)	No	Compute edition, Enterprise edition

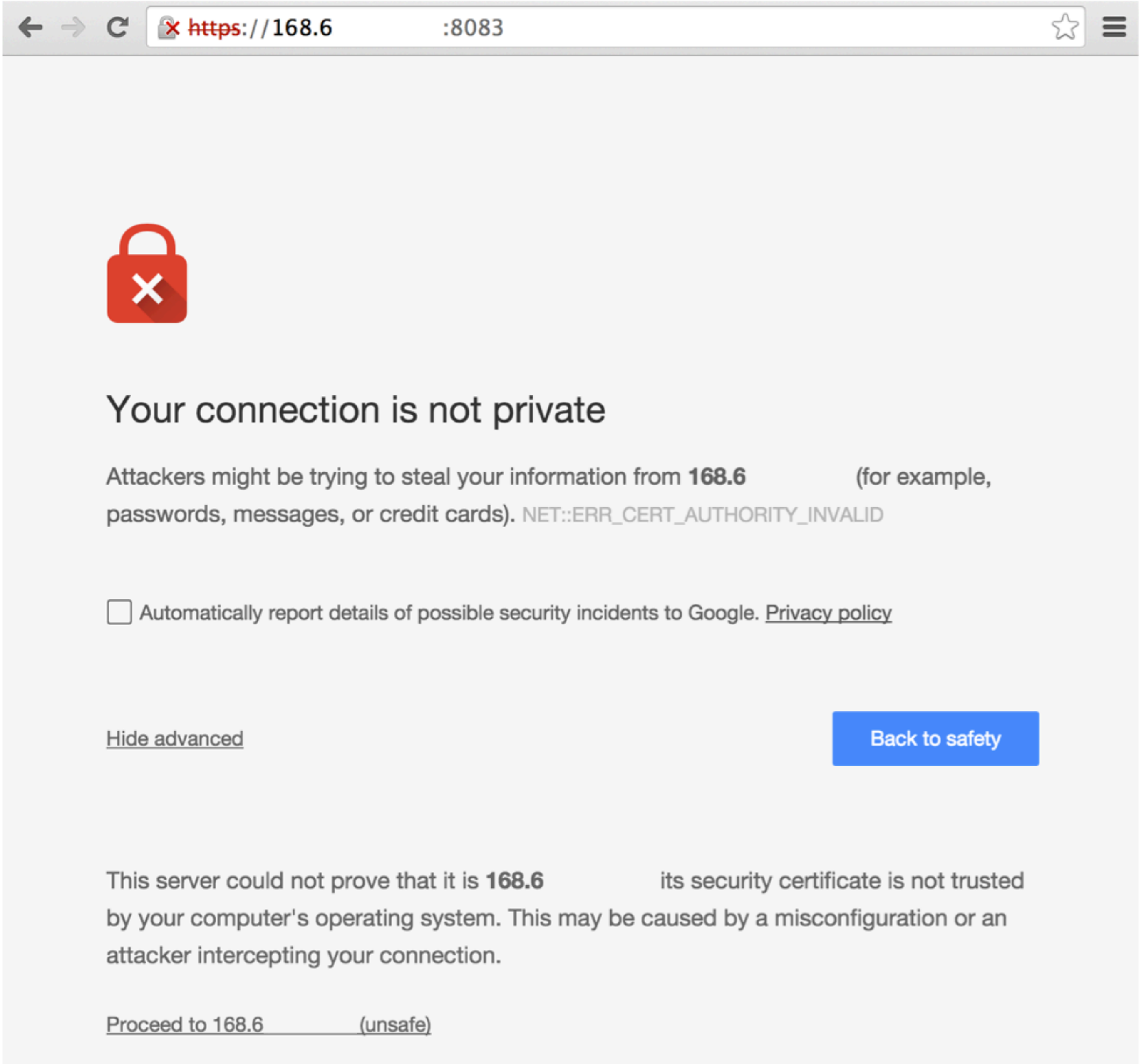
Console TLS communication certificates

You can secure access to Console with your own digital certificate. By default, Prisma Cloud accesses the Console's web portal and API with a self-signed certificate.

The self-managed certificate generated by Console is valid for three years. 90 days prior to expiration, Prisma Cloud will let you rotate it (a banner will appear at the top of the UI). After rotating Console's certificate, you must restart the Console.



When you access Console's web portal with this setup, for example, the browser flags the portal as untrusted with a warning message. The following screenshot shows the warning message in Chrome:



You can resolve these warnings by installing your own certificate that proves your server's identity to the client. With the proper certificate, users are taken directly to Console, and the green padlock in the address bar indicates that the site is trusted.

Creating certificates is outside the scope of this article. For more information about how SSL and certificates secure a site, see How does HTTPS actually work.

Configuration options

Prisma Cloud secures the communication between various actors and entities with certificates. These certificates are automatically generated and self-signed during the Prisma Cloud install process. They secure communication between:

- Users and the Console web portal
- Users and the Console API
- Console and the Prisma Cloud Intelligence Stream

The following options control the properties of the certificates generated during the installation process. The default values for these options are typically adequate.

Note that these settings only change the values used when creating self-signed certificates. Thus, users accessing the Console will still see warning messages because the certificates are not signed by a trusted certificate authority (CA). To configure the Console to use a certificate signed by a trusted CA, follow the steps later in this article.

These options can be found in *twistlock.cfg* under the General Configuration section:

Configuration option	Description
CONSOLE_CN	Specifies the Common Name to be used in the certificate generated by Prisma Cloud for the host that runs Console. The Common Name is typically your hostname plus domain name. For example, it might be www.example.com or example.com. (Default) By default, the Common Name is assigned the output from the command *hostname --fqdn*.
DEFENDER_CN	Specifies the Common Name to be used in the certificate generated by Prisma Cloud for the hosts that run Defender. (Default) By default, the Common Name is assigned the output from the command *hostname --fqdn*.

You can also control the Subject Alternative Names (SANs) in Console's certificate.

Securing access to Console with custom certificates

Secure access to Console with your own custom certificates.

Prerequisites:

- Your certs have been generated by a commercial Certificate Authority (CA) or with your own Public Key Infrastructure (PKI). You should have the following files on hand:
  - A *.pem* file, which contains your certificate and your Certificate Authority's intermediate certificates.
  - A *.key* file, which contains your private key.

STEP 1 | Have your signed certificate (*.pem* file) and private key (*.key* file) ready to be accessed and uploaded to Console.

Make sure that the private key starts and ends with:

```
----BEGIN PRIVATE KEY----
----END PRIVATE KEY----
```

or:

```
-----BEGIN RSA PRIVATE KEY-----
-----END RSA PRIVATE KEY-----
```

STEP 2 | Open Prisma Cloud Console in a browser.

STEP 3 | Navigate to **Manage > Authentication > System Certificates**.

STEP 4 | Concatenate your public certificate and private key into a single PEM file.

```
$ cat server.crt server.key > server-cert.pem
```

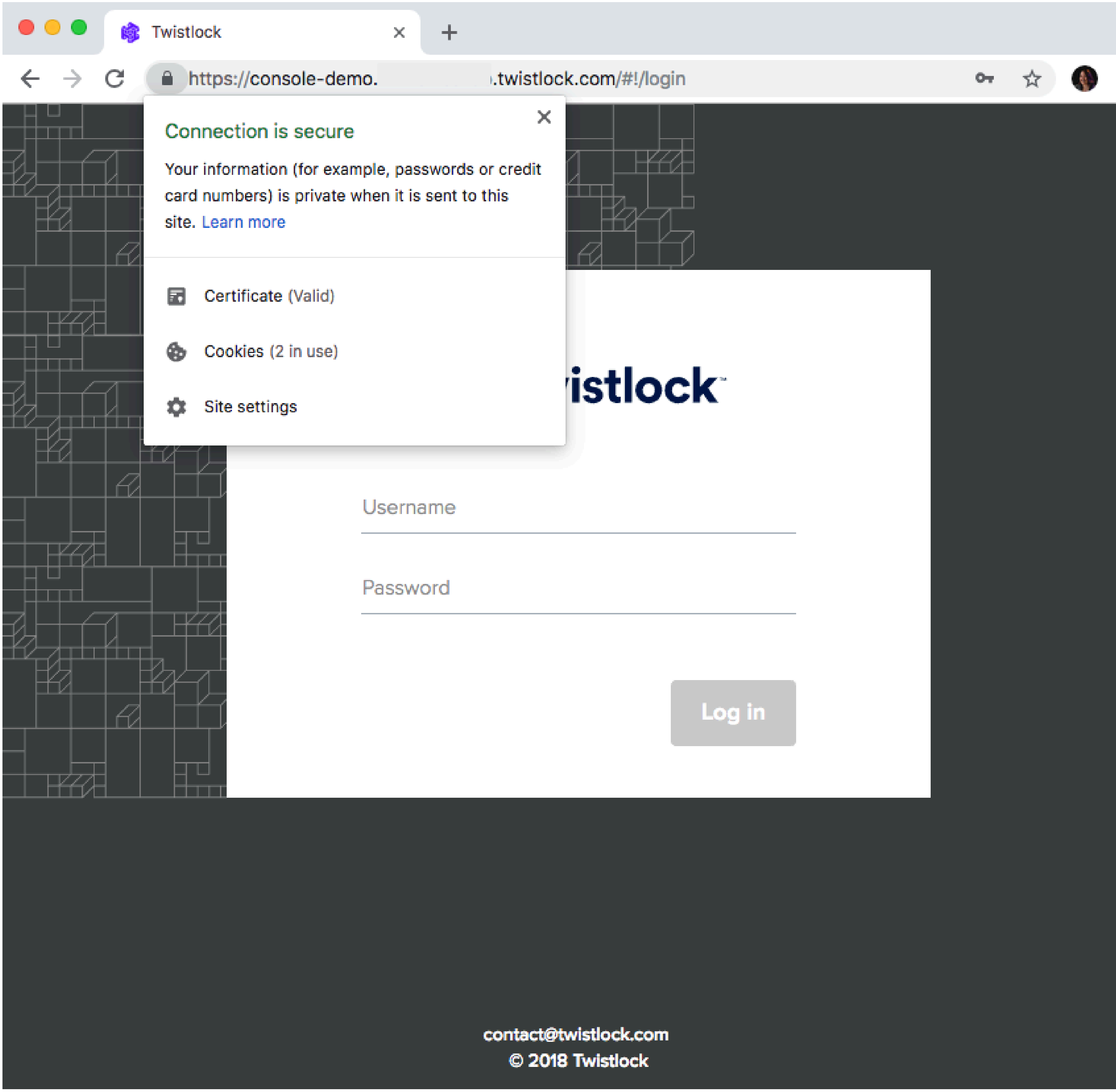
STEP 5 | Open the **TLS certificate for Console** section

1. Upload the PEM file into the **Concatenate public cert and private key (e.g., cat server-cert.pem server-key.pem)**
2. Click **Save**

STEP 6 | Verify that your certs have been correctly installed.

Open your browser, and navigate to: https://<CONSOLE_HOSTNAME>:8083

If you see the locked padlock icon, you have installed your certs correctly.



HTTP Public Key Pinning (HPKP) was a security feature that was used to tell a web client to associate a specific cryptographic public key with a certain web server to decrease the risk of Man In The Middle (MITM) attacks with forged certificates. This feature is no longer recommended. See https://developer.mozilla.org/en-US/docs/Web/HTTP/Public_Key_Pinning

Docker role-based access control certificates

These certificates settings are related to the Docker access control feature. Using the Docker access control, you can validate that Docker commands only run from remote machines through Defender on port 9998. Any user running Docker commands on port 9998 must be authenticated and authorized. By default, the Console generates certificates for users to authenticate to Defender. Any command run against Defender must also be explicitly allowed.

Prisma Cloud lets you use your own certificates for Docker access control. Customize the Docker access control certificates, by providing Prisma Cloud the CA that signs the client (user) certificates. You can also specify an explicit list for client-trusted certificates.

NOTES:

- External certification authority (CA) section will be visible only to an Admin role user.
- All trusted certificate information will be retrieved from the certificate itself, so the user doesn't have to manually add info such as CN, issuer, etc.
- Only the public portion of a user certificate should be added to the explicit trust list. Private keys are not required and should be excluded from this process.

Setting up your custom certs

To set up your custom certs:

STEP 1 | Open Console, and go to **Manage > Authentication > System certificates**.

STEP 2 | Open the **CA certificate** card.

1. Under **CA certificate**, upload CA certificate to trust.

   Once this configuration is enabled, users must copy their keys (both public and private) to the host they're using to run commands with docker or kubectl. Though the path can be referenced in each command, it's usually simpler to place them in the default directory that docker looks in for certificates (~/.docker).

   Each user certificate used with Prisma Cloud must have the user's CN embedded in the Subject field of the certificate. You can validate these settings by running the following command against the certificate:

```
$ openssl x509 -in .docker/cert.pem -text | grep Subj
Subject: CN=username
```

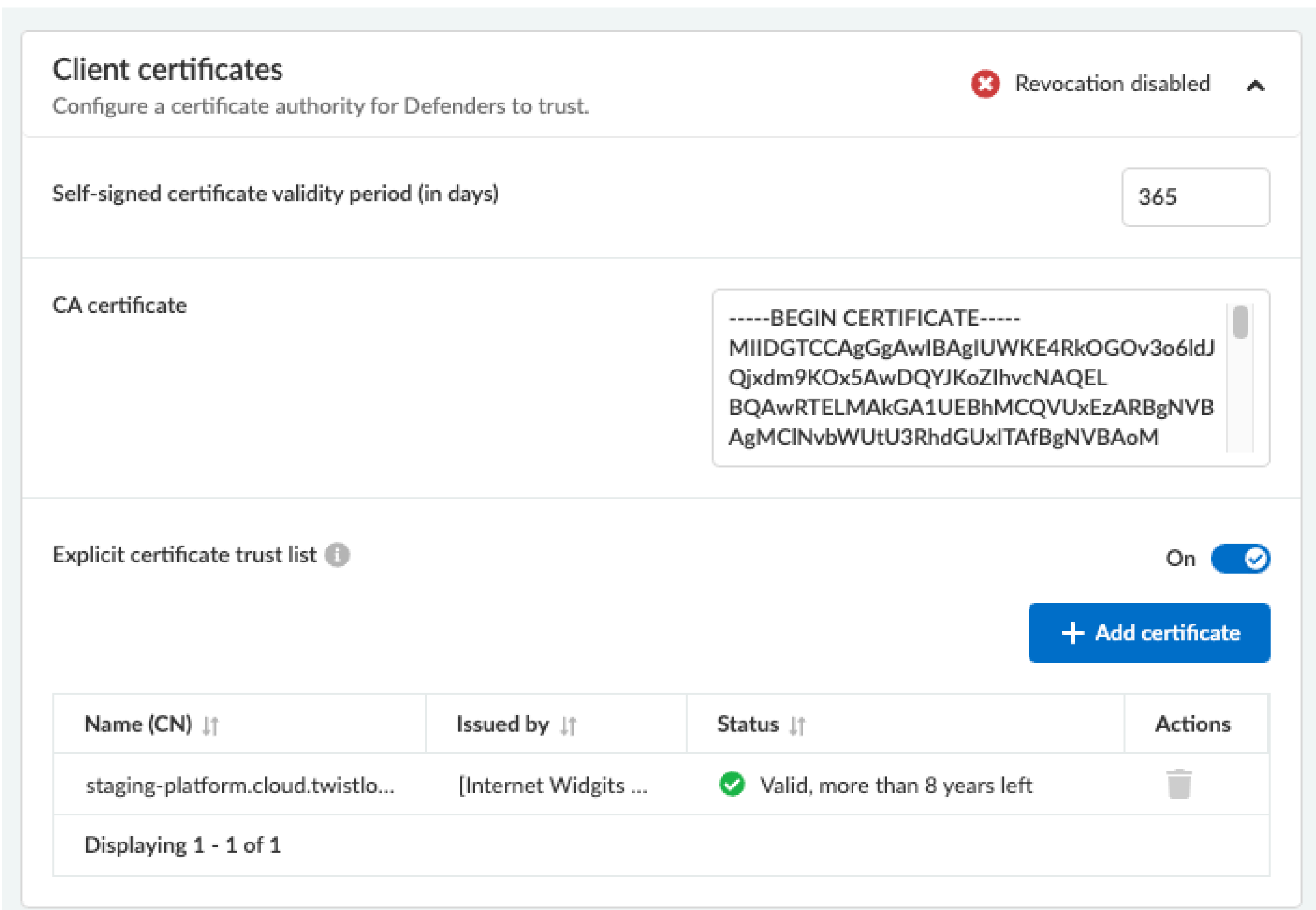
   Finally, Docker requires that the CA certificate used to sign the server certificate on the nodes Prisma Cloud is protecting must also be in the ~/.docker folder, in a file called ca.pem. Because the 'server' certificate used in this deployment model is still generated by Prisma Cloud, this means that on each host where you're running docker or kubectl commands, you must also add the CA certificate to this folder.

STEP 3 | You can also choose to set **Explicit certificate trust list** to **ON** (this configuration is optional)

An explicit certificate trust list allows you to create a list of explicitly trusted custom certificates. A typical use case of this feature would be when you have multiple certificates issued to a given user, but only want specific ones to be available for use with Prisma Cloud. By adding an explicit trust list, you can control what certificates can be used, as Prisma Cloud compares any certificates presented to it against the allowed trusted certificates list. This way, a user using a certificate not in the explicitly allowed list will not be able to use the certificate with Prisma Cloud, even if it was issued by a trusted CA. Note that this feature is valid only when a custom CA is configured. When enabled, this feature allows users to add new certificates to a table by uploading public certificates in PEM format.

STEP 4 | Click **Add certificate**, copy the PEM-formatted public certificate which was issued by the trusted CA, then click **Add**.

When a custom certificate is provided to Prisma Cloud, it first checks the certificate against this list. If the certificate is matched to an entry in the list, then the previously existing flow continues. If the certificate is not in the trusted list, then the authentication fails with an error 'Certificate not in certificate trust list configured in Prisma Cloud'.



Certificate-based authentication to Console

This feature allows the Console to verify the client's CA certificate when accessing the Console. Use certificates from an implicitly trusted CA for securing the TLS connection. To enable this feature, follow the steps below:

STEP 1 | Open Console, and go to **Manage > Authentication > System Certificates**.

STEP 2 | Open the **Certificate-based authentication to Console** card.

STEP 3 | Under **Console Authentication** upload the CA certificate(s) in PEM format, then click **Save**.

If you have multiple CAs, such as a root CA and several issuing CAs, you must add all these certificates into the PEM file. The order of certificates in the PEM file should be from the lowest tier of the hierarchy to the root. For example, if you have a 3 tier hierarchy that looks like this:

```
->RootCA
     ->IntermediateCA
          ->IssuingCA1
          ->IssuingCA2
```

Your PEM file should be ordered as IssuingCA1, IssuingCA2, IntermediateCA, RootCA. To create such a PEM file, you'd get the public keys of each CA in PEM format and concatenate them together:

```
$ cat IssuingCA1.pem IssuingCA2.pem IntermediateCA.pem RootCA.pem >
 CAs.pem
```

Console-Defender communication certificates

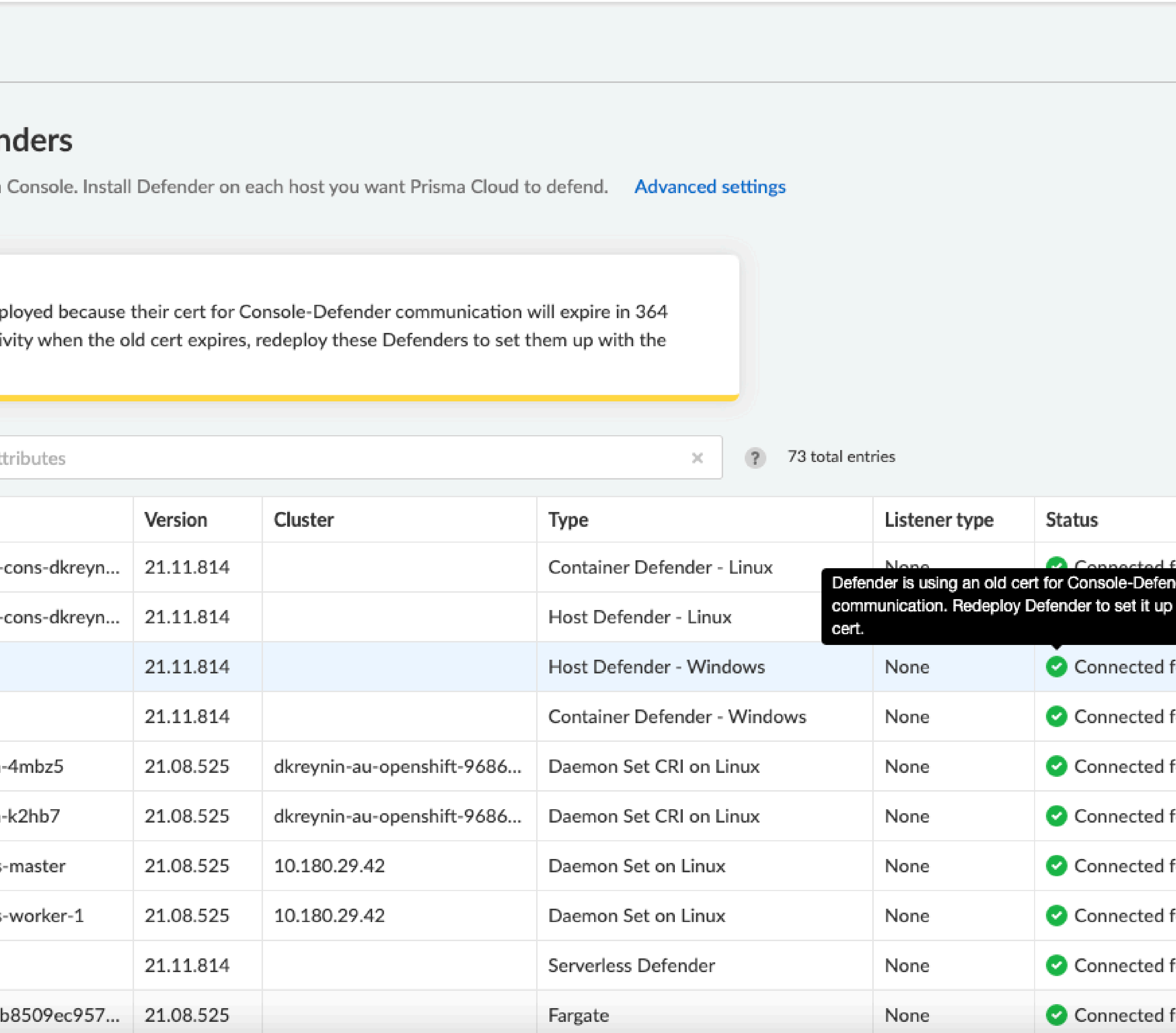
By design, Console and Defender don't trust each other and use certificates to mutually authenticate when Defender establishes a connection with Console. The certificates for Console-Defender communication are issued by the Defender CA (defender-ca.pem). The Defender CA is a self-signed CA, generated by Console, and it's valid for three years. Console is considered the server and Defender the client. Console generates certs for each party, and signs them with the Defender CA.

Prisma Cloud automatically rotates the Defender CA and related server and client certificates 1.5 years before the Defender CA expires. Console and Defender use the old certs until the old Defender CA expires.

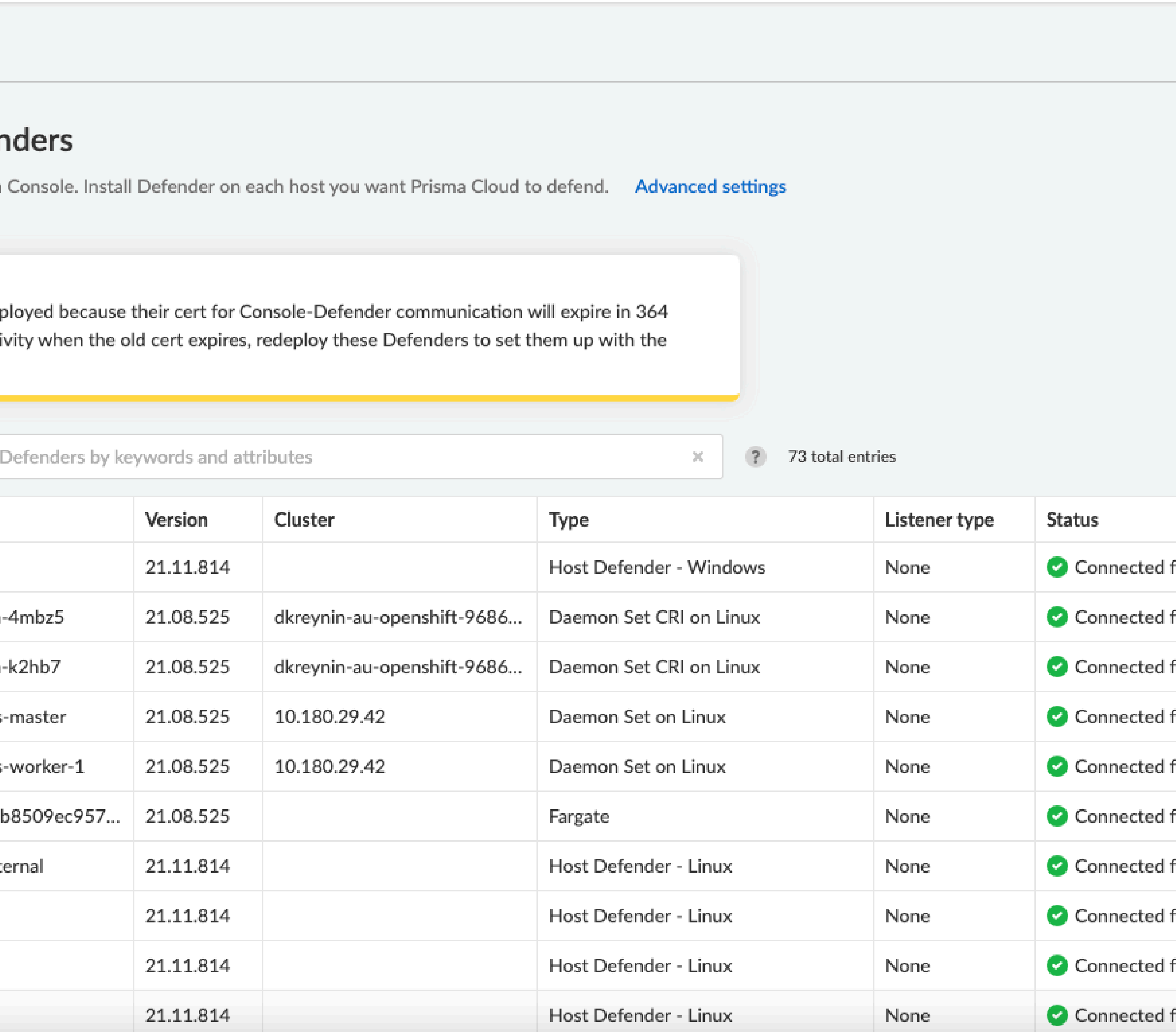
New Defenders, deployed after the certificates have been rotated, automatically get both the new and old certificates. Existing Defenders, however, must be redeployed to get the new certificates. Existing Defenders use the old certificates until they expire. Thereafter, these Defenders won't be able to establish a connection to Console until they're redeployed.

Single Defenders upgraded from the Console UI don't get newly rotated certificates. To set up single Defenders with the new certificate, you must manually redeploy them.

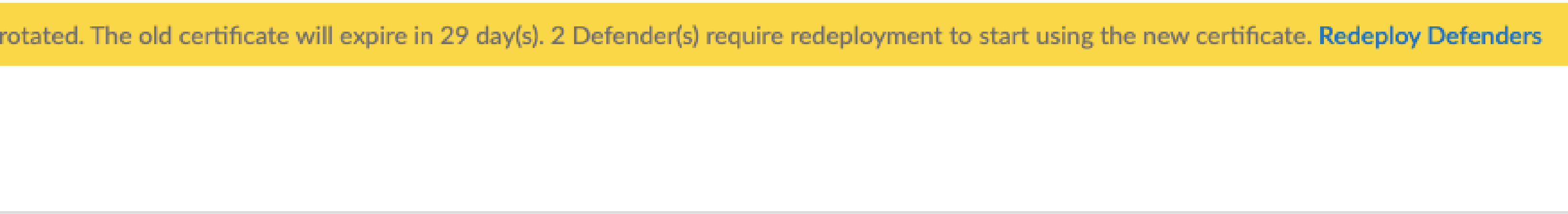
To identify which Defenders need to be redeployed, go to **Manage > Defenders > Manage > Defenders**. Use the **Status** column to identify the Defenders that are using an old certificate. Use the note at the top of the page to understand how many Defenders require redeployment, and when the old certificate will expire.



Use the **Using old certificate** filter on the Defenders list to see only the Defenders that are using an old certificate:



If you still have Defenders in your environment that are using an old certificate, which is about to expire in 60 days or less, you will get notified once entering the Console UI:



If the old certificate has expired, and you still have Defenders in your environment that are using the expired certificate, you will get notified once entering the Console UI. The **Status** column on the Defenders page will reflect the Defenders that are using an expired certificate. Use the **Certificate expired** filter on the Defenders list to see only the Defenders with expired certificates.

Additional technical details

This section provides additional technical details about how the certificates that secure Console-Defender communication are managed.

What is the rotation model?

When Console is first deployed, it generates a set of certs for Console-Defender communication - a Defender CA, a Defender server cert, and a Defender client cert (with keys). The certs are valid for three years. Console initiates the certificate rotation. Console rotates the certs 1.5 years before the Defender CA expires. Thereafter, Console holds two sets of certificates: old and new Console rotates the new certs 1.5 years before the new Defender CA expires. The old certs are deleted, the new certs become the old certs, and a new set of certs are created.

Newly deployed Defenders, after rotation, are deployed with two sets of certs: old and new. Defenders that aren't redeployed only have the old client certs and CA, and keep using them until they expire.

Until the old Defender CA expires, Console responds with the old Defender certs during the TLS handshake when Defender tries to connect to Console. As long as the old Defender CA is valid, Defender uses the old client cert for TLS handshakes. When the old certs expire, Defender uses the new certs for TLS handshakes.

Which certificates are rotated?

Console rotates the following files:

- *defender-ca.pem* — Rotated to defender-ca.pem.old, and then Console creates a new defender-ca.pem.
- *defender-server-cert.pem* and *defender-server-key.pem* — Rotated to defender-server-cert.pem.old and defender-server-key.pem.old, and then Console creates new ones.
- *defender-client-cert.pem* and *defender-client-key.pem* — Rotated to defender-client-cert.pem.old and defender-client-key.pem.old, and then Console creates new ones.

Are all certs rotated at the same time?

Yes, the Defender CA cert, server cert, and client cert are all rotated at the same time.

What triggers Console to regenerate and rotate the certs?

Console checks the expiration date of the Defender CA, and rotates all certs 1.5 years before the Defender CA expires.

What is the rotation frequency?

Once every 1.5 years.

What happens when you upgrade Prisma Cloud Compute?

When Console or Defenders are upgraded, the old, unexpired certificates remain on the system. Defenders that only have the old certificates are supported until the old Defender CA expires.

How can you programmatically determine that certs have been rotated?

Look for changes to the Defender certificates on the machine that runs Console. Certificates are stored in */var/lib/twistlock/certificates*.

Inspect the Defender CA cert for its expiration time. When the .old suffix is added to the cert file, you will know it has been rotated.

Can you manage the certificate lifecycle yourself?

Yes, for Compute Edition (self-hosted) only.

See Configure custom certs from a predefined directory.

SaaS Defenders connect to Console using an API token, not certs.

After certs have been rotated, what's returned from api/v<VERSION>/defenders/daemonset.yaml?

The DaemonSet yaml will include both sets of new and old certs:

New certs:

- defender-ca.pem
- defender-client-cert.pem
- defender-client-key.pem

Old certs:

- defender-ca.pem.old
- defender-client-cert.pem.old
- defender-client-key.pem.old

Which Defender types support certificate rotation?

Supported Defender types:

- Container Defenders (Windows and Linux)
- Host Defenders (Windows and Linux)
- DaemonSet Defenders
- App-Embedded Defenders, including Fargate

Serverless Defenders aren't supported. Serverless Defenders are always deployed with old, unexpired certs, even if new certs exist.

What happens the moment a Defender's old certs expire?

Defenders can switch to new certificates from old certificates at runtime. No restart is required.

Admission control certificates

Prisma Cloud provides a dynamic admission controller for Kubernetes built on the Open Policy Agent (OPA). The admission control certificate is used for the authentication between the Defenders and the admission webhook. When deploying the admission webhook, make sure it is configured with the right CA bundle, according to the Defender's admission certificate. See the webhook configuration section on the admission control article.

Configure Agentless Scanning

Edit on GitHub

Agentless scanning provides visibility into vulnerabilities and compliance risks on hosts by scanning the root volumes of snapshots. The agentless scanning architecture lets you inspect a host without having to install an agent or affecting its execution. To learn more about the architecture and scan results, see agentless scanning.

- Prerequisites
- Onboard GCP Accounts for Agentless Scanning
- Onboard AWS Accounts for Agentless Scanning
- Onboard Azure Accounts for Agentless Scanning
- Pre-flight Checks
- Bulk Actions
- Other Settings

Prerequisites

To configure agentless scanning you must ensure the following requirements are met.

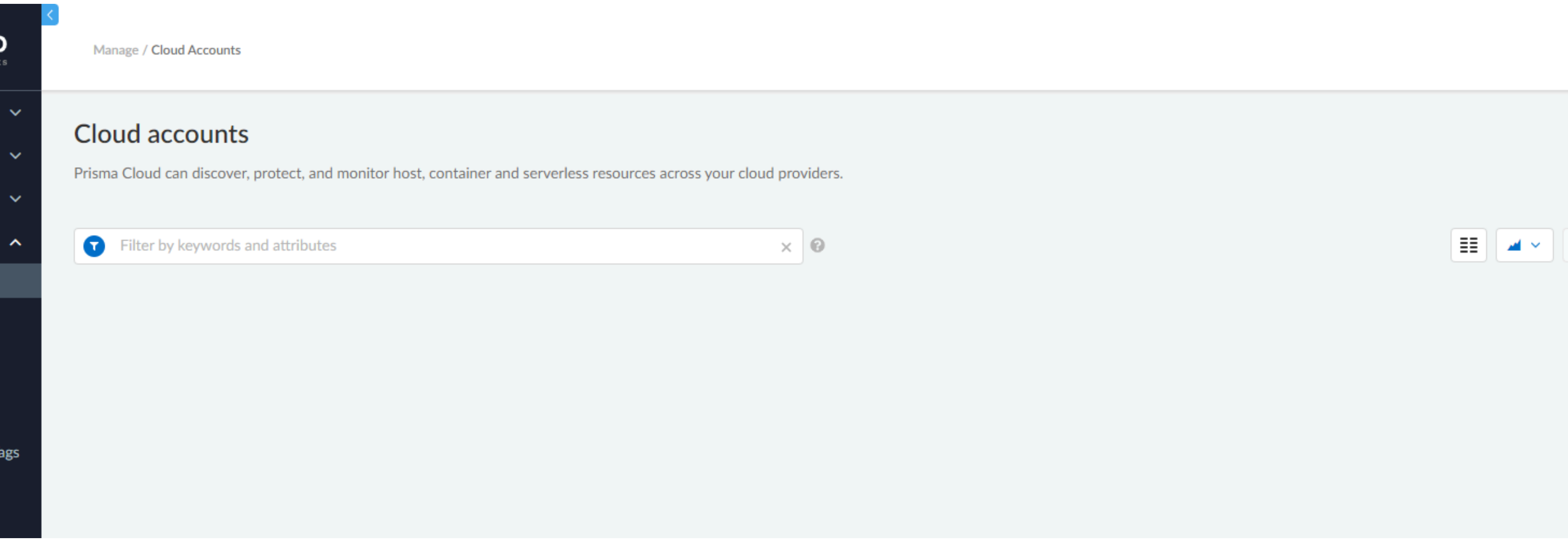
- Ensure you have permissions to create service keys and security groups in your cloud account.
- Ensure you have permissions to apply agentless permission templates to your cloud account.
- Ensure you can connect to the Prisma Cloud Console over HTTPS from your cloud account. If default security group is not available, create custom security group with custom VPC that allows connection for scanners from the account to Prisma Cloud Console.
- Unless you are using a proxy to connect to the Prisma Cloud Console, you must enable auto-assign public IPs on the subnet or security group you use to connect your cloud account to the Prisma Cloud Console.

To understand what permissions will be needed for agentless scanning, refer to our full permission list. The downloaded templates from Console add conditions around these permissions to ensure least privileged roles in your accounts. To learn more about the credentials you can use, go to our credentials store page.

Onboard GCP Accounts for Agentless Scanning

The following procedure shows the steps required to configure agentless scanning for a cloud account.

STEP 1 | Go to **Compute > Manage > Cloud accounts**.

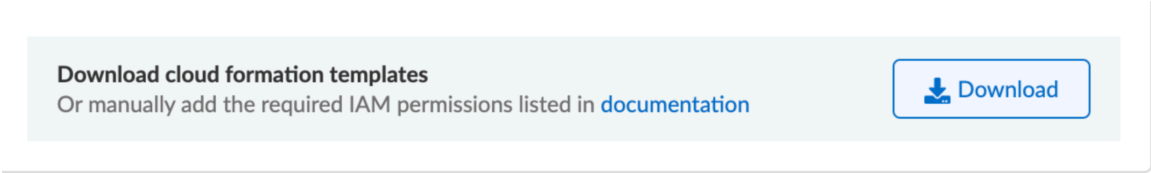


STEP 2 | Click on **Add Account** or click the **Edit** icon of an existing account.

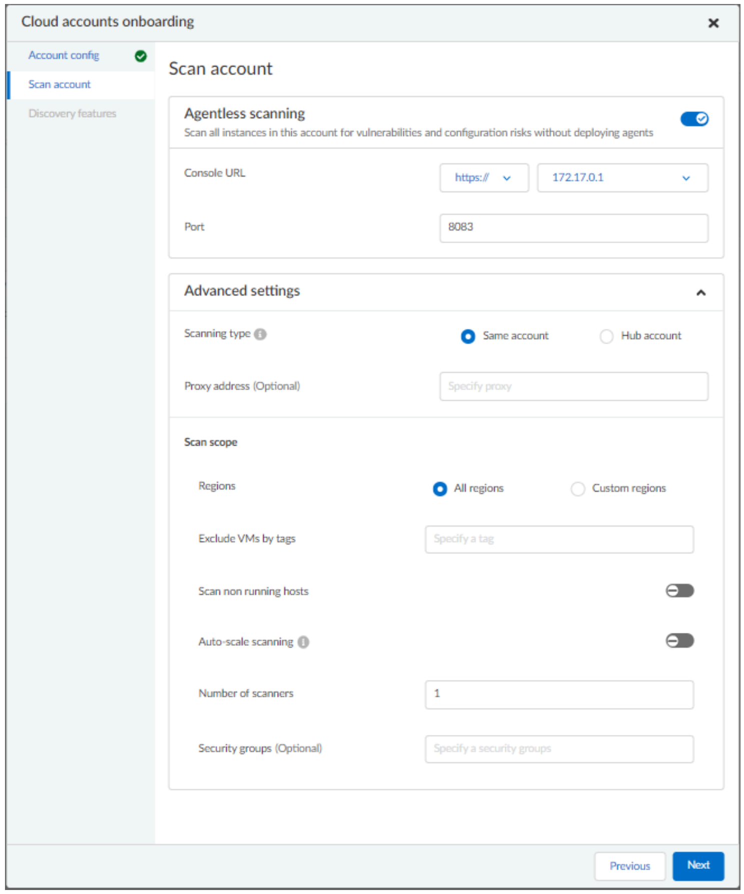
STEP 3 | Select your cloud provider.

1. GCP uses a service account and a service account key.

STEP 4 | If you are adding cloud account credentials, click the **Download** button to download its permission templates. Prisma Cloud validates the specified credentials and the download raises an error if the credentials are incorrect. To understand more about the downloaded template files and how they are used, refer to the permission templates.



STEP 5 | Review the default configuration values and make any needed changes.



1. **Console URL and Port:** Specify the Prisma Cloud Console URL and port that you will use to connect your cloud account to the Prisma Cloud Console.
2. **Scanning type:**
   1. **Same Account:** Scan hosts of a cloud account using the same cloud account.
   2. **Hub Account:** Scan hosts of a cloud account, known as the target account, using another cloud account, known as the hub account.

      For a detailed instructions for each of the scanning modes and their corresponding permission templates, refer to the scanning modes.
3. **HTTP Proxy:** To connect to the Prisma Cloud Console through a proxy, specify its URL.
4. **Regions:** Specify the regions to be scanned.
5. **Exclude VMs by tags:** Specify the tags used to ignore specific hosts. For example: *example:tag*
6. **Scan non-running hosts:** Enable to scan stopped hosts, that are not currently running.
7. **Auto-scale scanning:** When turned ON, Prisma Cloud automatically scales up / down multiple scanners for faster scans without any user-defined limits. Useful for large scale deployments.
8. **Number of scanners:** Define an upper limit to control the number of scanners Prisma Cloud can automatically spin up in your environment. Depending on the size of your

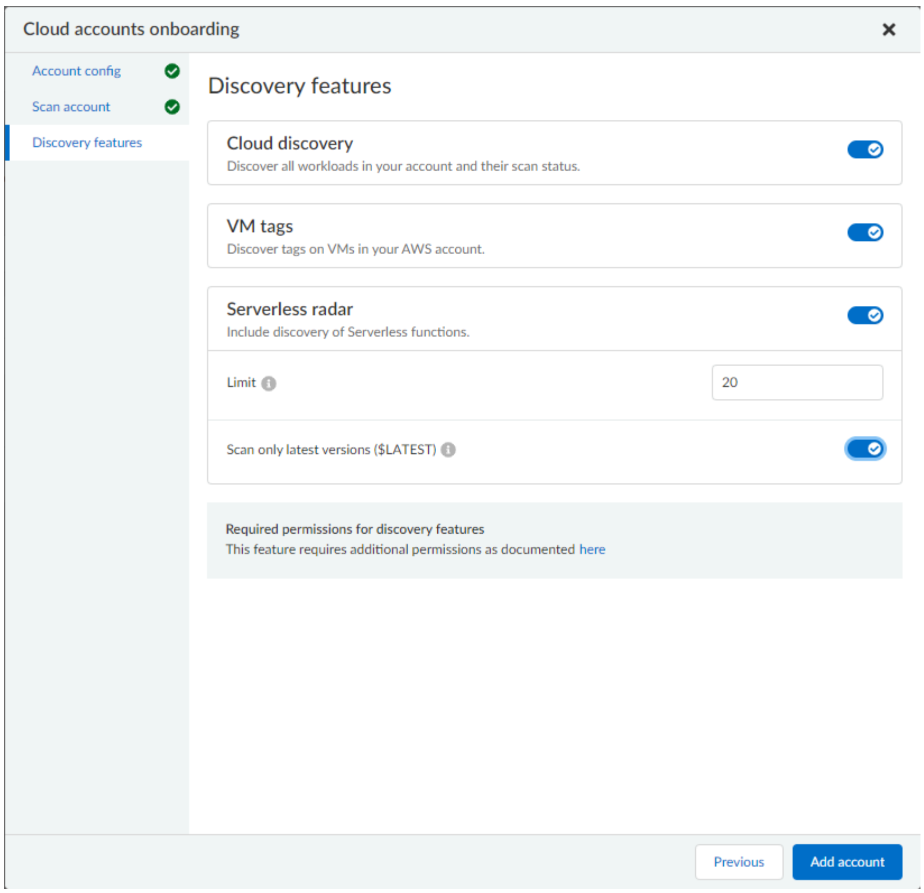
environment, Prisma cloud will scale up / down scanners within the given limit for faster scans.

9. **Security groups:**

    1. **GCP:** Subnet - If blank, Prisma Cloud uses the *default* subnet in your project to connect to the Prisma Cloud Console. If the *default* is not available, you must create and specify a custom subnet. Otherwise, the connection from your project to the Prisma Cloud Console fails and no scan results are shown.

STEP 6 | Enable or disable the **Discovery features** using the corresponding toggle.

STEP 7 | To complete the configuration, click the **Add account** button for new cloud accounts or the **Save** button for existing cloud accounts.



Onboard AWS Accounts for Agentless Scanning

Prisma Cloud gives you the flexibility to choose between agentless security and agent-based security using Defenders. Agentless scanning lets you inspect the risks and vulnerabilities of a virtual machine without installing an agent or affecting the execution of the instance. Prisma Cloud supports agentless scanning for vulnerabilities and compliance on AWS hosts. Agentless scanning for containers and clusters is in development. To learn more about how agentless scanning works, go to the How Agentless Scanning Works? page.

To onboard AWS account for agentless scanning you need to complete three tasks.

1. Add an AWS credential to the Prisma Cloud Compute Console.

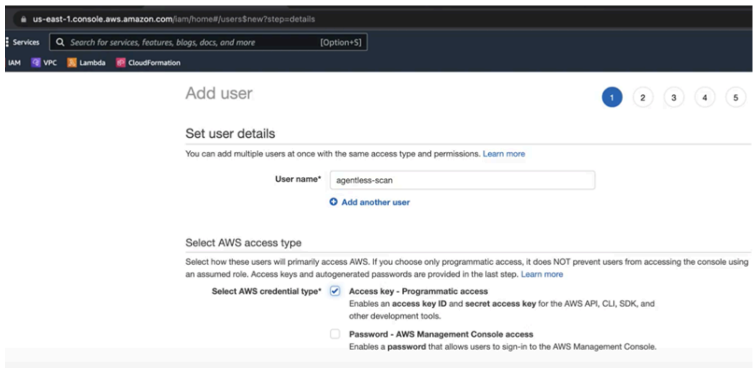
2. Apply the agentless scanning permission templates to the account for scanning.
3. Create a security group to connect your AWS account and the Prisma Cloud Console.

Add an AWS Credential to the Prisma Cloud Compute Console

Authenticate your AWS account using its IAM users for agentless scanning. Agentless scanning in AWS only supports IAM users that are using an access key for authentication. An access key consists of an access key ID and a secret key. Create an IAM user in AWS to serve as an identity that represents a person or service interacting with AWS. You must use both the access key ID and secret access key of an access key together to authenticate requests with AWS. For more detailed information on how to create and maintain IAM users, go to the AWS documentation.

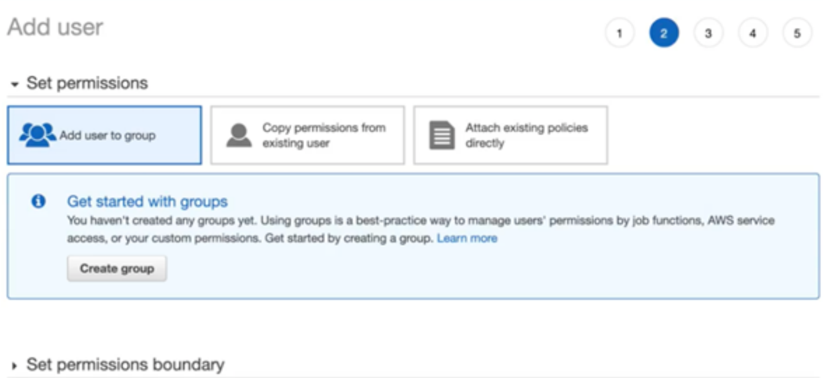
STEP 1 | Go to the IAM page for your AWS account at: https://console.aws.amazon.com/iam/

STEP 2 | Click Add user.



STEP 3 | Enter a user name and enable **Access key - Programmatic access**. Agentless scanning uses this access to call the APIs and scan your AWS account.

STEP 4 | Click **Next** to go to the **Set permissions** page.

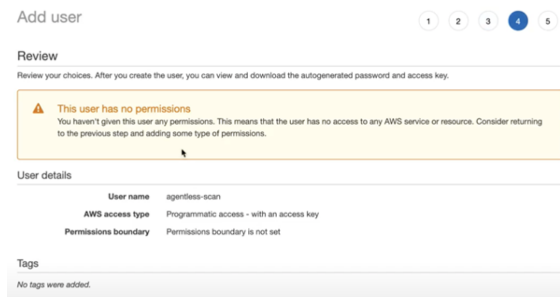


STEP 5 | Skip the **Set permissions** page. You can get the needed permission templates after validating your credentials in the Prisma Cloud Console. Click **Next**.

STEP 6 | Add tags as needed but no tags are needed for agentless scanning.

STEP 7 | Click **Review**.

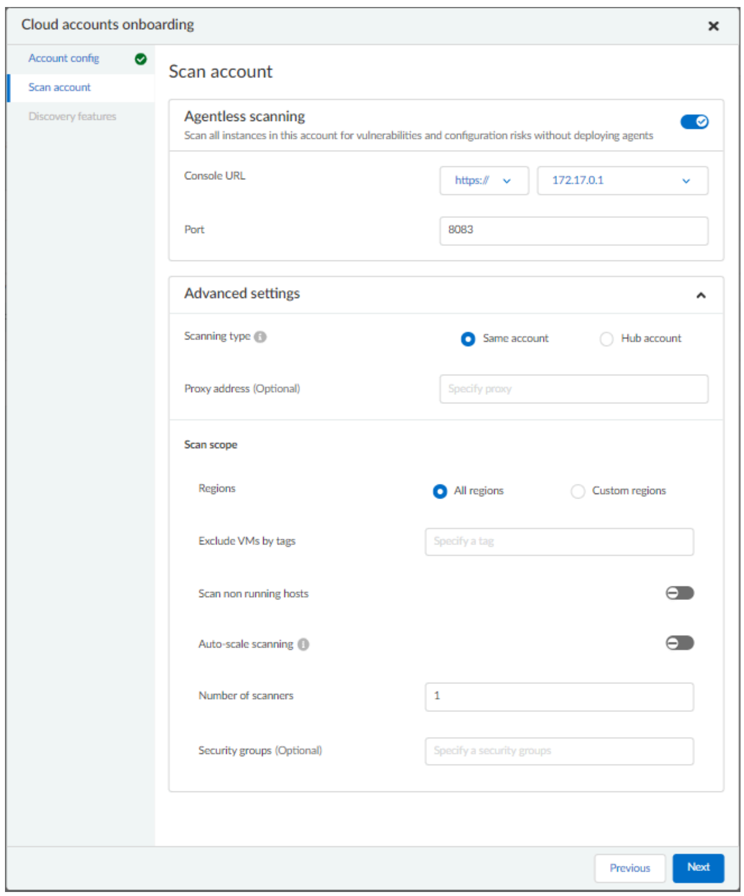
STEP 8 | Ignore the "This user has no permissions" warning and click **Create user**.



STEP 9 | Copy the **Access Key ID** and **Secret Key** from the AWS Console for this newly created user. You need to add this information when adding the credential to Prisma Cloud Compute Console.

STEP 10 | Go to the Prisma Cloud Compute Console.

STEP 11 | Go to **Manage > Cloud Accounts > Add Account**.



STEP 12 | Select AWS as the cloud provider and Access Key as the authentication type.

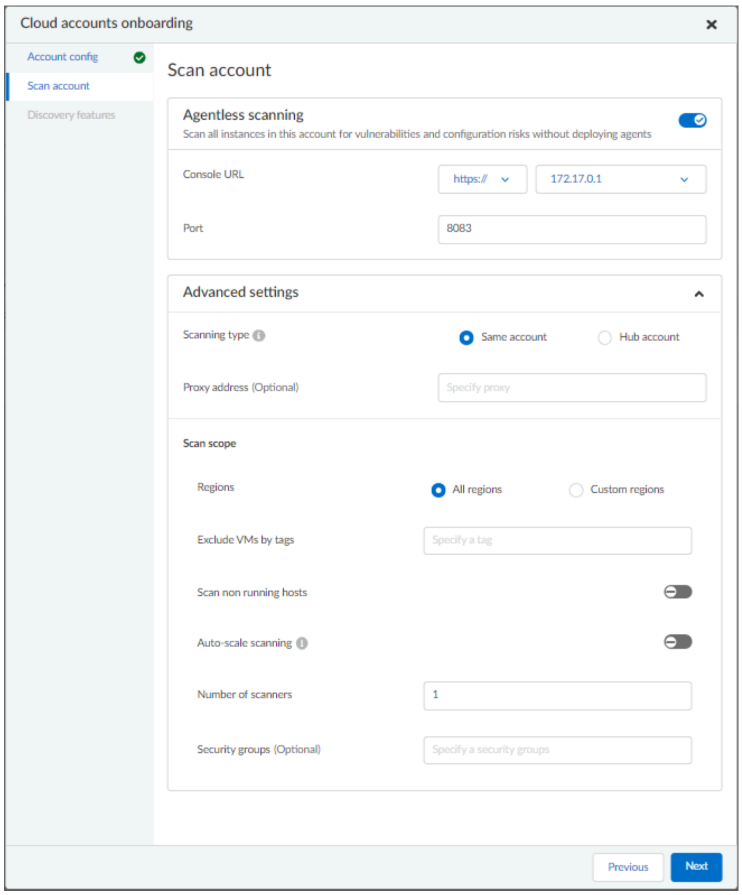
STEP 13 | Paste the Access key and Secret key for the newly created user that you copied from the AWS Console.

Following AWS best practices, you should rotate your keys every 90 days. Prisma Cloud raises an Alert when the age of the added credentials is greater than 90 days. If you follow this practice, rotate your keys at least every 90 days and update the credential in the Prisma Cloud Console.

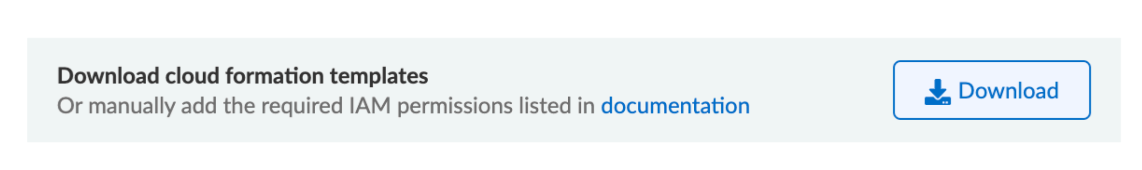
Apply the Agentless Scanning Permission Templates

After adding credentials for your AWS cloud account to the Prisma Cloud Compute Console, you need to configure agentless scanning.

STEP 1 | After adding the AWS IAM credential, click **Next** in the cloud account set up of the Prisma Cloud Compute Console.



STEP 2 | In the **Agentless scanning** tab, click the **Download** button to download agentless permission templates.



When you click Download the Prisma Cloud Console performs the following actions:

1. Validates the specified credentials and the download raises an error if the credentials are incorrect.
2. Multiple permission templates are downloaded as JSON files.

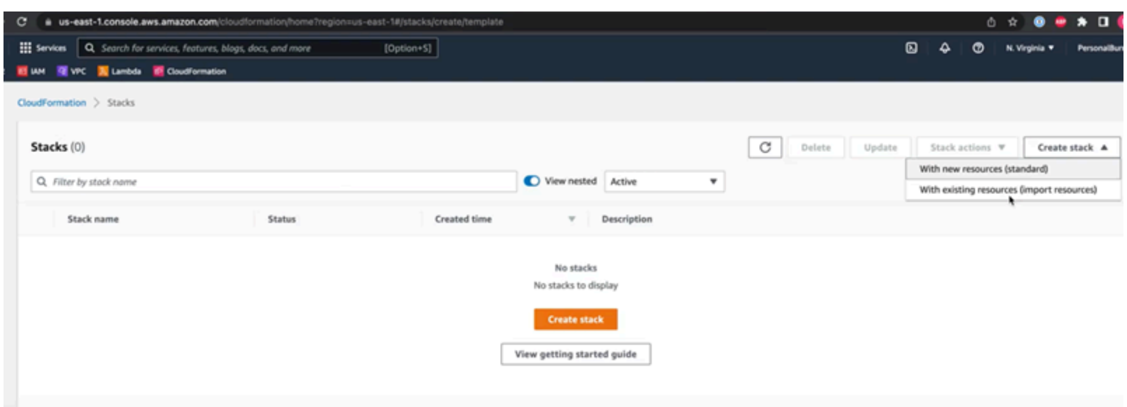
   The permission templates provide the permissions required by each cloud account for each of the scanning modes. Learn more about the permission included in the downloaded template files and how they are used in the permissions by feature. You can scan AWS accounts using the same account or the hub account scanning modes. If you want to use an existing AWS account, you can use the awsAgentlessPermissions.json permissions template to grant it the needed permissions.

Same Account Mode

Using the same account scanning mode, you scan all hosts of a cloud account belonging to the same AWS cloud account. This scanning mode keeps the snapshots within the same AWS account where the hosts run and spins up the scanners using that same account.

STEP 1 | To scan accounts using this mode, you apply the permission template ending in *_target_user_permissions.json* to the AWS cloud account. For detailed instructions on how to apply cloud formation templates, refer to the AWS documentation.

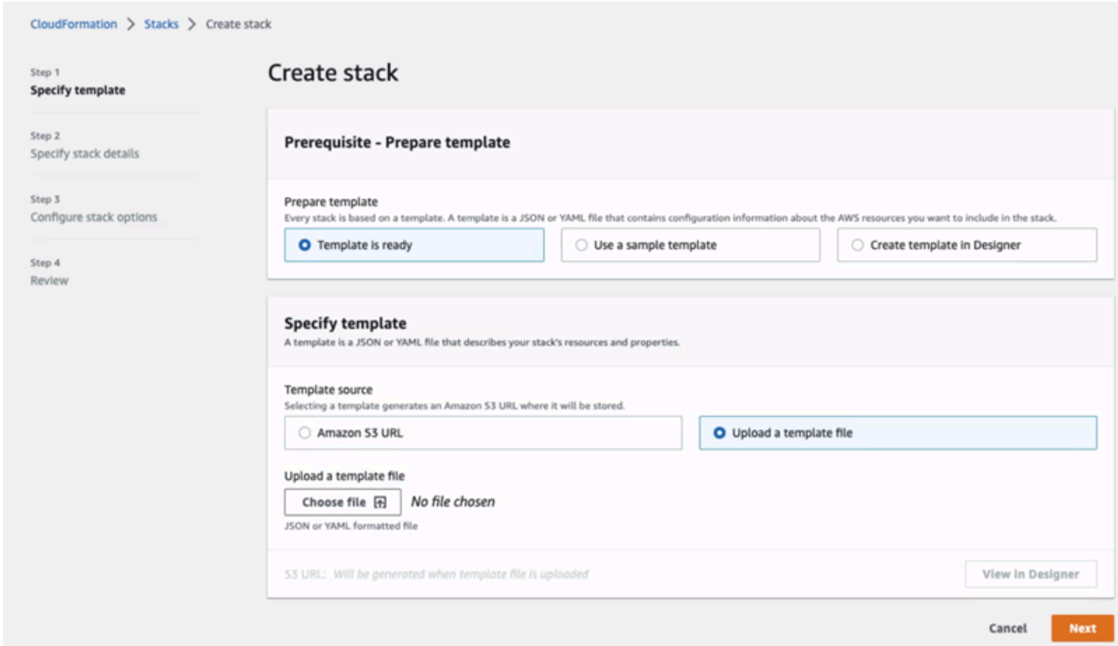
STEP 2 | Go to the AWS CloudFormation console for your account.



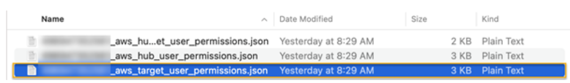
STEP 3 | Click the **Create stack** dropdown in the top right corner and select the **With new resources** option.

STEP 4 | Click the **Create stack** button.

1. Select the **Template is ready** and **Upload template file** options.

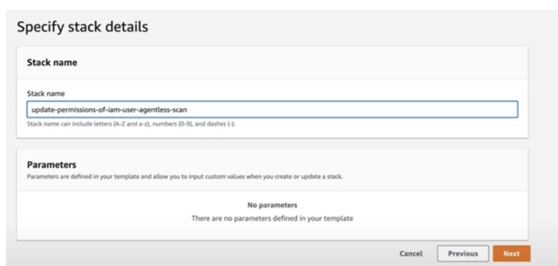


2. Under **Upload a template file**, click the **Chose file** button. Select the template that you downloaded from the Prisma Cloud Compute Console for agentless scanning ending with *_target_user_permissions.json*.



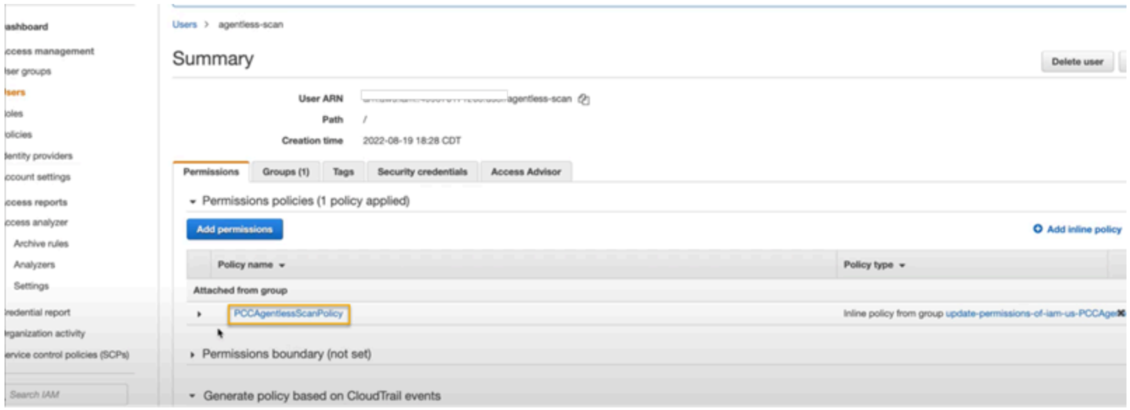
3. Click **Next**.

STEP 5 | Enter a **Stack name** for the agentless scanning IAM user you created.



STEP 6 | Click **Next** and use the default values in the following screens until you reach the final **Create Stack** page.

STEP 7 | Verify that the IAM user has the permissions applied. The permissions appear as *PCCAgentlessScanPolicy* in the **Permissions** tab for the IAM user.



Hub Account Mode

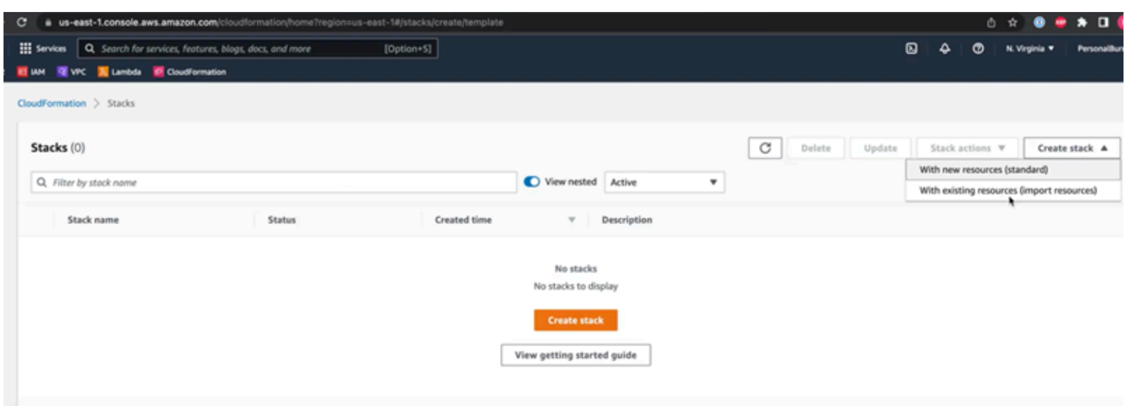
Using the hub account scanning mode, you scan all hosts in one or more cloud accounts, which are called target accounts, from another dedicated cloud account. This dedicated cloud account is called a hub account and it spins up the agentless scanners. To use the hub account mode, you must complete the following steps.

1. Add an AWS account to use as the hub account for agentless scanning to your Prisma Cloud Compute Console.
2. Add the AWS account or accounts that you want to scan using Prisma Cloud agentless scanning.

Add the Hub Account

STEP 1 | To add a hub account, apply the permission template ending in *_hub_user_permissions.json* to the AWS cloud account. For detailed instructions on how to apply cloud formation templates, refer to the AWS documentation.

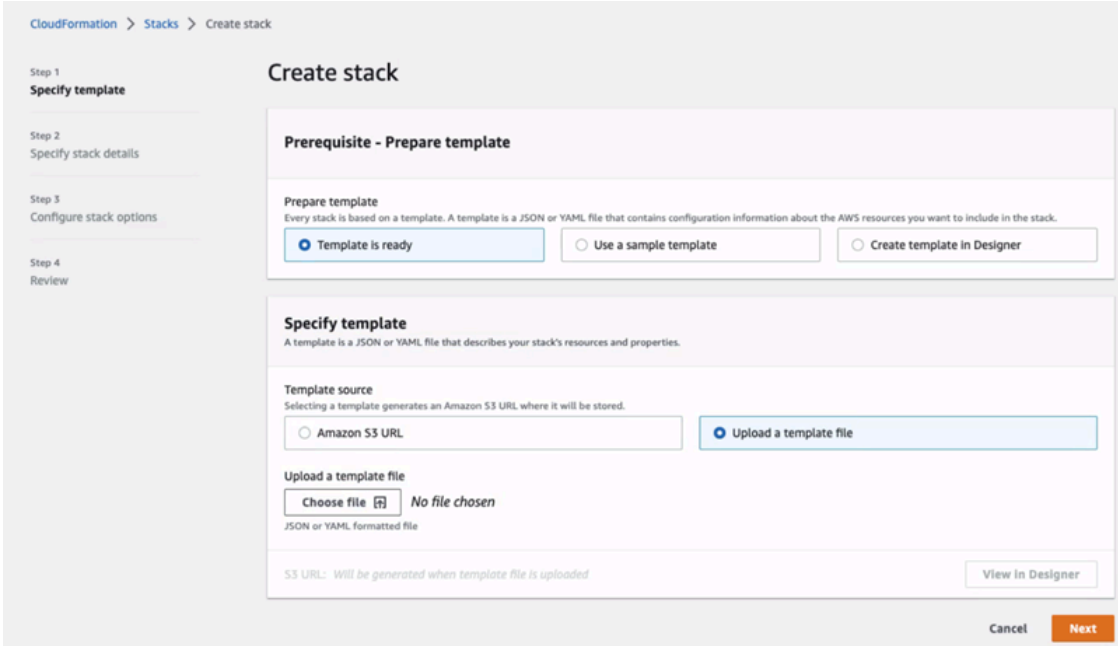
STEP 2 | Go to the AWS CloudFormation console for your account.



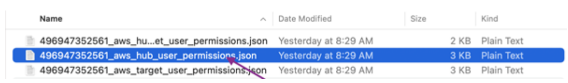
STEP 3 | Click the **Create stack** dropdown in the top right corner and select the **With new resources** option.

STEP 4 | Click the **Create stack** button.

1. Select the **Template is ready** and **Upload template file** options.

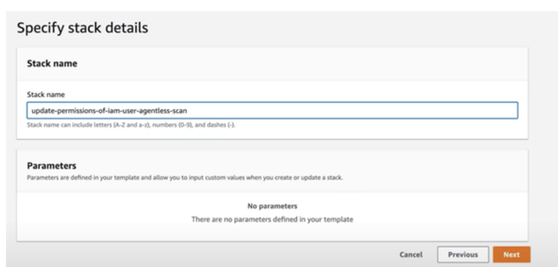


2. Under **Upload a template file**, click the **Chose file** button. Select the template that you downloaded from the Prisma Cloud Compute Console for agentless scanning ending with *_hub_user_permissions.json*.



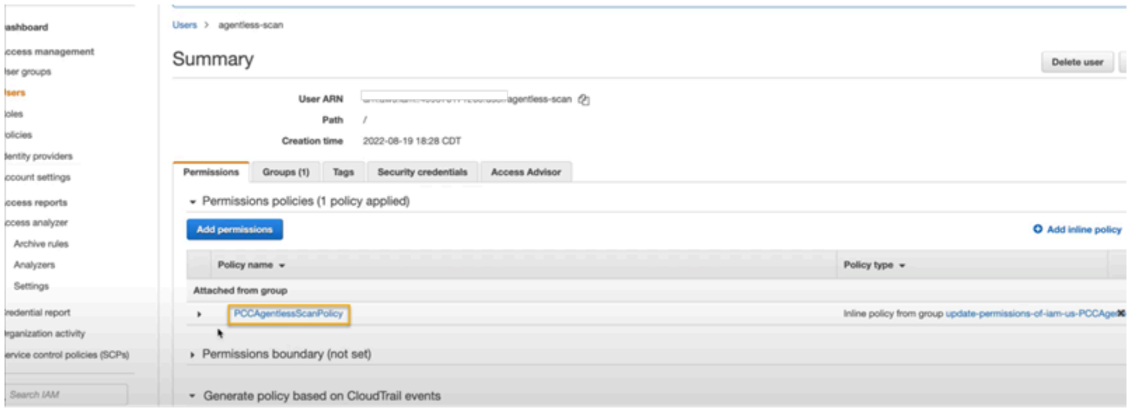
3. Click **Next**.

STEP 5 | Enter a **Stack name** for the agentless scanning IAM user you created.



STEP 6 | Click **Next** and use the default values in the following screens until you reach the final **Create Stack** page.

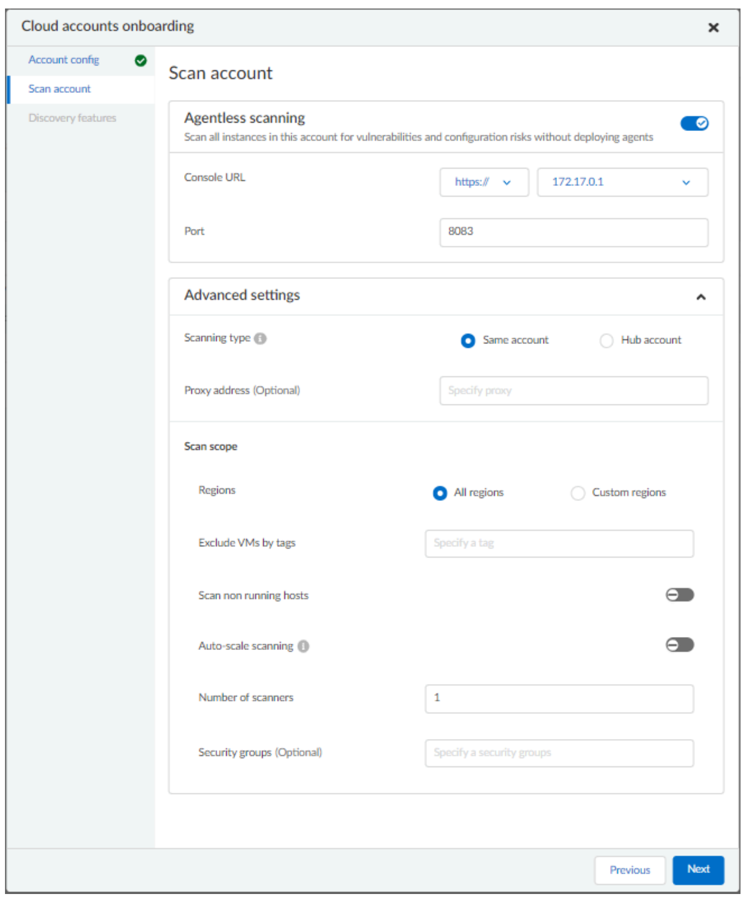
STEP 7 | Verify that the IAM user has the permissions applied. The permissions appear as *PCCAgentlessScanPolicy* in the **Permissions** tab for the IAM user.



When you add hub account credentials to the Prisma Cloud Console, you can turn off agentless scanning in the hub account unless you want to scan all hosts in that account as well. If that is the case, you must add the target user permissions to the hub account in addition to the hub account permissions.

STEP 8 | Go to the Prisma Cloud Compute Console.

STEP 9 | Go to **Manage** > **Cloud Accounts** > **Add Account**.



STEP 10 | Select AWS as the cloud provider and Access Key as the authentication type.

STEP 11 | Paste the Access key and Secret key for the newly created user that you copied from the AWS Console.

Following AWS best practices, you should rotate your keys every 90 days. Prisma Cloud raises an Alert when the age of the added credentials is greater than 90 days. If you follow this practice, rotate your keys at least every 90 days and update the credential in the Prisma Cloud Console.

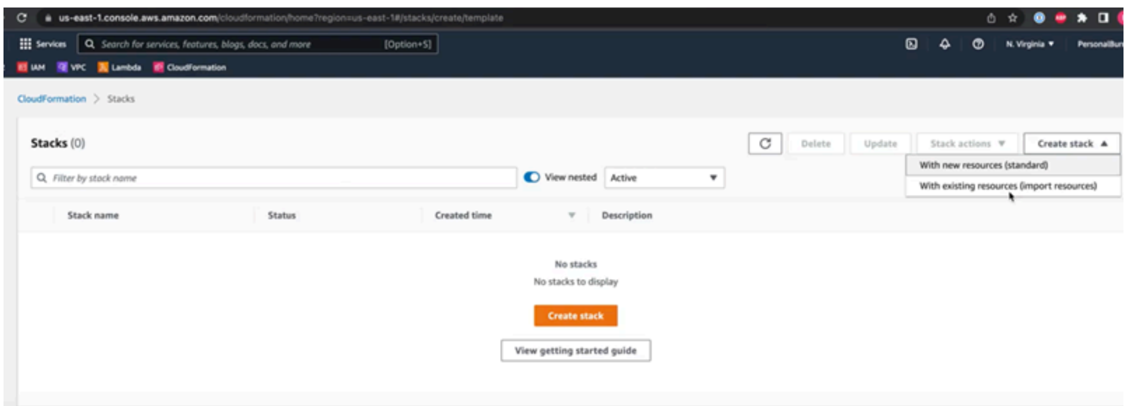
STEP 12 | Once you add the hub account to Prisma Cloud, you can then add the target accounts.

Add your Target Accounts

STEP 1 | To add a target account, you apply the permission template ending in *_target_user_permissions.json* to the AWS cloud account. For detailed instructions on how to apply cloud formation templates, refer to the AWS documentation.

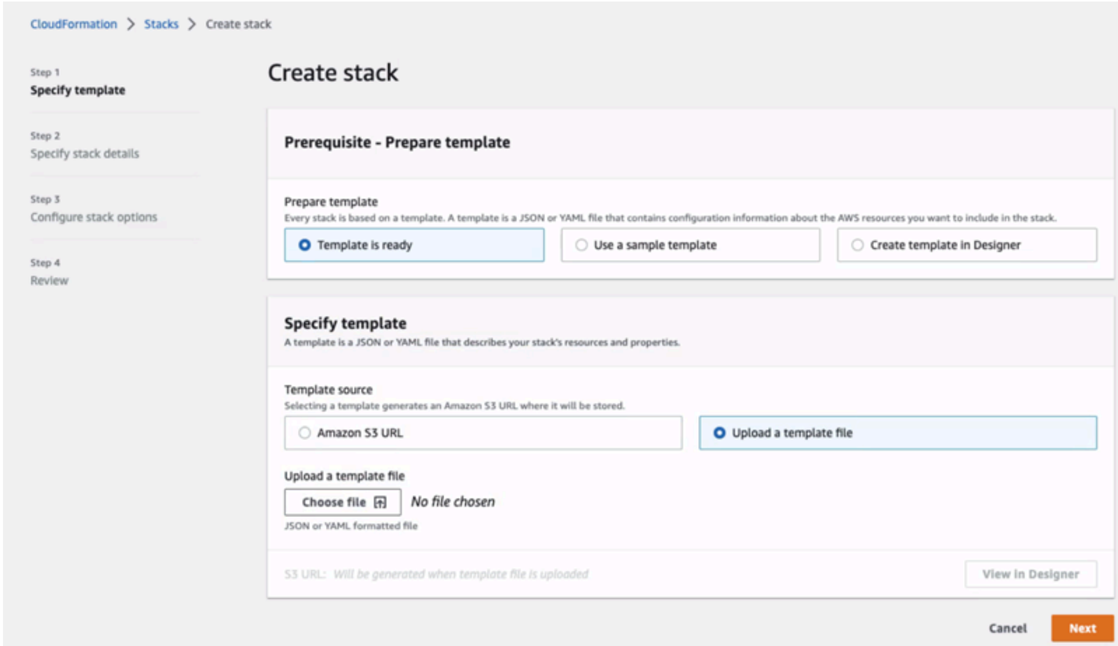
STEP 2 | Go to the AWS CloudFormation console for your account.

STEP 3 | Click the **Create stack** dropdown in the top right corner and select the **With new resources** option.

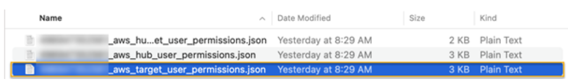


STEP 4 | Click the **Create stack** button.

1. Select the **Template is ready** and **Upload template file** options.

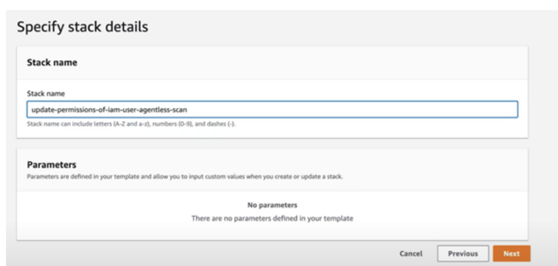


2. Under **Upload a template file**, click the **Chose file** button. Select the template that you downloaded from the Prisma Cloud Compute Console for agentless scanning ending with *_target_user_permissions.json*.



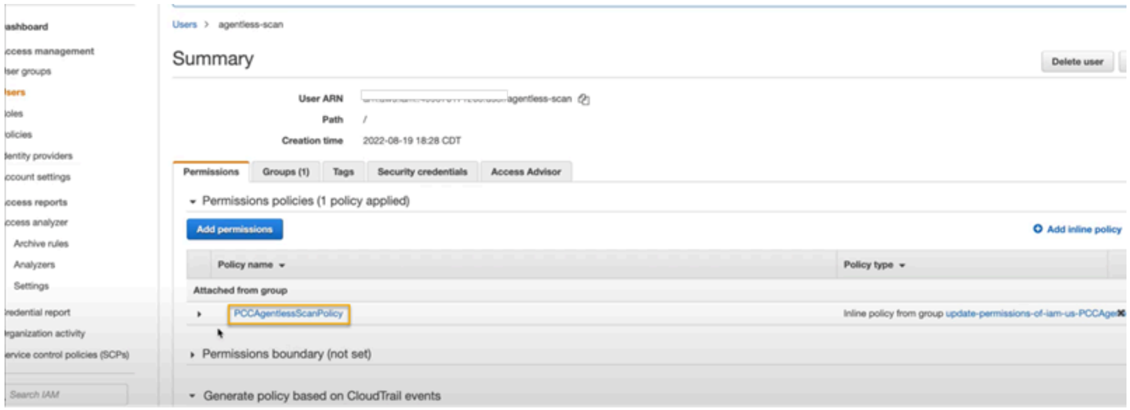
3. Click **Next**.

STEP 5 | Enter a **Stack name** for the agentless scanning IAM user you created.



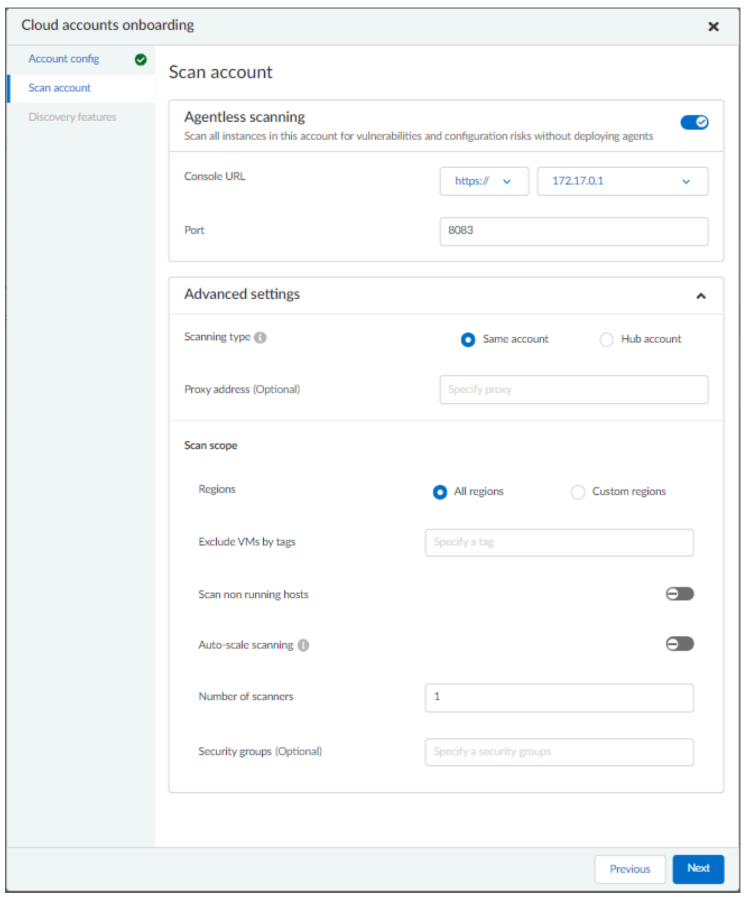
STEP 6 | Click **Next** and use the default values in the following screens until you reach the final **Create Stack** page.

STEP 7 | Verify that the IAM user has the permissions applied. The permissions appear as *PCCAgentlessScanPolicy* in the **Permissions** tab for the IAM user.



STEP 8 | Go to the Prisma Cloud Compute Console.

STEP 9 | Go to **Manage > Cloud Accounts > Add Account**.



STEP 10 | Select **AWS** as the cloud provider and **Access Key** as the authentication type.

STEP 11 | Paste the **Access key** and **Secret key** for the newly created user that you copied from the AWS Console.

> *Following AWS best practices, you should rotate your keys every 90 days. Prisma Cloud raises an Alert when the age of the added credentials is greater than 90 days. If you follow this practice, rotate your keys at least every 90 days and update the credential in the Prisma Cloud Console.*

STEP 12 | In the Agentless scanning tab, select the **Hub Account** option as the **Scanning type**.

STEP 13 | Select the hub account you want to use from the dropdown menu.

STEP 14 | Click **Next** to connect your AWS account with the Prisma Cloud Console.

Connect your AWS account with the Prisma Cloud Console

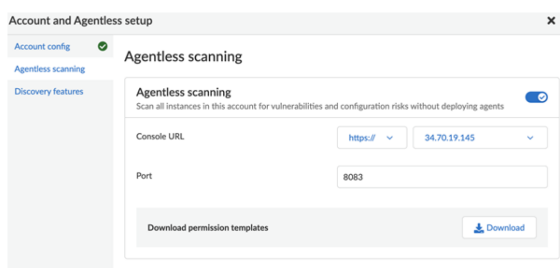
Prisma Cloud looks for the *default* security group that AWS creates to connect your AWS account to the Prisma Cloud Console for scanning. If the *default* security group is not available, you must create and specify a custom security group. Otherwise, the connection from your AWS account to the Prisma Cloud Console fails and no scan results are shown.

If you use the hub account scanning mode, you only need to create a security group in the hub account and not on each target account because the hub account is the only one that spins up the scanners. Complete the following steps to create the needed security group if the *default* is unavailable.

STEP 1 | Follow AWS instructions for creating a custom security group in the Amazon VPC Console.

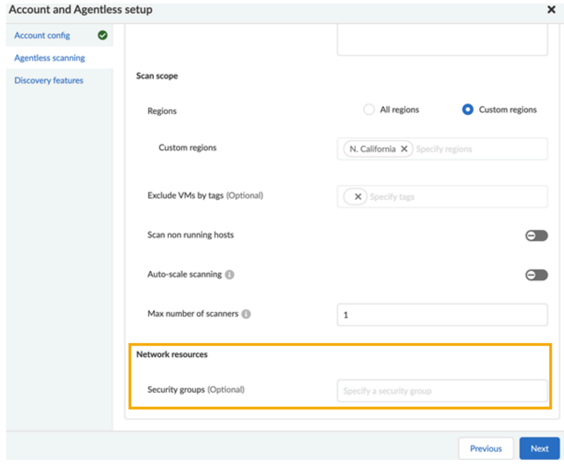
STEP 2 | Allow outbound connections to the Prisma Cloud Compute Console IP address and port. Complete these steps to find these values.

1. Go to the Prisma Cloud Console.
2. Go to **Manage > Cloud accounts**.
3. In the* Agentless scanning* tab, you can find the **Console URL** and **Port**.



STEP 3 | In the **Agentless scanning** tab, go to the **Advanced settings**.

STEP 4 | Enter the name of the **Security group** you created under **Network resources**.



STEP 5 | Set the advanced settings: The agentless scanning advanced settings allow you to make the following changes to the configuration to better suit your needs.

- **Console URL and Port**: Specify the Prisma Cloud Console URL and port that you use to connect your cloud account to the Prisma Cloud Console.
- **Scanning type**:
  - **Same Account**: Scan hosts of a cloud account using that same cloud account.
  - **Hub Account**: Scan hosts of a cloud account, known as the target account, using another cloud account, known as the hub account.
- **HTTP Proxy**: To connect to the Prisma Cloud Console through a proxy, specify the proxy's URL.
- **Regions**: Specify the regions you want to scan.
- **Exclude VMs by tags**: Specify the tags used to ignore specific hosts. For example: *example:tag*
- **Scan non-running hosts**: Enable to scan stopped hosts that are not currently running.
- **Auto-scale scanning**: When turned **ON**, Prisma Cloud automatically scales multiple scanners up or down for faster scans without any user-defined limits. Use this feature for large scale deployments.
- **Number of scanners**: Define an upper limit to control the number of scanners Prisma Cloud can automatically spin up in your environment. Depending on the size of your environment, Prisma cloud scales scanners up or down within the given limit for faster scans.
- **Security groups**: In AWS, you can enter a security group name
- **Cloud Discovery**: Use the toggle to enable or disable the cloud discovery features.

STEP 6 | Click the **Add account button** for new cloud accounts or the **Save button** for existing cloud accounts to complete the configuration.

Onboard Azure Accounts for Agentless Scanning

Agentless scanning lets you inspect the risks and vulnerabilities of a virtual machine without having to install an agent or affecting the execution of the instance. Prisma Cloud gives you the flexibility to choose between agentless and agent-based security using Defenders. Currently, Prisma Cloud supports agentless scanning on Azure hosts (containers and clusters coming soon next release) for vulnerabilities and compliance. To learn more about how agentless scanning works, refer to our article on Agentless scanning architecture.

This guide enables Agentless scanning for Prisma Cloud Compute Edition (PCCE or self-hosted) in Azure. The procedure shows you how to complete the following tasks.

STEP 1 | Create a role and a service principal in Azure.

STEP 2 | Configure agentless scanning in the Prisma Cloud console.

STEP 3 | Scan for vulnerabilities.

Create a Role and a Service Principal in Azure

STEP 1 | Log in to Azure with the Azure CLI.

STEP 2 | Download the azureAgentlessPermissions.json file.

STEP 3 | Determine your *subscriptionId* with the following Azure CLI command.

```
az account subscription list
```

STEP 4 | Replace *<subscriptionId>* in the *azureAgentlessPermissions.json* file with your Azure *subscriptionId*. You can find the field under the *"AssignableScopes": ["/subscriptions/ <subscriptionId>"]* element.

STEP 5 | Create the role using the JSON file with the following Azure CLI command.

```
az role definition create --role-definition
 azureAgentlessPermissions.json
```

STEP 6 | Create a service principal account with the following Azure CLI command.

```
az ad sp create-for-rbac --name PCEE-Agentless --role "Prisma Cloud
 Compute Agentless Scanner" --scope /subscriptions/<subscriptionId>
 --sdk-auth
```

STEP 7 | Copy and save the returned JSON object for the service principal, for example:

```
{
  "clientId": "<clientId>",
  "clientSecret": "<clientSecret>",
  "subscriptionId": "<subscriptionId>",
  "tenantId": "<tenantId>",
  "activeDirectoryEndpointUrl": "https://
login.microsoftonline.com",
  "resourceManagerEndpointUrl": "https://management.azure.com/",
  "activeDirectoryGraphResourceId": "https://graph.windows.net/",
  "sqlManagementEndpointUrl": "https://
management.core.windows.net:8443/",
  "galleryEndpointUrl": "https://gallery.azure.com/",
```



```
  "managementEndpointUrl": "https://management.core.windows.net/"
}
```

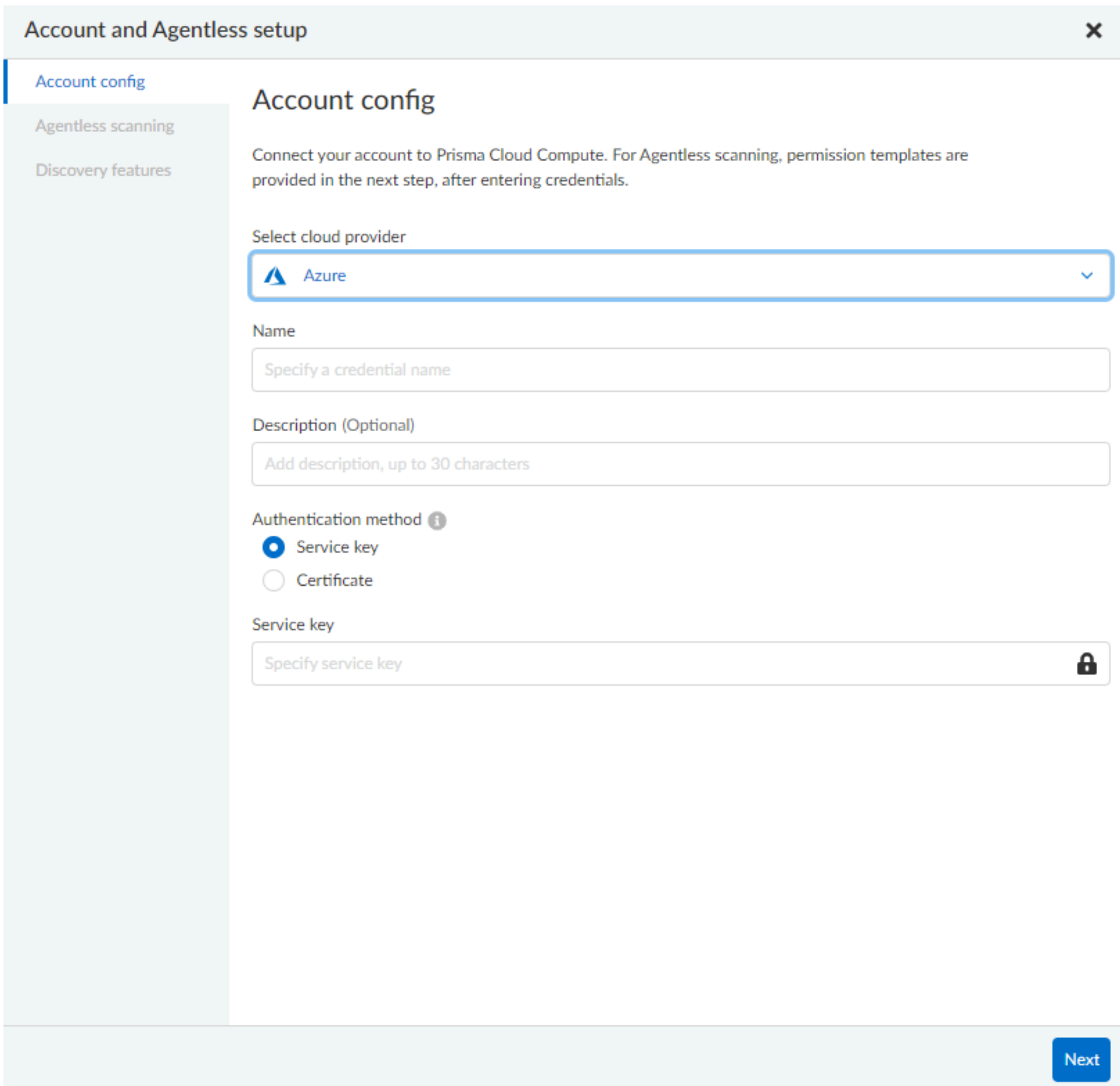
Configure Agentless Scanning in the Prisma Cloud Console

STEP 1 | Log in to your Prisma Cloud Compute Console.

STEP 2 | Go to **Manage > Cloud** Accounts.

STEP 3 | Click **+Add account**.

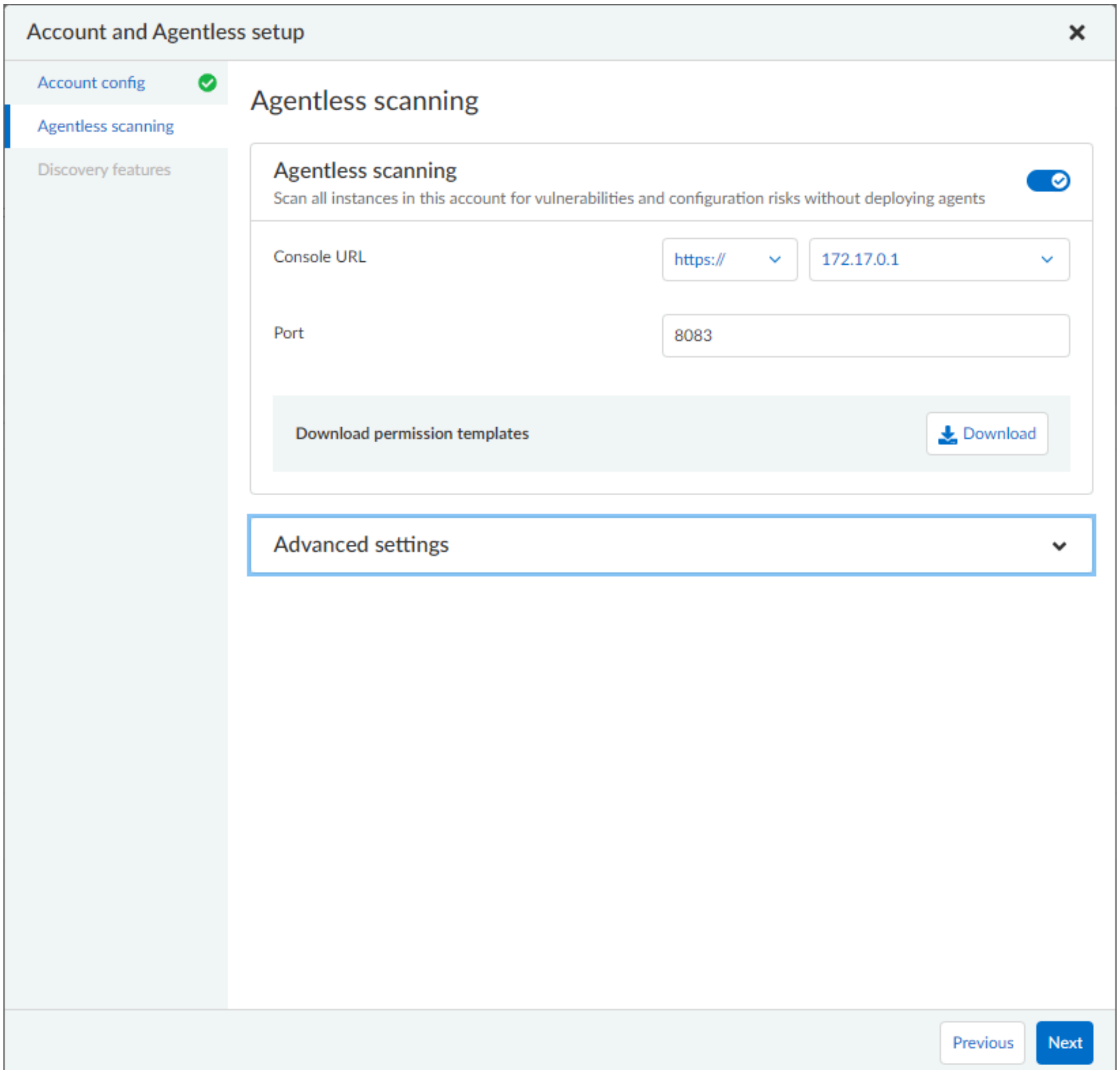
STEP 4 | Enter the needed information in the **Account config** pane.



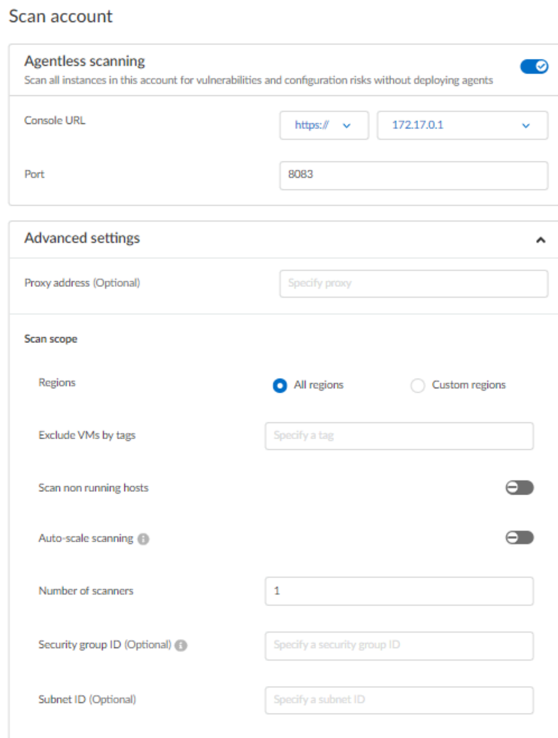
1. **Select Cloud provider**: Azure
2. **Name:** For example: PCC Azure Agentless
3. **Description:** Provide an optional string, for example: Kepler release
4. **Authentication method:** Service key
5. **Service Key:** Paste the JSON object for the service principal you created.

STEP 5 | Click Next.

STEP 6 | Complete the configuration in the **Scan account** pane:



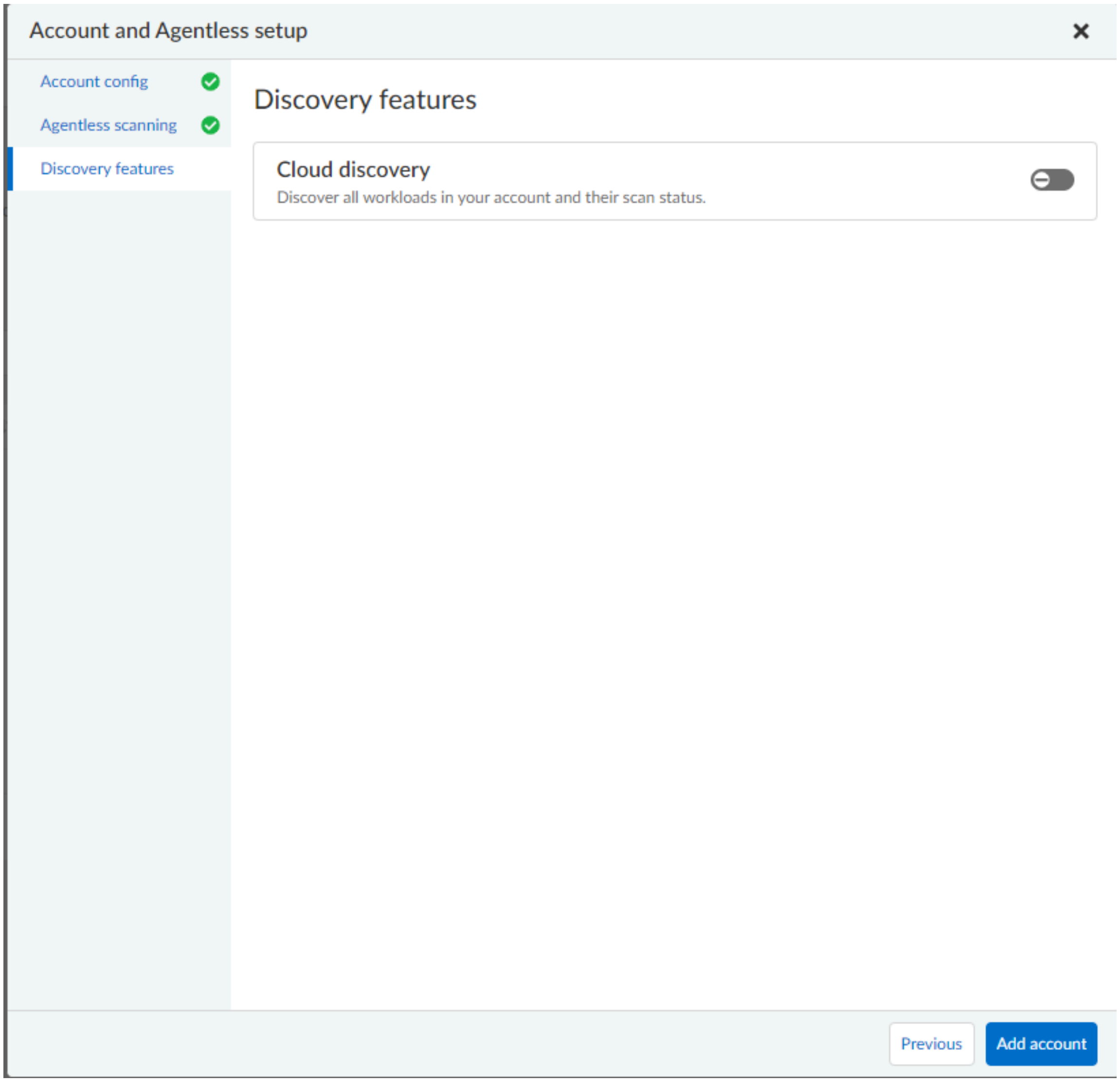
1. Enable **Agentless scanning**.
2. Set the **Console URL** and **Port** to the address of your Prisma Cloud console that can be reached from the internet. To create an address or FQDN reachable from the internet, complete the Subject Alternative Names procedure.
3. Expand* Advanced settings*.



1. If you use a proxy for traffic leaving your Azure tenant, enter the **Proxy** address and add it's Certificate Authority certificate.
2. Under **Scan scope** you can choose **All regions** to scan for VMs in all Azure regions. If you choose **Custom regions**, enter the Azure region in which you want Prisma Cloud to scan for VMs.
3. Enter tags under **Exclude VMs by tags** to further limit the scope of the scan.
4. Choose whether or not to **Scan non running hosts**
5. Choose whether or not to enable **Auto-scale scanning**. If you disable auto-scale, specify number of scanners Prisma Cloud should employ.
6. Enter the **Security group ID** and **Subnet ID** that are created to allow the Prisma Cloud console to communicate back with Azure.

STEP 7 | Click **Next**.

STEP 8 | In the **Discovery features** pane, disable **Cloud discovery**.



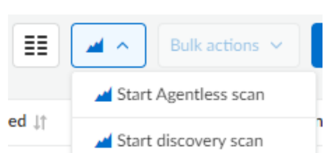
STEP 9 | Click **Add account**.

Scan for Vulnerabilities

STEP 1 | Go to **Manage > Cloud accounts.**

STEP 2 | Click the scan icon on the top right corner of the accounts table.

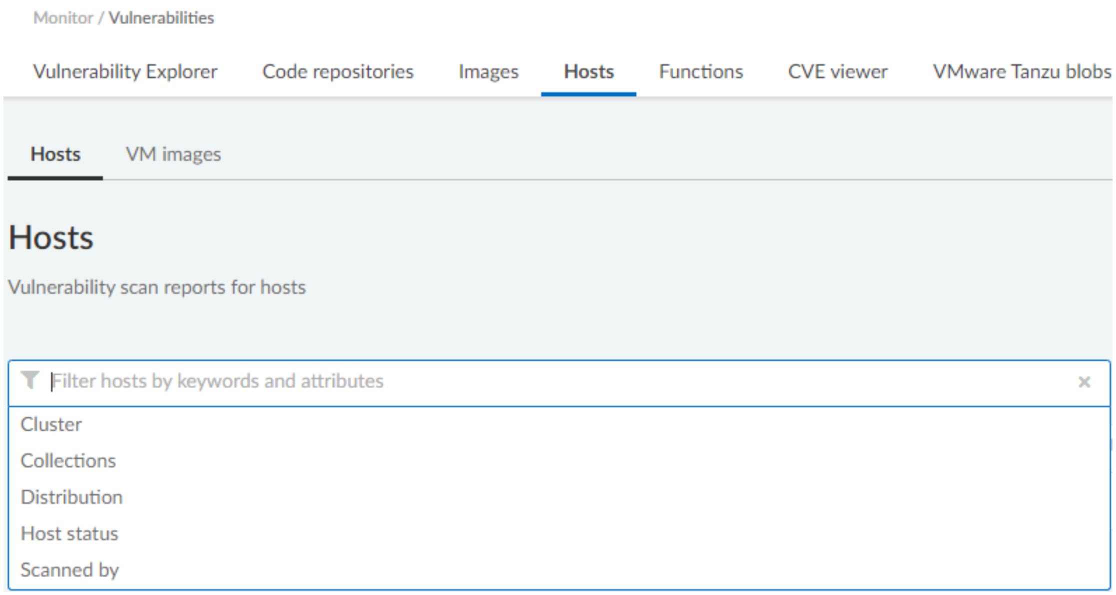
STEP 3 | Click Start Agentless scan



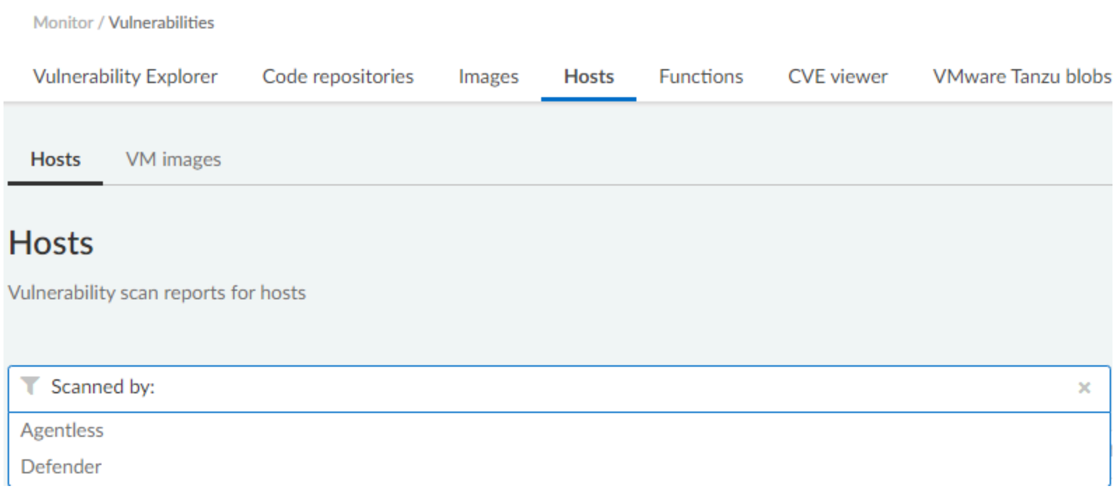
STEP 4 | Click the scan icon in the top right corner of the console to view the scan status.

STEP 5 | View the results.

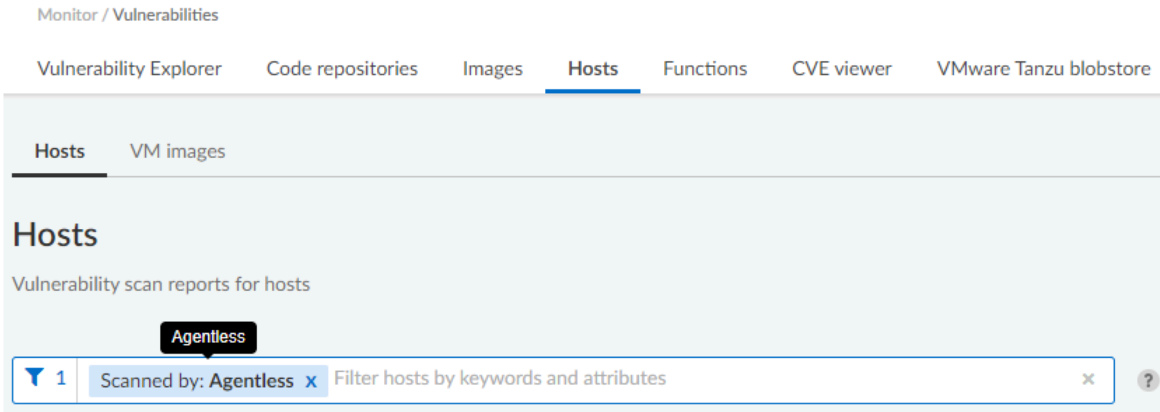
1. Go to **Monitor > Vulnerabilities > Hosts**.
2. Click on the **Filter hosts** text bar.



3. Select the **Scanned by** filter.

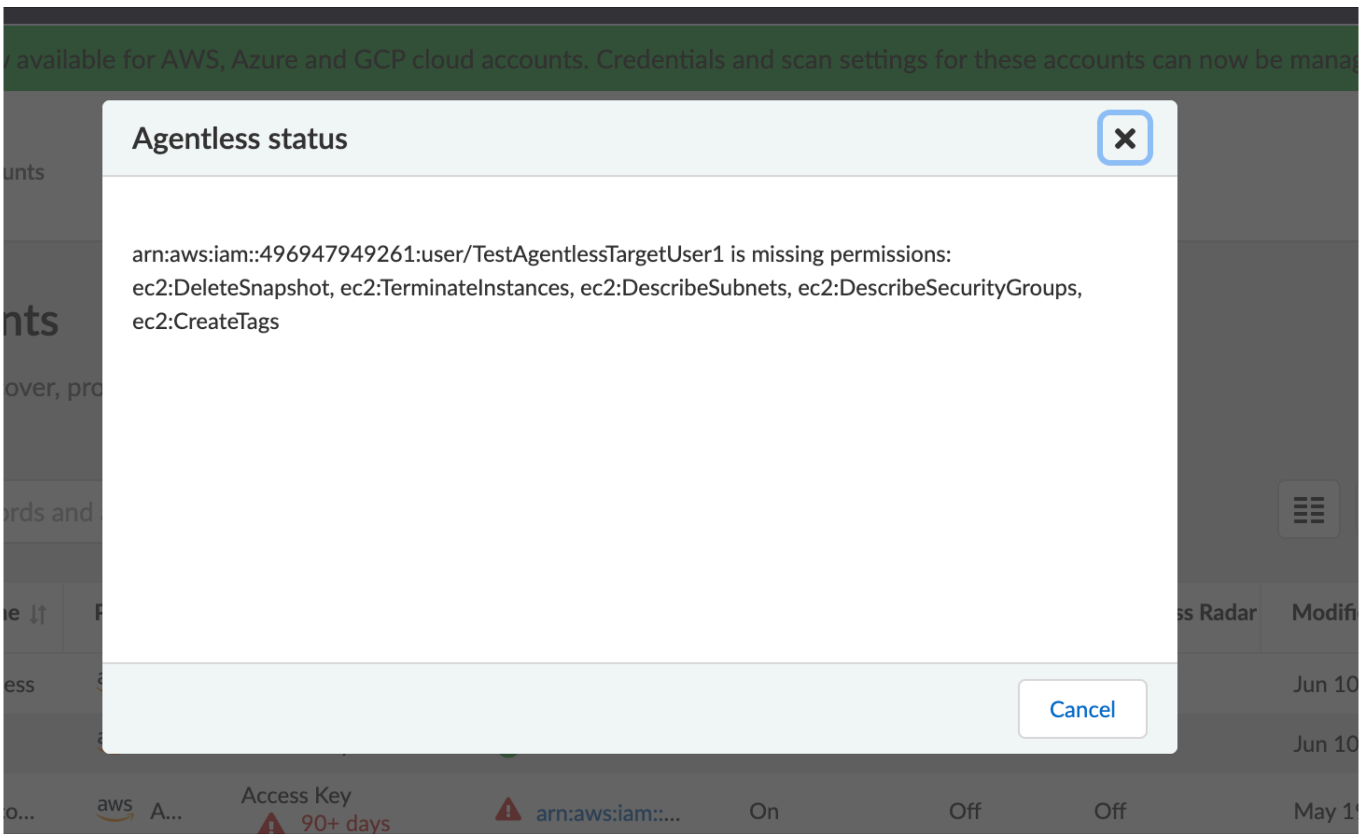


4. Select the **Agentless** filter.



Pre-flight Checks

Before scanning, Prisma Cloud performs pre-flight checks and shows any missing permissions. You can see the status of the credentials without waiting for the scan to fail. This gives you proactive visibility into errors and missing permissions allowing you to fix them to ensure successful scans. The following image shows the notification of a missing permission.



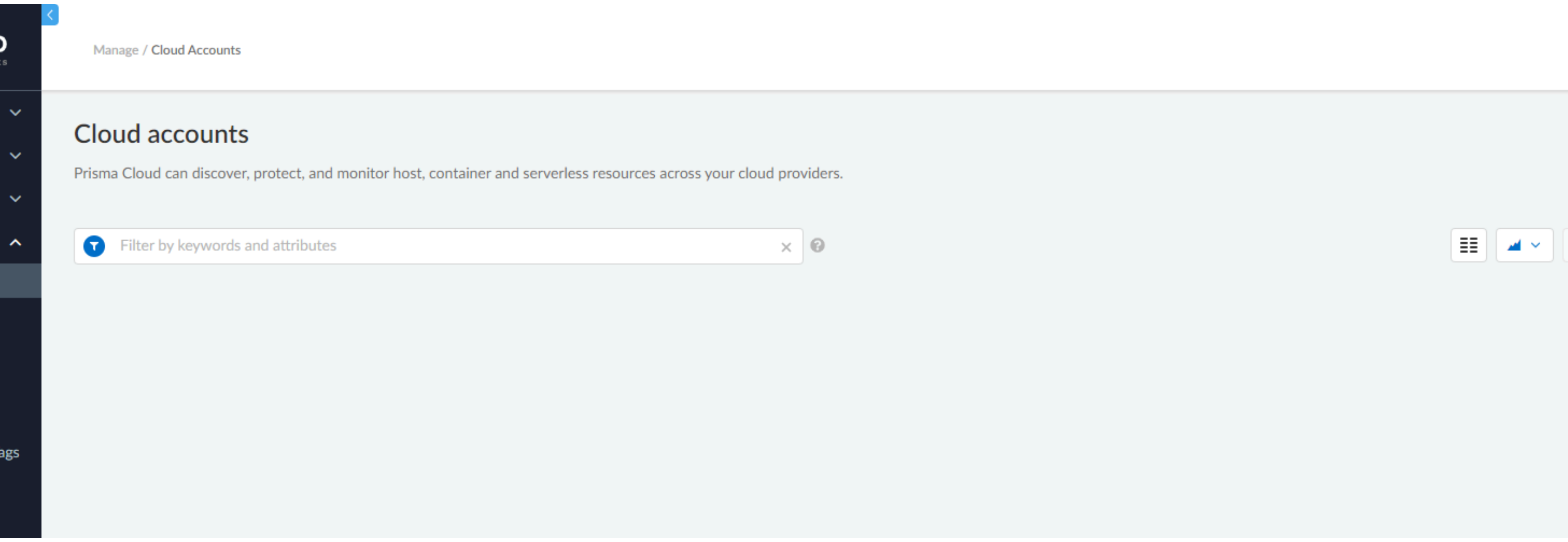
Scan Settings: Periodic scans occur every 24 hours by default. You can change the scan interval under **Manage - System > Scan - Agentless** setting. You can also perform on-demand scans by clicking the **Agentless scan** button on any of the Monitor pages or by selecting specific accounts under **Manage > Cloud accounts > Scan button** for bulk scanning.

Bulk Actions

Prisma Cloud supports performing agentless configuration at scale. Different cloud providers and authentication subtypes require different configuration fields, which also limits your ability to change accounts in bulk. The Prisma Cloud Console displays all the configuration fields that can be changed across all the selected accounts, and hides those that differ to prevent accidental misconfiguration.

The following procedure shows the steps needed to configure agentless scanning for multiple accounts at the same time.

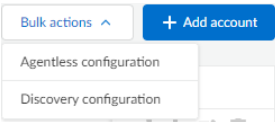
STEP 1 | Go to **Manage > Cloud accounts**



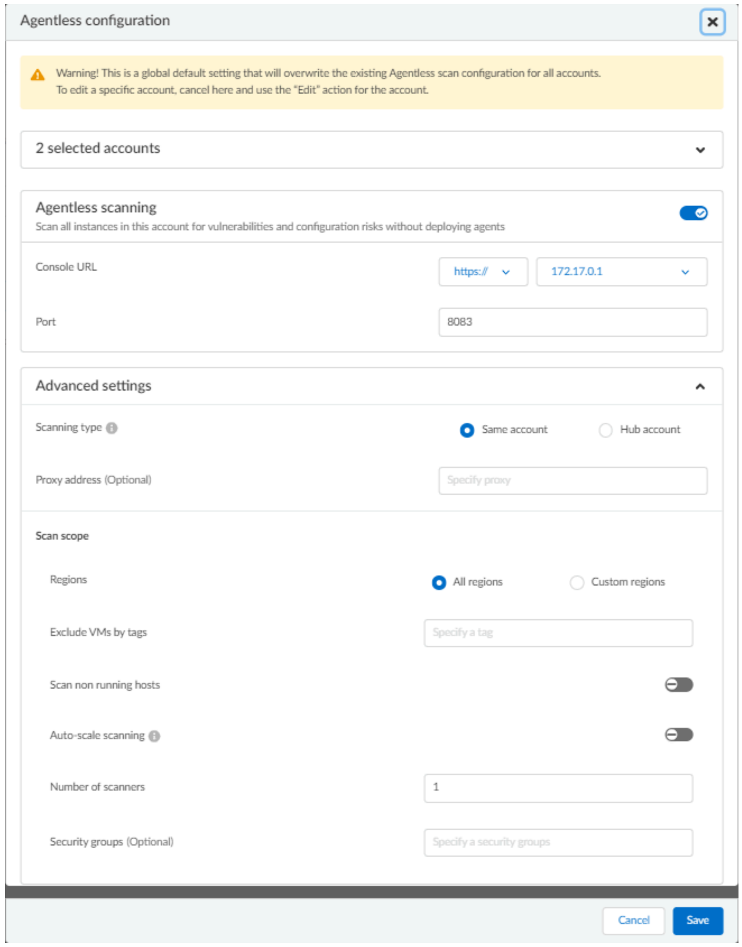
STEP 2 | Select multiple accounts.

STEP 3 | Click the **Bulk actions** dropdown.

STEP 4 | Select the **Agentless configuration** button.



STEP 5 | Change the configuration values for the selected accounts.

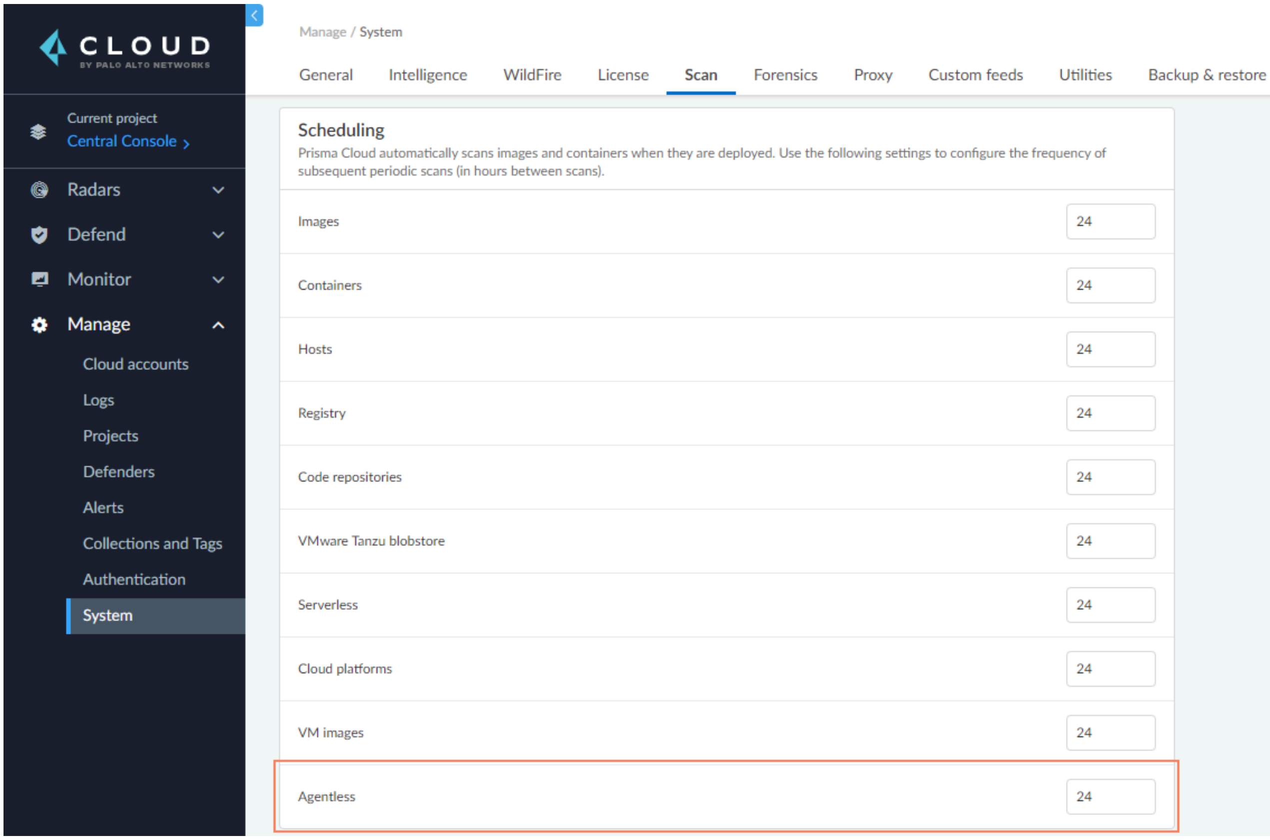


- Select **Save** to save the configuration for the selected accounts.

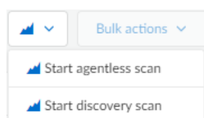
Other Settings

Use the **Cloud Account Manager** user role to grant full read and write access to all cloud account settings. This role can manage credentials, and change **Agentless Scanning** and **Cloud Discovery** configuration.

By default, agentless scans are performed every 24 hours, but you can change the interval on the **Manage > System > Scan** page under **Scheduling > Agentless**.



To manually trigger an agentless scan, click the **Trigger scan** dropdown and select the **Start agentless scan** option on the **Manage > Cloud accounts** page.



Agentless Scanning Modes

Edit on GitHub

There are two ways you can set up agentless scanning with Prisma Cloud.

- **Same Account**: Scan all hosts of a cloud account within the same cloud account. This mode spins up temporary scanning instances in the account.
- **Dedicated Account** Scan all hosts of a cloud account, called *target account*, from another dedicated cloud account, called *hub account*. This mode spins up temporary scanning instances in the hub rather than in the target(s).

Agentless scanning isn't supported for hosts running Windows. Agentless scanning doesn't support Azure hosts with an unmanaged operating system disk. Azure hosts with unmanaged operating system disks are skipped during the scan.

Scan Within the Same Cloud Account

1. Onboard cloud accounts with specific permissions required for agentless scanning.
2. Prisma Cloud lists instances in each account and creates snapshots for each instance.
3. Prisma Cloud starts spot instances, called *scanners*, within the same account, attaches snapshots, and performs the analysis.
4. Scanners send results to the Prisma Cloud Console.
5. Scanners and snapshots created by Prisma Cloud are deleted.
6. Process repeats for periodic scans.

Scan Within a Dedicated Cloud Account

1. Onboard cloud accounts with permissions for the hub account which perform the scan, and target accounts that are scanned by the hub account.
2. Prisma Cloud only spins up scanners in the dedicated hub account and attaches snapshots of instances from other accounts to the scanners in the hub account.
3. Scanners send results to the Prisma Cloud Console
4. Scanners then get deleted along with the snapshots that Prisma Cloud creates.
5. Process repeats for periodic scans.

AWS

When you click the **Download** button, multiple permission templates are downloaded as JSON files. These templates support the various permissions required by each of the cloud accounts for each of the scanning modes.

Same Account Mode

To scan accounts using this mode, download and apply the permission template that ends in *_target_user_permissions.json* to the AWS cloud account.

Dedicated Account Mode

You first configure and select a cloud account to serve as the hub account. You apply permissions templates from the hub account in Prisma Cloud to the hub account in AWS, and apply permissions templates from both the hub account and the target account in Prisma Cloud to the target account in AWS.

Hub Account

To use an account as a hub account, first add it to Cloud Accounts, downloading and applying the permission template that ends in *_hub_user_permissions.json* to that hub account in AWS.

Target Account

Download and apply the permission template that ends in *_hub_target_user_permissions.json* to that target account in AWS..

Azure

Download and apply the permission template to the Azure cloud account: there is no option for Hub Account Mode in Aure. Note that Prisma Cloud creates a dedicated *PCC* resource group to allow for easier cost calculations.

GCP

When you click the **Download** button, multiple permission templates are downloaded as JINJA files. These templates support the various permissions required by each of the cloud accounts for each of the scanning modes.

Same Account Mode

To scan accounts using this mode, download and apply the permission template that ends in *_target_user_permissions.yaml.jinja* to the GCP project.

Dedicated Account Mode

You first configure and select an cloud account to serve as the hub account. You apply permissions templates from the hub account in Prisma Cloud to the hub project in GCP, and apply permissions templates from both the hub account and the target account in Prisma Cloud to the target project in GCP.

Hub Account/Project

To use an account as a hub account, first add it to Cloud Accounts, downloading and applying the permission template that ends in *_hub_user_permissions.yaml.jinja* to that hub project in GCP.

To also allow that hub account to scan its own hosts, also apply the permission template that ends in *_target_user_permissions.yaml.jinja* to that hub project in GCP. Otherwise, disable Agentless scanning for that hub account in Cloud Accounts.

Target Account/Project

Download and apply the permission templates that end in *_hub_target_user_permissions.yaml.jinja* and *_hub_target_access_permissions.yaml.jinja* to the target project in GCP.

Configure scanning

Edit on GitHub

You can specify how often Prisma Cloud scans your environment for vulnerabilities and compliance issues. By default, Prisma Cloud scans your environment every 24 hours. Images are re-scanned when changes are detected. For example, pulling a new image triggers a scan.

Prisma Cloud scans for vulnerabilities and/or compliance issues in:

- Images
- VMware Tanzu blobstores
- Containers
- Serverless functions
- Hosts
- Cloud platforms
- Registries
- VM images

Scan intervals can be separately configured for each type of object.

Configuring scan intervals

The scan frequency is configurable. By default, Prisma Cloud scans your environment every 24 hours.

STEP 1 | Open Console.

STEP 2 | Go to **Manage > System > Scan**.

STEP 3 | Scroll down to the **Scheduling** section.

STEP 4 | Set the scan intervals for each type according to your requirements.

Scan intervals are specified in hours.

STEP 5 | Scroll to the bottom of the page, and click **Save**.

Last scan time

Console reports the last time Defender scanned your environment. Go to **Manage > Defenders > Manage**, and click a row in the table to get a detailed status report for each deployed Defender. The status column shows the last time Defender scanned the containers and images on the host where it runs. When Defender is delegated the registry scanner role, you can also see the last time your registry was scanned.

Component	Status
Connectivity	Connected since Mar 19, 2019 4:34:35 AM
Container scanning	Last scan on May 3, 2019 9:16:58 AM
Image scanning	Last scan on May 3, 2019 9:16:58 AM
Registry scanning	Last scan on May 3, 2019 10:56:53 AM
Filesystem	Enabled
Network	Enabled
Syscalls	Enabled
Processes	Enabled
Container Network Firewall	Enabled
Host Network Firewall	Enabled
Application Firewall	Enabled

Scan performance

Scanning for malware in archives in container images consumes a lot of resources. The scanner unpacks each archive to search for malicious software. Checksums must be indiviudally calculated for each file. Because of the performance impact and the way containers tend to be used, malware in archives is an unlikely threat. As such, **Scan for malware within archives in images** is disabled by default.

If this option is enabled, Prisma Cloud supports the following archive file types.

- ZIP
- GZ
- TAR
- WAR
- JAR
- EAR

Note: If the archive is over 512Mb, Prisma Cloud will not scan it.

Scan JavaScript components in manifest but not on disk

The purpose of this option is to show vulnerabilities in dependencies that might not exist on disk (which are often development dependencies).

Most Node.js packages contain a package.json that lists all of its dependencies (both dependencies, and devDependencies). When parsing a Node.js package discovered during a scan, if this option is enabled, Prisma Cloud appends the all packages found in each package.json to the list of packages to be assessed for vulnerabilities. This option isn't recommended for production scenarios because it can generate a significant number of false positives.

If this option is disabled (default), Prisma Cloud only evaluates the packages that are actually found on disk during scan. This is the recommended setting for production scenarios.

When scanning images with twistcli, use --include-js-dependencies to enable this option.

Unrated vulnerabilities

When **Show vulnerabilities that are of negligible severity** is enabled, the scanner reports CVEs that aren't scored yet or have a negligible severity. Negligible severity vulnerabilities don't pose a security risk, and are often designated with a status of "will not fix" or similar labels by the vendor. They are typically theoretical, require a very special (unlikely) situation to be exploited, or cause no real damage when exploited.

By default, this setting is disabled to strip unactionable noise from your scan reports.

Orchestration

Kubernetes and other orchestrators have control plane components implemented as containers. By default, Prisma Cloud doesn't scan orchestrator utility containers for vulnerability and compliance issues.

User certificate validity period

Edit on GitHub

User certificates identify a user, and are used to enforce access control policies. You can control how long user certificates are valid. By default, user certificates are valid for 365 days.

Configuring the validity period of user certificates

Configure the validity period of user certs.

STEP 1 | Open Console.

STEP 2 | Go to **Manage > Authentication > Certificates**.

STEP 3 | Under **Configuration**, enter a new value for **Number of days until expiration of certificate**.

STEP 4 | Click **Save**.

Expired user certificates

The following message is printed when you try to authenticate with an expired certificate. This example command tries to run *docker ps* on a remote host named prod_host1.

```
$ docker --tlsverify -H prod_host1:9998 ps
The server probably has client authentication (--tlsverify) enabled.
Please check your TLS client certification settings
```

Generating new certificates

When your certificates expire, you can generate new ones.

STEP 1 | Go to Console.

STEP 2 | Log in with your credentials to reauthenticate with Console. This step generates fresh certificates.

- If you integrated Prisma Cloud with LDAP, log in with your LDAP credentials.
- If you integrated with SAML, log in with your SAML credentials.
- If you are using Prisma Cloud users, log in with your Prisma Cloud user credentials.

STEP 3 | On the left menu, click **Manage > Authentication > User certificates**.

STEP 4 | Copy the installation script, and run it on your local machine.

The script installs fresh certificates on your machine.

STEP 5 | Verify that your certs are valid by running a Docker command on a host protected by Defender.

```
$ docker --tlsverify -H prod_host1:9998 ps
```


Enable HTTP access to Console

Edit on GitHub

By default, Prisma Cloud only creates an HTTPS listener for access to Console. In some circumstances, you may wish to enable an HTTP listener as well. Notice that accessing Console over plain, unencrypted HTTP isn't recommended, as sensitive information can be exposed.

Enabling an HTTP listener simply requires providing a value for it in twistlock.cfg. At first, your configuration file would look like this:

```
#############################################
#     Network configuration
#############################################
# Each port must be set to a unique value (multiple services cannot
 share the same port)
###### Management console ports #####
# Sets the ports that the Prisma Cloud management website listens on
# The system that you use to configure Prisma Cloud must be able to
 connect to the Prisma Cloud Console on these ports
# To disable a listener, leave the value empty (e.g.
 MANAGEMENT_PORT_HTTP=)
# Accessing Console over plain, unencrypted HTTP isn't recommended,
 as sensitive information can be exposed
MANAGEMENT_PORT_HTTP=
MANAGEMENT_PORT_HTTPS=8083
```

To enable the HTTP listener, your configuration file should look like this:

```
#############################################
#     Network configuration
#############################################
# Each port must be set to a unique value (multiple services cannot
 share the same port)
###### Management console ports #####
# Sets the ports that the Prisma Cloud management website listens on
# The system that you use to configure Prisma Cloud must be able to
 connect to the Prisma Cloud Console on these ports
# To enable the HTTP listener, set the value of MANAGEMENT_PORT_HTTP
 (e.g. MANAGEMENT_PORT_HTTP=8081)
# Accessing Console over plain, unencrypted HTTP isn't recommended,
 as sensitive information can be exposed
MANAGEMENT_PORT_HTTP=8081
MANAGEMENT_PORT_HTTPS=8083
```

After you've updated the configuration file, just rerun *twistlock.sh* for the changes to take effect. For example:

```
$ sudo ./twistlock.sh -s console
```

Set different paths for Defender and Console (with DaemonSets)

Edit on GitHub

When using daemon sets, Console is set up to store the Prisma Cloud config under */opt/twistlock*. By default, it uses this same config when installing the defenders. This article describes a work around solution to be able to set up different config paths for Console and Defenders using daemon sets

STEP 1 | Download Daemonset configurations for Defender.

The API to download Daemonset Configuration is:

```
/api/v1/defenders/daemonset.yaml?registry=${registry}&type=
${DEFENDER_TYPE}
&consoleaddr=${consoleaddr}&namespace=${namespace}
&orchestration=${orchestration}&ubuntu=${os_ubuntu}"
```

The parameters are:

- **registry** --

  the registry from where Kubernetes gets the image, where you pushed the image. In the example above, the value will be “gcr.io/projectA/”
- **type** --

  defender type - Daemon Set Docker on Linux or Daemon Set Kubernetes Node. (Daemon set Docker on Linux is the regular default Defender type, called in the UI Docker. Only difference being, unlike the default Defender, it does not listen to incoming traffic.
- **consoleaddr** --

  Name or IP address that Defenders use to connect to Console.
- **namespace** --

  the default when using the script is twistlock, but you can use whatever you want.
- **orchestration** --

  OpenShift or Kubernetes
- **ubuntu** --

  (ubuntu=true \ ubuntu=false), states if the cluster is running on ubuntu OS or not. If not provided, it’s assumed to be false.

STEP 2 | Edit the yaml file.

Make the necessary changes in this yaml file and upload this modified version of the yaml to the K8 controller.

Authenticate to Console with certificates

Edit on GitHub

Prisma Cloud supports certificate-based authentication for the Console UI and the API.

Prisma Cloud has always provided username / password based authentication. In addition to that, Prisma Cloud also supports certificate based authentication for the Console UI and the API. This is especially useful for those in government and financial services, who use multi-factor authentication technologies built on x.509 certificates. This is applicable to users authenticating via Active Directory accounts as well. This feature allows customers to be able to control the trusted CAs for signing certificates for authentication.

Setting up your certs

This procedure shows you how to set up Prisma Cloud for certificate-based authentication.

If you're using certificates to authenticate against Active Directory accounts, Prisma Cloud uses the UserPrincipalName field in the SAN to match the certificate to the user in Active Directory. This is the same process used by Windows clients for authentication, so for most customers, the existing smart card certificates you're already using can also be used for authentication to Prisma Cloud.

STEP 1 | Save the CA certificate(s) used to sign the certificates that you'll use for authentication to Prisma Cloud.

The certificate has to be in PEM format. If you have multiple CAs that issue certificates to your users, concatenate their PEM files together. For example, if you have Issuing CA 1 and Issuing CA 2, create a combined PEM file like this:

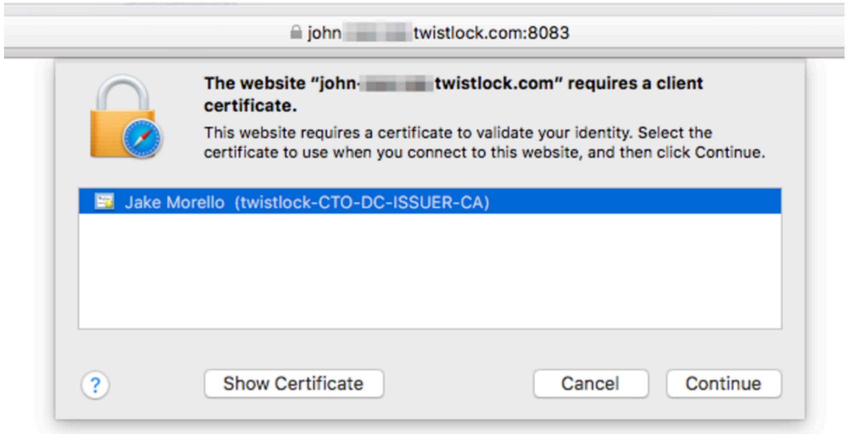
```
$ cat issuing-ca-1.pem issuing-ca-2.pem > issuing-cas.pem
```

STEP 2 | Log into Console, and go to **Manage > Authentication > System Certificates**.

STEP 3 | Scroll down to **Certificate-based authentication to Console**, and upload your CA certificate(s) in PEM format.

STEP 4 | Click **Save**.

STEP 5 | Open Console login page in your browser. When prompted select your user certificate.



What’s next?

See Assigning roles to learn how to add users and assign roles to them.

Configure custom certs from a predefined directory

Edit on GitHub

You can use your own certs to secure Prisma Cloud communication channels by simply copying your custom certs to a predefined directory that the Console and Defender containers mount at runtime. No additional configuration in the Console UI is required to use your custom certs. This mechanism is enabled by default.

The communication channels you can secure with this mechanism are:

- Console web UI and API over HTTPS

  By default, web and API clients connect to Console over HTTPS on port 8083. Out of the box, traffic is TLS encrypted using self-signed certs.

- Console-Defender communication over a WebSocket

  By default, Defender connects to Console over a WebSocket on TCP port 8084. Out of the box, traffic between Defender and Console is TLS encrypted using self-signed certs.

  By design, Console and Defender don't trust each other. When Defender initiates a connection with Console, mutual TLS is used to verify both parties.

In general, the keys and certs used for Defender-Console communication are considered an internal implementation detail. Prisma Cloud generates, manages, and rotates the keys internally. However, if your organization's policy calls for directly managing all keys and certs, Prisma Cloud gives you the control to do so.

How it works

Configure Console and Defender to use custom certs by saving the certs into a known, predefined directory. The predefined directory is */var/lib/twistlock/custom-certificates*. The directory must be mounted in the Console and Defender container file systems when the Prisma Cloud containers are started.

When establishing a connection, the directory is checked for the required certificates. If the required files exist, they are used to establish the connection.

If there's any error in the cert files (e.g., corrupt files, key-cert mismatch, etc.), the connection fails to be established, and an error is logged.

Prisma Cloud monitors the predefined directory for file system changes. If a relevant change is detected, the connection is restarted. For example, if console-cert.pem or console-key.pem are changed, the HTTPS listener is restarted. This system is designed to make rotating certificates easy by simply copying new certs to the predefined directory.

Loading a TLS configuration

When Console and Defender start, they create TLS configurations.

Depending on the environment, a number of scenarios are possible.

Scenario: Console/Defender starts and the predefined custom certificates directory doesn't exist and isn't mounted in the container.

In this case, the feature is switched off. Console/Defender uses its own self-signed certificates.

Scenario: Console/Defender starts and finds the predefined custom certificates directory mounted, but not all of the required files exist.

For example, Defender requires three files to secure Console-Defender communication: defender-ca.pem, defender-client-cert.pem, defender-client-key.pem. In this scenario, assume the defender-ca.pem file is missing.

In this case, Prisma Cloud initializes the connection using its own self-signed certificates, and then watches the predefined custom directory for changes. If there are changes to the predefined directory, Prisma Cloud checks if all certificate files exist. If they do, it tries to create a TLS configuration using those certs. If it succeeds, the connection is reset and new connection is established with the custom certs. If it fails, Prisma Cloud logs an error and keeps the existing connection.

Scenario: Console/Defender starts and finds the required certificates in the mounted directory

Prisma Cloud reads the certs and creates a TLS configuration. If reading a cert fails, Console/Defender logs the following messages, and exits:

```
 DEBU 2021-10-27T17:37:46.645 defender.go:1928  Using custom
 directory certificates
 CRIT 2021-10-27T17:37:46.645 defender.go:285  Failed to construct
 defender client TLS config error reading X509 key pair (/var/lib/
twistlock/custom-certificates/defender-client-cert.pem, /var/lib/
twistlock/custom-certificates/defender-client-key.pem): tls: failed
 to parse private key
```

Fixing misconfigurations

If there's an error loading your certs (for example, malformed key material), you can fix the problem by simply copying fixed certs to the predefined directory. As long as the Prisma Cloud container starts with the predefined directory mounted, it can watch the directory for changes, and try to reestablish a new connection with the latest certs when new files are detected

This, in conjunction with the verbose log messages, will help you get your configuration right.

Required certificate files

To secure a connection with your custom certs, Prisma Cloud looks for specific files in */var/lib/twistlock/custom-certificates*.

Console

For the Console HTTPS listener (for securing the web UI and API), the following files must be available in the predefined directory:

- *console-cert.pem*
- *console-key.pem*

For Console's WebSocket listener (for securing Console-Defender communication), the required files are:

- *defender-ca.pem*

- *defender-server-cert.pem*
- *defender-server-key.pem*

Defender

For Defender's WebSocket listener (for securing Console-Defender communication), the required files are:

- *defender-ca.pem*
- *defender-client-cert.pem*
- *defender-client-key.pem*

Log messages

Clear operational and error messages are sent to the Console and Defender logs so you can debug certificate issues.

On startup, the following message is logged, indicating the custom directory is mounted and being tracked for rotations:

```
Watching custom certificates directory: /var/lib/twistlock/custom-
certificates
```

If the directory was not mounted, the following message is logged:

```
Custom certificates watcher disabled: certificates directory is not
 mounted
```

Upon resetting a connection following a cert rotation, a message is logged, e.g.,

```
Defender custom certificates rotated, resetting connection
```

Any error in the cert files (e.g., corrupt files, key-cert mismatch, etc.) will result in failure to establish connection, and it's logged as an error.

Deployment patterns

This feature depends on your key material already being present on each node where Console and Defender run. Certificate enrollment, renewal, and management are strictly your responsibility.

The predefined custom certs directory must be mounted into Console/Defender's file system when the containers start (or restart). Console/Defender only requires read access to the predefined directory.

In general, you should have some kind of network storage (e.g., NFS), where you can centrally store and rotate your custom certs. All pods would mount the same network volume, so that when you rotate your certs, the latest files are available to all pods at the same time.

Onebox

STEP 1 | Before installing Onebox, create the predefined custom certs directory.

```
mkdir -p /var/lib/twistlock/custom-certificates
```

STEP 2 | Install Onebox.

STEP 3 | Check the Console and Defender logs.

A log message says the pre-created directory was identified and that it's being watched.

```
DEBU 2021-11-11T11:59:33.296 cert_watcher.go:45 Watching custom
 certificates directory: /var/lib/twistlock/custom-certificates
```

STEP 4 | Copy your custom certificates to the pre-created directory.

Both Console and Defender watch this directory for their certificates Connections are reset when relevant changes are detected.

Deploying Console and Defender in Kubernetes or OpenShift clusters

The following steps provide high-level guidance for deploying Prisma Cloud containers with your custom certs in your clusters.

STEP 1 | Use twistcli to generate a Defender DaemonSet YAML configuration file.

- Console on Kubernetes
- Console on OpenShift
- Defender DaemonSets on Kubernetes
- Defender DaemonSets for OpenShift

STEP 2 | Before deploying, open the YAML file, and add a volume mount for the predefined directory, */var/lib/twistlock/custom-certificates/*.

For example:

```
apiVersion: v1
kind: Pod
metadata:
  name: example-pod
spec:
  volumes:
    - name: example-pv-storage
      persistentVolumeClaim:
        claimName: example-pv-claim
  containers:
    - name: defender-container
      image: defender-image
      volumeMounts:
        - mountPath: "/var/lib/twistlock/custom-certificates/"
          name: example-pv-storage
```

Limitations

App-Embedded and Serverless Defenders currently do not support custom keys and certs for securing Console-Defender communication.

Customize terminal output

Edit on GitHub

Prisma Cloud lets you create rules that block access to resources or block the deployment of non-compliant containers.

For example, you might create a rule that blocks the deployment of any image that has critical severity vulnerabilities. By default, when you try to run non-compliant image, Prisma Cloud returns a terse response:

```
# docker -H :9998 --tls run -ti morello/docker-whale
docker: Error response from daemon: [Prisma Cloud] operation blocked
 by policy: (test-compliance), host has 19 compliance issues.
```

To help the operator better understand how to handle a blocked action, you can enhance Prisma Cloud's default response by

- Appending a custom message to the default message. For example, you could tell operators where to go to open a ticket.
- Configuring Prisma Cloud to return an itemized list of compliance issues rather than just a summary. This way, the operator does not need to contact the security team to determine which issues are preventing deployment. They are explicitly listed in the response.

Enhanced terminal output is available for rules created under:

- **Defend > Vulnerabilities > Policy**
- **Defend > Compliance > Policy**
- **Defend > Access** (Docker Engine and Kubernetes access control rules).

Specifying a custom message

This procedure shows you how to create an access control rule that blocks all users from running the *container_create* operation. You will configure the rule to emit the following custom message when an action is blocked:

```
Contact admin@example.com to get additional privileges
```

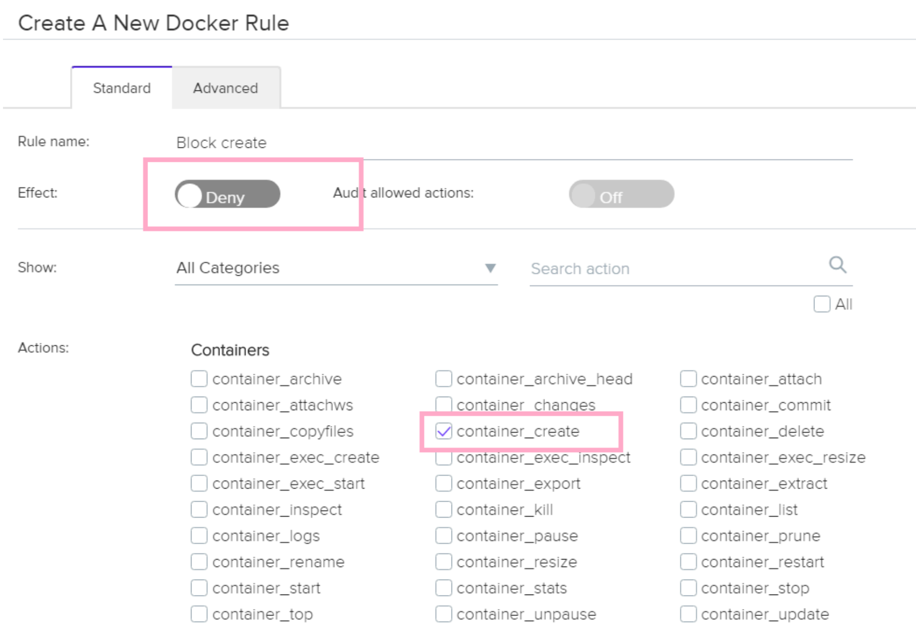
Although this procedure is specific to access control rules, the process for configuring custom messages for vulnerability and compliance rules is the same.

STEP 1 | Open Console.

STEP 2 | Go to **Defend > Access > Docker**, then click **New Docker rule**.

STEP 3 | In the new rule dialog, enter the following information:

1. In **Rule name**, enter a name.
2. Set **Effect** to **Deny**.
3. In **Show**, uncheck **All** to deselect all actions.
4. In **Actions**, check **container_create**.



5. Click on the **Advanced** tab.
6. In **Custom message for blocked requests**, enter **Contact** admin@example.com **to get additional privileges**.
7. Click **Save**.

STEP 4 | Test your setup by running a command that violates your access control rule.

1. Install your client certs.

   For more information, see Configure Docker client variables.
2. Try to run a container on a host protected by Prisma Cloud:

```
$ docker --tlsverify -H  <HOST>:9998 run ubuntu:latest
docker: Error response from daemon: [Prisma Cloud] The command
 container_create denied for user aqsa by rule Block create.
 Contact admin@example.com to get additional privileges.
See 'docker run --help'.
```

   Where <*HOST*> is the hostname or IP address for a host running Defender.

Output itemized list of compliance issues

You can configure vulnerability and compliance rules to return a detailed list of issues when Prisma Cloud blocks a deployment.

In this procedure, you create a vulnerability rule that prevents the deployment of any image that contains any type of vulnerable package.

Although this procedure is specific to vulnerability rules, the process for compliance rules is the same.

STEP 1 | Open Console.

STEP 2 | Create a new vulnerability rule (**Defend > Vulnerabilities > Policy**) or compliance rule (**Defend > Compliance > Policy**).

STEP 3 | In the new rule dialog, enter the following information:

1. Enter a rule name.
2. Specify conditions that trigger a block action.

   For example, for the **Image contains vulnerable OS packages** condition in a vulnerability rule, set the **Action** to **Block** and set the **Severity** threshold to **Low**.
3. Set **Terminal output verbosity for blocked requests** to **Detailed**.
4. Click **Save**.

STEP 4 | Test your setup by deploying an image with vulnerabilities.

On a host protected by Prisma Cloud, run an image with vulnerabilities.

```
$ docker run --rm -it ubuntu:14.04 sh
docker: Error response from daemon: [Prisma Cloud] Image operation
 blocked by policy: (sdf), has 44 vulnerabilities, [low:25
 medium:19].
Image           ID       CVE             Package   Version
   Severity   Status
=====           ==       ===             =======   =======
   ========   ======
ubuntu:14.04    4333f1   CVE-2017-2518   sqlite3   3.8.2-1ubuntu2.1
   medium     deferred
ubuntu:14.04    4333f1   CVE-2017-6512   perl      5.18.2-2ubuntu1.1
   medium     needed
   .
   .
   .
```

Collections

Edit on GitHub

Collections are predefined filters for segments of your environment. They're centrally defined, and they're used in rules and views across the product.

Collections are used to:

- Scope rules to target specific resources in your environment. For example, you might create a vulnerability rule that applies to all container images in an app called sock-shop. The rule might reference collectionA, which specifies *sock-shop*** in the image resource filter.
- Partition views. Collections provide a convenient way to browse data from related resources.
- Enforce which views specific users and groups can see. Collections can control access to data on a need-to-know basis. These are known as assigned collections.

Collections are created with pattern matching expressions that are evaluated against attributes such as image name, container name, host name, labels, function name, namespace, and more.

For labels, Prisma Cloud supports AWS tags, as well as distro attributes. Distro attributes are designed for central security teams that manage the policies in Console, but have little influence over the operational practices of the groups that run apps in the environments being secured. If the central security team can't rely on naming conventions or labels to apply policies that are OS-specific (e.g. different compliance checks for different OSs), they can leverage the distro attributes. Supported distro attributes are:

- Distro name — "osDistro:<value>" (e.g. "osDistro:Ubuntu")
- Distro version — "osVersion:<value>" (e.g. "osVersion:20.04")

Partitioning views

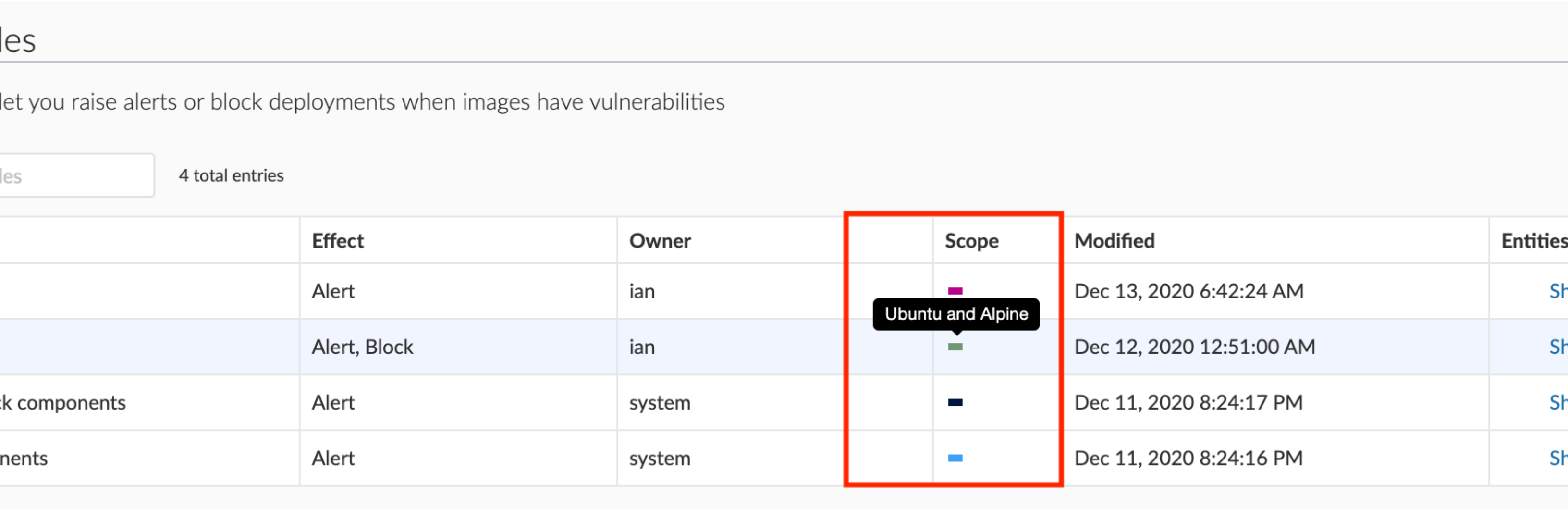
While a single Console manages data from Defenders spread across all hosts, collections let you segment that data into different views based on attributes.

Collections are useful when you have large container deployments with multiple teams working on multiple apps all in the same environment. For example, you might have a Kubernetes cluster that runs a shopping app, a travel app, and an expenses app. Different teams might be responsible for the development and operation of each app. An internal tools team might be responsible for the travel and expenses app, while a product team runs the shopping app.

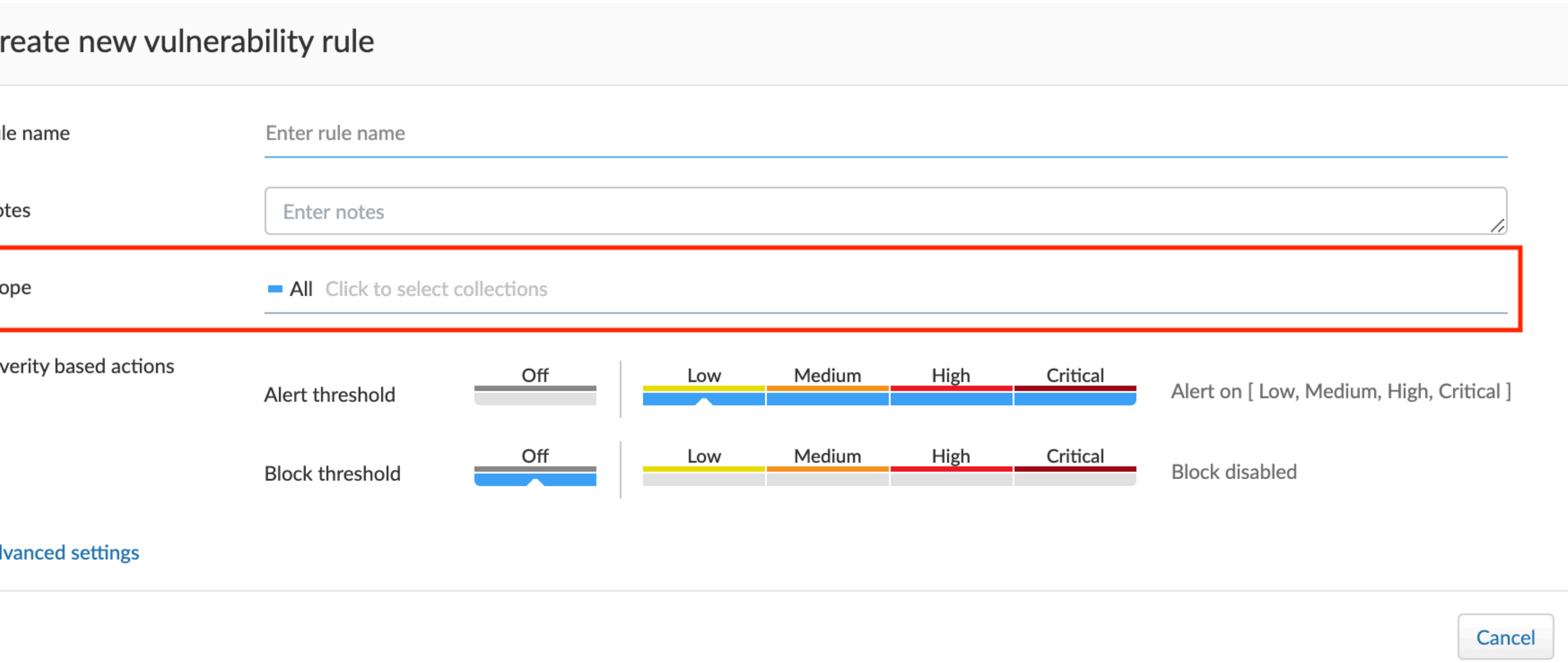
Selecting a collection reduces the scope displayed in Console to just the relevant resources. For example, the developer for the travel app only cares about vulnerabilities in the images that make up the travel app. All other vulnerabilities are just noise. Collections help focus the data.

Scoping rules

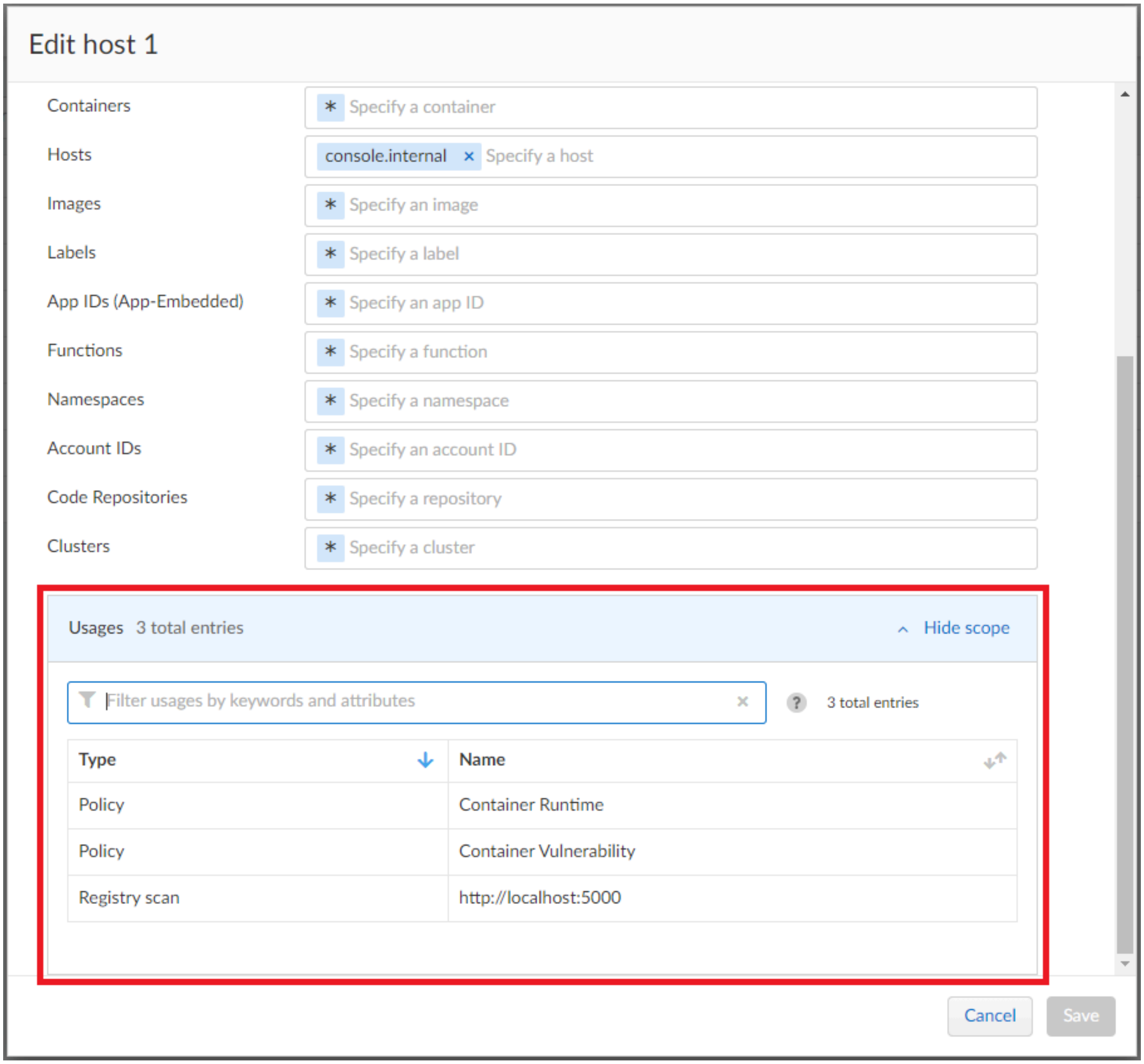
The scope of a rule is defined by referencing the relevant collections. Collections offer a centralized way to create and manage scope settings across the product. Collections make it easy to consistently reuse scope settings across policies. Policy tables give you a clear picture of what resources are being targeted in your rules.



When creating new rules, you can either select from a list of previously defined collections, or create a new one. By default, Prisma Cloud sets a rule's scope to the **All** collection, which captures all resources in the environment.



Collections cannot be deleted as long as they're being used by a rule. This mechanism ensures that rules are never left unscoped. Click on a specific collection to see how it's being used.



Importing and exporting rules

Rules can be exported from one Console and imported into another Console. When importing rules, any associated collections are also imported and created.

- If the imported rule uses a collection that doesn't exist in Console, the collection is automatically created.
- If the imported rule uses collection with a name that already exists, but with a different scope, the collection is created with the following name and description:
  - Name: <policyType> - <ruleName> <collectionName>
  - Description: Automatically generated collection for an imported rule/entity
- If the imported rule uses a collection that already exists, and a matching scope, the existing collection is used as-is.

Creating collections

You can create as many collections as you like. Collections cannot be nested. In tenant projects, collections are created and managed on a per-project basis.

Prisma Cloud ships with a built-in set called **All** that is not editable. The **All** collection contains all objects in the system. It is effectively the same as creating a collection manually and setting a wildcard (*) for each resource type (e.g., containers, images, hosts, labels, etc).

Collections can be created in **Manage > Collections and Tags > Collections**. Alternatively, collections can be created directly from a new rule dialog when you're setting the rule's scope. When creating collections from a new rule dialog, Prisma Cloud automatically disables any irrelevant scope fields. When selecting previously defined collections in a rule's scope field, any improperly scoped collections are hidden from display. For example, you can't select a collection that specifies serverless functions in a container runtime rule.

By default, new collections set a wildcard for each resource, effectively capturing all resources in the system. Customize the relevant fields to capture some segment of the universe of resources.

The labels field supports Docker labels, Kubernetes pod template labels, Kubernetes namespace labels, Kubernetes deployment labels, AWS tags, osDistro:<name> (for hosts), and osVersion:<version> (also for hosts).

To use Kubernetes namespace and deployment labels, enable the following setting when deploying Defenders: **Manage > Defenders > Deploy > DaemonSet > Collect Deployment and Namespace labels**.

To use AWS tags for hosts, enable the VM tags setting for relevant accounts under **Defend > Compliance > Cloud platforms**.

To scope App-Embedded policy rules (e.g., vulnerability, compliance, and runtime rules), use the collection's **App ID** field. For Fargate tasks protected by App-Embedded Defenders, you can additionally scope rules by image.

You cannot have collections that specify both containers and images. You must leave a wildcard in one of the fields, or else the collection won't be applied correctly. If you want to create collections that apply to both a container and an image, create two separate collections. The first collection should only include the container name, the second should only include the image name. Filtering on both collections at the same time will yield the desired result.

Filtering by cloud account ID for Azure Container Instances isn't currently supported.

To create a new collection:

STEP 1 | Open Console.

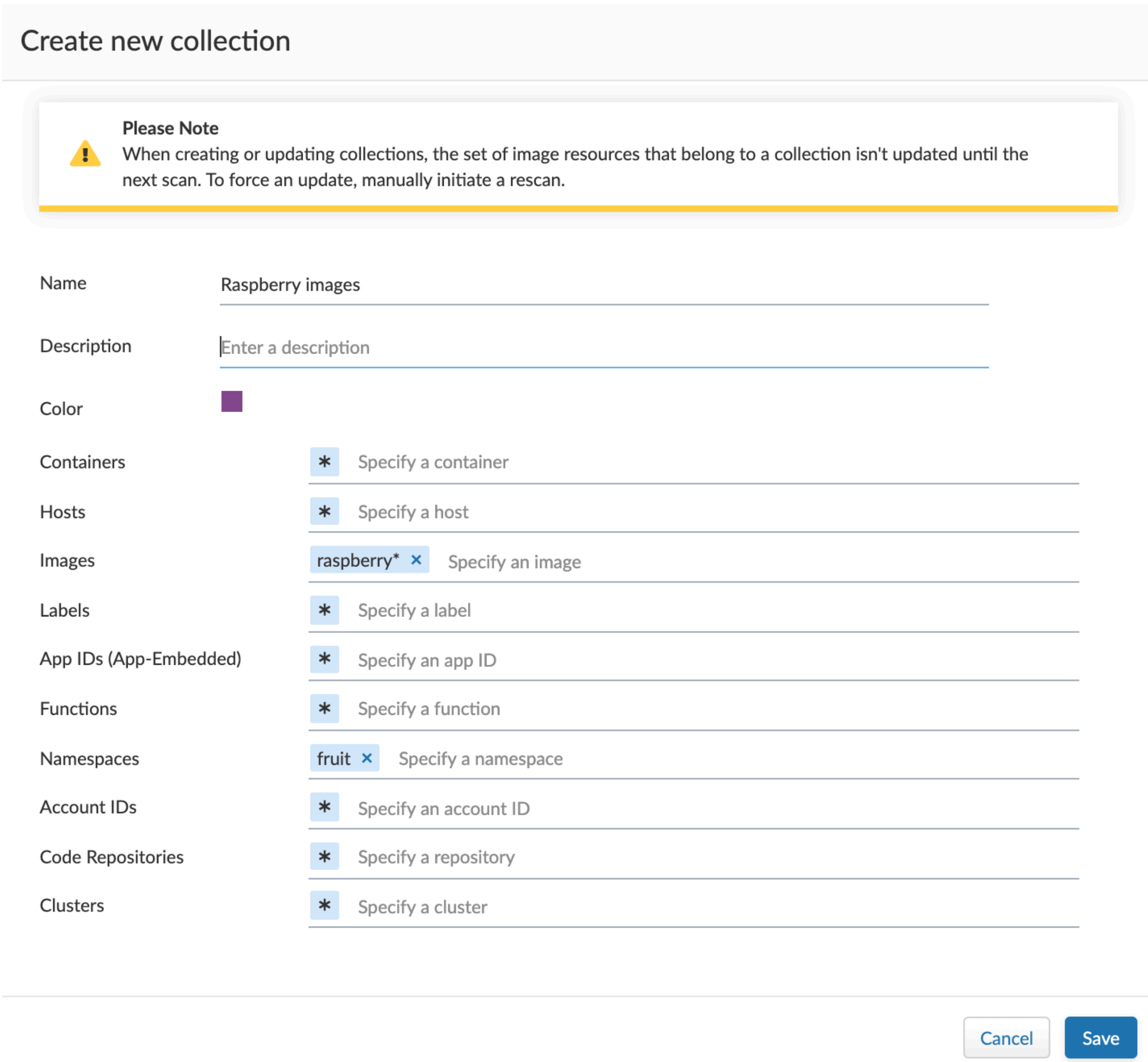
STEP 2 | Go to **Manage > Collections and Tags > Collections**.

STEP 3 | Click **Add collection**.

STEP 4 | In the **Create a new collection** dialog, enter a name, description, and then specify a filter to target specific resources.

For example, create a collection named **Raspberry images** that shows all *raspberry* images in the *fruit* namespace. Pick a color for easy visibility and differentiation.

The following collection selects all images that start with the string *raspberry*. You can also create collections that exclude resources. For more information on syntax that can be used in the filter fields (e.g., containers, images, hosts, etc), see Rule ordering and pattern matching.



STEP 5 | Click **Save**.

Assigned collections

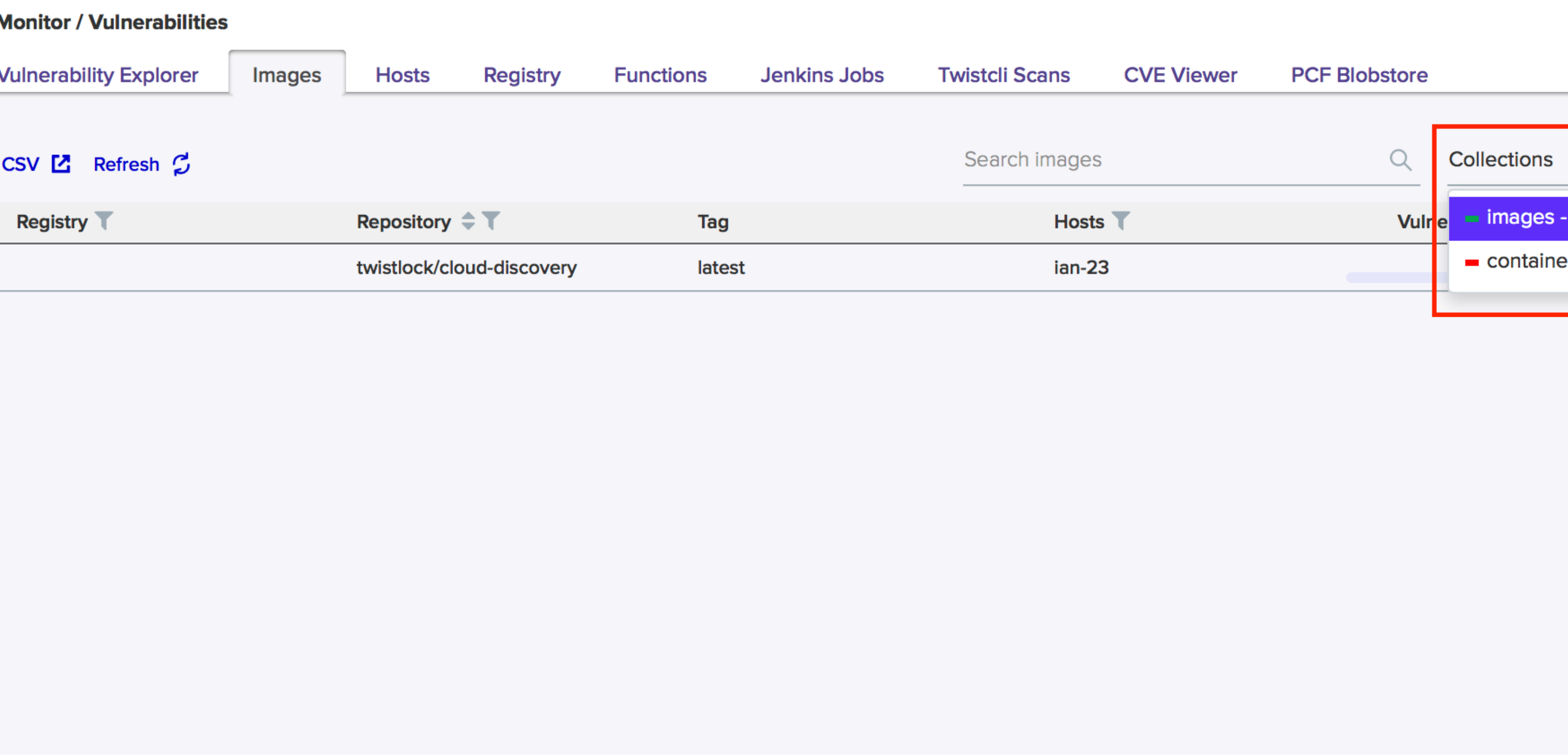
Collections provide a light-weight mechanism to provision least-privilege access to the resources in your environment. You can assign collections to specific users and groups to limit their view of data and resources in the environment.

Projects is the other mechanism for partitioning your environment. Projects are Prisma Cloud's solution for multi-tenancy. They let you provision multiple independent environments, and federate them behind a single Console URL, interface, and API. Projects take more effort to deploy than collections. Collections and Projects can work together. Collections can be utilized in both non-Project and Project-enabled environments.

By default, users and groups can access all collections and are not assigned with any collection.

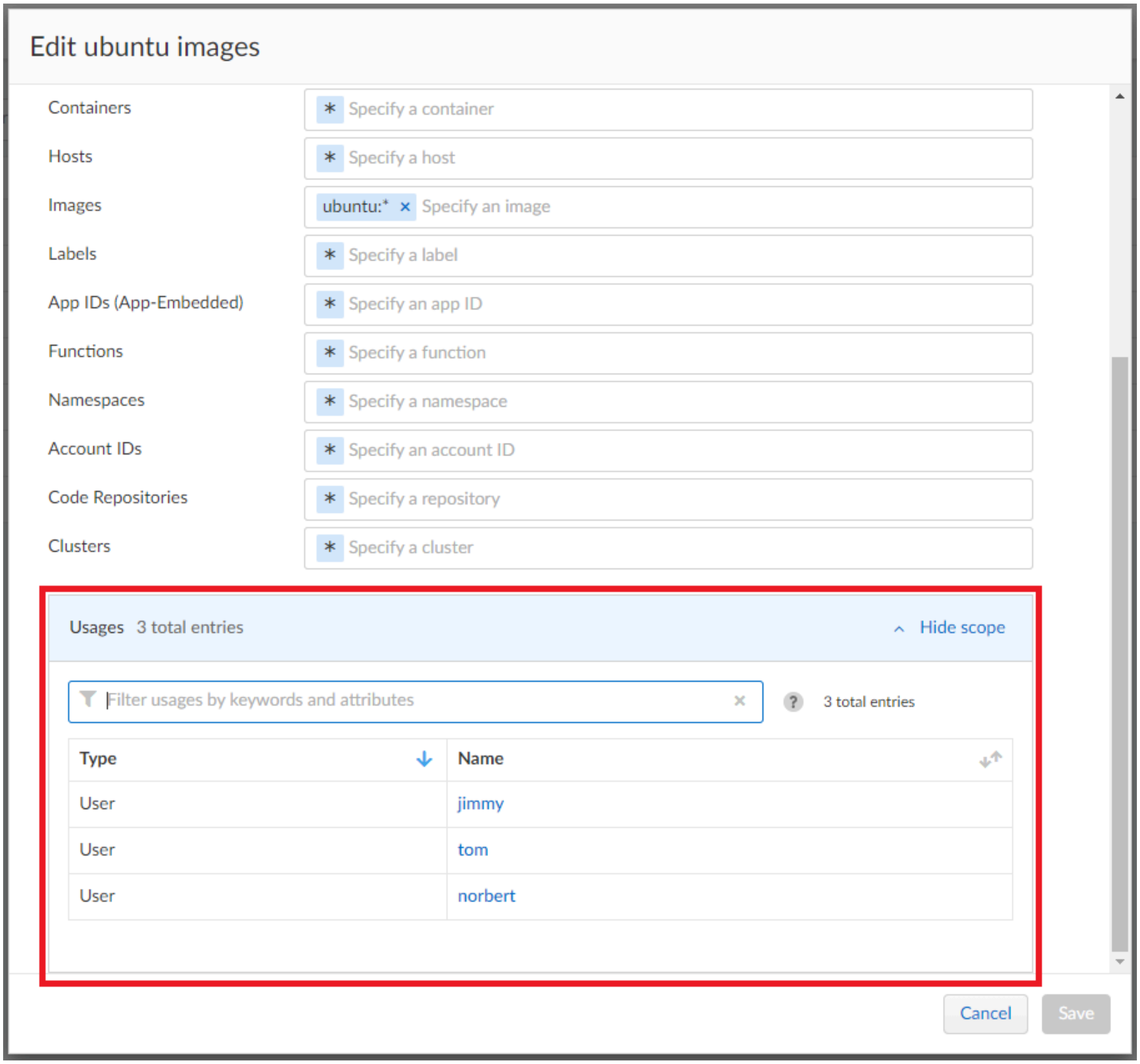
Users with admin or operator roles can always see all resources in the system. They can also see all collections, and utilize them to filter views. When creating users or groups with the admin or operator role, there is no option for assigning collections.

When creating users or groups with any other role, admins can optionally assign one more collections. These users can only see the resources in the collections they've been assigned.



If a user is assigned multiple system roles, either directly or through group inheritance, then the user is grante the higest role, and access to the assigned collections of all the groups to which the user belongs. If a user is assigned both system and custom roles, then the user will be randomly granted the rights of one of the groups, including its role and assigned collections.

Collections cannot be deleted as long as they've been assigned to users or groups. This enforcement mechanism ensures that users and groups are never left stateless. Click on a specific collection to see who is using them.



Changes to a user or group's assigned collections only take affect after users re-login.

Assigning collections

Assign collections to specific users and groups to restrict their view of data in the environment.

If a role allows access to policies, users with this role will be able to see all rules and all collections that scope rules under the Defend section, even if the user's view of the environment is restricted by assigned collections.

Collections can be assigned to local users, LDAP users, and SAML users. Collections can also be assigned to LDAP and SAML groups. They cannot be assigned to local groups.

When using Projects, Collections can only be assigned to users on each project. Users of the Central Console have access to all projects, and cannot be limited with assigned collections.

Prerequisites:

- You've already created one or more collections.
- (Optional) You've integrated Prisma Cloud with a directory service or SAML IdP.

STEP 1 | Open Console, and go to **Manage > Authentication > {Users | Groups}**.

STEP 2 | Click **Add users** or **Add group**.

STEP 3 | Select the **Auditor** or **DevOps User** role.

STEP 4 | In **Permissions**, select one or more collections. If left unspecified, the default permissions is **All collections**.

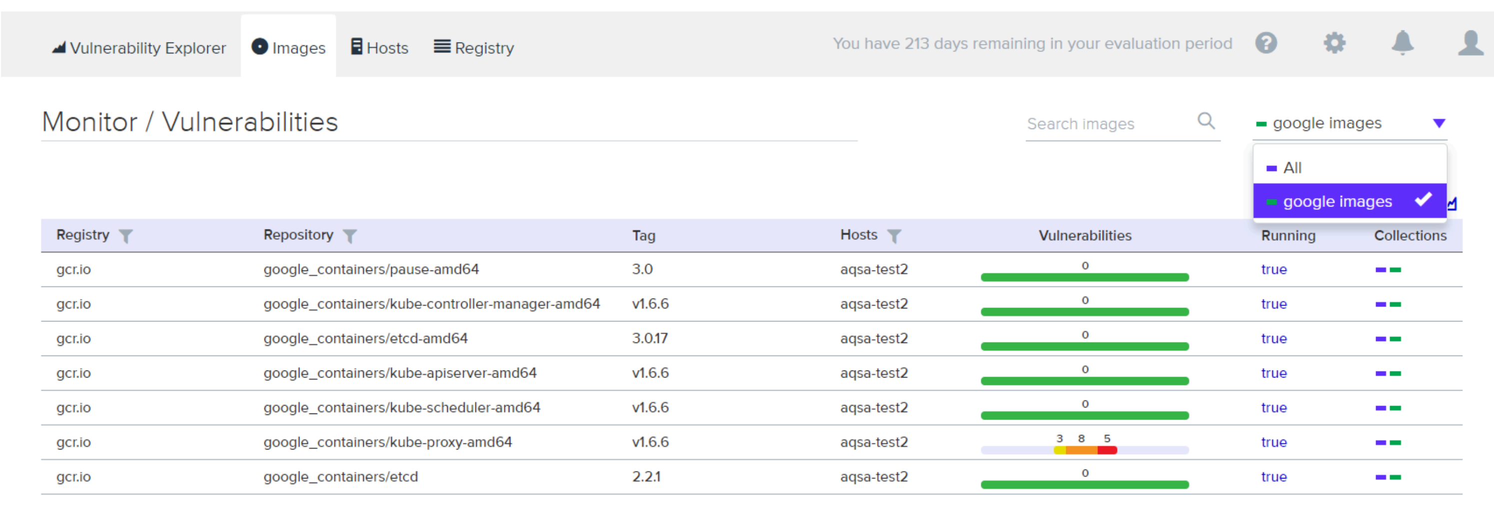
STEP 5 | Click **Save**.

Selecting a collection

Collections filter data in the **Monitor** section of Console.

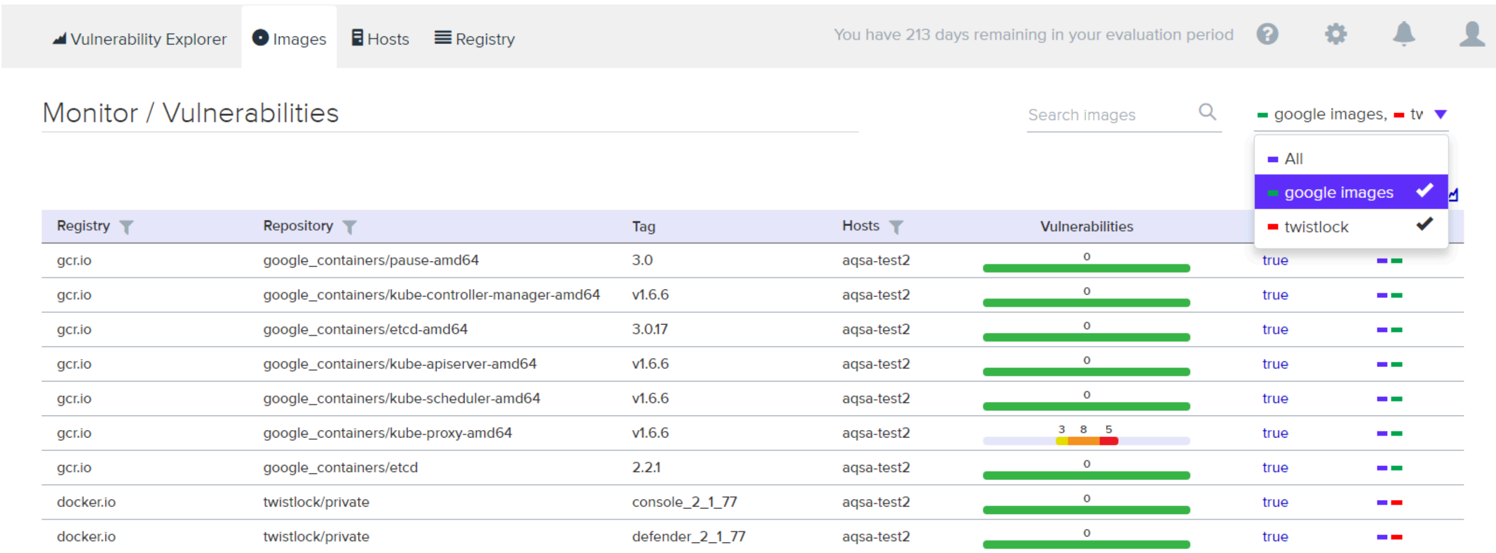
When a collection (or multiple collections) are selected, only the objects that match the filter are shown in those views. When a collection is selected, it remains selected for all views until it is explicitly disabled.

To select a collection, go to any view under **Monitor**. In the Collections drop-down list in the top right of the view, select a collection. In the following screenshot, the view is filtered based on the collection named **google images**, which shows all images that contain the string **google_containers**.



When multiple collections are selected, the effective scope is the union of each individual query.

Individual filters on each collection aren't applicable to all views. For example, a collection created with only functions won't include any resources when viewing hosts results. Similarly, a collection created with hosts won't filter images by hosts when viewing image results.



The **Collections** column shows to which collection a resource belongs. The color assigned to a collection distinguishes objects that belong to specific collections. This is useful when multiple collections are displayed simultaneously. Collections can also be assigned arbitrary text tags to make it easier for users to associate other metadata with a collection.

Limitations

Different views in Console are filtered by different resource types.

If a collection specifies resources that are unrelated to the view, filtering by this collection returns an empty result.

Section	View	Supported resources in collection
Monitor/ Vulnerabilities Monitor/ Compliance	Images	Images, Hosts, App IDs (App-Embedded), Namespaces, Clusters, Labels, Cloud Account IDs
Monitor/ Vulnerabilities Monitor/ Compliance	Registry images	Images, Hosts (of the scanner host), Labels, Cloud Account IDs
Monitor/ Vulnerabilities	Containers	Images, Containers, Hosts, Namespaces, Clusters, Labels, Cloud Account IDs

Section	View	Supported resources in collection
Monitor/ Compliance		
Monitor/ Vulnerabilities Monitor/ Compliance	Hosts	Hosts, Clusters, Labels, Cloud Account IDs
Monitor/ Vulnerabilities Monitor/ Compliance	VM images	VM images (under Images), Cloud Account IDs
Monitor/ Vulnerabilities Monitor/ Compliance	Functions	Functions, Cloud Account IDs, Labels (Region, AWS tag)
Monitor/ Vulnerabilities	Code repositories	Code repositories
Monitor/ Vulnerabilities	VMware Tanzu blobstore	Hosts (of the scanner host), Cloud Account IDs, Labels (tas-application-id, tas-application-name, tas-space-id, tas-space-name, tas-org-id, tas-org-name, tas-foundation)
Monitor/ Vulnerabilities	Vulnerability Explorer	Images, Hosts, Clusters, Labels, Functions, Cloud Account IDs
Monitor/ Compliance	Cloud Discovery	Cloud Account IDs
Monitor/ Compliance	Compliance Explorer	Images, Hosts, Namespaces, Clusters, Labels, Cloud Account IDs
Monitor/Events	Container audits	Images, Containers, Namespaces, Clusters, Container Deployment Labels (under Labels), Cloud Account IDs. (Cluster collections are not currently able to filter some events such as container audits, specifically.)
Monitor/Events	CNNS for Containers	Images (Destination image), Cloud Account IDs
Monitor/Events	WAAS for Containers	Images, Namespaces, Cloud Account IDs

Section	View	Supported resources in collection
Monitor/Events	Trust Audits	Images, Clusters, Cloud Account IDs
Monitor/Events	Admission Audits	Namespaces, Clusters, Cloud Account IDs
Monitor/Events	Docker Audits	Images, Containers, Hosts, Clusters, Cloud Account IDs
Monitor/Events	App-Embedded audits	App IDs (App-Embedded), Cloud Account IDs, Clusters, Images
Monitor/Events	WAAS for App-Embedded	App IDs (App Embedded), Cloud Account IDs
Monitor/Events	Host audits	Hosts, Clusters, Labels, Cloud Account IDs
Monitor/Events	CNNS for Hosts	Hosts (Source and Destination Hosts), Cloud Account IDs
Monitor/Events	WAAS for Hosts	Hosts, Cloud Account IDs
Monitor/Events	Host Log Inspection	Hosts, Clusters, Cloud Account IDs
Monitor/Events	Host File Integrity	Hosts, Clusters, Cloud Account IDs
Monitor/Events	Host Activities	Hosts, Clusters, Cloud Account IDs
Monitor/Events	Serverless audits	Functions, Cloud Account IDs, Labels (Region, Provider)
Monitor/Events	WAAS for Serverless	Functions, Cloud Account IDs, Labels (Region)
Monitor/Runtime	Container incidents	Images, Containers, Hosts, Namespaces, Clusters, Cloud Account IDs
Monitor/Runtime	Host incidents	Hosts, Clusters, Cloud Account IDs
Monitor/Runtime	Serverless incidents	Functions, Cloud Account IDs, Labels (Region)
Monitor/Runtime	App Embedded incidents	App IDs (App Embedded), Cloud Account IDs
Monitor/Runtime	Container models	Images, Namespaces, Clusters, Cloud Account IDs

Section	View	Supported resources in collection
Monitor/ Runtime	App-Embedded observations	App IDs, Images, Containers, Clusters, Account IDs, Regions (under Labels)
Monitor/ Runtime	Host observations	Hosts, Clusters, AWS tags (under Labels), OS tags (under Labels), Cloud Account IDs
Monitor/ Runtime	Image analysis sandbox	Images, Labels
Radar	Containers Radar	Images, Containers, Hosts, Namespaces, Clusters, Labels, Cloud Account IDs
Radar	Hosts Radar	Hosts, Clusters, AWS tags (under Labels), OS tags (under Labels), Cloud Account IDs
Radar	Serverless Radar	Functions, Cloud Account IDs, Labels (Region, AWS tag)
Manage	Defenders	Hosts, Clusters, Cloud Account IDs

Using Collections

After collections are created or updated, there are some views that require a rescan before you can see the change:

- Deployed Images vulnerabilities and compliance views
- Registry Images vulnerabilities and compliance views
- Code repositories vulnerabilities view
- Trusted images
- Cloud Discovery
- Vulnerability Explorer
- Compliance Explorer

After collections are created or updated, there are some views that are affected by the change only for future records. These views include historical records that keep their collections from creation time:

- Images and Functions CI results view
- Events views
- Incidents view
- Image analysis sandbox results view

Tags

Edit on GitHub

Tags are predefined labels that can help you manage the vulnerabilities in your environment. They are centrally defined and can be set to vulnerabilities and as policy exceptions.

Tags are used as:

- Vulnerability labels. They provide a convenient way to categorize the vulnerabilities in your environment.
- Policy exceptions. They can be a part of your rules in order to have a specific effect on tagged vulnerabilities.

Tags are useful when you have large container deployments with multiple teams working in the same environment. For example, you might have different teams handling different types of vulnerabilities. Then you can set tags in order to define responsibilities over vulnerabilities. Other uses would be to set the status of fixing the vulnerability, or to mark vulnerabilities to ignore when they are a known problem that can't be fixed in the near future.

Tag definition

You can define as many tags as you like.

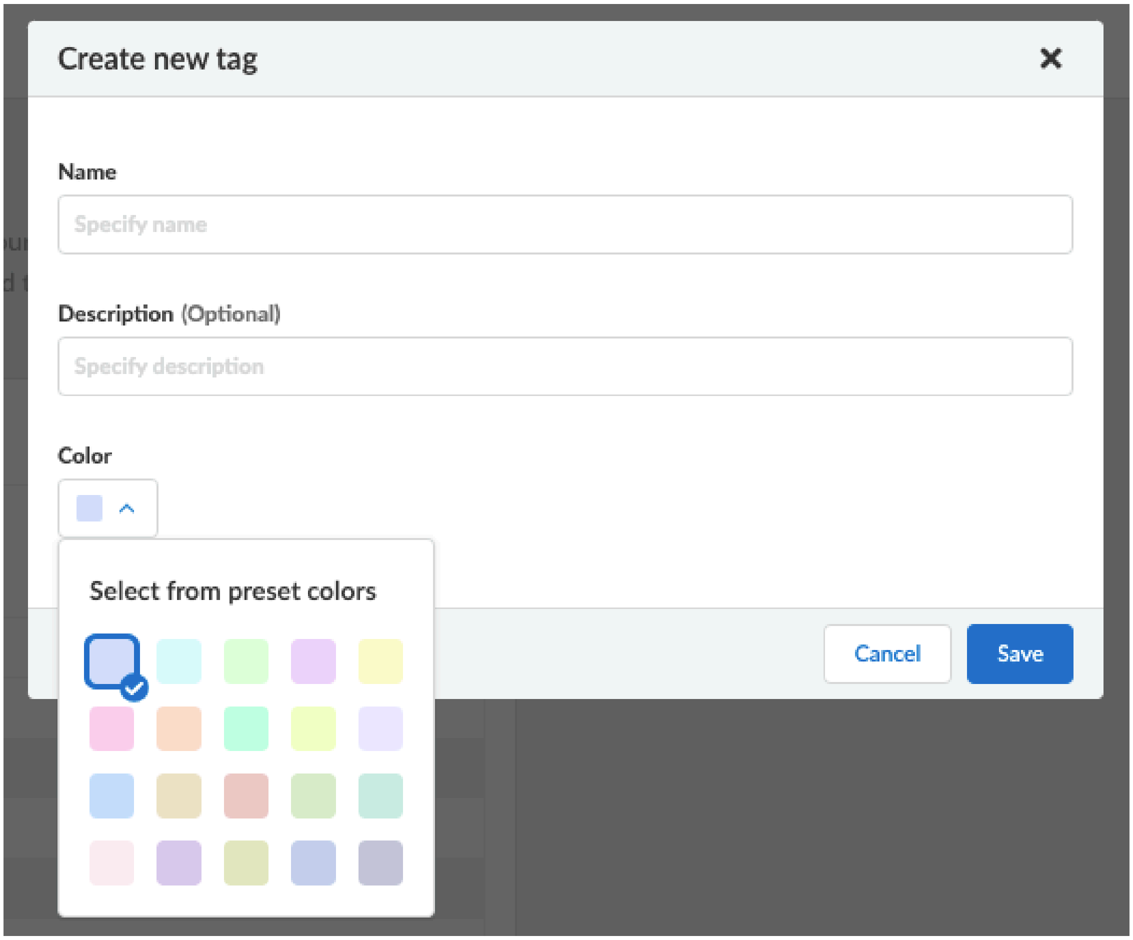
STEP 1 | To define a new tag, navigate to **Manage > Collections and Tags > Tags**.

Prisma Cloud ships with a predefined set of tags: Ignored, In progress, For review, DevOps notes. The predefined tags are editable and you can use them according to your needs.

STEP 2 | Click **Add Tag**.

STEP 3 | In the **Create new tag** dialog, enter a name and description.

STEP 4 | Pick a color for easy visibility and differentiation.



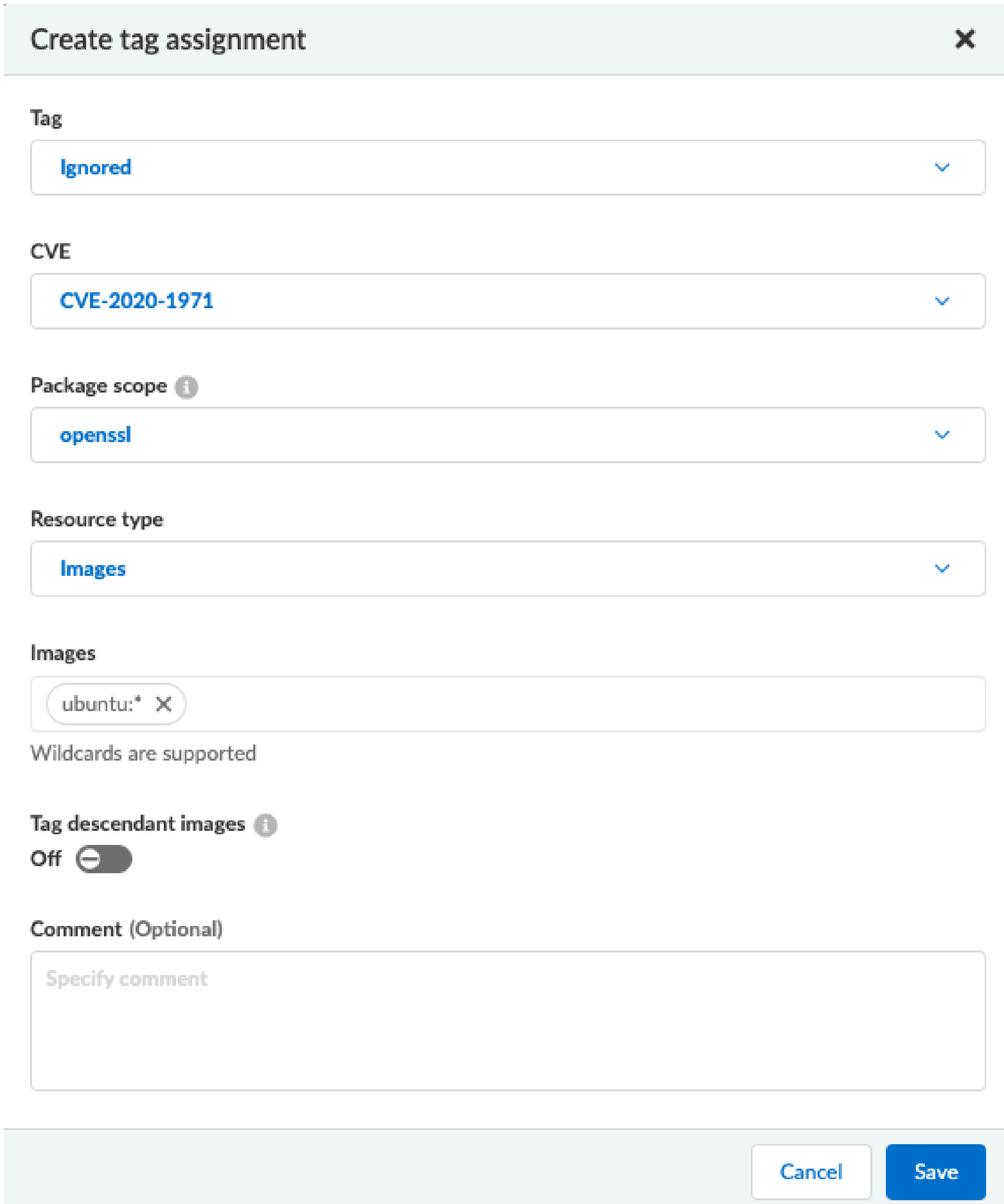
STEP 5 | Click **Save**.

Tag assignment

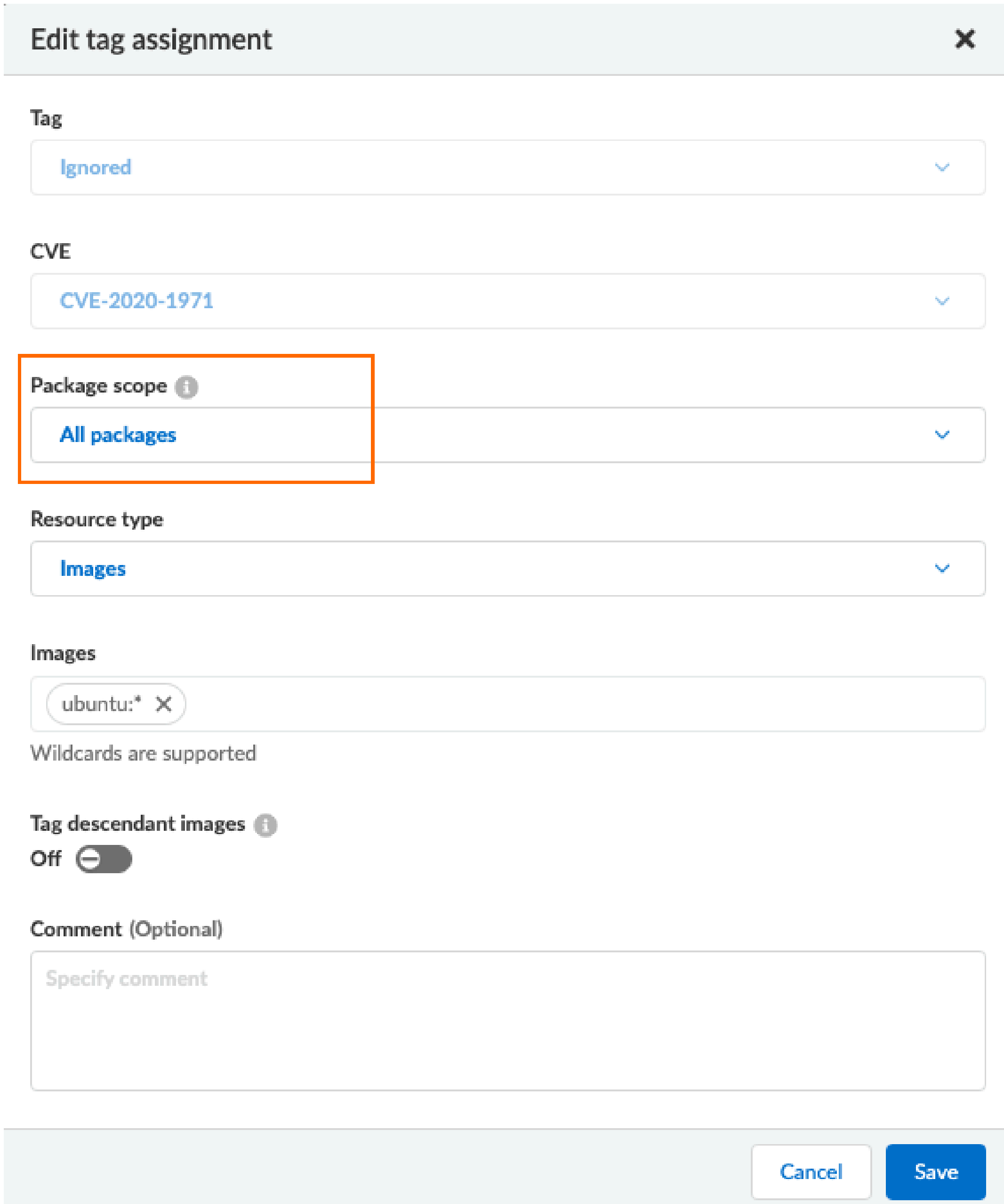
You can assign tags to vulnerabilities, and specify their scope based on CVE ID, packages and resources. Alternatively, you can manually tag vulnerabilities from scan reports.

Note that a tag assignment is uniquely identified by tag, CVE ID, package scope and resource type, therefore, you can not create multiple tag assignments for the same tag, CVE ID, package scope and resource type. To extend the scope of a tag applied to a CVE, edit its existing tag assignment to apply to more packages or resources.

For example, assign the tag *Ignored* to *CVE-2020-1971*, package *openssl*, and all *ubuntu* images as follows:



You can also adjust the scope of a tag assigned either from the tags management page or from scan reports. Click the **Edit** button to start editing the tag assignment. For example, extend the scope of the tag *Ignored* for *CVE-2020-1971* to all packages affected by this CVE by changing the **Package scope**:



As another example, after the *In progress* tag was assgined to *CVE-2019-14697* for specific *alpine* images from the scan reports, you can extend its scope so it will apply to all *alpine* images and their descendant images:

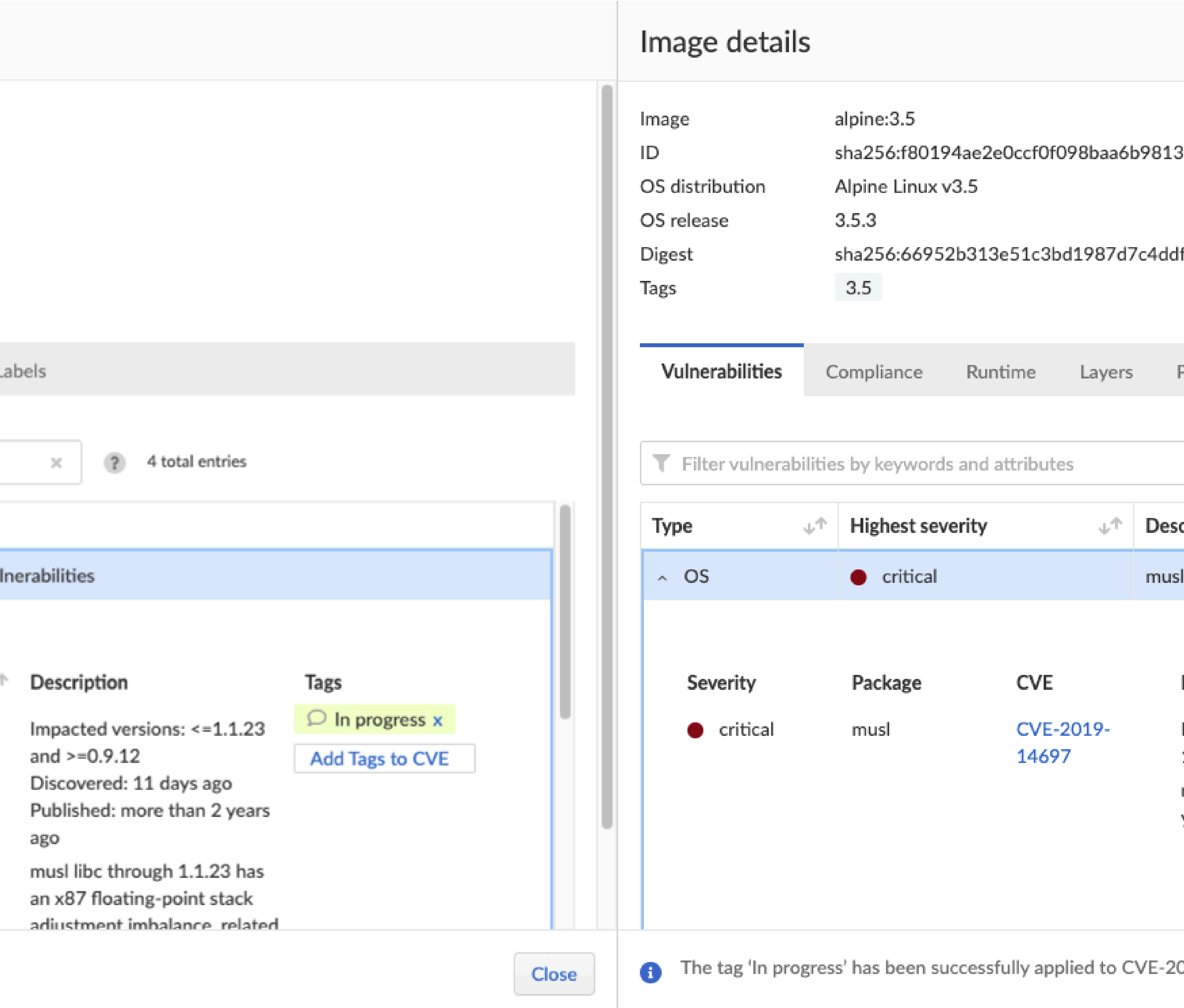
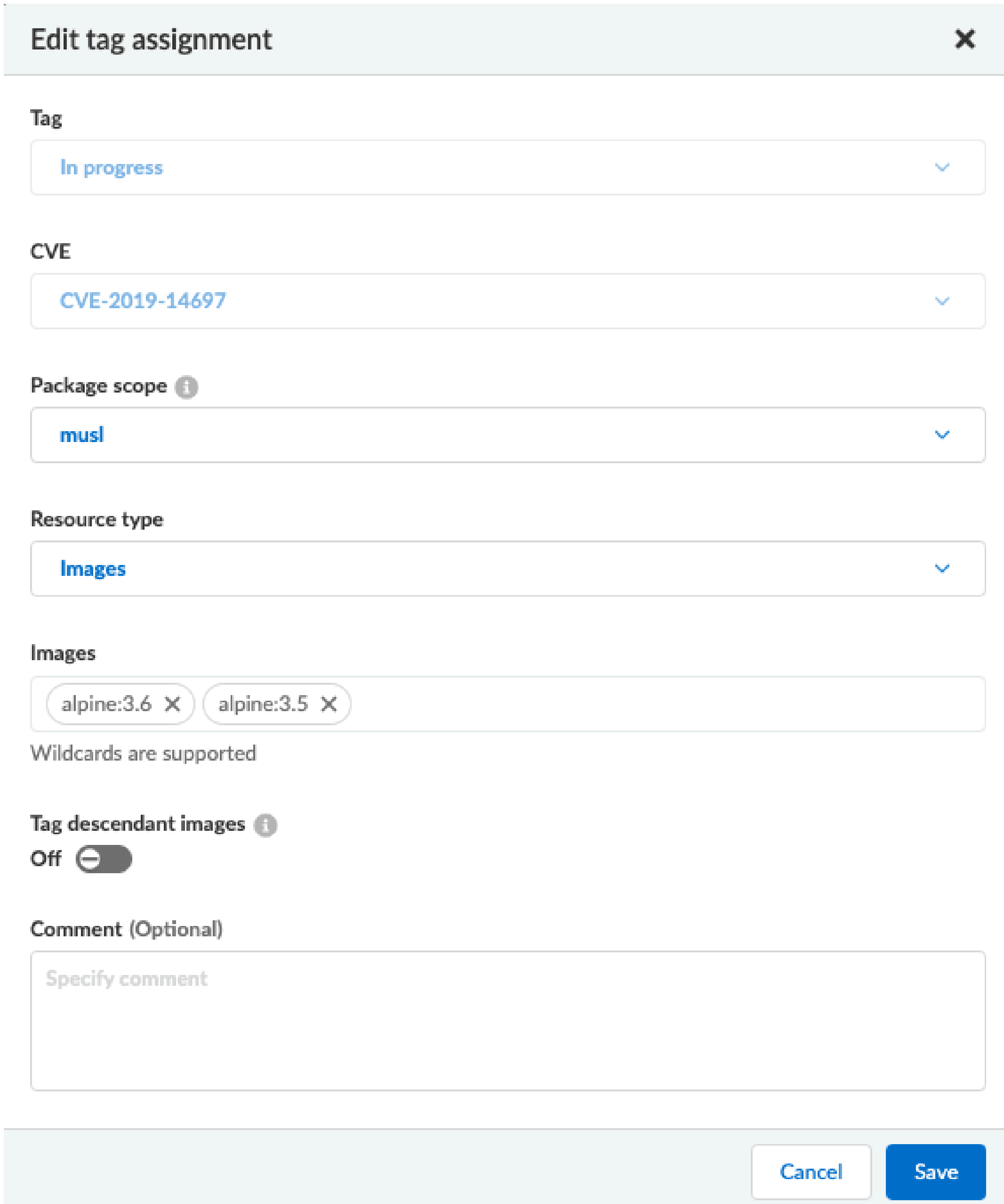
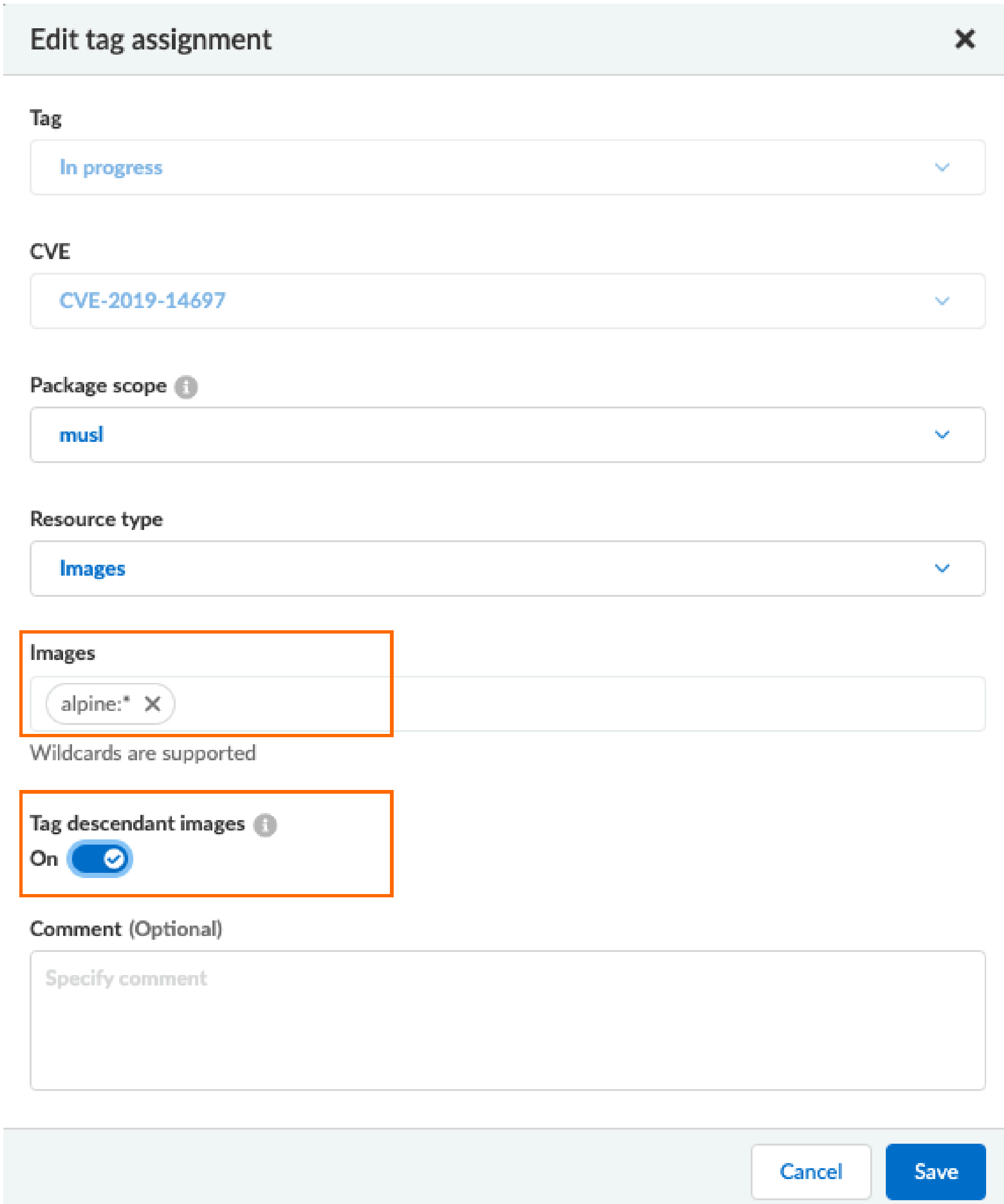
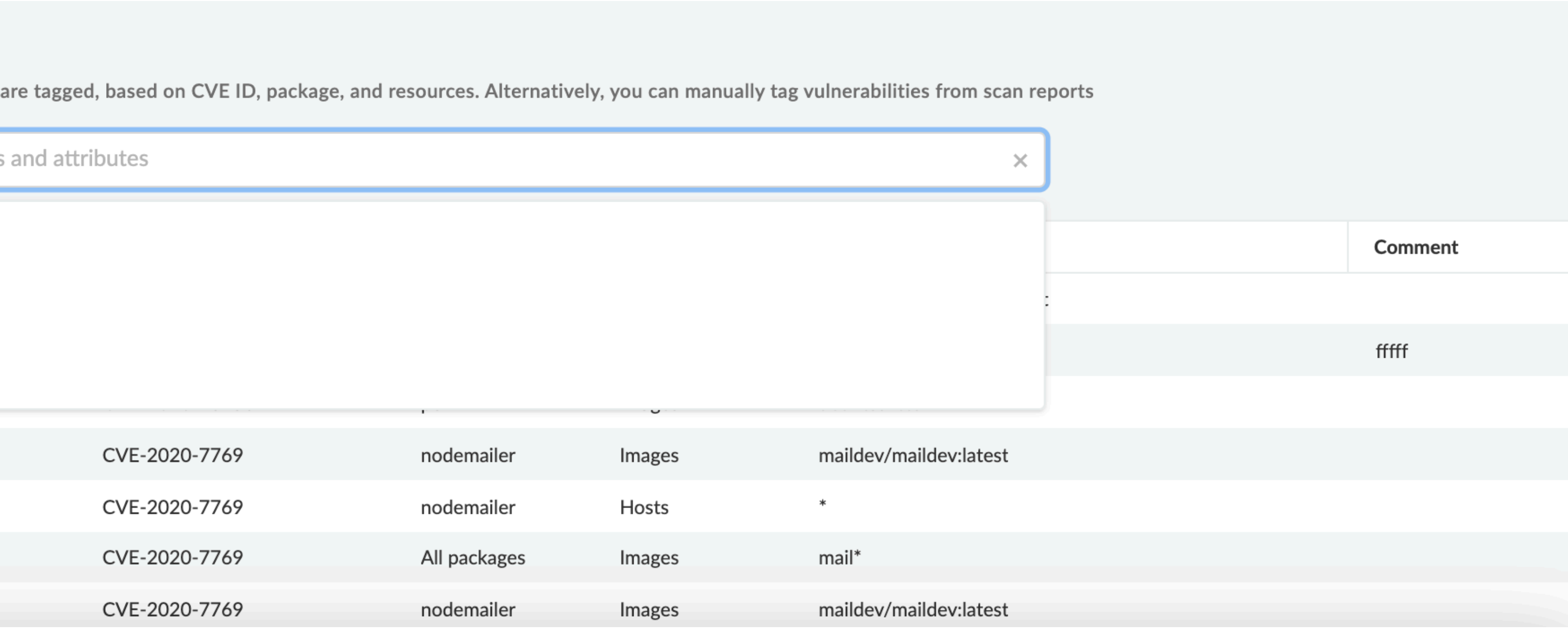


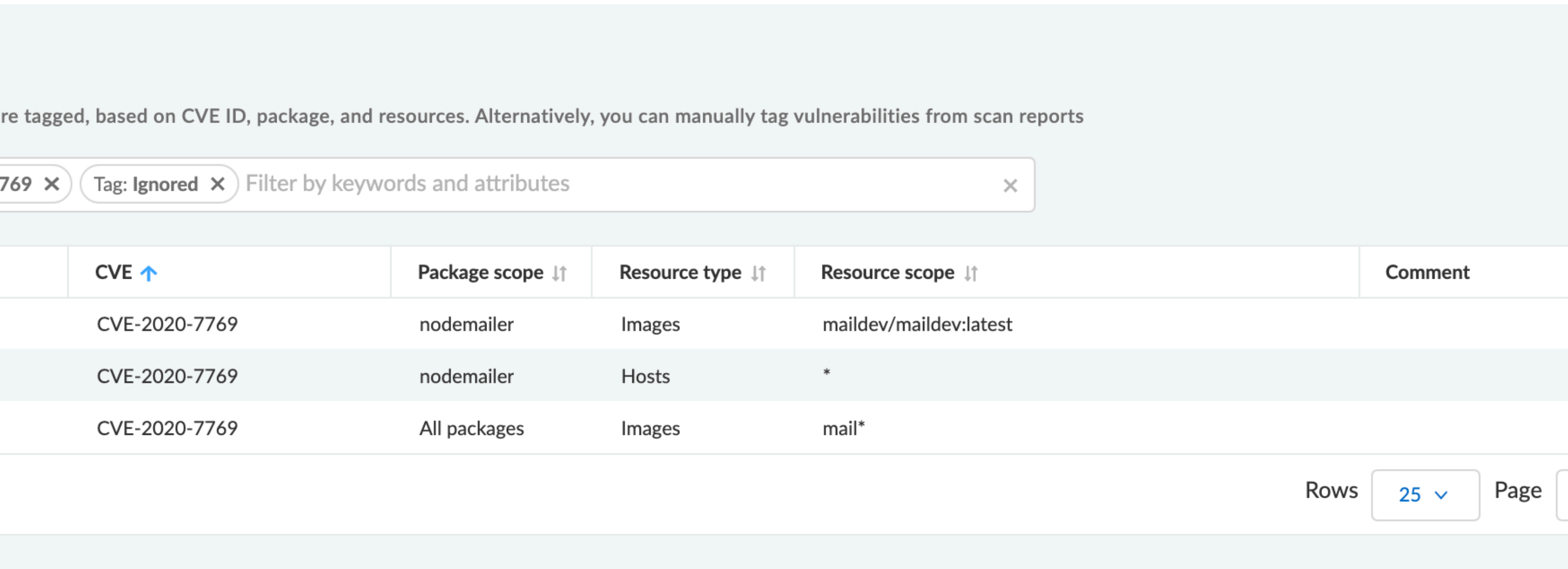
Image details
Image
alpine:3.5
ID
sha256:f80194ae2e0ccf0f098baa6b9813
OS distribution
Alpine Linux v3.5
OS release
3.5.3
Digest
sha256:66952b313e51c3bd1987d7c4ddf
Tags
3.5
Vulnerabilities
Compliance
Runtime
Layers
Filter vulnerabilities by keywords and attributes
Type
Highest severity
OS
critical
musl
Severity
Package
CVE
critical
musl
CVE-2019-14697
Labels
4 total entries
Description
Tags
Impacted versions: <=1.1.23 and >=0.9.12
Discovered: 11 days ago
Published: more than 2 years ago
musl libc through 1.1.23 has an x87 floating-point stack
In progress
Add Tags to CVE
Close
The tag 'In progress' has been successfully applied to CVE-20


Edit tag assignment
Tag
In progress
CVE
CVE-2019-14697
Package scope
musl
Resource type
Images
Images
alpine:3.6
alpine:3.5
Wildcards are supported
Tag descendant images
Off
Comment (Optional)
Specify comment
Cancel
Save



To easily navigate in multiple tag assignments, use the table filters on the **Tag assignment** table. Filter by CVE ID, tag, package scope, and resource type to quickly find all places a tag applies to.





STEP 1 | To assign tag to vulnerability, navigate to **Manage > Collections and Tags > Tags**.

STEP 2 | Click **Assign Tag**.

STEP 3 | In **Tag**, select the tag to assign.

STEP 4 | In **CVE**, select the CVE ID to assign the tag for.

STEP 5 | In **Package scope**, select the package to which the tag should apply. You can select **All packages** to apply the tag to all the packages affected by the CVE.

STEP 6 | In **Resource type**, select the type of resources to assign the tag for. You can select **All resources** to apply the tag to all the resources across your environment.

*VMware Tanzu droplets and running applications are being referenced as **Images**.*

STEP 7 | Once resource type is selected, specify the resources to which the tag should apply under **Images**, **Hosts**, **Functions**, or **Code repositories**. Wildcards are supported.

STEP 8 | (Optional) For images, turn on the **Tag descendant images** toggle to let Prisma Cloud automatically tag this CVE in all images where the base image is one of the images specified in the **Images** field.

For Prisma Cloud to be able to tag descendant images, first identify the base images in your environment under **Defend > Vulnerabilities > Images > Base images**.

STEP 9 | (Optional) In **Comment**, specify a comment for this tag assignment.

STEP 10 | Click **Save**.

Logon settings

Edit on GitHub

You can control how users access Prisma Cloud with logon settings.

Setting Console's token validity period

Prisma Cloud lets you set up long-lived tokens for access to the Console web interface and the API.

For security, users are redirected to the login page when an inactive Console session exceeds a configurable timeout. By default, the timeout is 30 minutes. This configurable timeout value also controls the validity period for API tokens.

For Console web interface tokens:

- If a user explicitly logs out, the claim to access Console is revoked.
- If Console is restarted, all users are automatically logged out.

Setting Console's token validity period

Tokens are issued to control access to both Console's web interface and the API. You can set a timeout for Console sessions and a validity period for API tokens.

STEP 1 | Open Console.

STEP 2 | Go to **Manage > Authentication > Logon**.

STEP 3 | Specify a value for **Timeout for inactive Console sessions**.

This value controls:

- Time, in minutes, that a Console session can be inactive. After the timeout expires, the user is redirected to the login page. In an active session, the token is automatically renewed when the time elapsed is greater than or equal to half the timeout value.
- Time, in minutes, that an API token is valid. After the token expires, a new one must be retrieved.

  The maximum value permitted for **Timeout for inactive Console sessions** is 71580 minutes.

STEP 4 | Click **Save**.

After you save your changes, Console redirects you to the login page for your changes to take effect.

Single sign-on to the Prisma Cloud Support

Prisma Cloud can allow single sign on and contextual help from the "?" button in the upper right hand corner of each Console page.

Our https://docs.twistlock.com site allows access when a valid token is issued from the Customer. Or in this case, the "?" contextual links can embed the token into the URL used to access the page.

STEP 1 | Open Console.

STEP 2 | Go to **Manage > Authentication > Logon**.

STEP 3 | Set the toggle for **Enable context sensitive help and single sign on to the Twistlock Support site**.

When set to on (default), the token will be embedded into the contextual help link. when set to off, it will not be and you will need to enter the token manually.

STEP 4 | Click **Save**.

After saving your changes, Console redirects you to the login page for your changes to take effect.

Basic authentication to Console and API

Twistlock lets you disable basic authentication to the Console and API. Basic authentication is used in connections from *twistcli*, the API, and Jenkins.

With *twistcli*, you need to use the '--*token*' option to authenticate with the Console for image scanning and other operations that access Console. This is the same token you receive form the /api/v1/authenticate API endpoint. For more inforomation, see the API documentation.

With the API, you would have use the authenticate endpoint to generate an authentication token to access any of the endpoints. Accessing the APi with Basic Authentication would not be allowed.

With Jenkins, there is no option at this point to use the Jenkins plugin and have basic authentication disabled. An option would be to use *twistcli* within Jenkins. this would require a step in the pipeline to retrieve an authentication token from the API for the scan to be completed.

STEP 1 | Open Console.

STEP 2 | Go to **Manage > Authentication > Logon**.

STEP 3 | Set the toggle for **Disable basic authentication to Console and API**.

When set to on, basic authentication will be disabled for the Console and API. You will not loose access to the Console from the login page. All of your user account will still be active and will still have access to login to the Console.

STEP 4 | Click **Save**.

After saving your changes, Console redirects you to the login page for your changes to take effect.

Strict certificate validation in Defender

Twistlock Console provides Defender installation scripts which use *curl* to transfer data from Console. By default, scripts copied from Console append the '-*k*' option, also known as '--*insecure*', to curl commands. This option lets curl proceed even if server connections are otherwise considered insecure.

Console provides a global option to disable the '-*k*' argument for curl commands.

STEP 1 | Open Console.

STEP 2 | Go to **Manage > Authentication > Logon**.

STEP 3 | Set the toggle for **Require strict certificate validation in Defender installation links**.

When set to **On**, Defender installation scripts copied from Console do not use the '-*k*' option with curl when transferring data from Console. In addition, the piped *sudo bash* command passes the '-*v*' option to defender.sh to secure secondary curl commands in the defender.sh script.

STEP 4 | Click **Save**.

After saving your changes, Console redirects you to the login page for your changes to take effect.

Strong passwords for local accounts

Twistlock can enforce the use of a strong password. A strong password has the following requirements:

- Cannot be the same as the username.
- Must be at least 12 characters.
- Must contain one of each of the following: uppercase character, lowercase character, number, special character.
- List of special characters: *~!@#$%^&*()-_=+|[{}];:'\",<.>/?"*

STEP 1 | Open Console.

STEP 2 | Go to **Manage > Authentication > Logon**.

STEP 3 | Set the toggle for **Require strong passwords for local accounts**.

When enabled, strong passwords are required for passwords of newly created accounts or when existing passwords are changed. Enabling this setting doesn't force existing accounts to change their password or disable access to any accounts.

STEP 4 | Click **Save**.

After saving your changes, Console redirects you to the login page for your changes to take effect.

Reconfigure Prisma Cloud

Edit on GitHub

In many cases, you will set up *twistlock.cfg* before you install Prisma Cloud. However, in some cases, you might want to change some parameters in *twistlock.cfg* after Prisma Cloud has already been installed. To reconfigure Prisma Cloud with an updated *twistlock.cfg*, run the *twistlock.sh* installer script again.

STEP 1 | Extract the release tarball to a new location on your host.

Make sure this location does not have any previous Prisma Cloud install files.

```
$ tar -xvf twistlock_<VERSION>.tar.gz
```

STEP 2 | Update *twistlock.cfg* with your new settings.

```
$ vim twistlock.cfg
```

STEP 3 | Reload twistlock.cfg.

```
$ sudo ./twistlock.sh onebox
```

This command assumes that both *twistlock.sh* and *twistlock.cfg* reside in the same directory. To specify a configuration file in a different directory, use the -c option.

The old configuration is stored in */var/lib/twistlock/scripts/twistlock.cfg.old*

Subject Alternative Names

Edit on GitHub

You can add or remove Subject Alternative Names (SANs) to Console's certificate. The subjectAltName extension is described in RFC6125. It defines a mechanism for adding identities to Public Key Infrastructure X.509v3 (PKIX) certificates.

Defender communicates with Console using Transport Layer Security (TLS). When Defender tries to establish a secure connection with Console, it must validate Console's identity. Defender checks a reference identity against the identifiers presented in Console's PKIX certificate, searching for a match. The reference identity is set when you deploy Defender. It's the name you configured Defender to use to connect to Console.

When deploying Prisma Cloud Console, setting up a DNS name for it is considered a best practice. RFC6125 says:

"IP addresses are not necessarily reliable identifiers for application services because of the existence of private internets [PRIVATE], host mobility, multiple interfaces on a given host, Network Address Translators (NATs) resulting in different addresses for a host from different locations on the network, the practice of grouping many hosts together behind a single IP address, etc. Most fundamentally, most users find DNS domain names much easier to work with than IP addresses, which is why the domain name system was designed in the first place."

Adding a SAN to Console's certificate

Add a SAN to Console's certificate directly from Console's web interface.

STEP 1 | Open Console.

STEP 2 | Go to **Manage > Defenders > Names**.

STEP 3 | Click **Add SAN**.

STEP 4 | Enter a DNS name or IP address.

STEP 5 | Click **Add**.

WildFire Settings

Edit on GitHub

WildFire is Palo Alto Networks' malware detection engine, and it provides malware detection for both known and unknown threats.

Wildfire analysis is provided without additional costs, but this may change in future releases. The service is available in Prisma Cloud for malware analysis as part of containers Continuous Integration (CI) and as runtime protection for containers and hosts. Access to WildFire is provided as a new subscription that is specific to Prisma Cloud Compute, and doesn't affect any existing WildFire subscriptions.

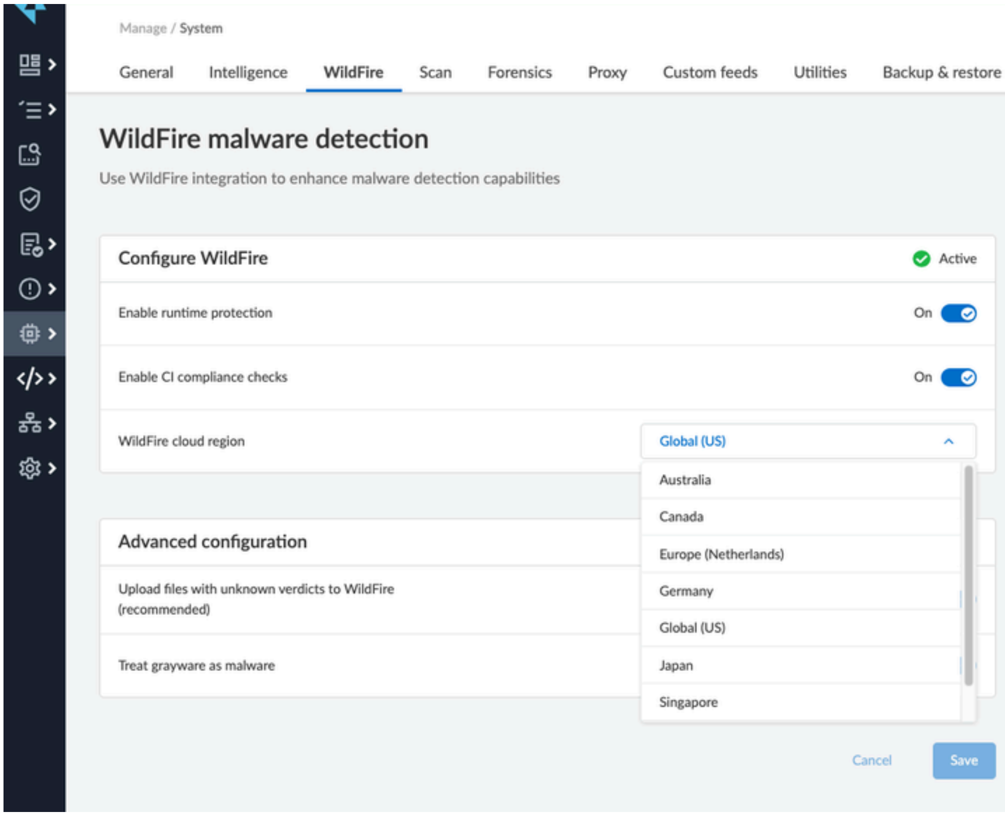
To check a file verdict, the file hash is calculated and sent to WildFire for a verdict. If a file with the specified hash was already uploaded for a verdict, Wildfire provides an instantaneous verdict. You can send unknown files to WildFire for a full analysis, which includes machine based static analysis, dynamic analysis with detonation of the file in a sandbox, and behavioral analysis.

WildFire supports the following verdict types: benign, malware, grayware, and unknown:

- **Benign:** The sample is safe and doesn't exhibit malicious behavior.
- **Grayware:** The sample doesn't pose a direct security threat, but might display otherwise obtrusive behavior. Grayware typically includes:
  - Adware
  - Spyware
  - Browser Helper Objects (BHOs).
- **Malicious:** The sample is malware and poses a security threat. Malware can include:
  - Viruses
  - Worms
  - Trojans
  - Remote Access Tools (RATs)
  - Rootkits
  - Botnets
- **Unknown:** The file hasn't been uploaded previously to Wildfire for analysis. Full analysis can be performed on file upload.

Configuration of the WildFire malware analysis service is done via **Manage > System > WildFire**.

- **Wildfire malware detection:** Enable WildFire malware detection.
- **Status:** Shows the current activation state of WildFire. The status is updated upon successful activation of the Wildfire service.
- **WildFire cloud region:** The WildFire service is available in multiple locations to meet local privacy requirements and reduce latency for communication to the service.



All Defenders connected to a given Prisma Cloud Console must use the same Wildfire service. This WildFire service is used for file verdicts and to upload files for full analysis. You should select the WildFire service closest to where most defenders are, or based on your privacy requirements. Defenders must be able to access the relevant WildFire service configured over https (port 443) based on the following URLs:

- Global (US): wildfire.paloaltonetworks.com
- Australia: au.wildfire.paloaltonetworks.com
- Canada: ca.wildfire.paloaltonetworks.com
- Europe (Netherlands): eu.wildfire.paloaltonetworks.com
- Japan: jp.wildfire.paloaltonetworks.com
- Singapore: sg.wildfire.paloaltonetworks.com
- United Kingdom: uk.wildfire.paloaltonetworks.com

For WildFire activation and license renewals, the Prisma Cloud Console must be able to access the Intelligence Stream (IS) server at https://intelligence.twistlock.com.

- **Use WildFire for runtime protection:** Enable WildFire malware scanning in runtime for containers and hosts. Go to the rule's **Anti-malware** tab, to configure the preferred effects per rule.
- **Use WildFire for CI compliance checks:** Enable WildFire malware scanning for containers CI checks. WildFire scans malware as part of Twistlock labs image check (ID 422).
- **Upload files with unknown verdicts to WildFire:** Determine whether files with unknown verdict are sent to WildFire for full analysis. When disabled, WildFire only provides verdicts for files that have been uploaded to WildFire via a different client.

- **Treat grayware as malware:** Use a more restrictive approach and treat files with grayware verdict as malware.

Currently Prisma Cloud Compute uses WildFire for file verdicts only in the following scenarios:

- Hosts runtime:
  - ELF files written to a Linux host file system in runtime, which are not deployed via a package manager.
  - Files must be smaller than 100MB due to the size limit of WildFire.
- Container runtime and CI:
  - ELF files written to a Linux container file system in runtime. Malware analysis not supported for other file types.

    During CI scanning, WildFire analyses only executable files that were not written as part of a package installation.
  - WildFire doesn't scan shared objects.
  - File must be smaller than 100MB due to the size limit of WildFire.

- *You can submit up to 5000 files per day, and get up to 50,000 verdicts on your submissions to the WildFire service.*
- *Wildfire is supported on Linux only.*

  Windows containers and hosts aren't currently supported.

Log Scrubbing

Edit on GitHub

Prisma Cloud Compute Runtime events may include sensitive information that's found in commands that are run by protected workloads, such as secrets, tokens, PII or other information considered to be personal by various laws and regulations.

Using the Runtime log scrubbing capabilities, you can filter such sensitive information and ensure that it is not included in the Runtime findings (Forensics, Incidents, audits, etc.).

You can filter your Runtime sensitive data out using the automatic scrubbing capability, as well as using custom scrubbing rules. Follow the documentation instructions to learn more about these two options.

Sensitive information from WAAS logs can be scrubed as well, see WAAS Log Scrubbing to learn more.

Automatically scrub secrets from runtime events

To help identify and filter secrets that commonly appear in the Runtime monitored commands, we added the capability to automatically scrub known sensitive phrases and words such as "secrets", "token", etc. from your events. The detected sensitive data will be replaced in the events by *"[*****]"*.

Automatically scrubbing secrets will be ***enabled*** *by default when upgrading Console from 21.08 to 22.01.*

Enable/Disable the automatic scrubbing:

Enable automatic log scrubbing.

STEP 1 | Open the Console, and go to **Manage > General**.

STEP 2 | Select the desired mode in the **Automatically scrub secrets from runtime events** toggle.

Add/Edit custom scrubbing rule

Create or edit log scrubbing rules.

STEP 1 | Open the Console, and go to **Manage > General**.

STEP 2 | In the **Custom log scrubber** section select Runtime or WAAS.

STEP 3 | Click on **Add rule** or select an existing rule.

STEP 4 | Enter the rule **Name**.

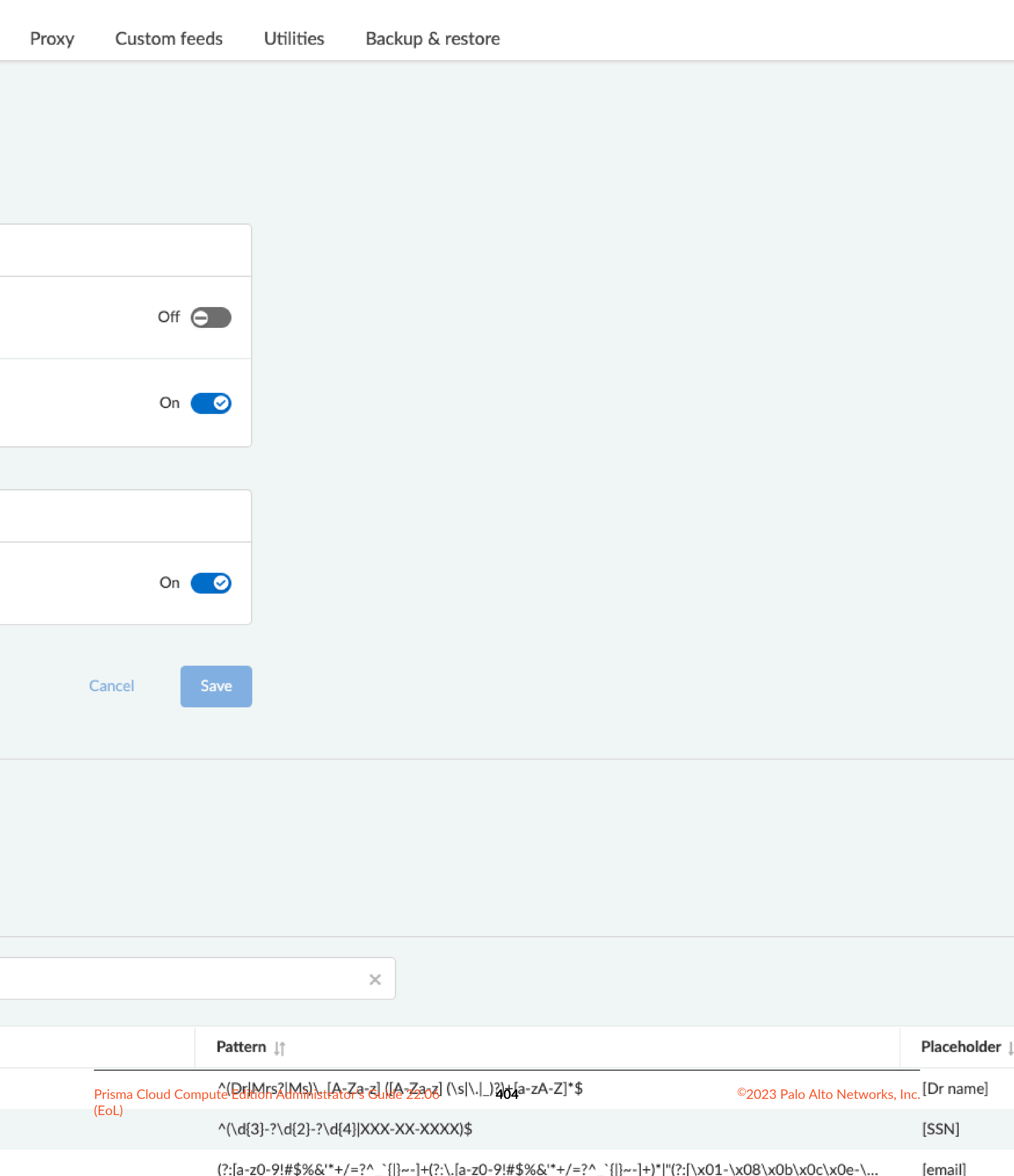
STEP 5 | Provide a matching **Pattern** in the form of a regular expression (re2), e.g. ^*sessionID$*, *key-[a-zA-Z]{8,16}*.

STEP 6 | Provide a **Placeholder** string e.g. *[scrubbed email]*.

1. Placeholder strings indicating the nature of the scrubbed data should be used as users will not be able to see the underlying scrubbed data.

STEP 7 | Click **Save**.

- *Data will now be scrubbed from any Runtime and WAAS event before it is written (either to the Defender log or syslog) and sent to the console.*
- *The automatic scrubbing and custom scrubbing are independent, meaning that you can choose to use each one of them separately.*
- *Data will be scrubbed only in messages that are generated while the scrubbing toggle or scrubbing rule are **enabled**. Messages that were generated **before** enabling one of the scrubbing configurations above or **after** disabling them, won't be scrubbed.*
- *The WAAS scrubbing rules are synced with the rules in **Defend > WAAS > Log scrubbing**.*
- *Serverless Runtime events are not scrubbed.*


Proxy
Custom feeds
Utilities
Backup & restore
Off
On
On
Cancel
Save
Pattern
Placeholder
^(Dr|Mrs?|Ms)\. [A-Za-z] ([A-Za-z] (\s|\.|_)?)+[a-zA-Z]*$
[Dr name]
^(\d{3}-?\d{2}-?\d{4}|XXX-XX-XXXX)$
[SSN]
(?:[a-z0-9!#$%&'*+/=?^_`{|}~-]+(?:\.[a-z0-9!#$%&'*+/=?^_`{|}~-]+)*|"(?:[\x01-\x08\x0b\x0c\x0e-\...
[email]

Clustered-DB

Edit on GitHub

Running Prisma Cloud Compute in the clustered-DB configuration lets you sync Console's database across multiple instances of Console. When using the clustered-DB deployment, all Consoles must be accessible with the same DNS name and connected to the same load balancer.

The Consoles in a clustered-DB pool can be deployed in the same region across different availability zones.

A new Prisma Cloud Compute self-hosted setup is required to create a clustered-DB deployment. That is, all Consoles must be new. Accordingly, all Defenders must be new as well.

Recommendations for your clustered-DB setup

The information in the following sections help you make informed decisions when selecting pool size, load balancer configuration, and storage types.

Number Of Consoles in the clustered-DB pool

The clustered-DB mechanism continuously requires the pool to choose a primary Console. This is done by the replica sets election mechanism. The election determines which member will become primary. Replica sets can trigger an election in response to a variety of events, such as: adding a new member, initiating a pool, and losing connectivity to the primary for more than the configured timeout.

An odd number of Consoles is recommended for better fault tolerance. The table below describes the fault tolerance for 3/5/7 Consoles:

Number of Members	Majority Required to Elect a New Primary	Fault Tolerance
3	2	1
4	3	1
5	3	2
6	4	2
7	4	3

Load balancer configuration

When configuring your load balancer, make sure to set up the following:

- TCP (Network) load balancer.
- TLS health through.
- Except for traffic on the used ports (8083, 8084 the default).

- To ensure LB can identify unavailable Consoles, we recommended that you set a health check to the Console ping API: https://<ip:port>/api/v1/_ping.

Performance implications

When deploying a clustered-DB setup on AWS, we recommend that you use EBS-optimized instances.

Custom certificates

You can provide a custom certificate to the clustered-DB Consoles pool. When configuring custom certificates, make sure that all Consoles and Defenders know all clustered-DB Consoles and the load balancer. To do so, add all Console addresses and load balancer names to the certificate's SAN.

All Consoles and Defenders must be signed with the same CA.

This is also relevant for configuring custom certs from a predefined directory.

Guidelines

Before you begin, make sure to follow these guidelines:

- Managing and monitoring a clustered-DB setup requires System Admin permissions.
- When using IAM roles, all Console nodes must be assigned with the same role.
- The clustered-DB pool should be created with fresh Consoles and Defenders.
- All Consoles should run the same **major and minor version**.
- All Consoles should be able to communicate with each other on port 27017.
- The number of the total Consoles in the pool should be between 3 to 7. See Number Of Consoles in the clustered-DB pool.
- Create a load balancer using the instructions described in Load balancer configuration This load balancer will be used for Defenders, twistcli, API and TLS communication with the clustered-DB pool.

Creating the clustered-DB pool

STEP 1 | Deploy the clustered-DB pool's Consoles without initializing them with usernames and passwords.

1. Option 1 - Deploy Console using YAML/Helm charts.

   ```
   ./twistcli console export kubernetes <optional parameters> --
   clustered-db
   ```

   ```
   kubectl create -f twistlock_console.yaml
   ```

   When deploying Consoles on the same cluster, you can use a statefulset deployment or use different namespaces for the Console installation. Make sure to change the YAML/Helm accordingly.
2. Option 2 - Deploy the Console as a single Console.
   1. Add *CLUSTERED_DB_ENABLED=true* configuration to the *twistlock.cfg* file
   2. Install the Console (not the onebox installation), without initializing it.

      For example *sudo ./twistlock.sh console*

STEP 2 | Configure the load balancer to send traffic to the Console services.

STEP 3 | Choose one of the Consoles and initialize it with a user, password, and license.

When initializing the Console, access it directly, rather than through the load balancer.

This first Console is considered the seed Console.

STEP 4 | Check the clustered-DB status against the initialized Console:

```
./twistcli clustered-db status --address https://<console
 address>:8083/ --password <password>
```

You should expect to see the following result:

```
Clustered DB status:
Last updated: 28 Mar 22 18:00 UTC
Load balancer address:
Members:
1. Address: <console name>, State: PRIMARY
```

STEP 5 | Add a load balancer to the clustered-DB.

The Console's address should be static and available for the other Consoles. You can choose a static IP address or a DNS name.

Run the following command to set up the clustered-DB load balancer address. The <console address> should be the initialized Console address:

```
./twistcli clustered-db configure --load-balancer-address <load-
balancer-address> --address https://<console address>:8083/ --
password <password>
```

STEP 6 | (Optional) Edit the seed address.

This command can be useful if the initialized Console address is not accessible for the other Consoles that you are about to add to the pool. Note you can use this command after adding the first Console to the pool (the seed Console), meaning that this command can't be executed after adding the other Consoles to the pool.

```
./twistcli clustered-db configure --seed-console-address <service-
name/service-name.namespace/host name/IP address> --password
 <password>
```

STEP 7 | Add the other Consoles to the pool.

The Consoles' addresses should be static and available for the other Consoles. You can choose a static IP address or a DNS name.

Run the following command in order to add members to the clustered-DB pool. You can add single or multiple addresses at once:

```
./twistcli clustered-db add --member-address <member adderss>
 --member-address <member adderss> … --address https://<console
 address>:8083/ --password <password>
```

STEP 8 | Check status the pool status:

Now it's possible to check the pool status against the load balancer address:

```
./twistcli clustered-db status --address https://<load balancer
 address>:8083/ --password <password>
```

Expected output:

```
Clustered DB status:
Last updated: 28 Mar 22 18:25 UTC
Load balancer address: load_balancer_address
Members:
1. Address: Console1_address, State: PRIMARY
2. Address: Console2_address, State: SECONDARY
3. Address: Console3_address, State: SECONDARY
```

Clustered-DB potential statuses

The clustered-DB status call restrains the status of the entire pool and the status for each one of the members. Status is a string representation of the member's state, from the cluster perspective. The last updated field represents the time when the status was last updated.

```
./twistcli clustered-db status --address https://<load balancer
 address>:8083/ --password <password>
```

See the available statuses in the list below:

Name	State Description
STARTUP	Not yet an active member of any set. All members start up in this state.
PRIMARY	The member in state primary the primary member of the pool. Eligible to vote.
SECONDARY	A member in state secondary is replicating the data store. Eligible to vote.
RECOVERING	Members either perform startup self-checks, or transition from completing a rollback or resync. Eligible to vote.
STARTUP2	The member has joined the set and is running an initial sync. Eligible to vote. NOTE this member is not eligible to vote and cannot be elected during the initial sync process.
UNKNOWN	The member's state, as seen from another member of the set, is not yet known.
DOWN	The member, as seen from another member of the set, is unreachable.
ROLLBACK	This member is actively performing a rollback. Eligible to vote. The member is not accessible during the rollback time frame.
REMOVED	This member was once in a replica set but was subsequently removed. This status can be available for a very short period of time after removing a member. When the remove action is complete, the member will no longer appear in the status.

Remove members

Follow the steps below to remove a member (Console) from the pool. Note that after removing a member, this Console cannot be reused.

STEP 1 | Remove the member from the LB settings.

STEP 2 | Remove a member from the pool using the following command:

```
./twistcli clustered-db remove --address https://<load balancer
 address> -u user -p password --member-address <member-address-to-
remove>
```

STEP 3 | Delete the removed Console instance, since it's not reusable.

After removing a member from the pool, the deleted Console will remain in the existing DB. This Console can't be added to the pool again since it's already initialized. You won't be able to return the same member to the pool, unless you delete the Console and create a new non-initialized one.

Console disconnection

If Console fails, the clustered-DB pool will choose a new primary Console. The primary selections might cause a short downtime to make the transition. The downtime period depends on different factors (sufficient number of members to vote, network latency, etc). Typically the process will take about 30 seconds.

If a single member is disconnected from the pool for a long period of time, it might take a while for it to return. This is due to possible DB differences. If the delta between the disconnected member and the pool DB is significant, the member DB will be restored from the current pool DB.

Upgrade clustered-DB Consoles

All Consoles should run the same **major and minor version** (i.e., exactly the same x.y.z version). All clustered-DB Consoles should be upgraded within a reasonable amount of time, to make sure that all of them will run with the same version shortly. For example, if the DB was upgraded to x.y.z +1, members still running the previous version x.y.z won't be able to become primary. They just replicate the DB.

Limitations

The clustered-DB setup has the following limitations:

- You cannot deploy projects when using clustered-DB.
- Consoles running on Fargate aren't supported.
- Backup and restore: clustered-DB can track only periodic backups only (daily, weekly, monthly), but not on demand. The backup is taken from all of the clustered-DB pool members.

Permissions by feature

Edit on GitHub

When you set up Prisma Cloud Compute to secure your cloud workloads, you'll need to ensure you've granted Prisma Cloud the right permissions. The following tables list the permissions required for each of Compute's protection capabilities.

AWS

Feature	Protection Mode	Permissions	Condition	Prisma Cloud Templates Status	Role/Policy
Registry Scan	Monitor AND Monitor & Protect	Update both read-only & read-write templates			
		ecr:GetAuthorizationToken		Verified	PrismaCloud-ReadOnly-Policy-Compute
		ecr:BatchCheckLayerAvailability		Verified	PrismaCloud-ReadOnly-Policy-Compute
		ecr:GetDownloadUrlForLayer		Verified	PrismaCloud-ReadOnly-Policy-Compute
		ecr:GetRepositoryPolicy		Verified	arn:aws:iam::aws:policy/SecurityAudit
		ecr:DescribeRepositories		Verified	arn:aws:iam::aws:policy/SecurityAudit
		ecr:ListImages		Verified	arn:aws:iam::aws:policy/SecurityAudit
		ecr:DescribeImages		Verified	arn:aws:iam::aws:policy/SecurityAudit
		ecr:BatchGetImage		Verified	PrismaCloud-ReadOnly-

Feature	Protection Mode	Permissions	Condition	Prisma Cloud Templates Status	Role/Policy
					Policy-Compute
		ecr:GetLifecyclePolicy		Verified	arn:aws:iam::aws:policy/SecurityAudit
		ecr:GetLifecyclePolicyPreview		Verified	PrismaCloud-ReadOnly-Policy-Compute
		ecr:ListTagsForResource		Verified	arn:aws:iam::aws:policy/SecurityAudit
		ecr:DescribeImageScanFindings		Verified	arn:aws:iam::aws:policy/SecurityAudit
Serverless Scan	Monitor AND Monitor & Protect	Update both read-only & read-write templates			
		lambda:ListFunctions		Verified	arn:aws:iam::aws:policy/SecurityAudit
		lambda:GetFunction		Verified	PrismaCloud-ReadOnly-Policy-Compute
		iam:GetPolicy		Verified	arn:aws:iam::aws:policy/SecurityAudit
		iam:GetPolicyVersion		Verified	arn:aws:iam::aws:policy/SecurityAudit
		iam:GetRole		Verified	arn:aws:iam::aws:policy/SecurityAudit
		iam:GetRolePolicy		Verified	arn:aws:iam::aws:policy/SecurityAudit
		iam:ListAttachedRolePolicies		Verified	arn:aws:iam::aws:policy/SecurityAudit

Feature	Protection Mode	Permissions	Condition	Prisma Cloud Templates Status	Role/Policy
		iam:ListRolePolicies		Verified	arn:aws:iam::aws:policy/ SecurityAudit
		lambda:GetLayerVersion		Verified	PrismaCloud-ReadOnly-Policy-Compute
		kms:Decrypt		Verified	PrismaCloud-ReadOnly-Policy-Compute
Serverless Auto Defend	Monitor & Protect ONLY	Update read-write templates ONLY			
		lambda:PublishLayerVersion		Verified	PrismaCloud-Remediation-Compute-Policy-ServerlessAutoDefend
		lambda:UpdateFunctionConfiguration		Verified	PrismaCloud-IAM-Remediation-Policy
		lambda:GetLayerVersion		Verified	PrismaCloud-ReadOnly-Policy-Compute
		lambda:GetFunctionConfiguration		Verified	arn:aws:iam::aws:policy/ SecurityAudit
		iam:SimulatePrincipalPolicy		Verified	arn:aws:iam::aws:policy/ SecurityAudit
		lambda:GetFunction		Verified	PrismaCloud-ReadOnly-Policy-Compute

Feature	Protection Mode	Permissions	Condition	Prisma Cloud Templates Status	Role/Policy
		lambda:ListFunctions		Verified	arn:aws:iam::aws:policy/SecurityAudit
		iam:GetPolicyVersion		Verified	arn:aws:iam::aws:policy/SecurityAudit
		iam:GetRole		Verified	arn:aws:iam::aws:policy/SecurityAudit
		iam:ListRolePolicies		Verified	arn:aws:iam::aws:policy/SecurityAudit
		iam:ListAttachedRolePolicies		Verified	arn:aws:iam::aws:policy/SecurityAudit
		iam:GetRolePolicy		Verified	arn:aws:iam::aws:policy/SecurityAudit
		iam:GetPolicy		Verified	arn:aws:iam::aws:policy/SecurityAudit
		lambda:ListLayerVersions		Verified	arn:aws:iam::aws:policy/SecurityAudit
		lambda:ListLayers		Verified	arn:aws:iam::aws:policy/SecurityAudit
		lambda:DeleteLayerVersion		Verified	PrismaCloud-Remediation-Compute-Policy-ServerlessAutoDefend
		kms:Decrypt		Verified	PrismaCloud-ReadOnly-Policy-Compute
		kms:Encrypt		Verified	PrismaCloud-Remediation-Compute-Policy-AgentlessScanning

Feature	Protection Mode	Permissions	Condition	Prisma Cloud Templates Status	Role/Policy
		kms:CreateGrant		Verified	PrismaCloud-Remediation-Compute-Policy-AgentlessScanning
Serverless Radar	Monitor & Protect ONLY	Update read-write templates ONLY			
		cloudwatch:DescribeAlarms		Verified	arn:aws:iam::aws:policy/SecurityAudit
		iam:GetPolicyVersion		Verified	arn:aws:iam::aws:policy/SecurityAudit
		iam:GetRole		Verified	arn:aws:iam::aws:policy/SecurityAudit
		iam:GetPolicy		Verified	arn:aws:iam::aws:policy/SecurityAudit
		iam:GetRolePolicy		Verified	arn:aws:iam::aws:policy/SecurityAudit
		iam:ListRolePolicies		Verified	arn:aws:iam::aws:policy/SecurityAudit
		iam:ListAttachedRolePolicies		Verified	arn:aws:iam::aws:policy/SecurityAudit
		lambda:ListFunctions		Verified	arn:aws:iam::aws:policy/SecurityAudit
		lambda:GetFunction		Verified	PrismaCloud-ReadOnly-Policy-Compute
		lambda:ListAliases		Verified	arn:aws:iam::aws:policy/SecurityAudit
		lambda:ListEventSourceMappings		Verified	arn:aws:iam::aws:policy/SecurityAudit

Feature	Protection Mode	Permissions	Condition	Prisma Cloud Templates Status	Role/Policy
		lambda:GetPolicy		Verified	arn:aws:iam::aws:policy/ SecurityAudit
		kms:Decrypt		Verified	PrismaCloud-ReadOnly-Policy-Compute
		logs:DescribeSubscriptionFilters		Verified	arn:aws:iam::aws:policy/ SecurityAudit
		s3:GetBucketNotification		Verified	arn:aws:iam::aws:policy/ SecurityAudit
		elasticloadbalancing:DescribeListeners		Verified	arn:aws:iam::aws:policy/ SecurityAudit
		elasticloadbalancing:DescribeTargetGroups		Verified	arn:aws:iam::aws:policy/ SecurityAudit
		elasticloadbalancing:DescribeListenerCertificates		Verified	arn:aws:iam::aws:policy/ SecurityAudit
		elasticloadbalancing:DescribeRules		Verified	arn:aws:iam::aws:policy/ SecurityAudit
		cloudfront:ListDistributions		Verified	arn:aws:iam::aws:policy/ SecurityAudit
		events:ListRules		Verified	arn:aws:iam::aws:policy/ SecurityAudit
		apigateway:GET		Verified	arn:aws:iam::aws:policy/ SecurityAudit
VM Tags Discovery	Monitor AND Monitor & Protect	Update both read-only & read-write templates			
		ec2:DescribeTags		Verified	arn:aws:iam::aws:policy/ SecurityAudit

Feature	Protection Mode	Permissions	Condition	Prisma Cloud Templates Status	Role/Policy
VM Images Scan	Monitor & Protect ONLY	Update read-write templates ONLY			
		ec2:CreateSecurityGroup		Verified	PrismaCloud-Remediation-Compute-Policy-AMIScan
		ec2:DescribeSecurityGroups		Verified	arn:aws:iam::aws:policy/SecurityAudit
		ec2:RevokeSecurityGroupEgress		Verified	PrismaCloud-Remediation-Compute-Policy-AMIScan
		ec2:AuthorizeSecurityGroupIngress		Verified	PrismaCloud-Remediation-Compute-Policy-AMIScan
		ec2:DeleteSecurityGroup		Verified	PrismaCloud-Remediation-Compute-Policy-AMIScan
		ec2:RunInstances		Verified	PrismaCloud-Remediation-Compute-Policy-AMIScan
		ec2:DescribeInstances		Verified	arn:aws:iam::aws:policy/SecurityAudit
		ec2:TerminateInstances		Verified	PrismaCloud-Remediation-Compute-

Feature	Protection Mode	Permissions	Condition	Prisma Cloud Templates Status	Role/Policy
					Policy-AMIScan
		ec2:DescribeImages		Verified	arn:aws:iam::aws:policy/SecurityAudit
		ec2:CreateTags		Verified	PrismaCloud-Remediation-Compute-Policy-AMIScan
		ec2:AuthorizeSecurityGroupEgress		Verified	PrismaCloud-Remediation-Compute-Policy-AMIScan
		ec2:DescribeSubnets		Verified	arn:aws:iam::aws:policy/SecurityAudit
		ec2:DescribeVpcs		Verified	arn:aws:iam::aws:policy/SecurityAudit
		ec2:DescribeInstanceTypeOfferings		Verified	arn:aws:iam::aws:policy/SecurityAudit
Host Auto-Defend	Monitor & Protect ONLY	Update read-write templates ONLY			
		ec2:DescribeImages		Verified	arn:aws:iam::aws:policy/SecurityAudit
		ec2:DescribeInstances		Verified	arn:aws:iam::aws:policy/SecurityAudit
		ssm:SendCommand		Verified	PrismaCloud-Remediation-Compute-Policy-HostAutoDefend

Feature	Protection Mode	Permissions	Condition	Prisma Cloud Templates Status	Role/Policy
		ssm:DescribeInstanceInformation		Verified	arn:aws:iam::aws:policy/SecurityAudit
		ssm:ListCommandInvocations		Verified	PrismaCloud-Remediation-Compute-Policy-HostAutoDefend
		ssm:CancelCommand		Verified	PrismaCloud-Remediation-Compute-Policy-HostAutoDefend
		ssm:CreateAssociation		Verified	PrismaCloud-Remediation-Compute-Policy-HostAutoDefend
		ec2:DescribeRegions		Verified	arn:aws:iam::aws:policy/SecurityAudit
		ec2:DescribeTags		Verified	arn:aws:iam::aws:policy/SecurityAudit
Alert Provider	Monitor AND Monitor & Protect	Update both read-only & read-write templates			
		securityhub:BatchImportFindings		Verified	PrismaCloud-ReadOnly-Policy-Compute
Secrets Manager					
Agentless Scanning	Monitor & Protect	Update read-write templates ONLY			

Feature	Protection Mode	Permissions	Condition	Prisma Cloud Templates Status	Role/Policy
Scanning within the same Account (Individual Account Permissions)		ec2:CreateSnapshots		Verified	PrismaCloud-Remediation-Compute-Policy-AgentlessScanning
		ec2:DescribeSnapshots		Verified	arn:aws:iam::aws:policy/SecurityAudit
		ec2:DeleteSnapshot	{"StringEquals": {"ec2:ResourceTag/created-by": "prismacloud-agentless-scan"}	Verified	PrismaCloud-Remediation-Compute-Policy-AgentlessScanning
		ec2:TerminateInstances	{"StringEquals": {"ec2:ResourceTag/created-by": "prismacloud-agentless-scan"}	Verified	PrismaCloud-Remediation-Compute-Policy-AgentlessScanning
		ec2:CreateTags	{"StringEquals": {"ec2:ResourceTag/created-by": "prismacloud-agentless-scan"}	Verified	PrismaCloud-Remediation-Compute-Policy-AgentlessScanning
		ec2:DescribeSubnets		Verified	arn:aws:iam::aws:policy/SecurityAudit
		ec2:DescribeSecurityGroups		Verified	arn:aws:iam::aws:policy/SecurityAudit
		ec2:DescribeVolumes		Verified	arn:aws:iam::aws:policy/SecurityAudit
		ec2:DescribeInstances		Verified	arn:aws:iam::aws:policy/SecurityAudit

Feature	Protection Mode	Permissions	Condition	Prisma Cloud Templates Status	Role/Policy
		ec2:RunInstances		Verified	PrismaCloud-Remediation-Compute-Policy-AgentlessScanning
		ec2:DescribeInstanceStatus		Verified	arn:aws:iam::aws:policy/SecurityAudit
		ssm:GetParameters		Verified	PrismaCloud-Remediation-Compute-Policy-AgentlessScanning
		sts:DecodeAuthorizationMessage		Verified	PrismaCloud-Remediation-Compute-Policy-AgentlessScanning
		sts:GetCallerIdentity		Verified	PrismaCloud-Remediation-Compute-Policy-AgentlessScanning
		kms:Decrypt		Verified	PrismaCloud-Remediation-Compute-Policy-AgentlessScanning
		kms:GenerateDataKeyWithoutPlaintext		Verified	PrismaCloud-Remediation-Compute-Policy-AgentlessScanning
		kms:ReEncryptFrom		Verified	PrismaCloud-Remediation-Compute-Policy-AgentlessScanning

Feature	Protection Mode	Permissions	Condition	Prisma Cloud Templates Status	Role/Policy
		kms:Encrypt		Verified	PrismaCloud-Remediation-Compute-Policy-AgentlessScanning
		kms:ReEncryptTo		Verified	PrismaCloud-Remediation-Compute-Policy-AgentlessScanning
		kms:DescribeKey		Verified	arn:aws:iam::aws:policy/SecurityAudit
		kms:CreateGrant		Verified	PrismaCloud-Remediation-Compute-Policy-AgentlessScanning
		iam:SimulatePrincipalPolicy		Verified	arn:aws:iam::aws:policy/SecurityAudit
Scanning within a dedicated account (Hub setup)					
Permissions for Account being scanned by the Hub Account		ec2:CreateSnapshots		Verified	PrismaCloud-Remediation-Compute-Policy-AgentlessScanning
		ec2:DeleteSnapshot	{"StringEquals": {"ec2:ResourceTag/created-by": "prismacloud-agentless-scan"}	Verified	PrismaCloud-Remediation-Compute-Policy-AgentlessScanning

Feature	Protection Mode	Permissions	Condition	Prisma Cloud Templates Status	Role/Policy
		ec2:ModifySnapshotAttribute	{"StringEquals": {"ec2:ResourceTag/created-by": "prismacloud-agentless-scan"}	Verified	PrismaCloud-Remediation-Compute-Policy-AgentlessScanning
		ec2:CreateTags	{"StringEquals": {"ec2:ResourceTag/created-by": "prismacloud-agentless-scan"}	Verified	PrismaCloud-Remediation-Compute-Policy-AgentlessScanning
		ec2:DescribeVolumes		Verified	arn:aws:iam::aws:policy/SecurityAudit
		ec2:DescribeInstances		Verified	arn:aws:iam::aws:policy/SecurityAudit
		ec2:DescribeSnapshots		Verified	arn:aws:iam::aws:policy/SecurityAudit
		kms:Decrypt		Verified	PrismaCloud-Remediation-Compute-Policy-AgentlessScanning
		kms:GenerateDataKeyWithoutPlaintext		Verified	PrismaCloud-Remediation-Compute-Policy-AgentlessScanning
		kms:ReEncryptFrom		Verified	PrismaCloud-Remediation-Compute-Policy-AgentlessScanning
		kms:Encrypt		Verified	PrismaCloud-Remediation-Compute-

Feature	Protection Mode	Permissions	Condition	Prisma Cloud Templates Status	Role/Policy
					Policy-AgentlessScanning
		kms:ReEncryptTo		Verified	PrismaCloud-Remediation-Compute-Policy-AgentlessScanning
		kms:DescribeKey		Verified	arn:aws:iam::aws:policy/SecurityAudit
		kms:CreateGrant		Verified	PrismaCloud-Remediation-Compute-Policy-AgentlessScanning
		iam:SimulatePrincipalPolicy		Verified	arn:aws:iam::aws:policy/SecurityAudit
		sts:GetCallerIdentity		Verified	PrismaCloud-Remediation-Compute-Policy-AgentlessScanning
		sts:DecodeAuthorizationMessage		Verified	PrismaCloud-Remediation-Compute-Policy-AgentlessScanning
Scan Hub Account Permissions		ec2:TerminateInstances	{"StringEquals": {"ec2:ResourceTag/created-by": "prismacloud-agentless-scan"}		PrismaCloud-Remediation-Compute-Policy-AgentlessScanning
		ec2:CreateTags	{"StringEquals": {"ec2:ResourceTag/created-by": "prismacloud-	Verified	PrismaCloud-Remediation-Compute-Policy-AgentlessScanning

Feature	Protection Mode	Permissions	Condition	Prisma Cloud Templates Status	Role/Policy
			agentless-scan"}		
		ssm:GetParameters		Verified	PrismaCloud-Remediation-Compute-Policy-AgentlessScanning
		ec2:DescribeSubnets		Verified	arn:aws:iam::aws:policy/SecurityAudit
		ec2:DescribeSecurityGroups		Verified	arn:aws:iam::aws:policy/SecurityAudit
		ec2:DescribeInstances		Verified	arn:aws:iam::aws:policy/SecurityAudit
		ec2:RunInstances		Verified	PrismaCloud-Remediation-Compute-Policy-AgentlessScanning
		ec2:DescribeInstanceStatus		Verified	arn:aws:iam::aws:policy/SecurityAudit
		kms:Decrypt		Verified	PrismaCloud-Remediation-Compute-Policy-AgentlessScanning
		kms:GenerateDataKeyWithoutPlaintext		Verified	PrismaCloud-Remediation-Compute-Policy-AgentlessScanning
		kms:ReEncryptFrom		Verified	PrismaCloud-Remediation-Compute-Policy-AgentlessScanning

Feature	Protection Mode	Permissions	Condition	Prisma Cloud Templates Status	Role/Policy
		kms:Encrypt		Verified	PrismaCloud-Remediation-Compute-Policy-AgentlessScanning
		kms:ReEncryptTo		Verified	PrismaCloud-Remediation-Compute-Policy-AgentlessScanning
		kms:DescribeKey		Verified	arn:aws:iam::aws:policy/SecurityAudit
		kms:CreateGrant		Verified	PrismaCloud-Remediation-Compute-Policy-AgentlessScanning
		ssm:GetParameters		Verified	PrismaCloud-Remediation-Compute-Policy-AgentlessScanning
		sts:DecodeAuthorizationMessage		Verified	PrismaCloud-Remediation-Compute-Policy-AgentlessScanning
		sts:GetCallerIdentity		Verified	PrismaCloud-Remediation-Compute-Policy-AgentlessScanning
		iam:SimulatePrincipalPolicy		Verified	arn:aws:iam::aws:policy/SecurityAudit

GCP

Feature	Protection Mode	Permissions	Prisma Cloud Templates Status	Role/Policy
Host Auto Defend	Monitor & Protect ONLY	Update read-write templates ONLY		
		osconfig.patchJobs.exec	Verified	Prisma Cloud Viewer
		osconfig.patchJobs.list	Verified	Prisma Cloud Viewer
		osconfig.patchJobs.get	Verified	Prisma Cloud Viewer
		storage.buckets.create	Verified	Prisma Cloud Viewer
		storage.buckets.delete	Verified	Prisma Cloud Viewer
		storage.objects.create	Verified	Prisma Cloud Viewer
		storage.objects.delete	Verified	Prisma Cloud Viewer
		storage.objects.get	Verified	Prisma Cloud Viewer
		storage.objects.list	Verified	Prisma Cloud Viewer
		compute.disks.get	Verified	Prisma Cloud Viewer
		compute.instances.list	Verified	Prisma Cloud Viewer
		compute.zones.list	Verified	Prisma Cloud Viewer
		compute.projects.get	Verified	Prisma Cloud Viewer

Feature	Protection Mode	Permissions	Prisma Cloud Templates Status	Role/Policy
GCR Scan	Monitor AND Monitor & Protect	Update both read-only & read-write templates		
		artifactregistry.repositories.list	Verified	roles/viewer
		artifactregistry.repositories.get	Verified	roles/viewer
		artifactregistry.repositories.downloadArtifacts	Verified	roles/viewer
		artifactregistry.files.list	Verified	roles/viewer
		artifactregistry.files.get	Verified	roles/viewer
		artifactregistry.packages.list	Verified	roles/viewer
		artifactregistry.packages.listTagBindings	Verified	roles/viewer
		artifactregistry.repositories.listEffectiveTags	Verified	roles/viewer
		artifactregistry.packages.list	Verified	roles/viewer
		artifactregistry.tags.list	Verified	roles/viewer
		artifactregistry.tags.get	Verified	roles/viewer
		artifactregistry.versions.list	Verified	roles/viewer
		artifactregistry.versions.get	Verified	roles/viewer
Cloud Discovery	Monitor AND Monitor & Protect	Update both read-only & read-write templates		
		roles/storage.objectViewer	Verified	roles/viewer
		roles/container.clusterViewer	Verified	roles/viewer
		roles/cloudfunctions.viewer	Verified	roles/viewer

Feature	Protection Mode	Permissions	Prisma Cloud Templates Status	Role/Policy
		compute.instances.list	Verified	Prisma Cloud Viewer
		compute.zones.list	Verified	Prisma Cloud Viewer
		compute.projects.get	Verified	Prisma Cloud Viewer
VM Images Scan	Monitor & Protect ONLY	Update read-write templates ONLY		
		compute.disks.create	Verified	Prisma Cloud Viewer
		compute.images.get	Verified	Prisma Cloud Viewer
		compute.images.list	Verified	Prisma Cloud Viewer
		compute.images.useReadOnly	Verified	Prisma Cloud Viewer
		compute.instances.create	Verified	Prisma Cloud Viewer
		compute.instances.delete	Verified	Prisma Cloud Viewer
		compute.instances.get	Verified	Prisma Cloud Viewer
		compute.instances.list	Verified	Prisma Cloud Viewer
		compute.instances.setMetadata	Verified	Prisma Cloud Viewer
		compute.instances.setTags	Verified	Prisma Cloud Viewer
		compute.networks.get	Verified	Prisma Cloud Viewer

Feature	Protection Mode	Permissions	Prisma Cloud Templates Status	Role/Policy
		compute.networks.updatePolicy	Verified	Prisma Cloud Viewer
		compute.networks.use	Verified	Prisma Cloud Viewer
		compute.networks.useExternalIp	Verified	Prisma Cloud Viewer
		compute.subnetworks.use	Verified	Prisma Cloud Viewer
		compute.subnetworks.useExternalIp	Verified	Prisma Cloud Viewer
Serverless Scanning	Monitor AND Monitor & Protect	Update both read-only & read-write templates		
		cloudfunctions.functions.sourceCodeGet	Pending - to be added to Prisma templates	
		cloudfunctions.functions.get	Verified	roles/viewer
		cloudfunctions.functions.list	Verified	roles/viewer
		cloudfunctions.locations.get	Verified	roles/viewer
		cloudfunctions.locations.list	Verified	roles/viewer
		cloudfunctions.operations.get	Verified	roles/viewer
		cloudfunctions.operations.list	Verified	roles/viewer
		cloudfunctions.runtimes.list	Verified	roles/viewer
Agentless Scanning	Monitor & Protect	Update read-write templates ONLY		
Scanning within the same project				

Feature	Protection Mode	Permissions	Prisma Cloud Templates Status	Role/Policy
Individual project Permissions		compute.disks.create	Verified	Prisma Cloud Viewer
		compute.instances.create	Verified	Prisma Cloud Viewer
		compute.instances.delete	Verified	Prisma Cloud Viewer
		compute.instances.get	Verified	roles/viewer
		compute.instances.setLabels	Verified	Prisma Cloud Viewer
		compute.instances.setMetadata	Verified	Prisma Cloud Viewer
		compute.networks.use	Verified	Prisma Cloud Viewer
		compute.networks.useExternalIp	Verified	Prisma Cloud Viewer
		compute.subnetworks.useExternalIp	Verified	Prisma Cloud Viewer
		compute.snapshots.create	Verified	Prisma Cloud Viewer
		compute.snapshots.delete	Verified	Prisma Cloud Viewer
		compute.snapshots.list	Verified	roles/viewer
		compute.snapshots.setLabels	Verified	Prisma Cloud Viewer
		compute.snapshots.useReadOnly	Verified	roles/viewer
		compute.subnetworks.use	Verified	Prisma Cloud Viewer
		compute.disks.get	Verified	roles/viewer
		compute.projects.get	Verified	roles/viewer

Feature	Protection Mode	Permissions	Prisma Cloud Templates Status	Role/Policy
		compute.instances.list	Verified	roles/viewer
		compute.zones.list	Verified	roles/viewer
		compute.disks.createSnapshot		
Project being Scanned by the Hub Account				
		compute.disks.get	Verified	roles/viewer
		compute.projects.get	Verified	roles/viewer
		compute.instances.list	Verified	roles/viewer
		compute.zones.list	Verified	roles/viewer
		compute.disks.createSnapshot	Pending - to be added to Prisma templates	
Hub project target project access		compute.disks.createSnapshot	Pending - to be added to Prisma templates	
Since snapshots are created in the hub account in GCP, we need to give permission for the hub service account in the target account This template will need to be applied in the target account				
Hub Project		compute.disks.create	Verified	Prisma Cloud Viewer

Feature	Protection Mode	Permissions	Prisma Cloud Templates Status	Role/Policy
		compute.instances.create	Verified	Prisma Cloud Viewer
		compute.instances.delete	Verified	Prisma Cloud Viewer
		compute.instances.get	Verified	roles/viewer
		compute.instances.setLabels	Verified	Prisma Cloud Viewer
		compute.instances.setMetadata	Verified	Prisma Cloud Viewer
		compute.networks.use	Verified	Prisma Cloud Viewer
		compute.networks.useExternalIp	Verified	Prisma Cloud Viewer
		compute.subnetworks.useExternalIp	Verified	Prisma Cloud Viewer
		compute.snapshots.create	Verified	Prisma Cloud Viewer
		compute.snapshots.delete	Verified	Prisma Cloud Viewer
		compute.snapshots.list	Verified	roles/viewer
		compute.snapshots.setLabels	Verified	Prisma Cloud Viewer
		compute.snapshots.useReadOnly	Verified	roles/viewer
		compute.subnetworks.use	Verified	Prisma Cloud Viewer
		compute.instances.list	Verified	roles/viewer
		compute.zones.list	Verified	roles/viewer

Azure

Feature	Protection Mode	Permissions	Prisma Cloud Templates Status	Role/Policy
Host Auto Defend	Monitor & Protect ONLY			
		Microsoft.Compute/ virtualMachines/ runCommand/ action	Verified	Prisma Cloud custom role
		Microsoft.Compute/ locations/ operations/read	Verified	Reader
		Microsoft.Resources/ subscriptions/ locations/read	Verified	Reader
Serverless Scanning	Monitor AND Monitor & Protect			
		Microsoft.Web/ sites/Read	Verified	Reader
		Microsoft.Web/ sites/config/list/ Action	Verified	Prisma Cloud custom role
		Microsoft.web/ sites/functions/ action	Pending - to be added to Prisma templates	
		Microsoft.web/ sites/functions/ read	Verified	Reader
		Microsoft.Web/ sites/ publishxml/ Action	Verified	Prisma Cloud custom role
Cloud Discovery	Monitor AND Monitor & Protect			

Feature	Protection Mode	Permissions	Prisma Cloud Templates Status	Role/Policy
		Microsoft.ContainerRegistry/registries/read	Verified	Reader
		Microsoft.ContainerRegistry/registries/metadata/read	Verified	Reader
		Microsoft.ContainerService/managedClusters/read	Verified	Reader
		Microsoft.Web/sites/Read	Verified	Reader
		Microsoft.ContainerInstance/containerGroups/read	Verified	Reader
		Microsoft.ContainerInstance/containerGroups/containers/exec/action	Pending - to be added to Prisma templates	
		Microsoft.Compute/virtualMachines/read	Verified	Reader
VM Images Scan	Monitor & Protect ONLY	Update read-write templates ONLY		
		Microsoft.Compute/locations/publishers/artifacttypes/offers/skus/versions/read	Verified	Reader
		Microsoft.Compute/images/read	Verified	Reader
		Microsoft.Compute/galleries/read	Verified	Reader

Feature	Protection Mode	Permissions	Prisma Cloud Templates Status	Role/Policy
		Microsoft.Compute/galleries/images/read	Verified	Reader
		Microsoft.Compute/galleries/images/versions/read	Verified	Reader
		Microsoft.Resources/subscriptions/resourceGroups/read	Verified	Reader
		Microsoft.Resources/subscriptions/resourceGroups/write	Verified	Prisma Cloud custom role
		Microsoft.Resources/subscriptions/resourceGroups/delete	Pending - to be added to Prisma templates	
		Microsoft.Network/networkSecurityGroups/read	Verified	Reader
		Microsoft.Network/networkSecurityGroups/write	Verified	Network Contributor
		Microsoft.Network/networkSecurityGroups/join/action	Verified	Network Contributor
		Microsoft.Network/networkSecurityGroups/delete	Verified	Network Contributor
		Microsoft.Network/networkInterfaces/read	Verified	Reader

Feature	Protection Mode	Permissions	Prisma Cloud Templates Status	Role/Policy
		Microsoft.Network/networkInterfaces/write	Verified	Network Contributor
		Microsoft.Network/networkInterfaces/join/action	Verified	Network Contributor
		Microsoft.Network/networkInterfaces/delete	Verified	Network Contributor
		Microsoft.Compute/disks/write	Verified	Prisma Cloud custom role
		Microsoft.Compute/disks/delete	Verified	Prisma Cloud custom role
		Microsoft.Network/virtualNetworks/subnets/read	Verified	Reader
		Microsoft.Network/virtualNetworks/subnets/join/action	Verified	Network Contributor
		Microsoft.Compute/virtualMachines/read	Verified	Reader
		Microsoft.Compute/virtualMachines/write	Verified	Prisma Cloud custom role
		Microsoft.Compute/virtualMachines/start/action	Pending - to be added to Prisma templates	
		Microsoft.Compute/virtualMachines/delete	Verified	Prisma Cloud custom role
		Microsoft.KeyVault/vaults/keys/read	Verified	Reader

Feature	Protection Mode	Permissions	Prisma Cloud Templates Status	Role/Policy
		Microsoft.KeyVault/vaults/keys/wrap/action	Pending - to be added to Prisma templates	
		Microsoft.KeyVault/vaults/keys/unwrap/action	Pending - to be added to Prisma templates	
Agentless Scanning	Monitor & Protect	Update read-write templates ONLY		
Scanning within the same Account (Individual Account Permissions)		Microsoft.Resources/subscriptions/resourceGroups/read	Verified	Prisma Cloud custom role
		Microsoft.Resources/subscriptions/resourceGroups/write	Verified	Prisma Cloud custom role
		Microsoft.Network/networkInterfaces/read	Verified	Network Contributor
		Microsoft.Network/networkInterfaces/write	Verified	Network Contributor
		Microsoft.Network/networkInterfaces/delete	Verified	Network Contributor
		Microsoft.Network/networkInterfaces/join/action	Verified	Network Contributor
		Microsoft.Network/networkSecurityGroups/read	Verified	Network Contributor

Feature	Protection Mode	Permissions	Prisma Cloud Templates Status	Role/Policy
		Microsoft.Network/networkSecurityGroups/write	Verified	Network Contributor
		Microsoft.Network/networkSecurityGroups/delete	Verified	Network Contributor
		Microsoft.Network/networkSecurityGroups/join/action	Verified	Network Contributor
		Microsoft.Network/virtualNetworks/read	Verified	Network Contributor
		Microsoft.Network/virtualNetworks/write	Verified	Network Contributor
		Microsoft.Network/virtualNetworks/delete	Verified	Network Contributor
		Microsoft.Network/virtualNetworks/subnets/read	Verified	Network Contributor
		Microsoft.Network/virtualNetworks/subnets/join/action	Verified	Network Contributor
		Microsoft.Compute/disks/read	Verified	Reader
		Microsoft.Compute/disks/write	Verified	Prisma Cloud custom role
		Microsoft.Compute/disks/delete	Verified	Prisma Cloud custom role
		Microsoft.Compute/disks/beginGetAccess/action	Verified	Prisma Cloud custom role

Feature	Protection Mode	Permissions	Prisma Cloud Templates Status	Role/Policy
		Microsoft.Compute/snapshots/read	Verified	Reader
		Microsoft.Compute/snapshots/write	Verified	Prisma Cloud custom role
		Microsoft.Compute/snapshots/delete	Verified	Prisma Cloud custom role
		Microsoft.Compute/virtualMachines/read	Verified	Reader
		Microsoft.Compute/virtualMachines/write	Verified	Prisma Cloud custom role
		Microsoft.Compute/virtualMachines/delete	Verified	Prisma Cloud custom role
		Microsoft.Compute/virtualMachines/instanceView/read	Verified	Reader

Authentication

Edit on GitHub

Prisma Cloud provides broad enterprise identity support. It can integrate with a number of identity providers, including Active Directory, OpenLDAP, Ping, Okta, Shibboleth, Azure AD, and Google G Suite, so you can implement single sign-on for the Prisma Cloud Console. Prisma Cloud supports simultaneous integration with multiple identity providers. For example, you might want to integrate with both Okta and GitHub to support users that login from both platforms.

Prisma Cloud ships with prebuilt roles to provide least privilege access to your DevOps and security teams. Use assigned collections to precisely control what data teams can view or use built-in multi-tenancy to securely isolate entire business units or geographies within the same Console.

Pluggable cryptography lets you bring your own certificates, not just for TLS, but also for smart card authentication to Console.

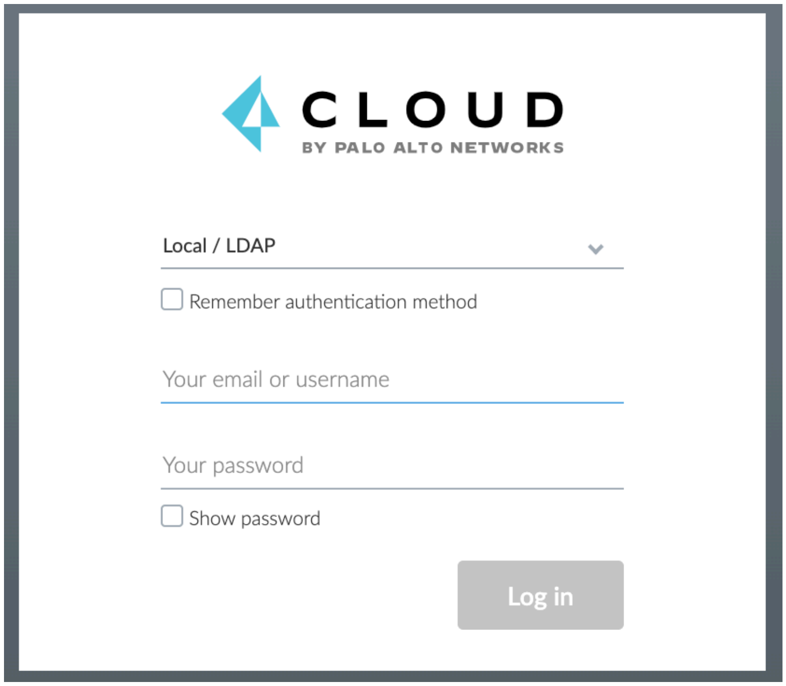
A credentials store providers a single secure location to store service accounts for integration with the various cloud providers. Define them once in the credentials store, and then reuse them throughout Console for your various integrations.

- Logging into Prisma Cloud
- Integrating with an IdP
- Integrate with Active Directory
- Integrate with OpenLDAP
- Integrate Prisma Cloud with Open ID Connect
- Integrate with Okta via SAML 2.0 federation
- Integrate Google G Suite via SAML 2.0 federation
- Integrate with Azure Active Directory via SAML 2.0 federation
- Integrate with PingFederate via SAML 2.0 federation
- Integrate with Windows Server 2016 & 2012r2 Active Directory Federation Services (ADFS) via SAML 2.0 federation
- Integrate Prisma Cloud with GitHub
- Integrate Prisma Cloud with OpenShift
- Non-default UPN suffixes
- Compute user roles
- Assign roles
- Credentials store
- Cloud accounts

Logging into Prisma Cloud

Edit on GitHub

Prisma Cloud Console supports multiple authentication methods. Check with your administrator to see how sign-in has been implemented for your organization, then choose the appropriate method from the drop-down list.



The options are:

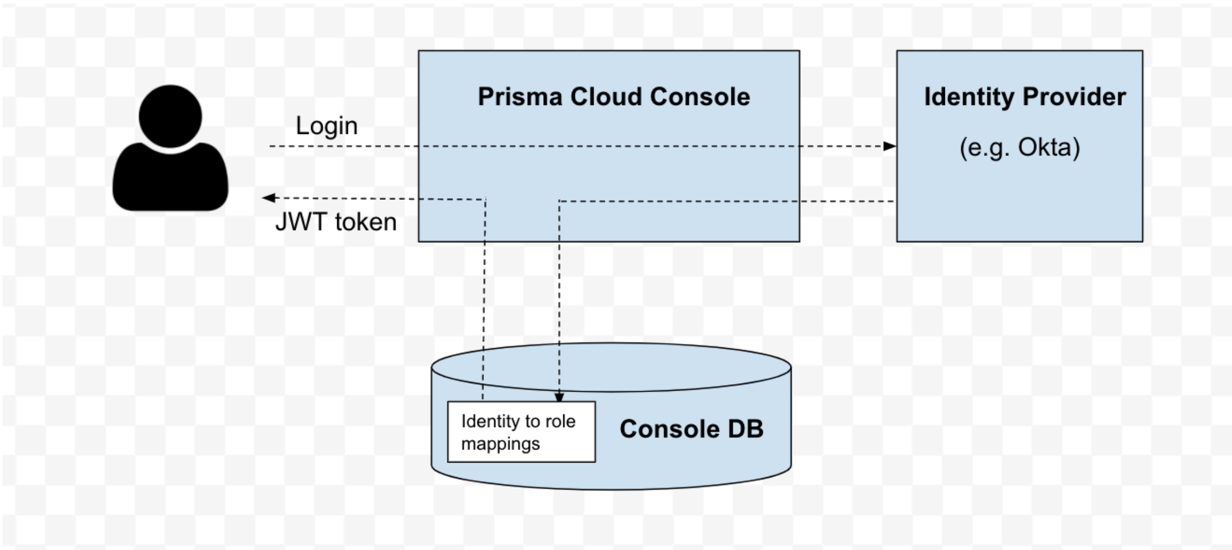
- **Local/ LDAP** — Users are evaluated against Console's database before the LDAP database. By default, initial admin users are created in Console's local database, so choose this option when you're logging in with your first user. If you integrate with a central identity provider, you can always delete the initial admin user, so that all users authenticate in compliance with your organization's policy (e.g., 2FA).

  If the same username exists in both databases, it's not possible to login with the LDAP user.
- **SAML** — Security Assertion Markup Language (SAML) is an open standard that enables single sign-on. Prisma Cloud supports all standard SAML 2.0 providers.
- **OAuth** — Prisma Cloud currently supports GitHub and OpenShift for OAuth login. For the OAuth login flow, Prisma Cloud gets permission from the user to query their information (username and email) from GitHub or OpenShift, and then checks the local database to determine if the user is authorized to access Prisma Cloud Console. If so, Prisma Cloud issues a token to the user to access Console.
- **OpenID Connect** — OpenID Connect is a simple identity layer on top of the OAuth 2.0 protocol. Prisma Cloud supports all standard OpenID Connect providers.

Login flow

If you integrate Prisma Cloud with an identity provider (IdP), the user's identity is verified by the IdP, and the role is mapped in Prisma Cloud Console.

If you don't want to integrate with an IdP, Prisma Cloud lets you create "local" users and groups, where the Console itself both authenticates and authorizes users.



Direct login URL

Direct login URLs are supported for SAML, OAuth and OIDC. When you use the direct login URL, the client doesn't need the extra step of selecting an auth provider from the Prisma Cloud login page.

Set type in the direct login URL:

```
https://<CONSOLE>:<PORT>/api/v1/authenticate/identity-redirect-url?
type=<oauth/oidc/saml>&redirect=true
```


Integrating with an IdP

Edit on GitHub

Prisma Cloud Compute can integrate with a number of identity providers (IdP).

Integrating with an IdP lets you reuse the identities and groups already defined in the IdP. From there, administrators can configure granular access control policies to the Prisma Cloud Compute UI and API, and users can access the system using their standard corporate credentials.

Integrate with Active Directory

Edit on GitHub

Prisma Cloud can integrate with Active Directory (AD), an enterprise identity directory service.

If your AD environment uses alternative UPN suffixes (also referred to as explicit UPNs), see Non-default UPN suffixes *to understand how to use them with Prisma Cloud.*

LDAP group names are case sensitive in Prisma Cloud.

With AD integration, you can reuse the identities and groups centrally defined in Active Directory, and extend your organization's access control policy to manage the data users can see and the things they can do in the Prisma Cloud Console.

For more information about Prisma Cloud's built-in roles, see User Roles.

Configuration options

The following configuration options are available:

Configuration option	Description
Enabled	Enables or disables integration with Active Directory. In Console, use the slider to enable (ON) or disable (OFF) integration with AD. By default, integration with AD is disabled.
URL	Specifies the path to your LDAP server, such as an Active Directory Domain Controller. The format for the LDAP server path is: <PROTOCOL>://<HOST>:<PORT> Where <PROTOCOL> can be ldap or ldaps. For an Active Directory Global Catalog server, use ldap. For performance and redundancy, use a load balanced path. Example: ldap://ldapserver.example.com:3268
Search Base	Specifies the search query base path for retrieving users from the directory. Example: dc=example,dc=com
User identifier	User name format when authenticating sAMAccountName = DOMAIN\sAMAccountName userPrincipalName = user@ad.example.com

Configuration option	Description
	The Active Directory domain name must be provided when using sAMAccountName due to domain trust behavior.
Account UPN	Console Account UPN Specifies the username for the Prisma Cloud service account that has been set up to query Active Directory. Specify the username with the User Principal Name (UPN) format: <USERNAME>@<DOMAIN> Example: twistlock_service@example.com
Account Password	Specifies the password for the Prisma Cloud service account.

Integrating Active Directory

Integrate Active Directory after you have installed Prisma Cloud.

STEP 1 | Open Console, then go to **Manage > Authentication > Identity Providers**.

STEP 2 | Set **Integrate LDAP users and groups with Prisma Cloud** to **Enabled**.

STEP 3 | Specify all the parameters for connecting to your Active Directory service.

1. For **Authentication** type, select **Active Directory**.
2. In **Path to LDAP service**, specify the path to your LDAP server.

   For example: *ldap://ldapserver.example.com:3268*
3. In **Search Base**, specify the base path to the subtree that contains your users.

   For example: *dc=example,dc=com*
4. In Service Account UPN and Service Account Password, specify the credentials for your service account.

   Specify the username in UPN format: <USERNAME>@<DOMAIN>

   For example, the account UPN format would be: *twistlock_service@example.com*
5. If you connect to Active Directory with ldaps, paste your CA certificate (PEM format) in the CA Certificate field.

   This enables Prisma Cloud to validate the LDAPS certificate to prevent spoofing and man- in-the-middle attacks. If this field is left blank, Prisma Cloud will not perform validation of the LDAPS certificate.

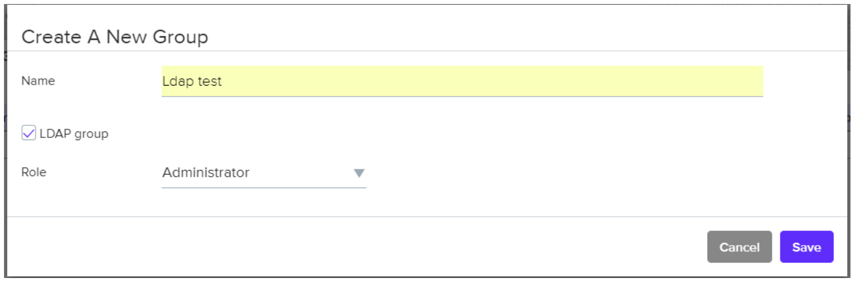
STEP 4 | Click **Save**.

Adding Active Directory group to Prisma Cloud

To grant authentication to users in an Active Directory group, add the AD group to Prisma Cloud.

STEP 1 | Navigate to **Manage > Authentication > Groups** and click **Add group**.

STEP 2 | In the dialog, enter AD group name and select **LDAP group**.



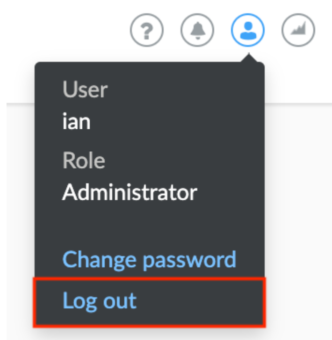
STEP 3 | Grant a role to members of the group.

Verifying integration with Active Directory

Verify the integration with AD.

STEP 1 | Open Console.

STEP 2 | If you are logged into Console, log out.



STEP 3 | At Console's login page, enter the UPN and password of an existing Active Directory user.

If the log in is successful, you are directed to the view appropriate for the user's role. If you have the Access User role, you are directed to a single page, where you can download certs for Docker client role-based access control.

Integrate with OpenLDAP

Edit on GitHub

Prisma Cloud can integrate with OpenLDAP, an open source implementation of the Lightweight Directory Access Protocol.

Integrating Prisma Cloud with OpenLDAP lets users access Prisma Cloud using their LDAP credentials, and lets admins define granular access control rules to Docker Engine or Kubernetes using existing LDAP identities.

With OpenLDAP integration, you can:

- Re-use the identities and groups already set up in your OpenLDAP directory.
- Extend your organization's access control logic to the management of Docker containers.

For example, you could specify that only members of the group Dev Ops Admins can start and stop containers in the production environment. For more information, see the article for setting up role-based access control for Docker Engine.

Integrating OpenLDAP

This procedure shows you how to integrate OpenLDAP with Prisma Cloud.

Prerequisites:

- You have installed OpenLDAP 2.4.44 or later. Prisma Cloud has been tested with version 2.4.44. Integration with older versions should work as well, but isn't officially supported.

STEP 1 | In your LDAP directory, create a service account that has admin privileges and that can run ldapsearch queries.

This admin account will be used by Prisma Cloud to authenticate users in your LDAP directory. It should be able to control the entire domain, and should therefore be created under the root OU.

STEP 2 | Verify that the service account can query your LDAP directory.

Run ldapsearch, passing it the credentials for your service account, and query your directory for a user:

```
$ ldapsearch -x \
  -b dc=example,dc=com \
  -D "cn=<SA-CN>,dc=example,dc=com" \
  -w <SA-PASS>
```



```
"(cn=<some-user-cn>)"
```

Where:

- ***<SA-CN>*** --

  Common name for the Prisma Cloud service account.
- ***<SA-PASS>*** --

  Password for the Prisma Cloud service account.
- ***<some-user-cn>*** --

  Common name for a user in your LDAP directory.

STEP 3 | Open Console, and go to **Manage > Authentication > Identity Providers > LDAP**.

STEP 4 | Set **Integrate LDAP users and groups with Prisma Cloud** to **Enabled**.

STEP 5 | For **Authentication type**, select **OpenLDAP**.

STEP 6 | For **Path to LDAP service**, enter the LDAP server and port number in the following format:

For secure connections over TLS: *ldaps://<server-dns>:<port-number>*.

For insecure connections: *ldap://<server-dns>:<port-number>*

STEP 7 | For **Search base**, enter the base DN for your users and groups.

STEP 8 | (OPTIONAL) For **User identifier**, specify an attribute to be used to match users.

For example, enter uid to match users based on their user IDs.

STEP 9 | For **Service account UPN**, enter the DN for your Prisma Cloud service account.

STEP 10 | For **Service account password**, enter the password for the Prisma Cloud service account.

STEP 11 | For **CA certificate**, provide the CA certificate used to sign the LDAPS certificate on the server.

Prisma Cloud uses the CA certificate to validate the LDAPS certificate and prevent man-in-the-middle attacks. If you are using an insecure connection or do not wish to validate the LDAPS certificate, leave this field blank.

STEP 12 | Click **Save**.

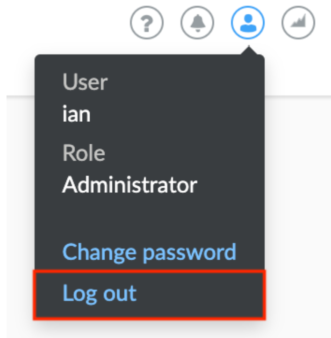
Console verifies all your parameters with the server. If a connection cannot be established, an error message is shown and no parameters are saved.

Verifying integration with OpenLDAP

Verify the integration with OpenLDAP.

STEP 1 | Open Console.

STEP 2 | If you are logged into Console, log out.



STEP 3 | Log in to Console using the credentials of an existing OpenLDAP user.

If the log in is successful, you are directed to the view appropriate for the user's role. If you have the Access User role, you are directed to a single page, where you can download certs for Docker client role-based access control.

Integrate Prisma Cloud with Open ID Connect

Edit on GitHub

OpenID Connect is a standard that extends OAuth 2.0 to add an identity layer. Prisma Cloud supports integration with any standard Open ID Connect (OIDC) provider that implements both OpenID connect core and OpenID connect discovery. Prisma Cloud supports the authorization code flow only.

This page includes instructions to integrate with the following providers:

- PingOne
- Okta
- Azure Active Directory

Use the **https://<CONSOLE>:<PORT>/api/v1/authenticate/callback/oidc** URL only to configure the integration between services. The API is not included in our reference guide because the URL is only enabled as a configuration value.

PingOne

Integrate with PingOne.

You need to configure Compute as an OIDC app. When configuring your app:

- The Start SSO URL must point to the **https://<CONSOLE>:<PORT>/callback** URL.
- The Redirect URI must point to the **https://<CONSOLE>:<PORT>/api/v1/authenticate/callback/oidc** URL.
- UserInfo must include *sub*, *idpid*, and *name*.
- All of the following scopes must be included for OpenID.
  - OpenID Connect (openid)
  - OpenID profile
  - OpenID Email
  - OpenID address
  - OpenID Phone
  - Groups

Update Ping callback URL

Update the callback URL.

STEP 1 | Log into the Ping web portal.

STEP 2 | Click **Applications**, and then click the **OIDC** tab.

STEP 3 | Click on the arrow button nest for your app.

STEP 4 | Click on the pencil icon on the right side.

STEP 5 | Click on **Authentication Flow**.

STEP 6 | In **REDIRECT URIS**, enter the following URL to enable the service-to-service integration:

https://<CONSOLE>:<PORT>/api/v1/authenticate/callback/oidc.

Create new user and join to group

STEP 1 | In the Ping web portal, click **Users**, and then click the **Users** tab.

STEP 2 | Click **Add users**, and choose the **Create New User** option.

STEP 3 | Fill the fields for **Password**, **Username** (should be your email), **First Name**, **Last Name**, and **Email**.

STEP 4 | In the **Membership** field, click **Add**, and choose a group.

STEP 5 | Click **Save**.

Okta

Integrate with Okta.

- Initiate Login URI (Okta) must point to **https://<CONSOLE>:<PORT>/callback**.
- Redirect URI must point to the **https://<CONSOLE>:<PORT>/api/v1/authenticate/callback/oidc** URL.
- UserInfo must include sub, idpid, name.
- Scopes:
  - All of the following scopes must be included for OpenID: OpenID Connect (openid), OpenID profile, OpenID Email, OpenID address, OpenID Phone, Groups.
  - All of the following scopes must be included for Okta: okta.groups.manage, okta.groups.read.

Update Okta callback URL

Update the callback URL.

STEP 1 | Log into Okta.

STEP 2 | Click on **Applications** and click on your application.

STEP 3 | Click the **General** tab, and then click **Edit**.

STEP 4 | Update **Login redirect URIs**. Enter the following URL to enable the service-to-service integration:

https://<CONSOLE>:<PORT>/api/v1/authenticate/callback/oidc

STEP 5 | Click **Save**.

Configure Okta as an Identity Provider

Configure Okta as an identity provider in Prisma Cloud with the following steps.

STEP 1 | Log into Prisma Cloud Console.

STEP 2 | Go to **Manage > Authentication > Identity Providers > OpenID Connect**.

STEP 3 | Enable OpenID Connect.

STEP 4 | Fill in the settings.

1. For **Client ID**, enter the client ID.
2. For **Client Secret**, enter the client secret.
3. For **Issuer URL**, enter:

   https://sso.connect.pingidentity.com/<CLIENT_ID>.
4. For **Group scope**, select **groups**.
5. (Optional) Enter your certificate.
6. Click **Save**.

Azure Active Directory (AD)

To integrate with Azure Active Directory (AD), you must register Prisma Cloud as an Open ID Connect (OIDC) application in Azure and configure Azure AD as an identity provider in Prisma Cloud.

STEP 1 | Go to your Azure console.

STEP 2 | Find the Azure AD service.

STEP 3 | Click the **app registration** button and select **New registration**

STEP 4 | Enter a name and select **Accounts in this organizational directory only** as the supported account type.

STEP 5 | Under **Redirect URI** select **Web console URL** enter the following URL to enable the service-to-service integration: **https://<CONSOLE>:<PORT>/api/v1/authenticate/callback/oidc**

STEP 6 | Click on **Register the app**.

STEP 7 | To add the secret for the client, go to **certificates & secrets**.

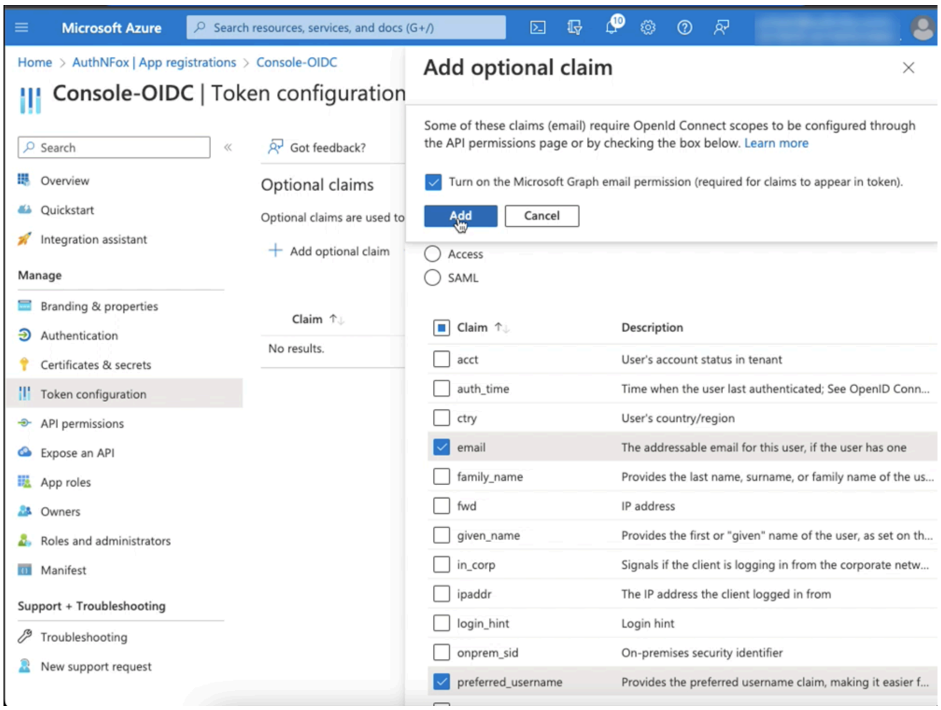
STEP 8 | Add a new secret for the client, copy and store it for later use.

You can only view the value of the secret when you create it. Copy and store the secret safely for later use.

Configure Groups in Azure AD

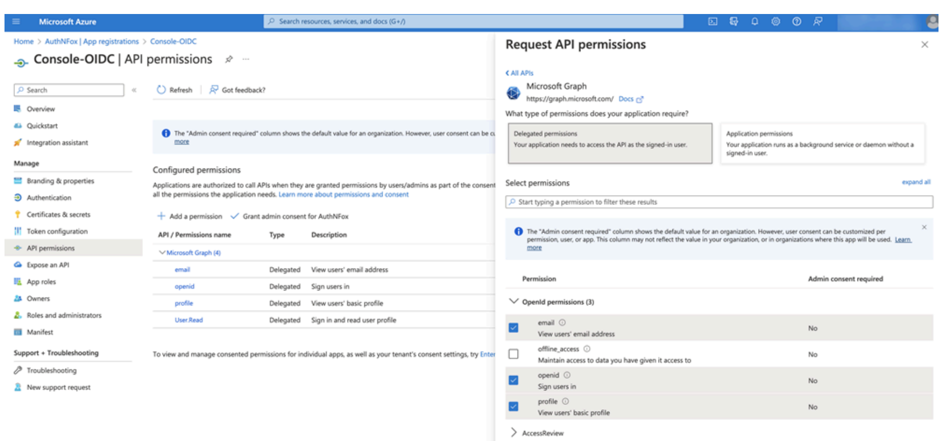
STEP 1 | To add the needed claim, go to **Token Configuration**.

1. Select **Add group claim**
2. Select the **Groups assigned to the application** option.
3. Keep the default values and click **Add**.
4. Click **Add optional claim** and select **Token type - ID**.
5. Select the **email** and **preferred_username** claims.
6. Turn on the Microsoft Graph email permission, while saving these claims.



STEP 2 | Go to the **API permissions** and click **Add a permission**.

1. Under **Microsoft API** select **Microsoft Graph**.
2. Select **Delegated permissions**
3. Select **email, openid, profile**.



STEP 3 | To create the needed application group, go to **Groups** in the Azure AD console.

STEP 4 | Create a new group and keep the default values.

Assign the Created Group to the Prisma Cloud Console

STEP 1 | Go to **Enterprise applications** in the Azure AD console.

STEP 2 | Find the application you registered.

STEP 3 | Click on **Properties** and check the **Assignment required** option.

STEP 4 | Click on **Assign users and groups**.

STEP 5 | Click add and select the previously created group.

STEP 6 | Click add and select your user.

STEP 7 | Go to **App registrations** in the Azure AD console.

STEP 8 | Click on **Your owned registered app**.

STEP 9 | Find the application you registered and click on **Endpoints**.

STEP 10 | Open the OpenID Connect metadata JSON file.

STEP 11 | Copy the value under Issuer URL from the JSON file, for example: **https://login.microsoftonline.com/<TENANT_ID>/v2.0**

Configure Azure AD as an Identity Provider

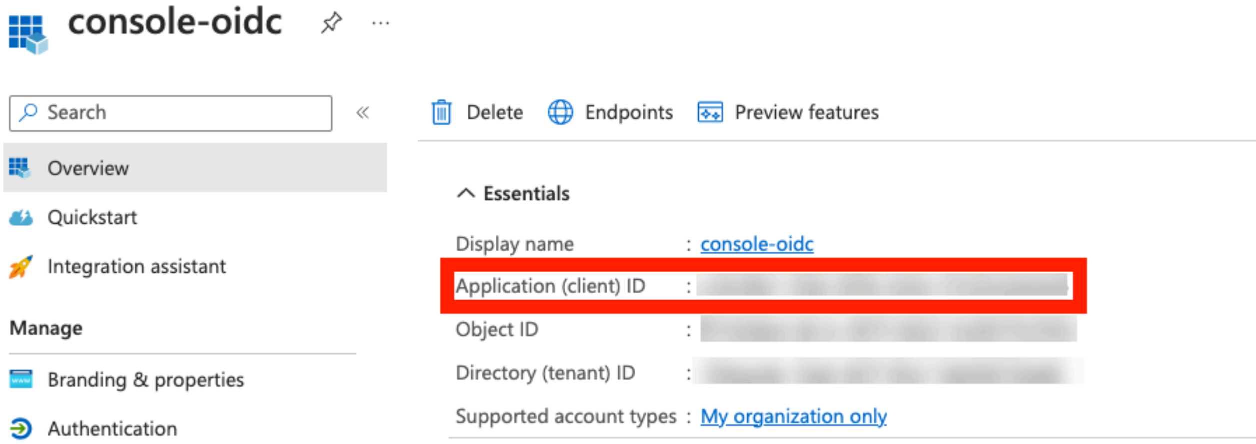
After you register Prisma Cloud as an Open ID Connect (OIDC) application in Azure, complete the following steps to configure Azure AD as an identity provider.

STEP 1 | Go to **Manage > Authentication > Identity Providers** in your Prisma Cloud Console.

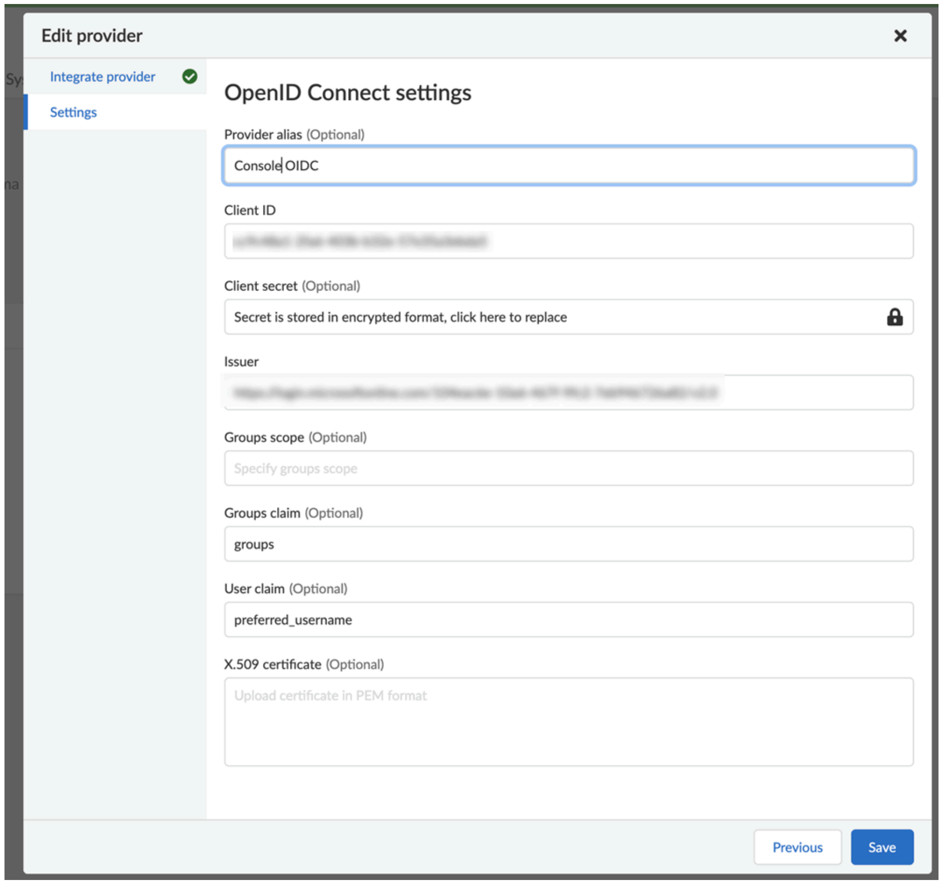
STEP 2 | Enable OpenID Connect.

STEP 3 | Enter the following information in the settings fields.

1. **Client ID**: Use the **Application (Client) ID** found in the Azure Console under **Azure AD > App registrations > Overview**.



2. **Client Secret**: The secret for the client that you created for the application and stored safely for later use.
3. **Issuer URL**: The endpoint of the application registered in Azure AD, for example **https://login.microsoftonline.com/<TENANT_ID>/v2.0**
4. **Group scope**: Leave this field blank.
5. **Group claim**: Set this field to *groups*. This allows Prisma Cloud to populate the specific group names automatically.
6. **User claim**: The optional claim for the user. Set this field to *preferred_username* for group based OIDC authentication, it is used for the audit logs.



STEP 4 | Click **Save**.

Prisma Cloud to OIDC user identity mapping

If you intend to use the group mapping method, skip to the Prisma Cloud to OIDC provider group mapping task. Create a user for every user that should access Prisma Cloud. The Open ID Connect specification requires every username to match with a configured username in the Prisma Cloud database. Prisma Cloud uses attributes that come from OIDC to perform this match, for example you can use *sub*, *username* or *email*. You should use whichever value the provider is configured to send to Prisma Cloud when you configure users.

STEP 1 | Go to **Manage > Authentication > Users**.

STEP 2 | Click **Add User**.

STEP 3 | Set **Username** to the GitHub user name.

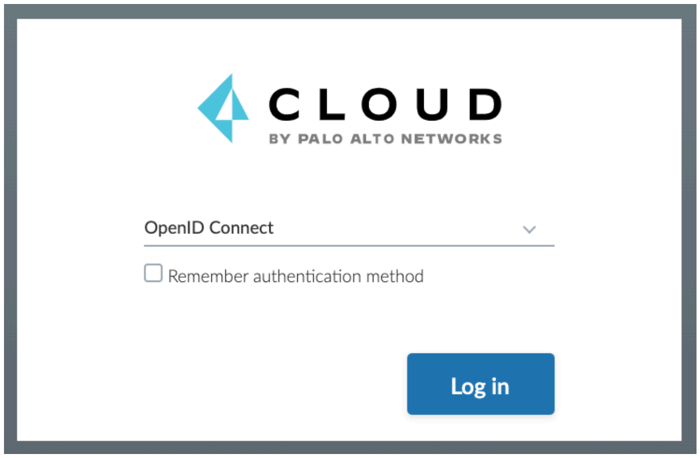
STEP 4 | Set **Auth method** to **OpenID Connect**.

STEP 5 | Select a role for the user.

STEP 6 | Click **Save**.

STEP 7 | Test logging into Prisma Cloud Console.

1. Logout of Prisma Cloud.
2. On the login page, select **OpenID Connect**, and then click **Login**.



3. You're redirected to your OIDC provider to authenticate.
4. After successfully authenticating, you're logged into Prisma Cloud Console.

Prisma Cloud to OIDC provider group mapping

When you use groups to assign roles in Prisma Cloud you don't have to create individual Prisma Cloud accounts for each user. The group value configured on the Compute side should reflect the name of the group scope in the OIDC provider. It might be something different than groups.

Groups can be associated and authenticated with by multiple identity providers. If you use Azure Active Directory (AAD), a user can't be part of more than 200 groups at once.

STEP 1 | Go to **Manage > Authentication > Groups**.

STEP 2 | Click **Add Group**.

STEP 3 | In **Name**, enter an OpenShift group name. For AAD use Azure Group's **Object ID** as the group name.

STEP 4 | In **Authentication method**, select **External Providers**.

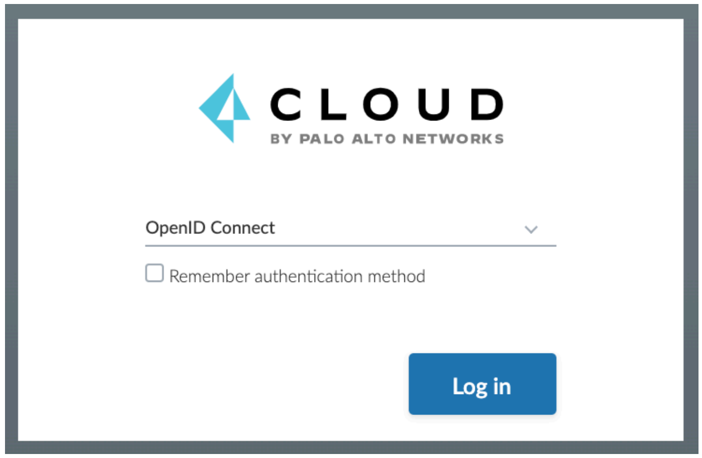
STEP 5 | In **Authentication Providers**, select **OpenID Connect group**.

STEP 6 | Select a role for the members of the group.

STEP 7 | Click **Save**.

STEP 8 | Test logging into Prisma Cloud Console.

1. Logout of Prisma Cloud.
2. On the login page, select **OpenID Connect**, and then click **Login**.



3. You're redirected to your OIDC provider to authenticate.
4. After successfully authenticating, you're logged into Prisma Cloud Console.

Integrate with Okta via SAML 2.0 federation

Edit on GitHub

Many organizations use SAML to authenticate users for web services. Prisma Cloud supports the SAML 2.0 federation protocol to access the Prisma Cloud Console. When SAML support is enabled, administrators can log into Console with their federated credentials. This article provides detailed steps for federating your Prisma Cloud Console with Okta.

The Prisma Cloud/Okta SAML federation flow works as follows:

1. Users browse to Prisma Cloud Console.
2. Their browsers are redirected to the Okta SAML 2.0 endpoint.
3. They enter their credentials to authenticate. Multi-factor authentication can be enforced at this step.
4. A SAML token is returned to Prisma Cloud Console.
5. Prisma Cloud Console validates the SAML token's signature and associates the user to their Prisma Cloud account via user identity mapping or group membership.

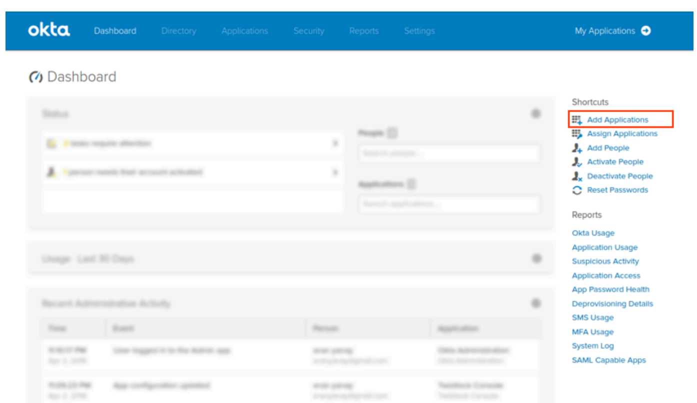
Integrating Prisma Cloud with SAML consists of setting up your IdP, then configuring Prisma Cloud to integrate with it.

Setting up Prisma Cloud in Okta

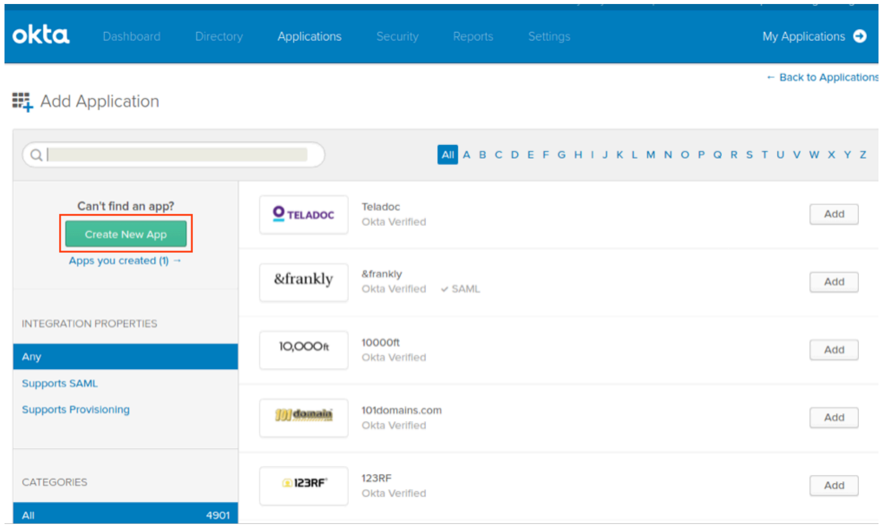
Set up Prisma Cloud in Okta.

STEP 1 | Log into the Okta admin dashboard.

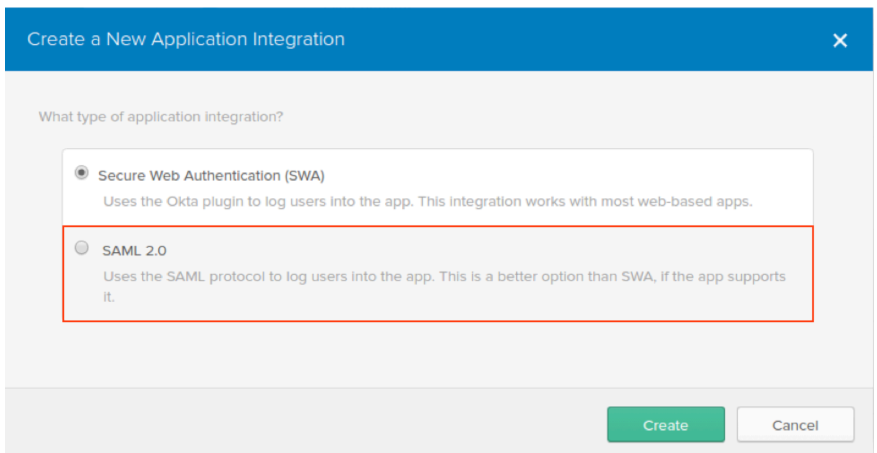
STEP 2 | On the right, click **Add Applications**.



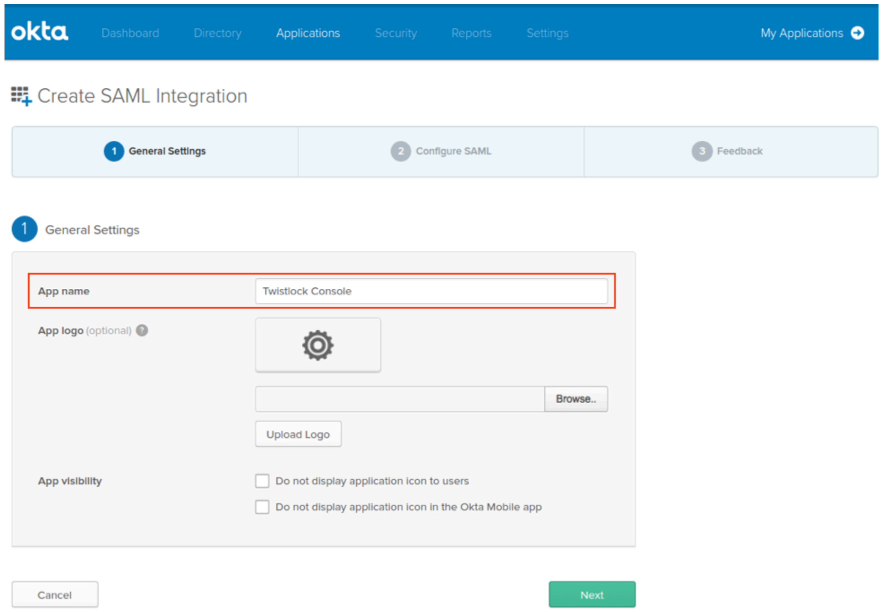
STEP 3 | On the left, click **Create new app**.



STEP 4 | Select **SAML 2.0**, and then click **Create**.



STEP 5 | In the **App name** field, enter **Prisma Cloud Console**, then click **Next**.



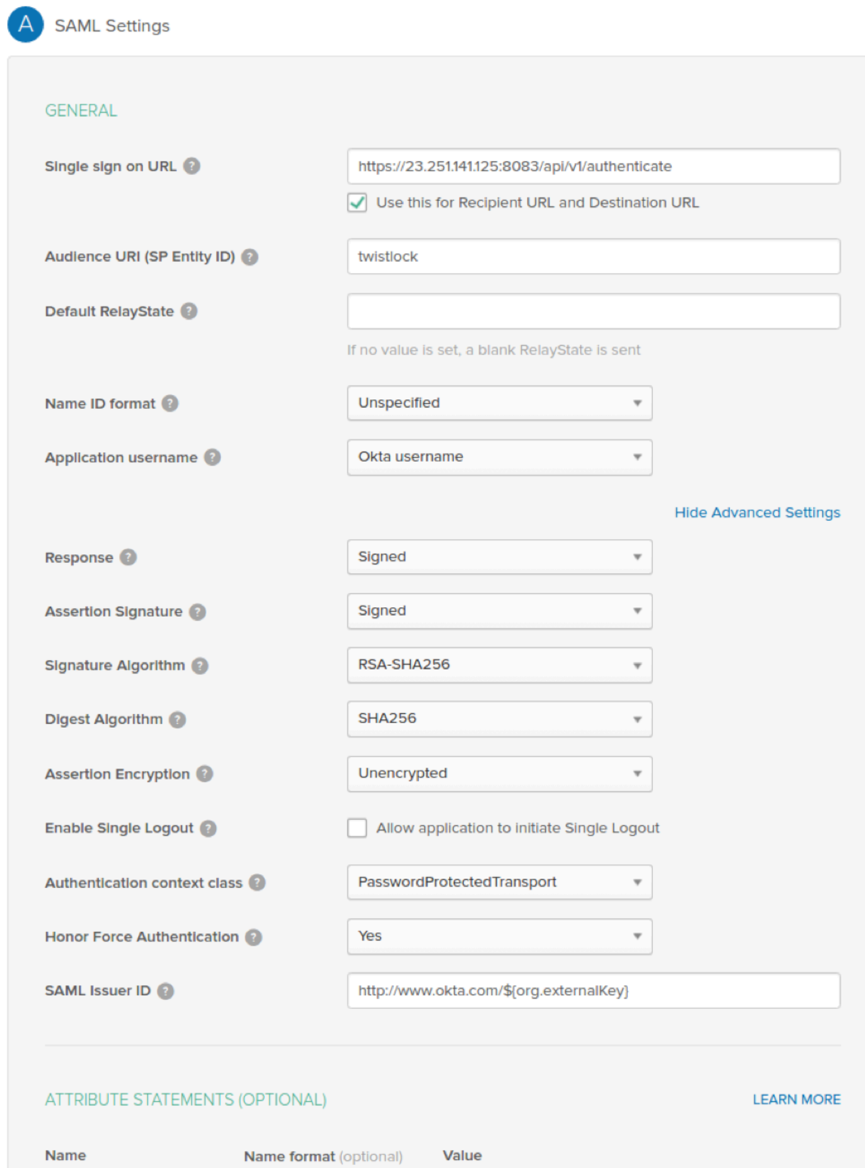
STEP 6 | In the SAML Settings dialog:

1. In the **Single Sign On URL** field, enter **https://<CONSOLE_ADDR>:8083/api/v1/authenticate.**

   Note that if you have changed the default port you use for the HTTPS listener, you'd need to adjust the URL here accordingly. Additionally, this URL must be visible from the

Okta environment, so if you're in a virtual network or behind a load balancer, it must be configured to forward traffic to this port and it's address is what should be used here.

2. Select **Use this for Recipient URL and Destination URL**.
3. In the field for **Audience Restriction**, enter **twistlock** (all lowercase).
4. Expand **Advanced Settings**.
5. Verify that **Response** is set to **Signed**.
6. Verify that **Assertion Signature** is set to **Signed**.

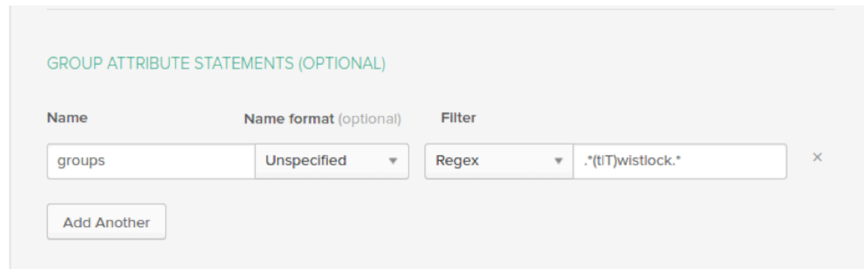


STEP 7 | (Optional) Add a group.

Setting up groups is optional. If you set up group attribute statements, then permission to access Prisma Cloud is assessed at the group level. If you don't set up group attribute statements, them permission to access Prisma Cloud is assessed at the user level.

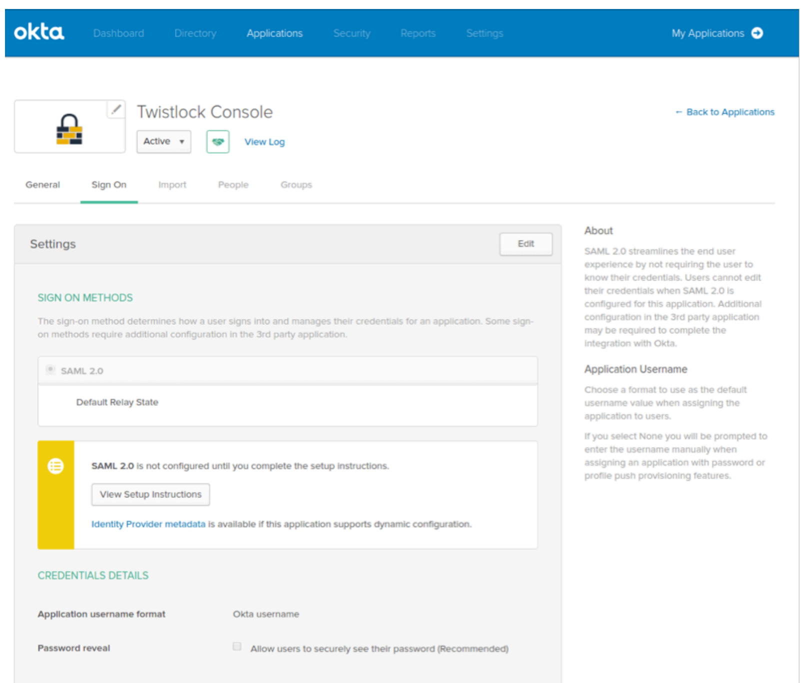
1. Scroll down to the **GROUP ATTRIBUTE STATEMENTS** section.
2. In the **Name** field, enter **groups**.
3. In filter drop down menu, select **Regex** and enter a regular expression that captures all the groups defined in Okta that you want to use for access control rules in Prisma Cloud.

   In this example, the regular expression **.*(t|T)wistlock.*** is used to include all groups prepended with either Prisma Cloud or twistlock. You should enter your own desired group name here. If you have just one group, such as YourGroup, then just enter **YourGroup**. Regular expressions are not required. If you have multiple groups, you can use a regular expressions, such as **(group1|group2|group3)**.

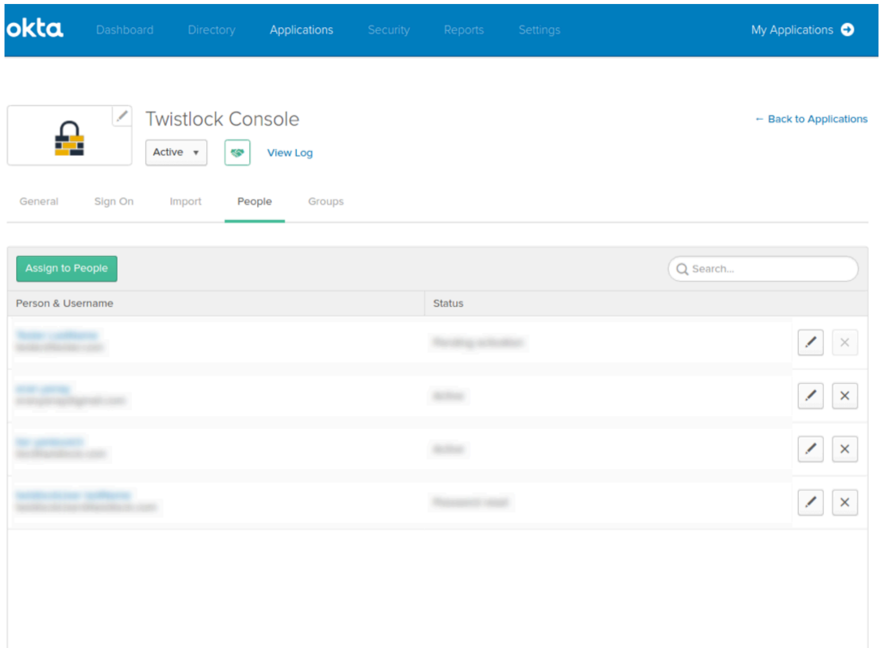


STEP 8 | Click **Next**, and then click **Finish**.

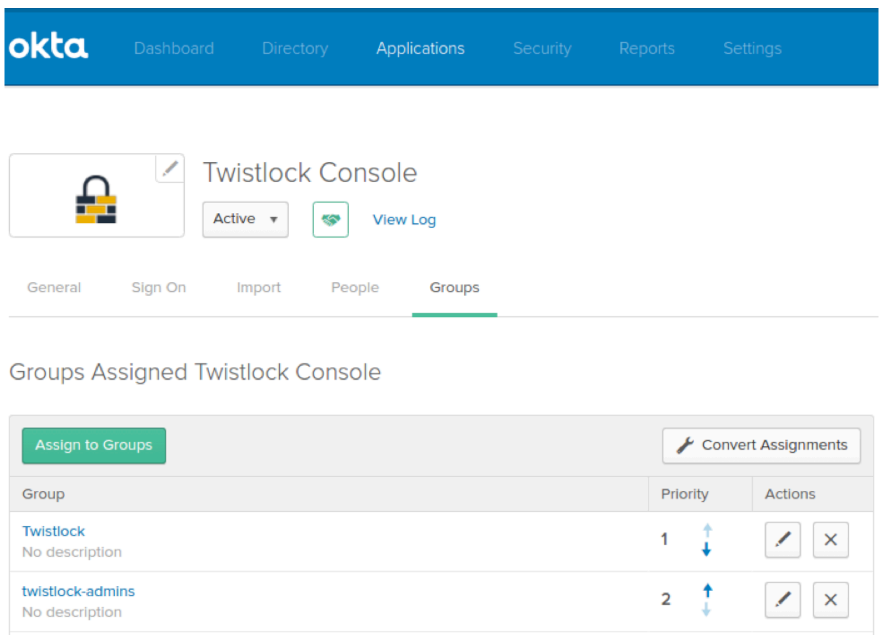
You are directed to a summary page for your new app.



STEP 9 | Click on the **People** tab, and add users to the Prisma Cloud app.



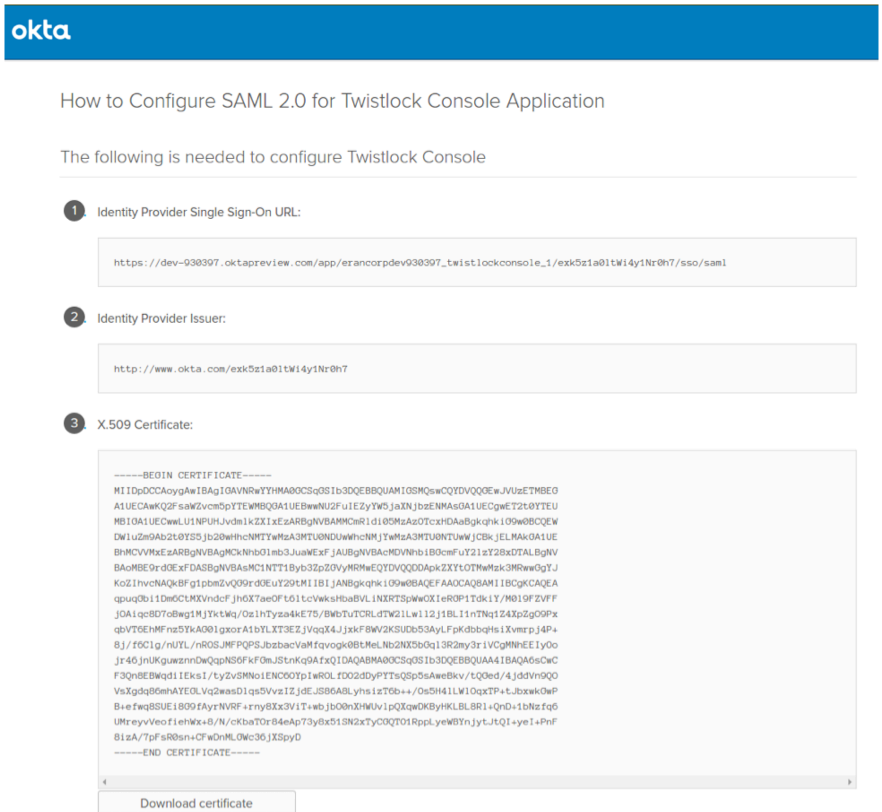
STEP 10 | Click on the **Groups** tab, and add groups to the Prisma Cloud app.



STEP 11 | Click on the **Sign On** tab and click **View setup instructions**.

The following values are used to configure Prisma Cloud Console, so copy them and set them aside.

- Identity Provider Single Sign-On URL
- Identity Provider Issuer
- X.509 Certificate



Configuring Console

Configure Prisma Cloud Console.

STEP 1 | Open Console, and login as admin.

STEP 2 | Go to **Manage > Authentication > Identity Providers > SAML**.

STEP 3 | Set **Integrate SAML users and groups with Prisma Cloud** to **Enabled**.

STEP 4 | Set **Identity provider** to **Okta**.

STEP 5 | Copy the following values from Okta and paste them into their corresponding fields in Console:

- Identity Provider Single Sign-On URL
- Identity Provider Issuer
- X.509 Certificate

STEP 6 | In **Audience**, enter **twistlock**.

STEP 7 | Click **Save**.

Granting access by group

Grant access to Prisma Cloud Console by group. Each group must be assigned a role. You can optionally use these groups to define RBAC rules for controlling who can run which Docker Engine commands in your environment.

STEP 1 | Open Console.

STEP 2 | Define a SAML group.

1. Go to **Manage > Authentication > Groups**.
2. Click **Add group**.
3. In the **Name** field, enter a group name.

   The group name must exactly match the group name in the SAML IdP. Console does not verify if that the value entered matches a group name in the SAML IdP.
4. Select the **SAML group** checkbox.
5. Select a role.
6. Select a project(s) - Optional.
7. Click **Save**.

Granting access by user

Grant access to Prisma Cloud Console by user. Each user must be assigned a role. You can optionally use these user to define RBAC rules for controlling who can run which Docker Engine commands in your environment.

STEP 1 | Open Console.

STEP 2 | Define a SAML user.

1. Go to **Manage > Authentication > Users**.
2. Click **Add user**.
3. In the **Username** field, enter a user name.

   The username must exactly match the username in the SAML IdP. Console does not verify if that the value entered matches a user name in the SAML IdP.
4. Select **SAML** as the Auth method
5. Select a role.
6. (Optional) Select a project(s).
7. Click **Save**.

Integrate Google G Suite via SAML 2.0 federation

Edit on GitHub

Many organizations use SAML to authenticate users for web services. Prisma Cloud supports the SAML 2.0 federation protocol to access the Prisma Cloud Console. When SAML support is enabled, users can log into Console with their federated credentials. This article provides detailed steps for federating your Prisma Cloud Console with Google G Suite.

The Prisma Cloud/G Suite SAML federation flow works as follows:

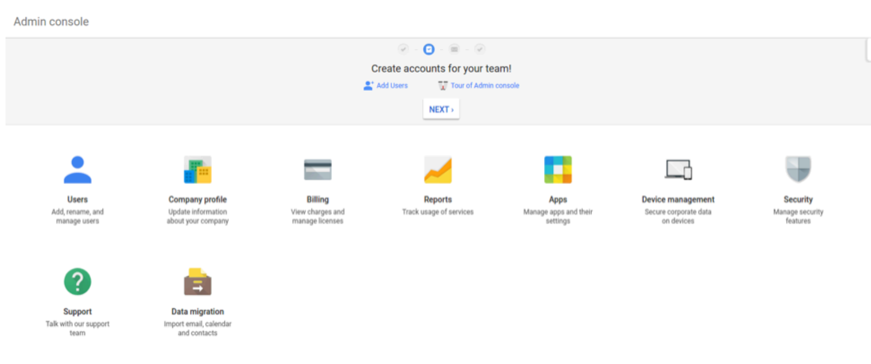
1. Users browse to Prisma Cloud Console.
2. Their browsers are redirected to the G Suite SAML 2.0 endpoint.
3. They enter their credentials to authenticate. Multi-factor authentication can be enforced at this step.
4. A SAML token is returned to Prisma Cloud Console.
5. Prisma Cloud Console validates the SAML token's signature and associates the user to their Prisma Cloud account via user identity mapping or group membership.

Setting up Google G Suite

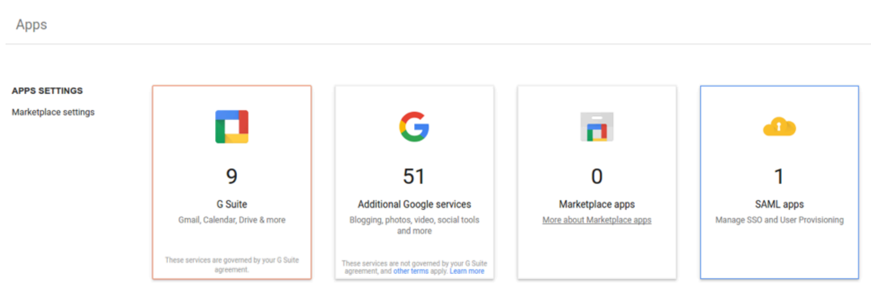
Prisma Cloud supports SAML integration with Google G Suite.

STEP 1 | Log into your G Suite admin console.

STEP 2 | Click on **Apps**.



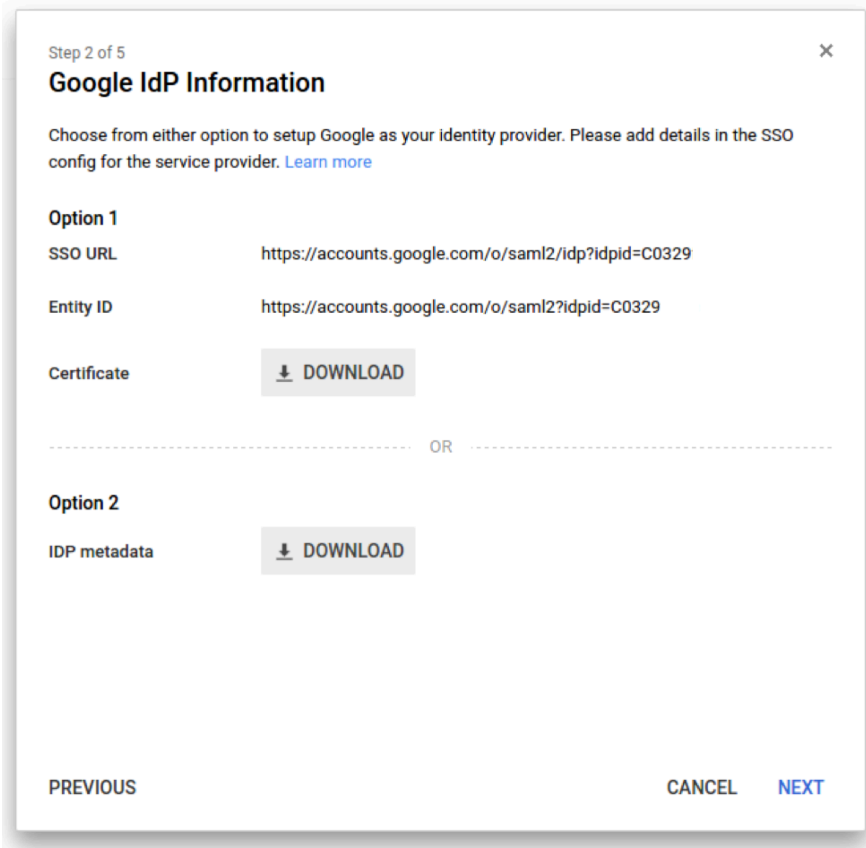
STEP 3 | Click on **SAML apps**.



STEP 4 | Click the **+** button at the bottom to add a new app.

STEP 5 | Click **SETUP MY OWN CUSTOM APP** at the bottom of the dialog.

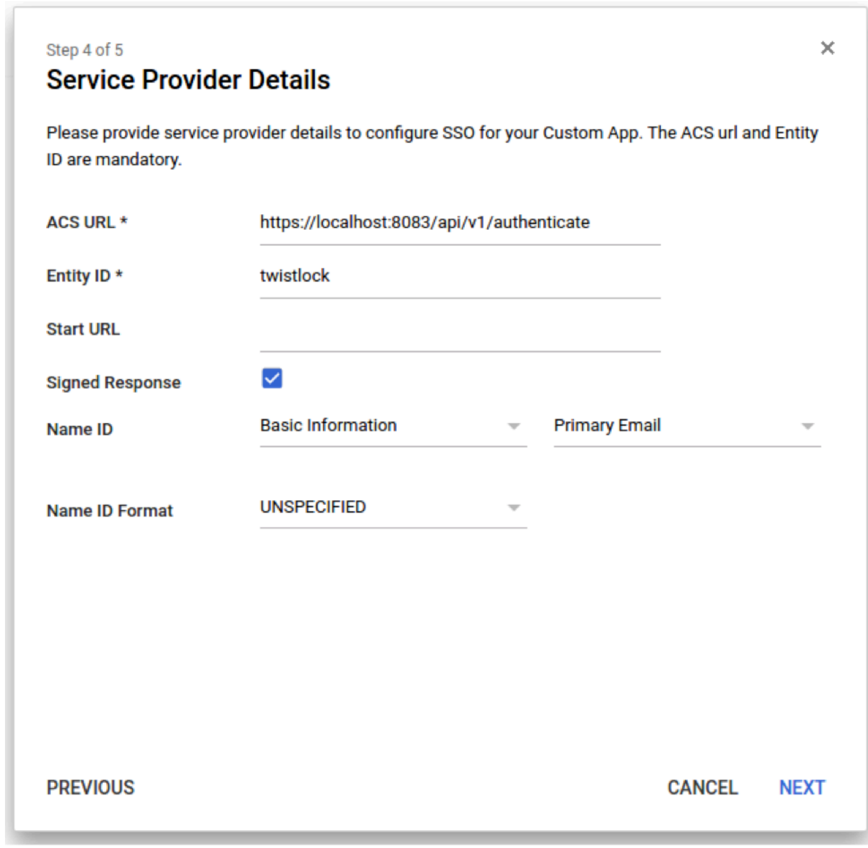
STEP 6 | Copy the **SSO URL** and **Entity ID**, and download the certificate. You will need these later for setting up the integration in Prisma Cloud Console. Click **NEXT**.



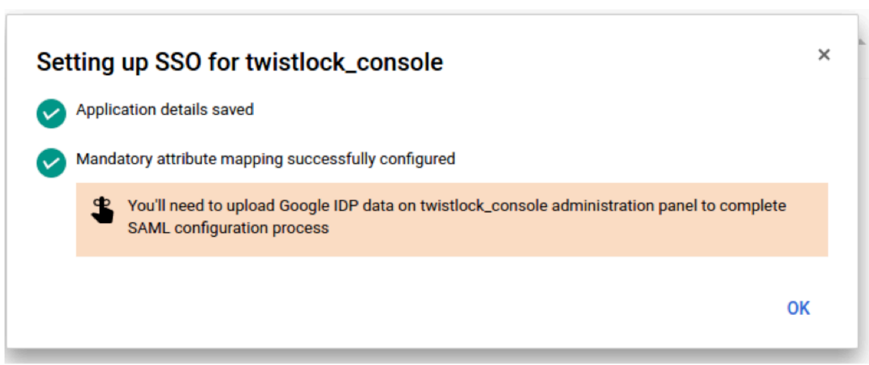
STEP 7 | Enter an **Application Name**, such as **Prisma Cloud**, then click **NEXT**.

STEP 8 | In the Service Provider Details dialog, enter the following details, then click **NEXT**.

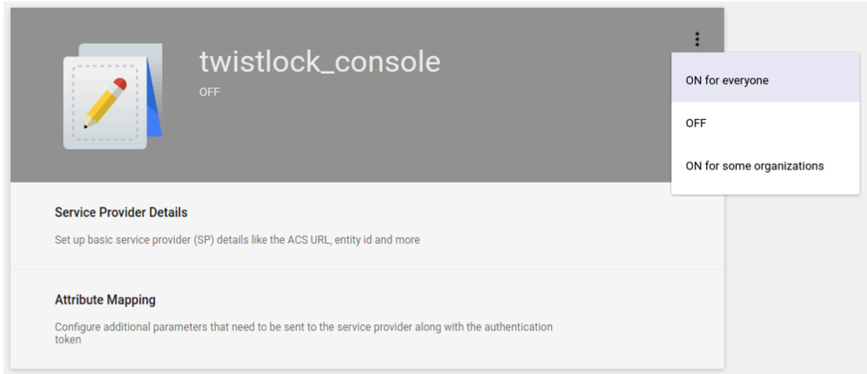
1. In **ACS URL**, enter: **https://<CONSOLE_IPADDR | CONSOLE_HOSTNAME>:8083/api/v1/authenticate**.
2. In **Entity ID**, enter: **twistlock**.
3. Enable **Signed Response**.



STEP 9 | Click **FINISH**, then **OK**.



STEP 10 | Turn the application to on. Select either **ON** for everyone or **ON for some organizations**.



Setting up Prisma Cloud

Set up Prisma Cloud for G Suite integration.

STEP 1 | Log into Console, then go to **Manage** > **Authentication** > **Identity Providers** > **SAML**.

STEP 2 | Set **Integrate SAML users and groups with Prisma Cloud** to **Enabled**.

STEP 3 | Set **Identity provider** to **G Suite**.

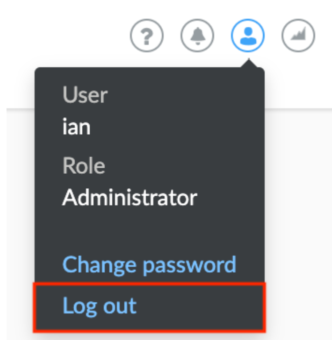
STEP 4 | Set up the following parameters:

1. Paste the SSO URL, Entity ID, and certificate that you copied during the G Suite set up into the **Identity Provider single sign-on URL**, **Identity provider issuer**, and **X.509 certificate** fields.
2. Set **Audience** to match the application Entity ID configured in G Suite. Enter **twistlock**.
3. Click **Save**.

STEP 5 | Go to **Manage** > **Authentication** > **Users**, and click **Add user**.

STEP 6 | In the **Username** field, enter the G Suite email address the user you want to add. Select a role, then click **Save**. Be sure **Create user in local Prisma Cloud account database** is **Off**.

STEP 7 | Log out of Console.



You will be redirected into G Suite and you might need to enter your credentials. After that, you will be redirected back into Prisma Cloud and authenticated as a user.

Integrate with Azure Active Directory via SAML 2.0 federation

Edit on GitHub

Many organizations use SAML to authenticate users for web services. Prisma Cloud supports the SAML 2.0 federation protocol to access the Prisma Cloud Console. When SAML authentication is enabled, users can log into the Console with their federated credentials. This article provides detailed steps for federating your Prisma Cloud Console with your Azure Active Directory (AAD) tenant's Identity Provider.

The Prisma Cloud/Azure Active Directory SAML federation workflow is as follows:

1. User browses to their Prisma Cloud Console.
2. The user's browser is redirected to the Azure Active Directory SAML 2.0 endpoint.
3. The user enters their AAD credentials to authenticate. Multi-factor authentication can be enforced at this step.
4. An AAD SAML token is returned to the user's Prisma Cloud Console.
5. Prisma Cloud Console validates the Azure Active Directory SAML token's signature and associates the user to their Prisma Cloud account via user identity mapping or group membership. Prisma Cloud supports SAML groups for Azure Active Directory federation.

The Azure Portal may change the Enterprise Application SAML federation workflow over time. The concepts and steps outlined in this document can be applied to any Non-gallery application.

The Prisma Cloud Console is integrated with Azure Active Directory as a federated SAML Enterprise Application. The steps to set up the integration are:

- Configure Azure Active Directory
  - Prisma Cloud User to AAD User identity mapping
  - Prisma Cloud Groups to AAD Group mapping
    - Add permissions to allow Prisma Cloud Console to query the Azure Active Directory API
- Configure Prisma Cloud Console
  - Prisma Cloud User to AAD User identity association
  - Group mapping without calling Azure Active Directory API
  - Group mapping with calling Azure Active Directory API

Configure Azure Active Directory

Prerequisites:

- Required Azure Active Directory SKU: Premium
- Required Azure Active Directory role: Global Administrator

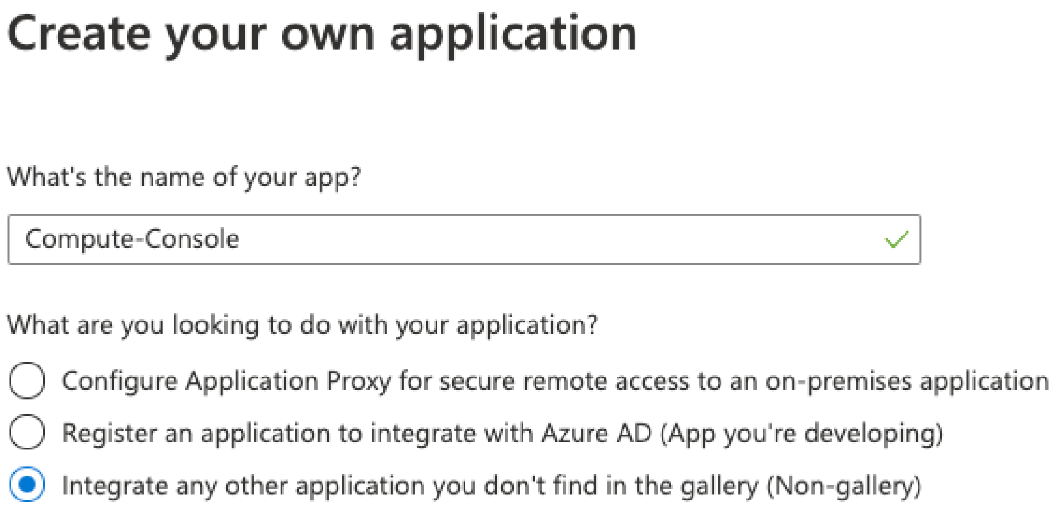
STEP 1 | Log onto your Azure Active Directory tenant (https://portal.azure.com)

STEP 2 | Go to *Azure Active Directory > Enterprise Applications*

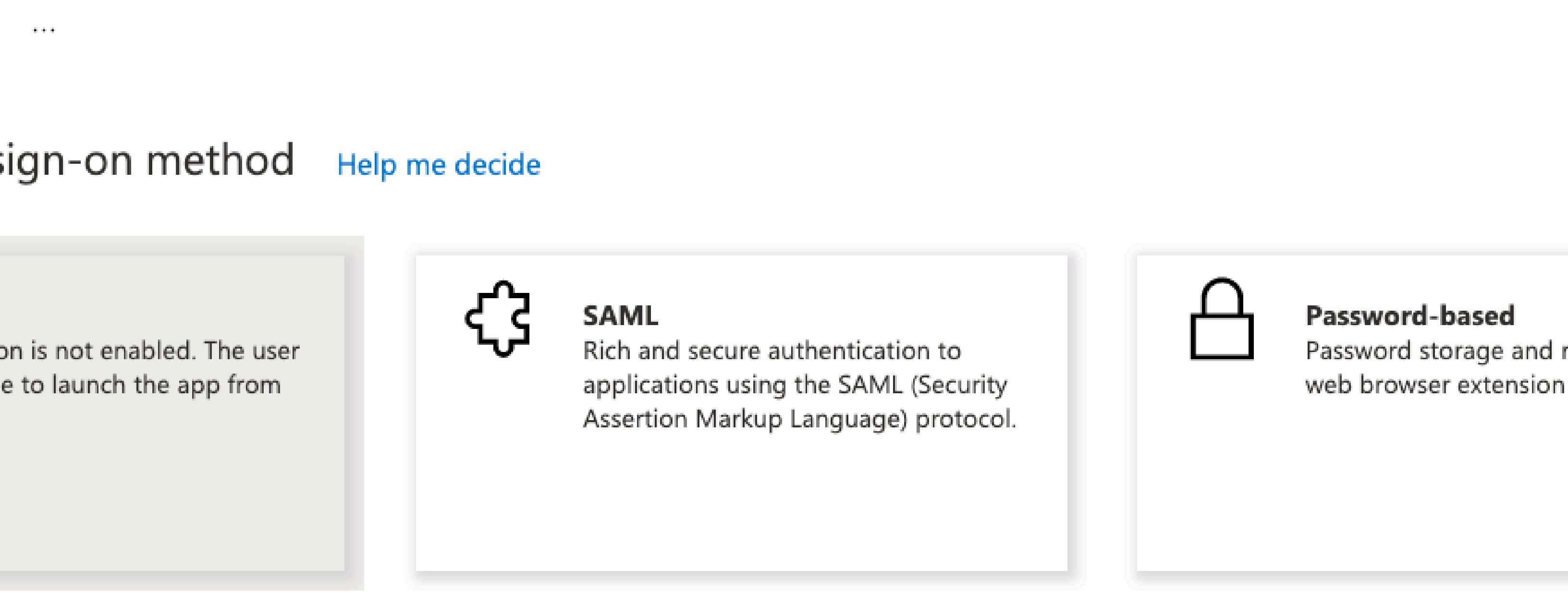
STEP 3 | On the top left of the window pane, click **+ New Application**

STEP 4 | Select **+ Create your own application** on the top left of the window pane

STEP 5 | In the *Name* field enter **Compute-Console**, select the *Integrate any other application you don't find in the gallery (Non-gallery)* radio button and then click **Create**. In this example I am using "Compute-Console" as the application's identifier.

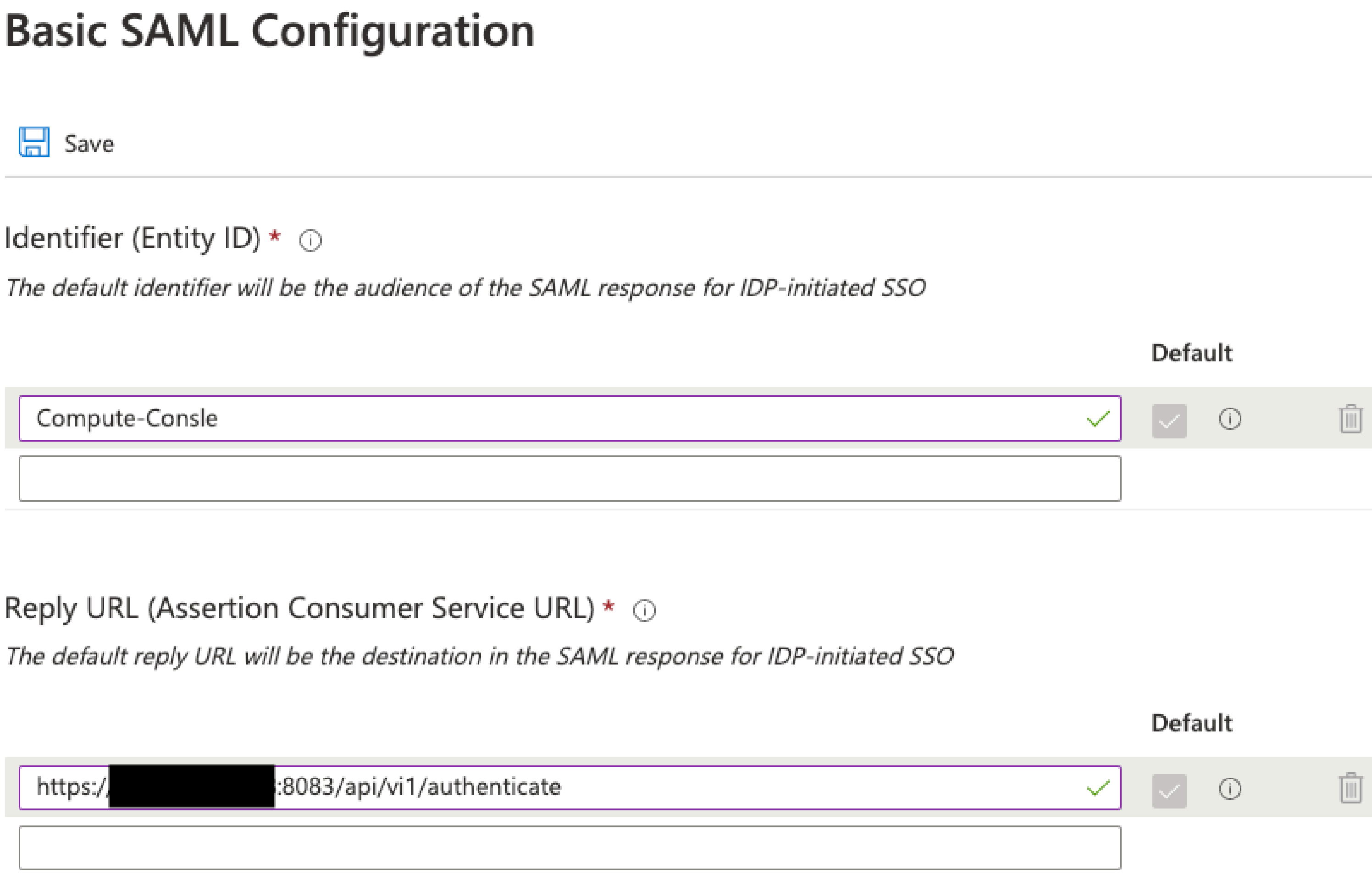


STEP 6 | The *Compute-Console* overview page will appear, select **2. Single sign-on** and then choose **SAML**



STEP 7 | Section #1 *Basic SAML Configuration*:

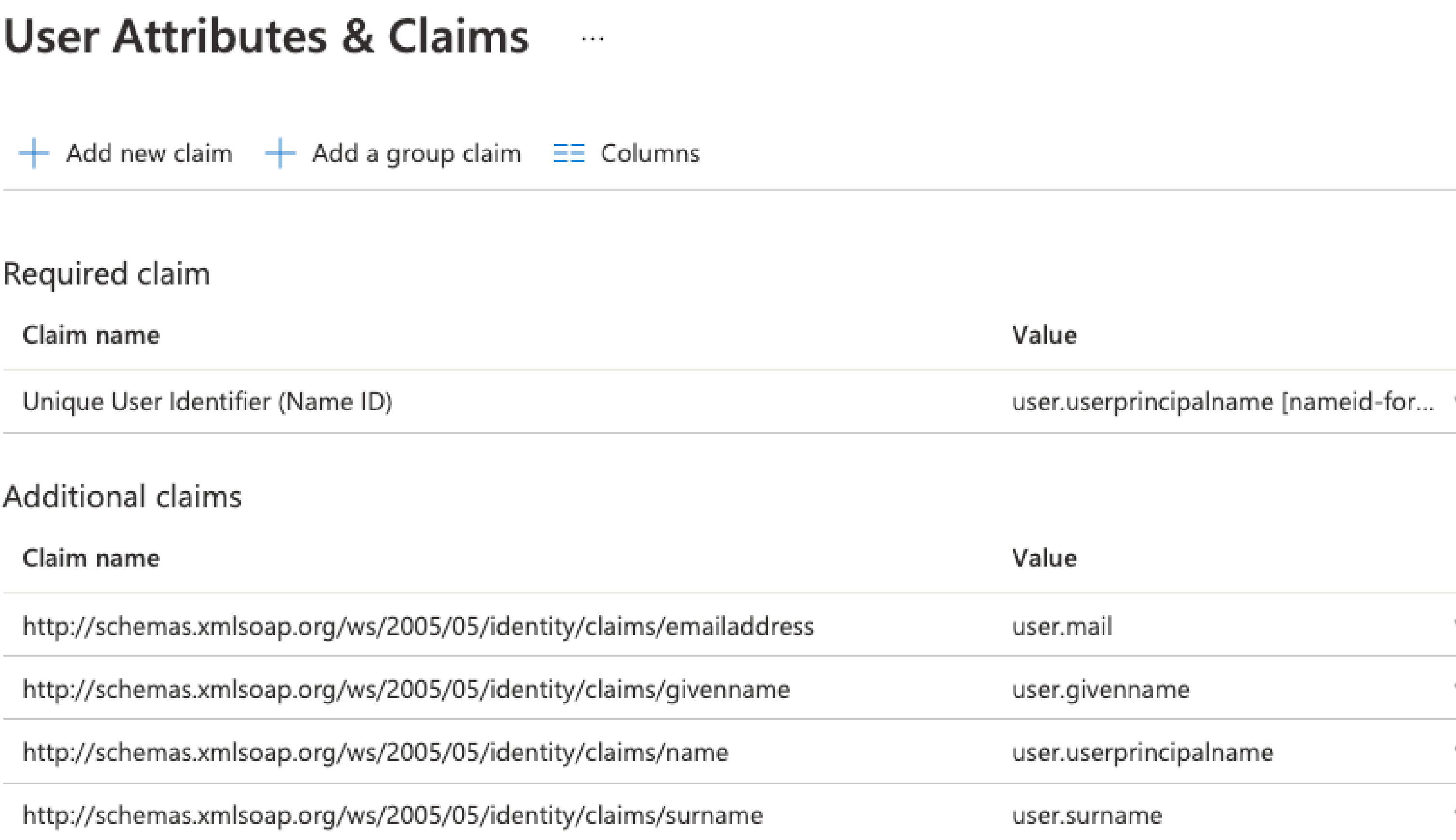
1. *Identifier*: **Compute-Console** Set to your Console's unique Audience value. You will configure this value within your Prisma Cloud Console at a later step.
2. *Reply URL*: **https://<FQDN_of_your_Prisma Cloud_Console>:8083/api/v1/authenticate**



STEP 8 | Section #2 *User Attributes & Claims*:

Select the Azure AD user attribute that will be used as the user account name within Prisma Cloud. This will be the NameID claim within the SAML response token. We recommend using the default value.

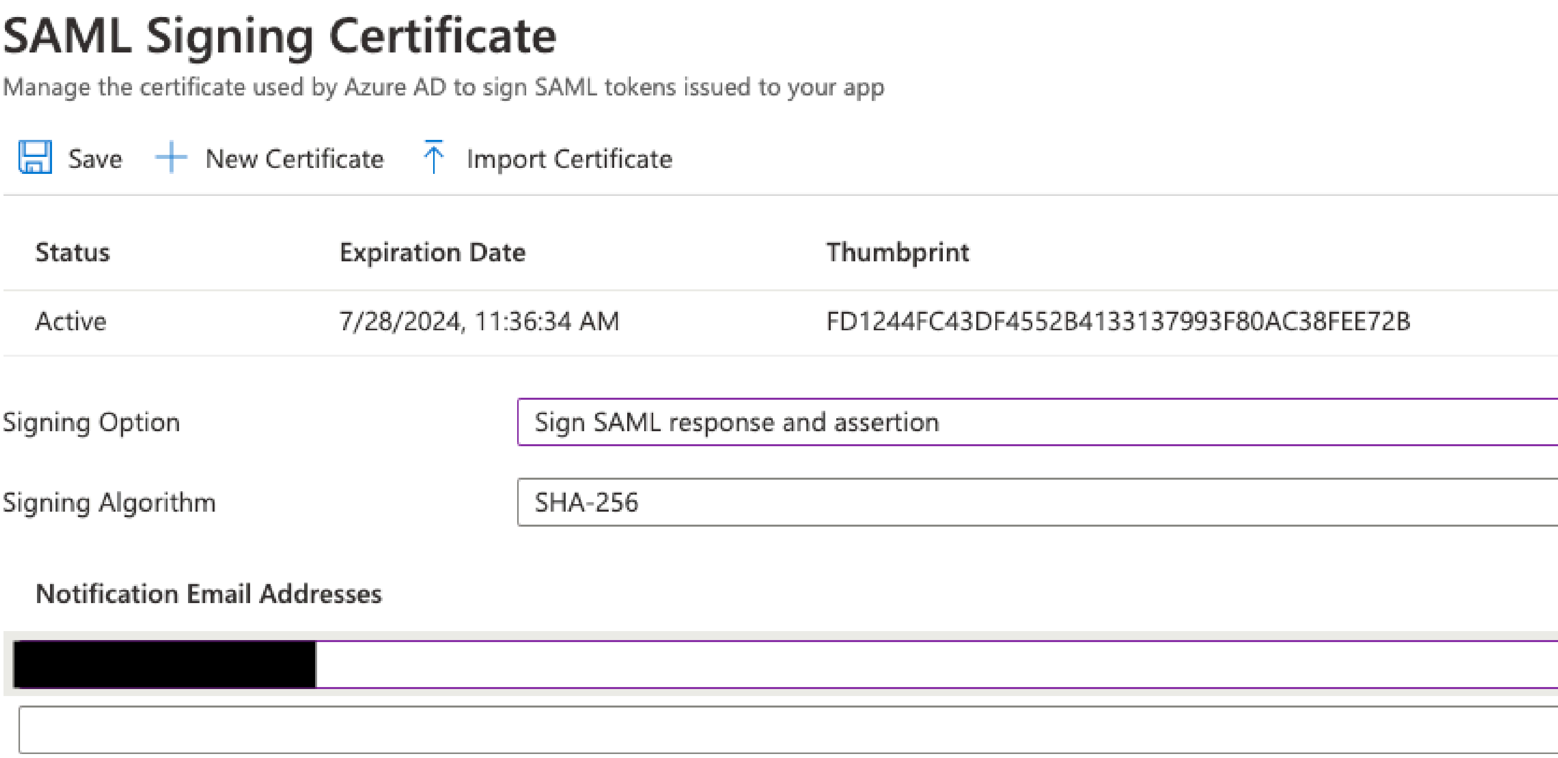
1. *Unique User Identifier (Name ID)*: **user.userprincipalname [nameid-format:emailAddress]**



Even if you are using AAD Groups to assign access to Prisma Cloud set the NamedID claim.

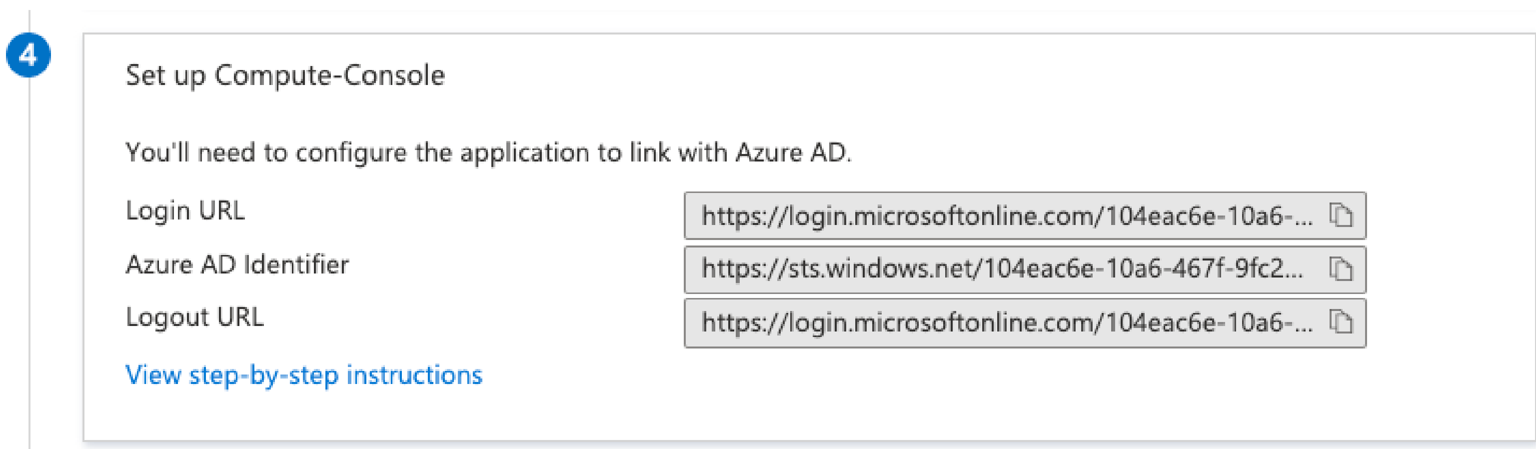
STEP 9 | Section #3 *SAML Signing Certificate*:

1. Select **Download: Certificate (Base64)**
2. Select the edit icon
3. Set *Signing Option*: **Sign SAML Response and Asertion**



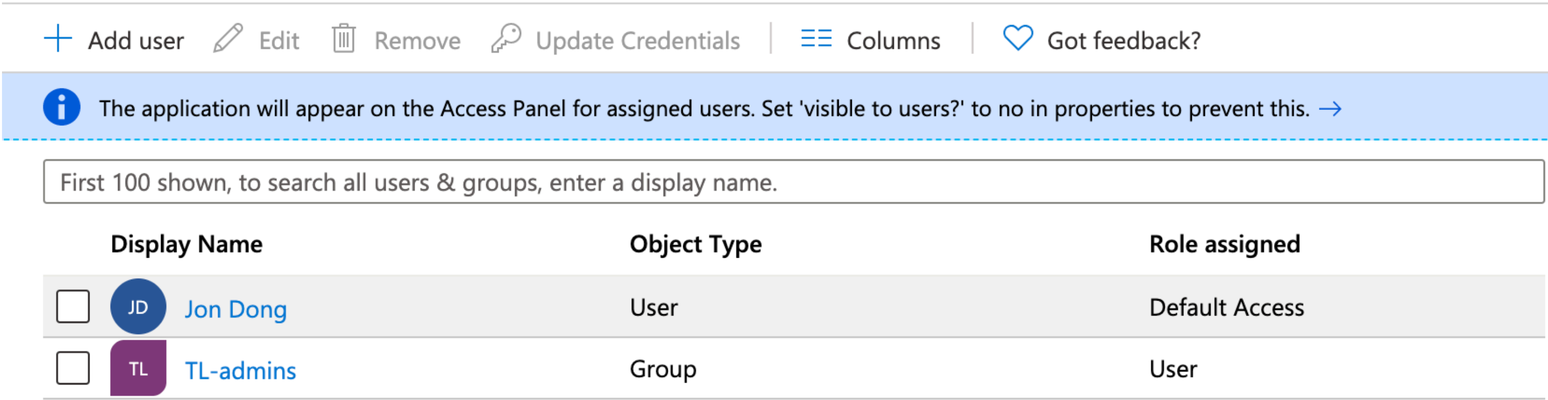
STEP 10 | Section #4 *Set up Compute-Console*:

Save the value of of *Login URL* and *Azure AD Identifier*. You will use these values for the configuration of the Prisma Cloud Console in a later step.



STEP 11 | Copy the *Application ID*. You can find this within the *Properties* tab in the Manage section of the application.

STEP 12 | Click on *1. Assign users and groups* within the Manage section of the application. Add the users and/or groups that will have the right to authenticate to Prisma Cloud Console.



Prisma Cloud User to AAD User identity mapping

If you plan to map Azure Active Directory users to Prisma Cloud user accounts go to Prisma Cloud User to AAD User identity association.

Prisma Cloud Groups to AAD Group mapping

When you use Azure Active Directory groups to map to Prisma Cloud SAML groups, do not create users in the Prisma Cloud Console. Configure the AAD SAML application to send group membership (http://schemas.microsoft.com/ws/2008/06/identity/claims/groups) claims within the SAML response token. When you enable AAD group authentication the Prisma Cloud user to AAD user identity method of association will be ignored.

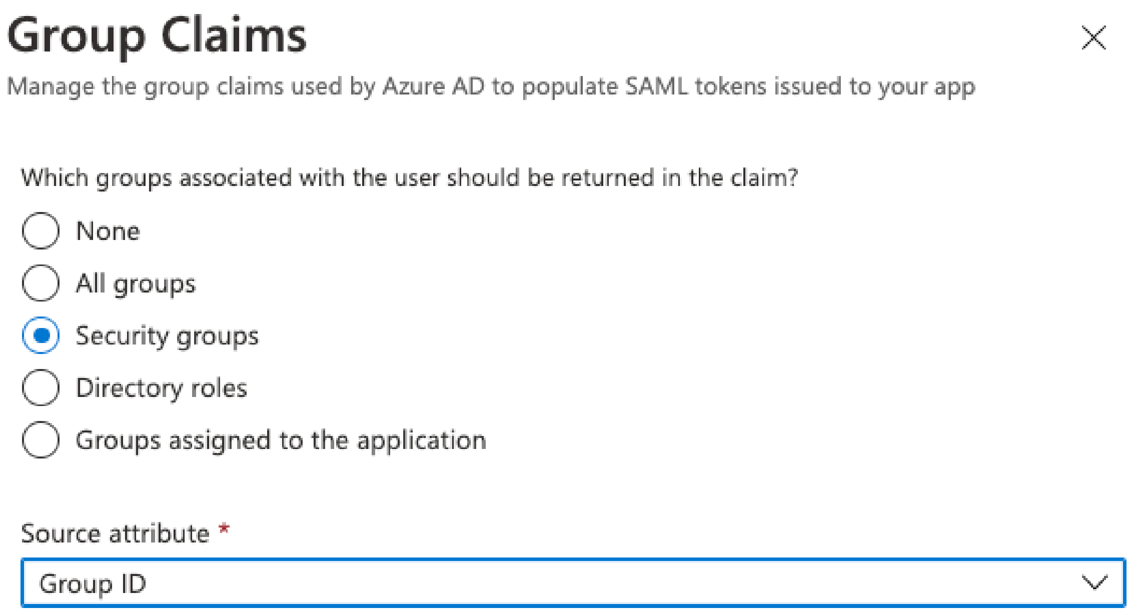
Prisma Cloud Compute version 22_06 now uses the Microsoft Graph API

When the Azure Active Directory SAML response returns a group claim it contains the user's group OIDs as the values. When adding AAD groups within the Console using the group's name the Console will perform a call to the Microsoft Graph API endpoint (https://graph.microsoft.com) to determine the OID of the group. Therefore you will need to configure the Console to query the Azure Active Directory API. For users whose group membership exceeds 150 groups the Console will have to perform an Microsoft Graph API call to query for the full group membership of the user. In this scenario it is recommended to use ApplicationGroups to emit only the groups that are explicitly assigned to the application and the user is a member of.

Prisma Cloud Compute version 21_08 and higher supports the scenerio in which the Console is unable to call the Microsoft Graph API. The AAD group's OID is supplied as the *OID* value when configuring the Console's SAML groups.

STEP 1 | Configure the application to send group claims within the SAML response token:

1. In Azure go to *Azure Active Directory > Enterprise applications > Compute-Console*
2. Under Manage click *Single sign-on*
3. Click the edit for section **2. User Attributes & Claims**
4. Click **Add a group claim**
5. Select the **Security groups** radio button
6. Set *Source attribute* to **Group ID**



STEP 2 | Assign the group to the application

1. In Azure go to *Azure Active Directory > Enterprise applications > Compute-Console*
2. Under Manage click *Users and groups*
3. Click **+ Add user/group**
4. Under *Users and groups* click **None Selected**
5. Select the group to be used for authentication to the Console and click **Select**
6. At the *Add Assignment* window click **Assign**

   If you plan not to use the Azure Active Directory API call functionality to determine the group's OID based upon the supplied group name and/or scenarios in which a user's group membership is greater than 150 groups go to Group mapping without calling Azure Active Directory API. Otherwise, continue with the following steps.

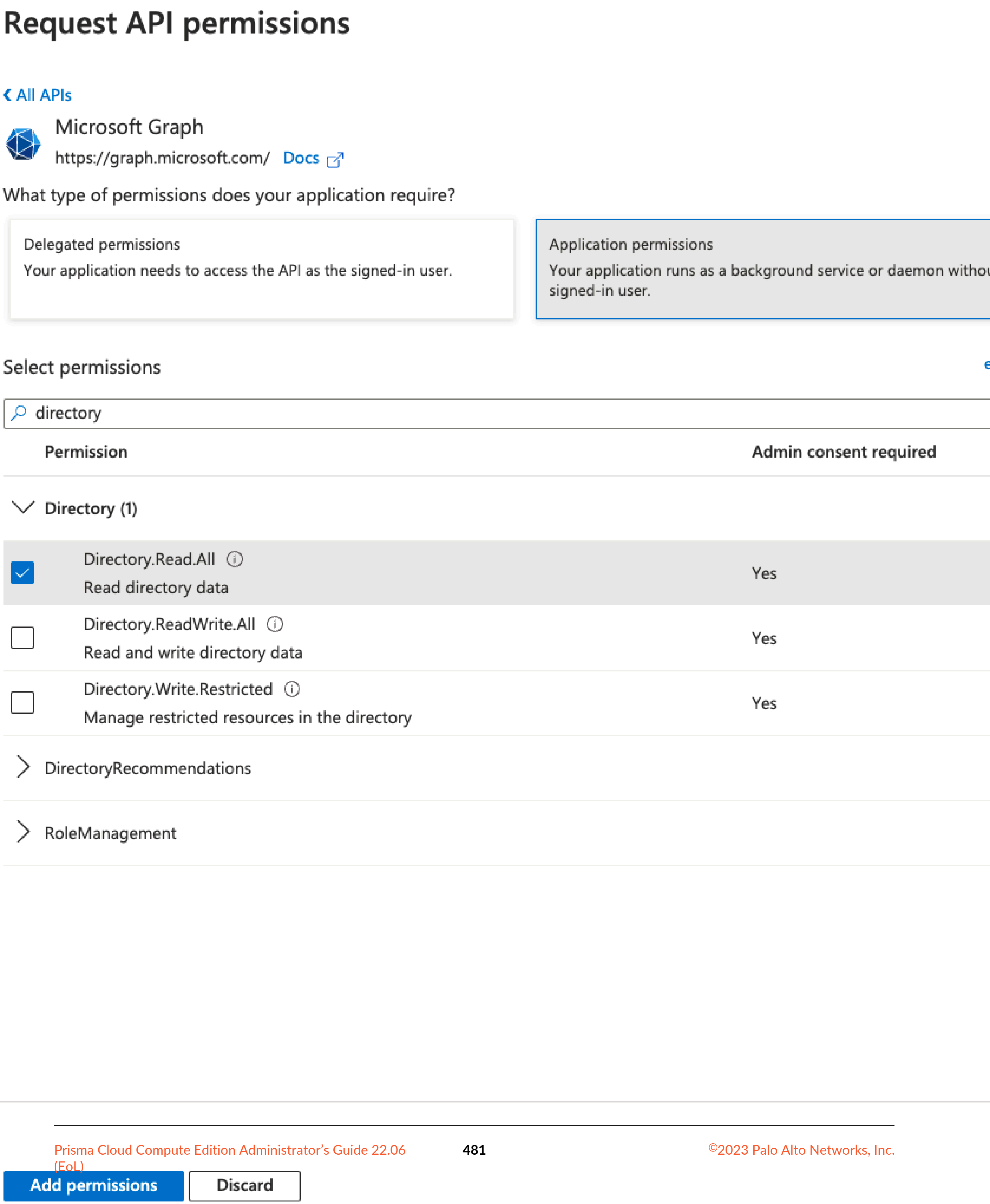
Add permissions to allow Prisma Cloud Console to query the Azure Active Directory API

Add these permissions to allow Prisma Cloud Console to query the Azure Active Directory API. These permissions are required in the following scenarios.

- Your Azure Active Directory (AAD) has users that belong to more than 150 groups.
- You add groups in the Prisma Cloud Console without their Object ID (OID).

STEP 1 | Set Application permissions:

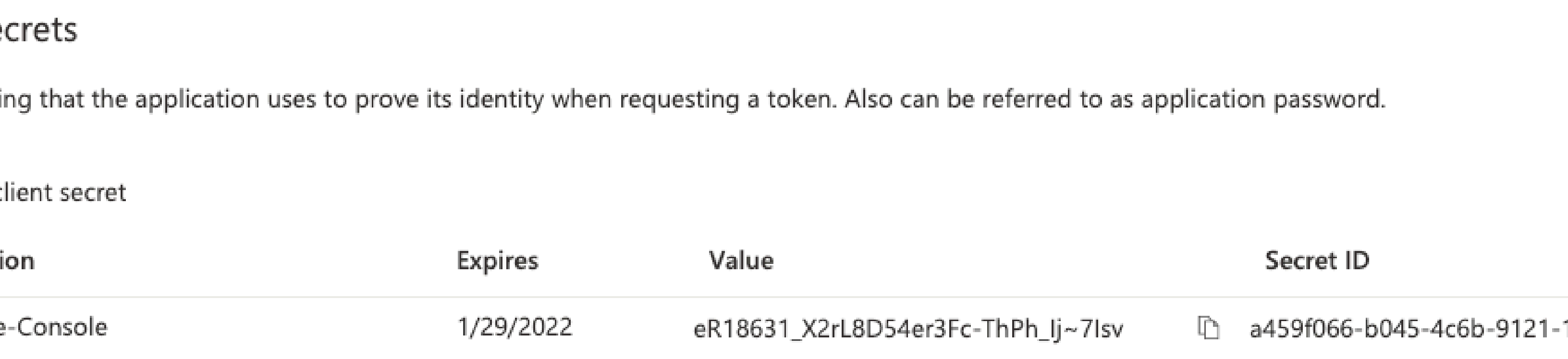
1. In Azure go to *Azure Active Directory* > *App registrations* > *Compute-Console*
2. Under the *Manage* section, go to *API Permissions*
3. Click on **Add a Permission**
4. Click on **Microsoft Graph**
5. *Select permissions*: **Application Permissions: Directory.Read.All**


Request API permissions
All APIs
Microsoft Graph
https://graph.microsoft.com/ Docs
What type of permissions does your application require?
Delegated permissions
Your application needs to access the API as the signed-in user.
Application permissions
Your application runs as a background service or daemon withou
signed-in user.
Select permissions
directory
Permission
Admin consent required
Directory (1)
Directory.Read.All
Read directory data
Yes
Directory.ReadWrite.All
Read and write directory data
Yes
Directory.Write.Restricted
Manage restricted resources in the directory
Yes
DirectoryRecommendations
RoleManagement
Add permissions
Discard

6. Click *Add Permissions*
7. Click *Grant admin consent for Default Directory* within the Configured permissions blade

STEP 2 | Create Application Secret

1. Under the Manage section, go to *Certificates & secrets*
2. Click on **New client secret**
3. Add a *secret description*
4. *Expires*: **Never**
5. Click *Add*
6. Make sure to save the secret *value* that is generated before closing the blade



Allow several minutes for these permissions to propagate within AAD.

Continue the configuration by going to Group mapping with calling Azure Active Directory API

Configure Prisma Cloud Console

Configure Prisma Cloud Compute Console.

Prisma Cloud User to AAD User identity association

Configure Prisma Cloud Console's SAML settings for user identity based logon.

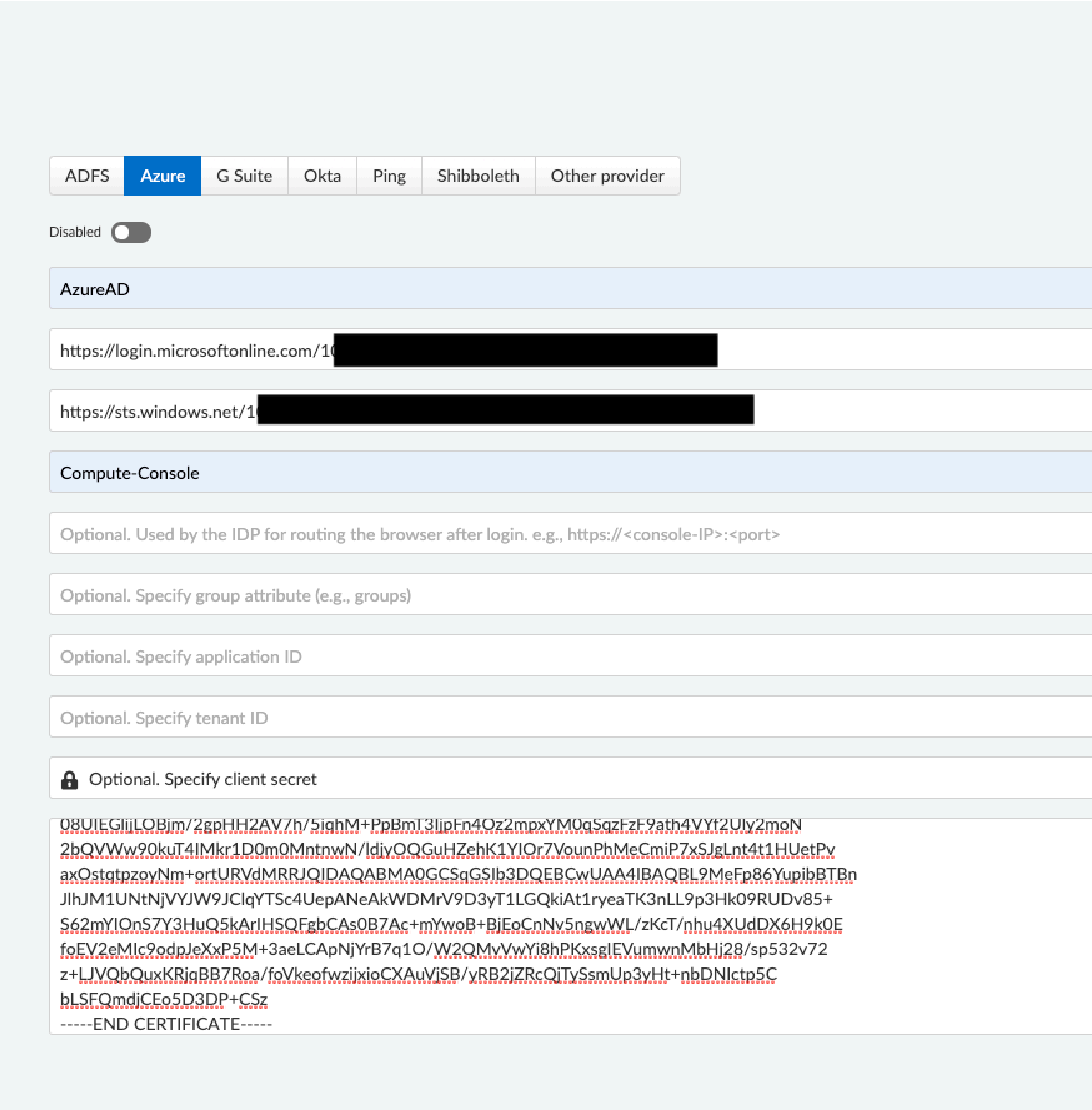
STEP 1 | Log into Prisma Cloud Console as an administrator

STEP 2 | Go to **Manage > Authentication > Identity Providers > SAML**

STEP 3 | Set **SAML settings** to **Enabled**

STEP 4 | Set **Identity Provider** to **Azure**

1. In **Provider alias** enter an identifier for this SAML provider (e.g. AzureAD)
2. In **Identity provider single sign-on URL** enter the Azure AD provided **Login URL**
3. In **Identity provider issuer** enter the Azure AD provided **Azure AD Identifier**
4. In **Audience** enter **Compute-Console**
5. In **X.509 certificate** paste the Azure AD SAML **Signing Certificate Base64** into this field



STEP 5 | Click **Save**

Map an Azure Active Directory user to a Prisma Cloud account

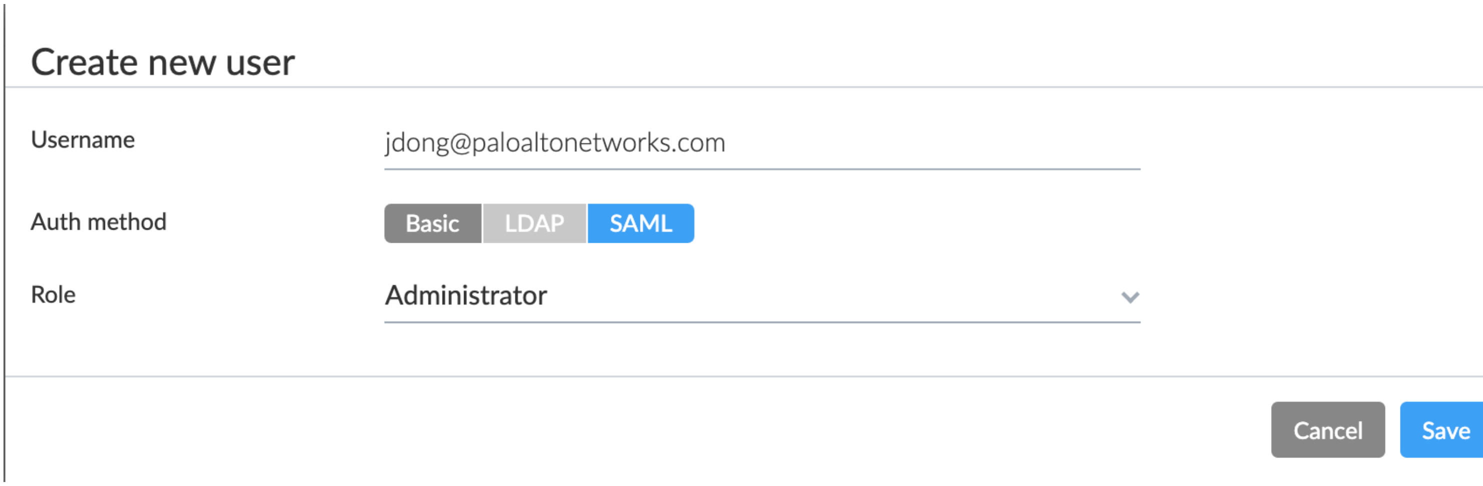
Map an Azure Active Directory user to a Prisma Cloud account.

STEP 1 | Go to **Manage > Authentication > Users**

STEP 2 | Click **Add user**

STEP 3 | **Create a New User**

1. **Username**: Azure Active Directory *userprincipalname*
2. **Auth Method**: Select **SAML**
3. **Role**: Select the appropriate role for the user



4. Click **Save**

Group mapping without calling Azure Active Directory API

In this configuration the Console will not call the Microsoft Graph API to determine the group's AAD OID based upon the group name supplied. If a user's security group membership is greater than 150 groups and the Console is unable to perform the Microsoft Graph API query it is recommended to to use ApplicationGroups.

Configure Prisma Cloud Console's SAML settings for group based logon.

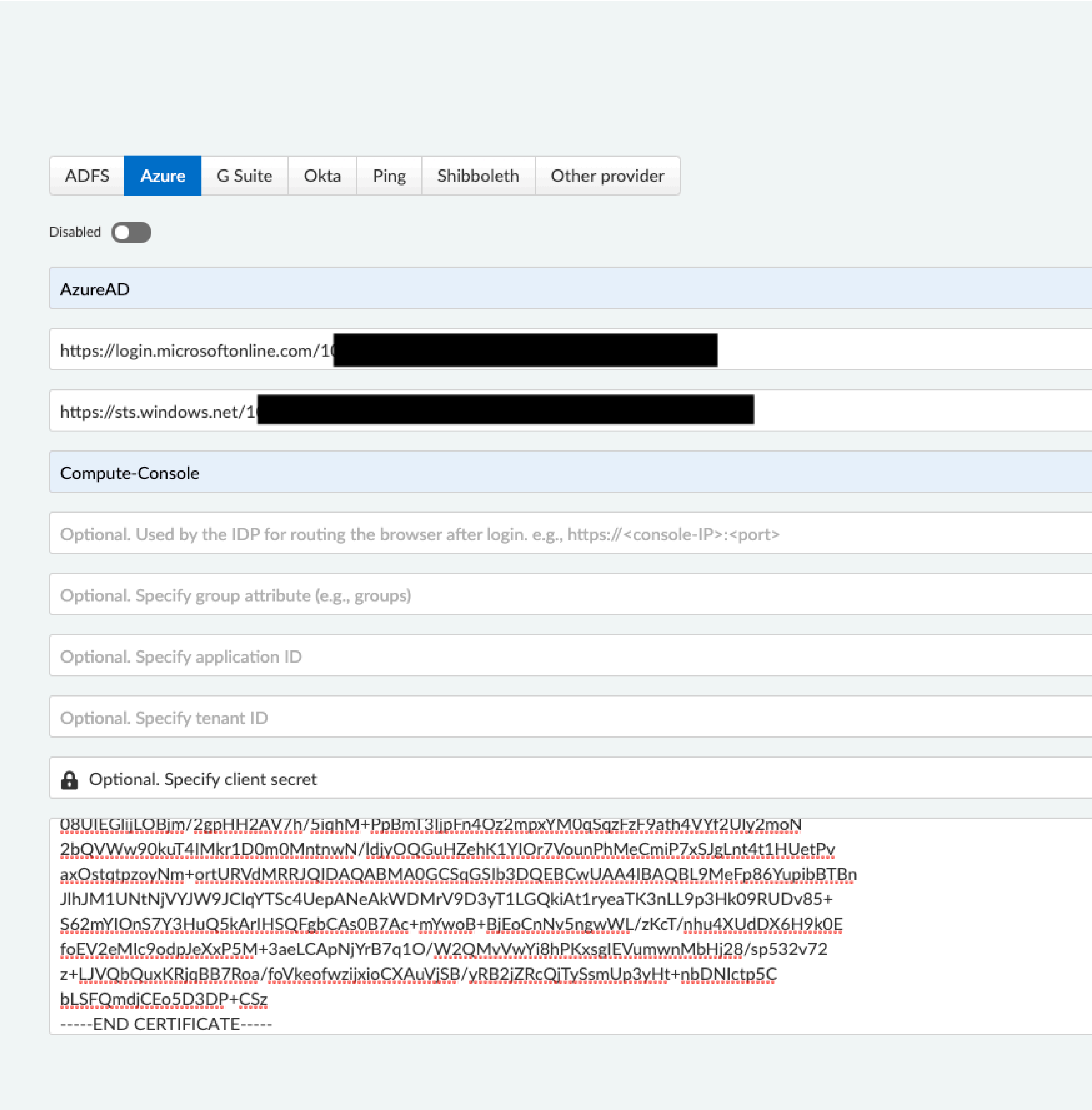
STEP 1 | Log into Prisma Cloud Console as an administrator

STEP 2 | Go to **Manage > Authentication > Identity Providers > SAML**

STEP 3 | Set **SAML settings** to **Enabled**

STEP 4 | Set **Identity Provider** to **Azure**

1. In **Provider alias** enter an identifier for this SAML provider (e.g. AzureAD)
2. In **Identity provider single sign-on URL** enter the Azure AD provided **Login URL**
3. In **Identity provider issuer** enter the Azure AD provided **Azure AD Identifier**
4. In **Audience** enter **Compute-Console**
5. In **X.509 certificate** paste the Azure AD SAML **Signing Certificate Base64** into this field



STEP 5 | Click **Save**

Assign the AAD group OID to a role

Assign the AAD group OID to a role.

STEP 1 | Go to **Manage > Authentication > Groups**

STEP 2 | Click **Add Group**

STEP 3 | Enter a display name for the group (e.g. AAD_SAML_admins)

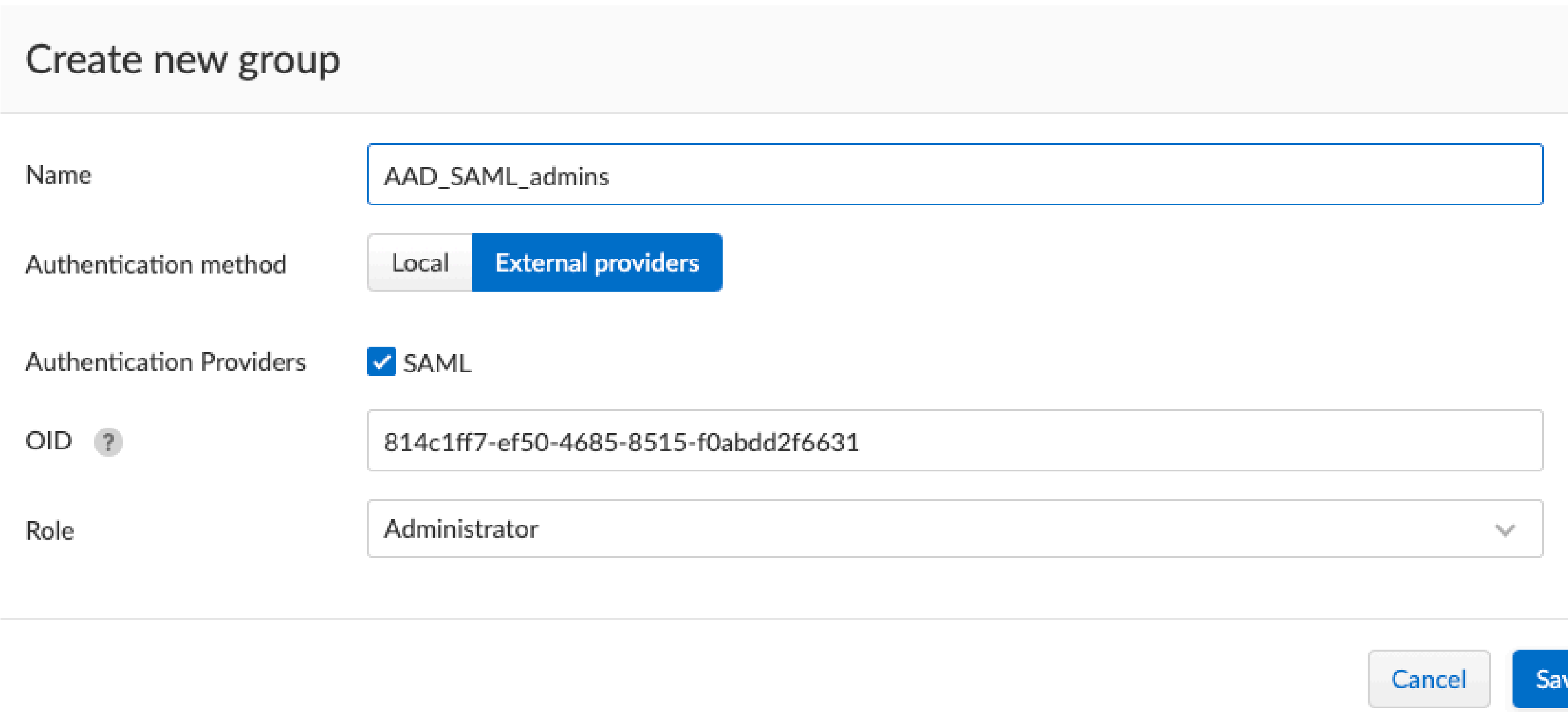
STEP 4 | Select *Authentication method* **External providers**

STEP 5 | Select *Authentication Providers* **SAML**

STEP 6 | Enter the AAD OID of the group within the *OID* field

STEP 7 | Select the Prisma Cloud role for the group

STEP 8 | Click **Save**



Group mapping with calling Azure Active Directory API

Azure Active Directory SAML response will send the user's group membership as OIDs and not the name of the group. When a group name is added, Prisma Cloud Console will query the Microsoft Graph API to determine the OID of the group entered. For users whose group membership exceeds 150 groups the Console will perform an Microsoft Graph API call to query for the full group membership of the user. Ensure your Prisma Cloud Console is able to reach the Microsoft Graph API endpoint (https://graph.microsoft.com).

STEP 1 | Log into Prisma Cloud Console as an administrator

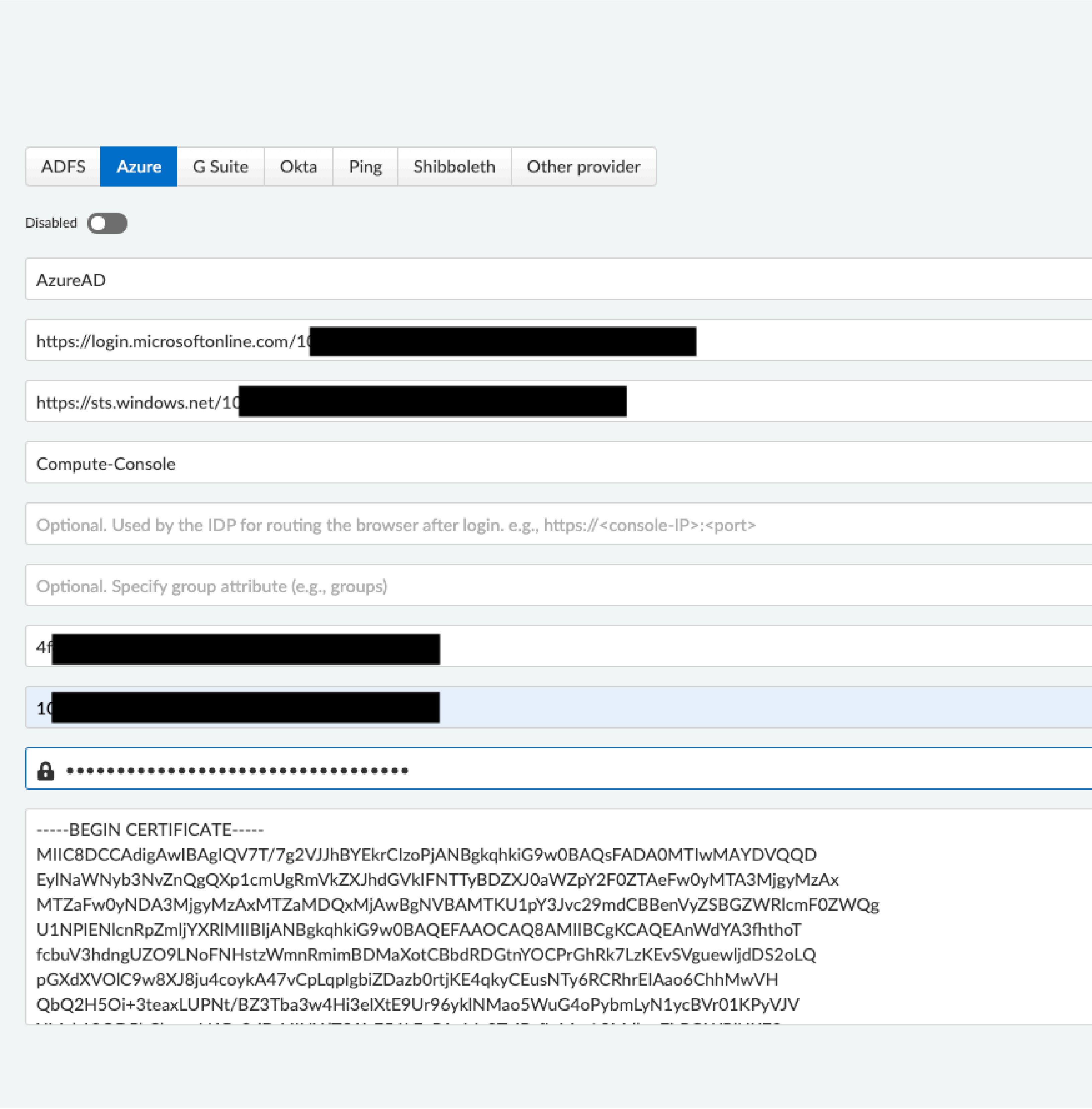
STEP 2 | Go to **Manage > Authentication > Identity Providers > SAML**

STEP 3 | Set **SAML settings** to **Enabled**

STEP 4 | Set **Identity Provider** to **Azure**

1. In **Provider alias** enter an identifier for this SAML provider (e.g. AzureAD)
2. In **Identity provider single sign-on URL** enter the Azure AD provided **Login URL**
3. In **Identity provider issuer** enter the Azure AD provided **Azure AD Identifier**
4. In **Audience** enter **Compute-Console**
5. Enter the **Application ID** of the *Compute-Console* AAD application
6. Enter the **Tenant ID** of your Azure Active Directory
7. Enter the **Application Secret value** for permission to Azure Active Directory API
8. In **X.509 certificate** paste the Azure AD SAML **Signing Certificate Base64** into this field

STEP 5 | Click **Save**



Assign the AAD group name to a role

Assign the AAD group name to a role.

STEP 1 | Go to **Manage > Authentication > Groups**

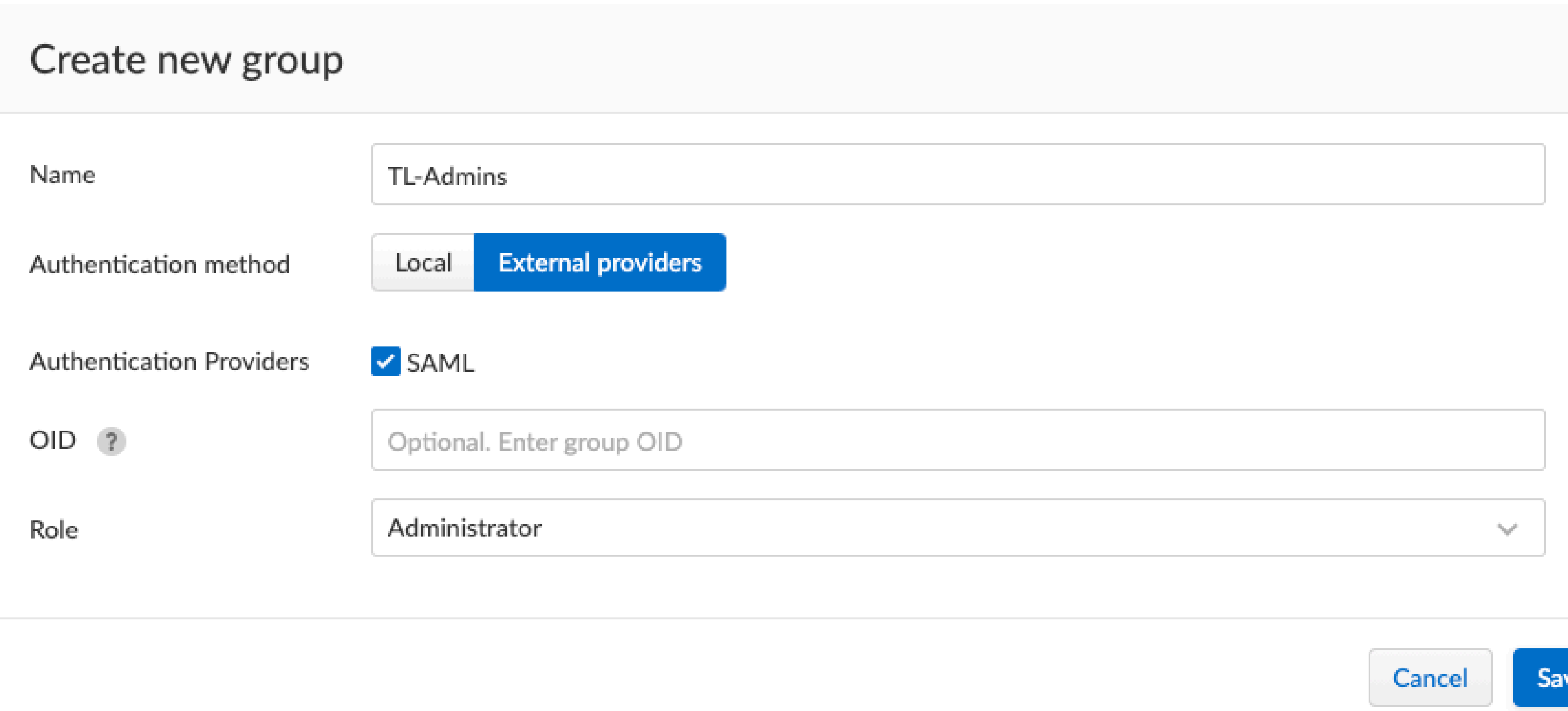
STEP 2 | Click **Add Group**

STEP 3 | Enter the name of the AAD group

STEP 4 | Click the **SAML group** radio button

STEP 5 | Select the Prisma Cloud role for the group

STEP 6 | Click **Save**



Test logging into Prisma Cloud Console via Azure Active Directory SAML federation. Leave your existing session logged into Prisma Cloud Console in case you encounter issues. Open a new incognito browser window and go to **https://<CONSOLE>:8083** and select SAML authentication method.

Integrate with PingFederate via SAML 2.0 federation

Edit on GitHub

Many organizations use SAML to authenticate users for web services. Prisma Cloud supports the SAML 2.0 federation protocol to access the Prisma Cloud Console. When SAML support is enabled, users can log into the Console with their federated credentials. This article provides detailed steps for federating your Prisma Cloud Console with your PingFederate v8.4 Identity Provider (IdP).

The Prisma Cloud/PingFederate SAML federation flow works as follows:

1. Users browse to Prisma Cloud Console.
2. Their browsers are redirected to the PingFederate SAML 2.0 endpoint.
3. They enter their credentials to authenticate. Multi-factor authentication can be enforced at this step.
4. A PingFederate SAML token is returned to Prisma Cloud Console.
5. Prisma Cloud Console validates the SAML token's signature and associates the user to their Prisma Cloud account via user identity mapping or group membership.

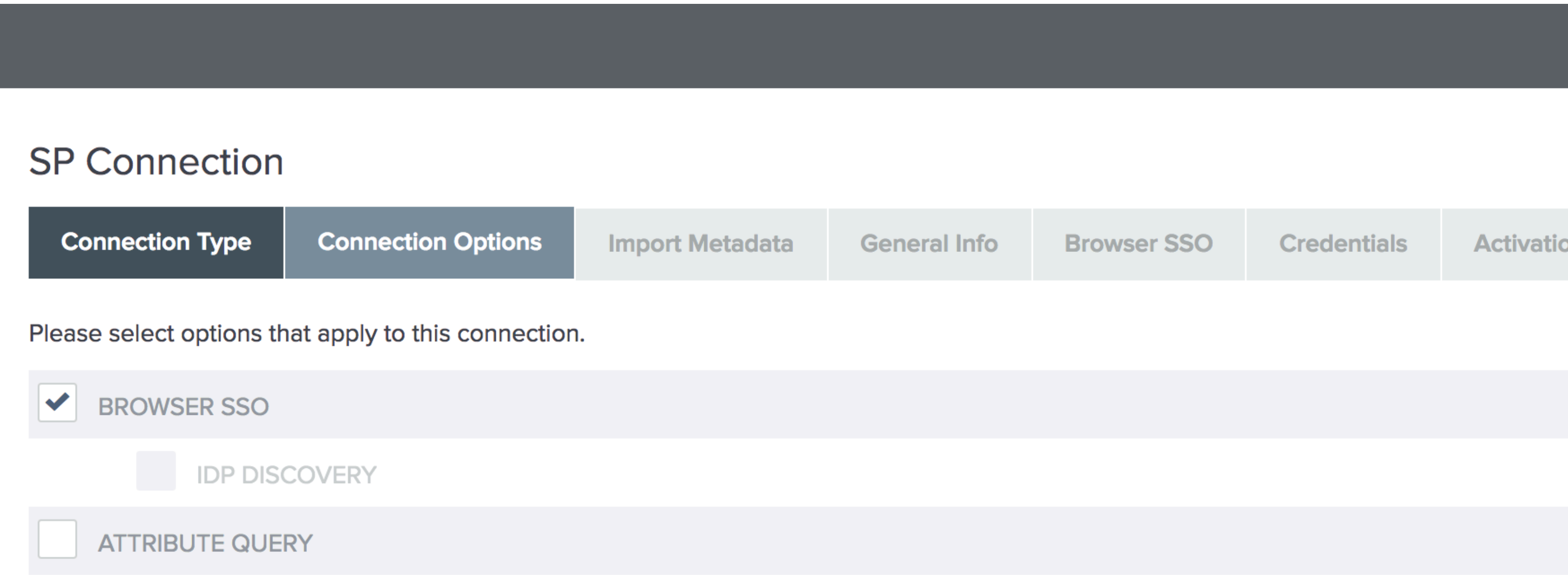
Prisma Cloud Console is integrated with PingFederate as a federated SAML Service Provider. The steps to set up the integration are:

- Configure PingFederate
- Configure Prisma Cloud Console

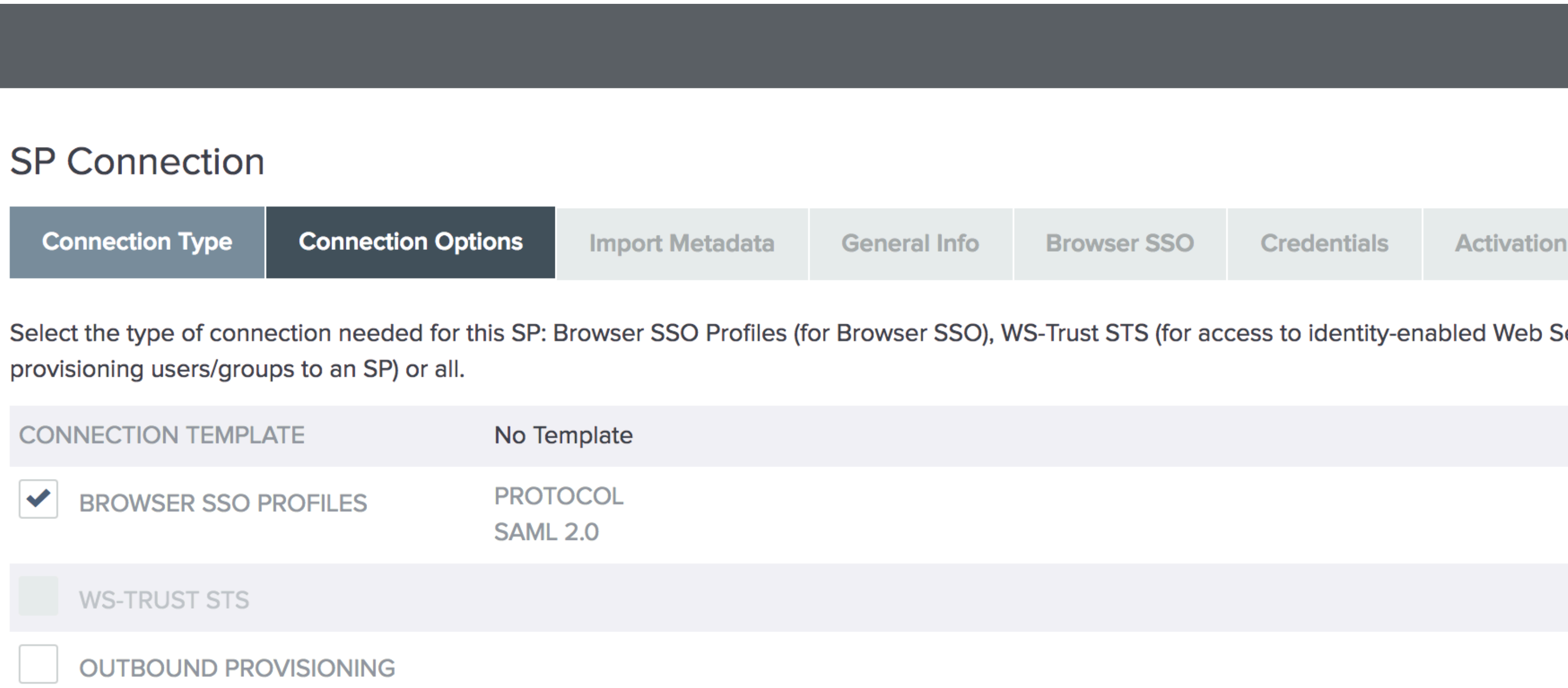
Configure PingFederate

STEP 1 | Logon to PingFederate

STEP 2 | Go to **IdP Configuration > SP Connection > Connection Type**, and select **Browser SSO**.



STEP 3 | Go to **IdP Configuration > SP Connection > Connection Options**, and select **Browser SSO Profiles SAML 2.0**.



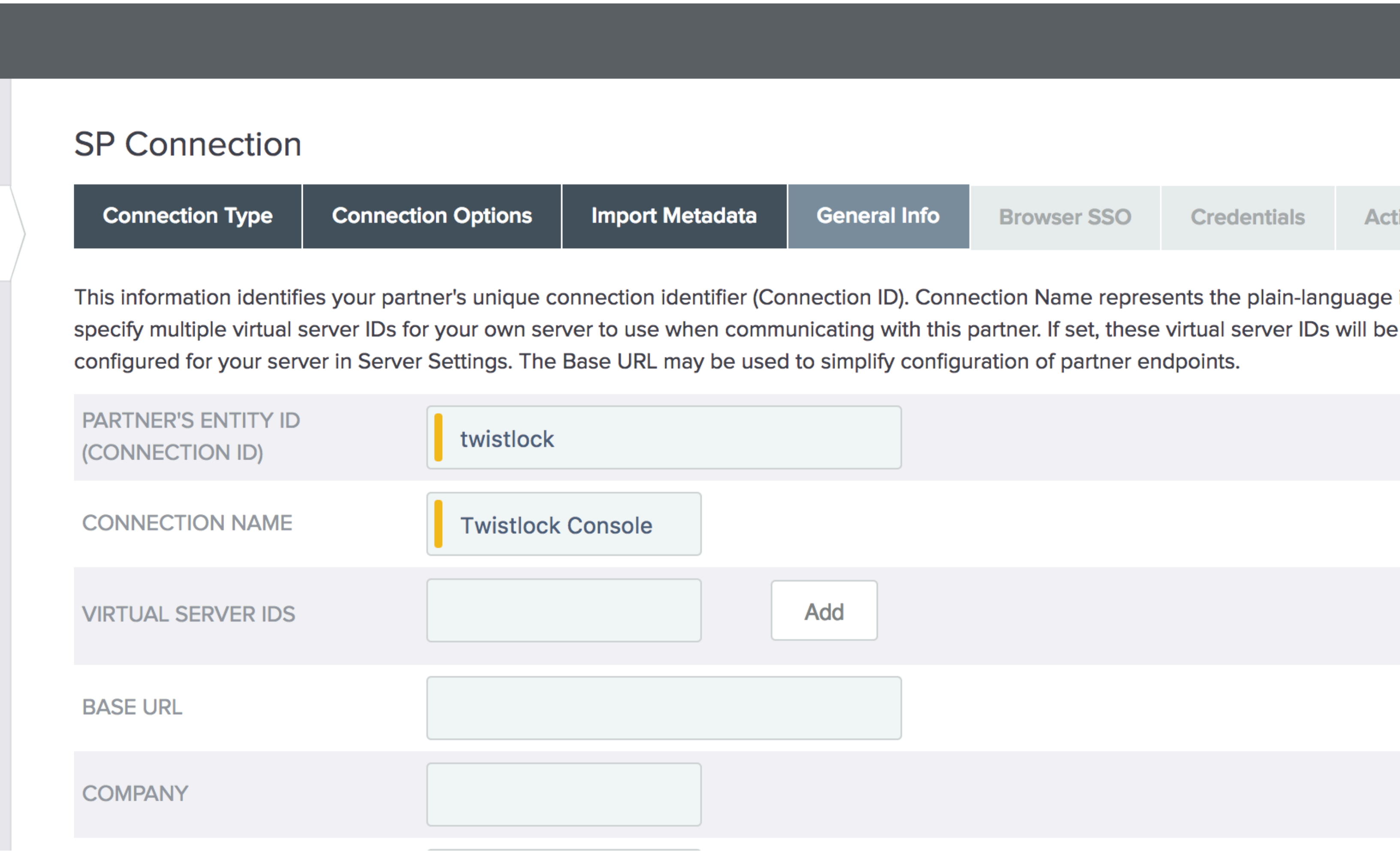
STEP 4 | Skip the **Import Metadata** tab.

STEP 5 | Go to **IdP Configuration > SP Connection > General Info**.

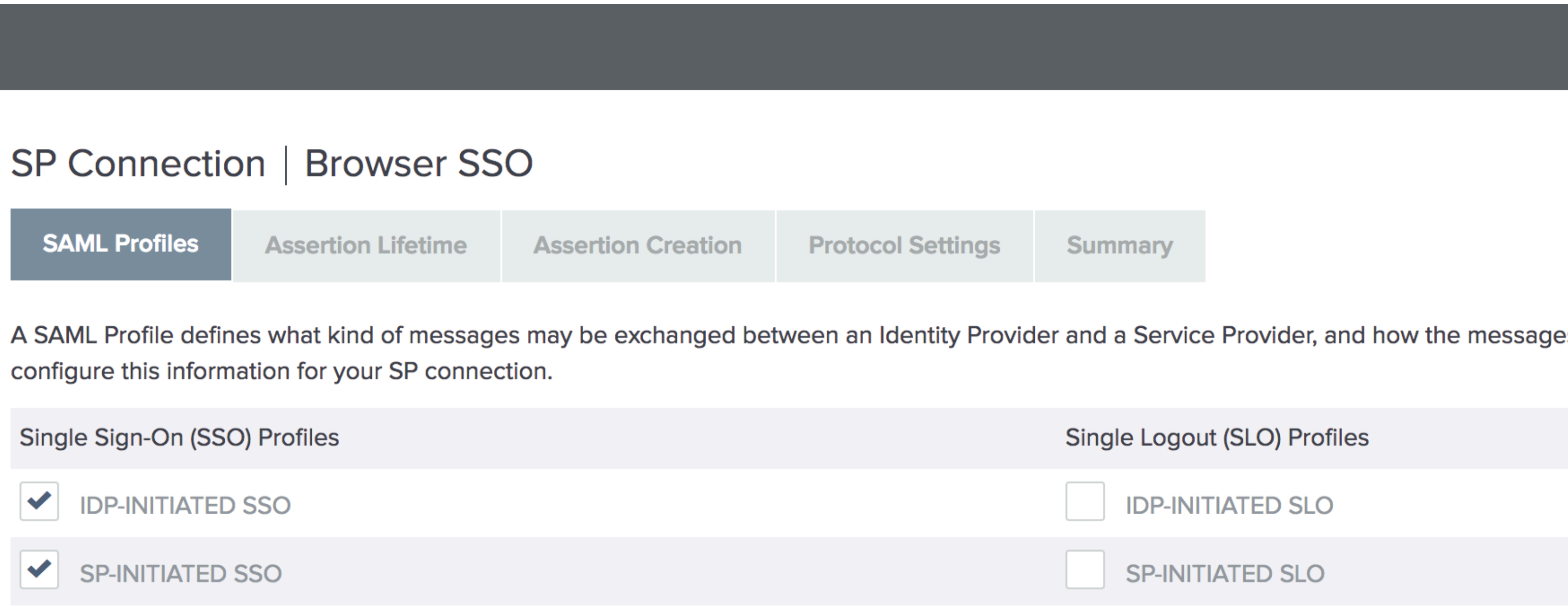
1. In **Partner's Entity ID**, enter **twistlock**.

   By default, the Partner's Entity ID is "twistlock". When configuring the SAML Audience in the Prisma Cloud Console, the default value is "twistlock". If you choose a different value here, be sure to set the same value in your Console.

2. In **Connection Name**, enter **Prisma Cloud Console**.
3. Click **Add**.

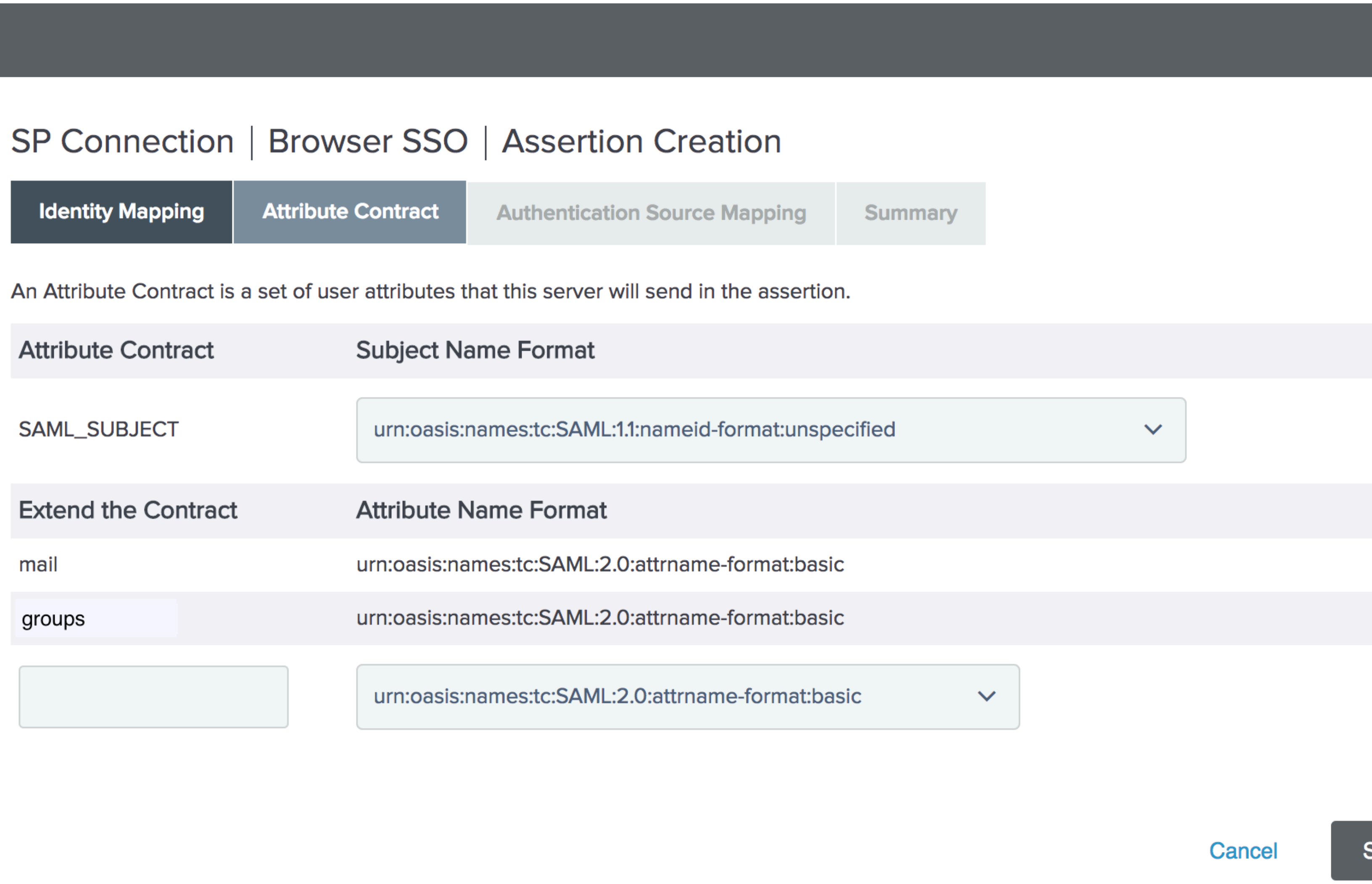


STEP 6 | In **Browser SSO > SAML Profiles**, select both **IDP-INITIATED SSO** and **SP-INITIATED SSO**.



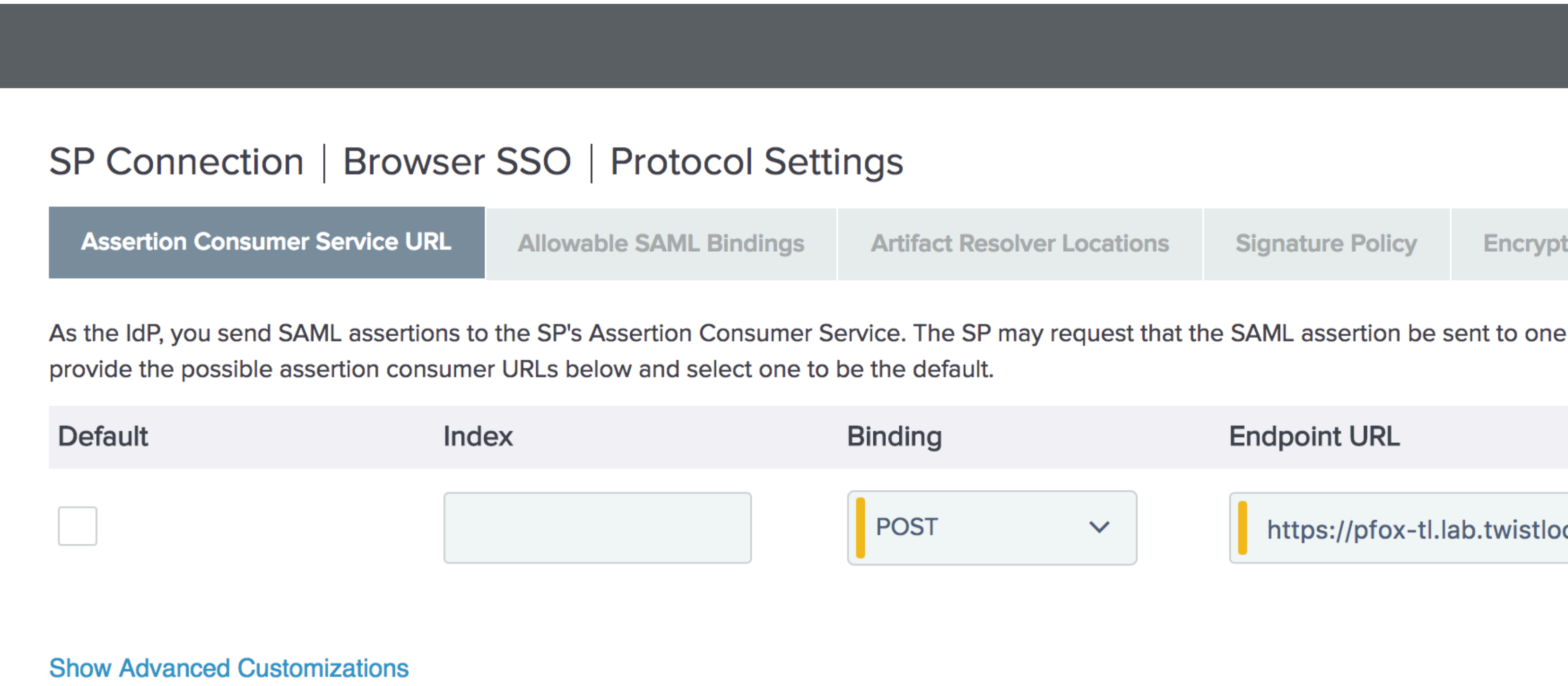
STEP 7 | Go to **Assertion Creation** and set **SAML_SUBJECT** to **SAML 1.1 nameid-format**.

In this example you mapped the user's email address to the SAML_SUBJECT attribute which matches the user's Prisma Cloud account. If you are using group-to-Prisma Cloud-role associations, add **groups** to the list of attributes to be returned in the SAML token.

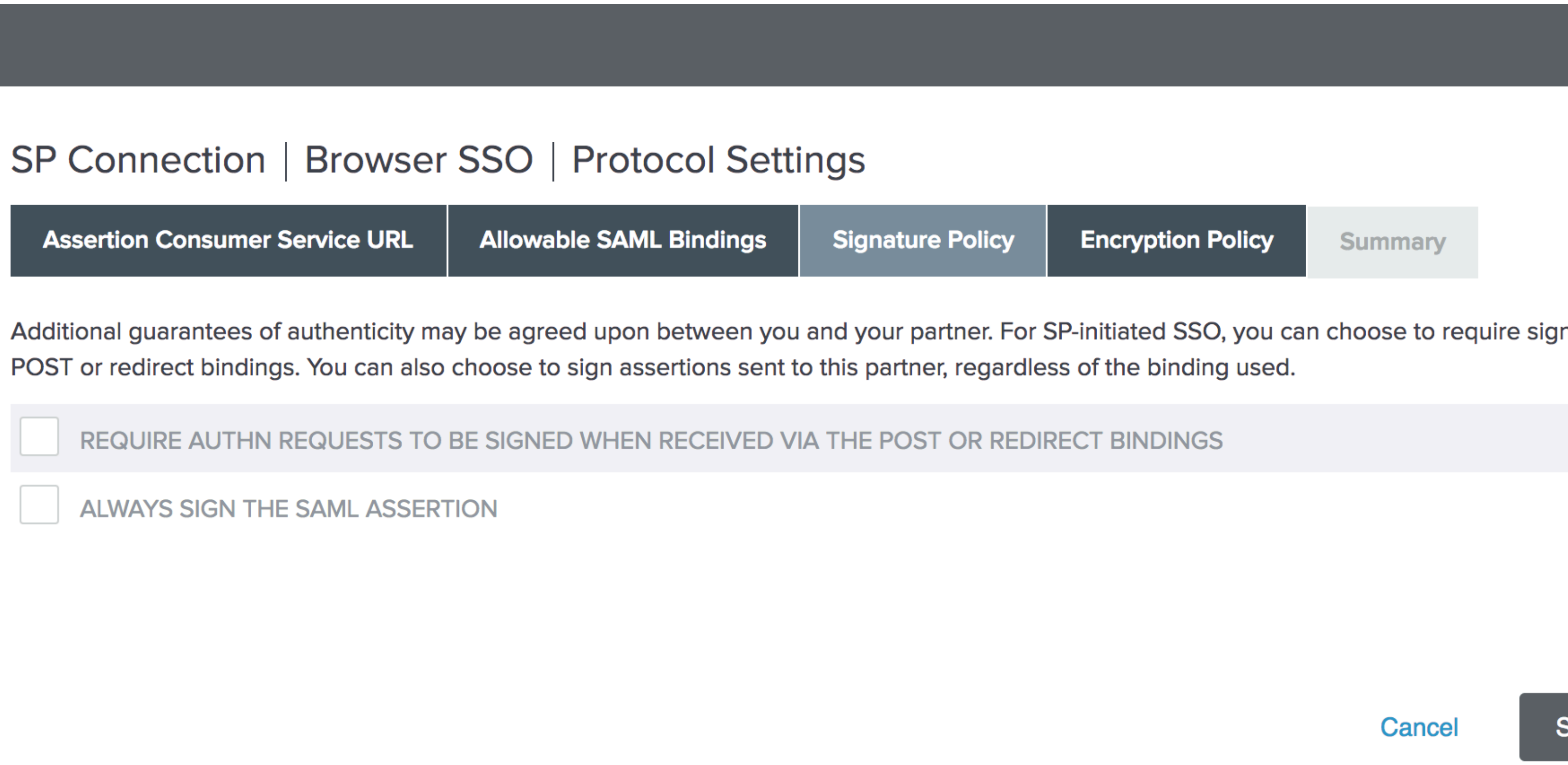


STEP 8 | In **IdP Configuration > SP Connection > Browser SSO > Protocol Settings > Assertion Consumer Service URL**, specify an assertion consumer URL.

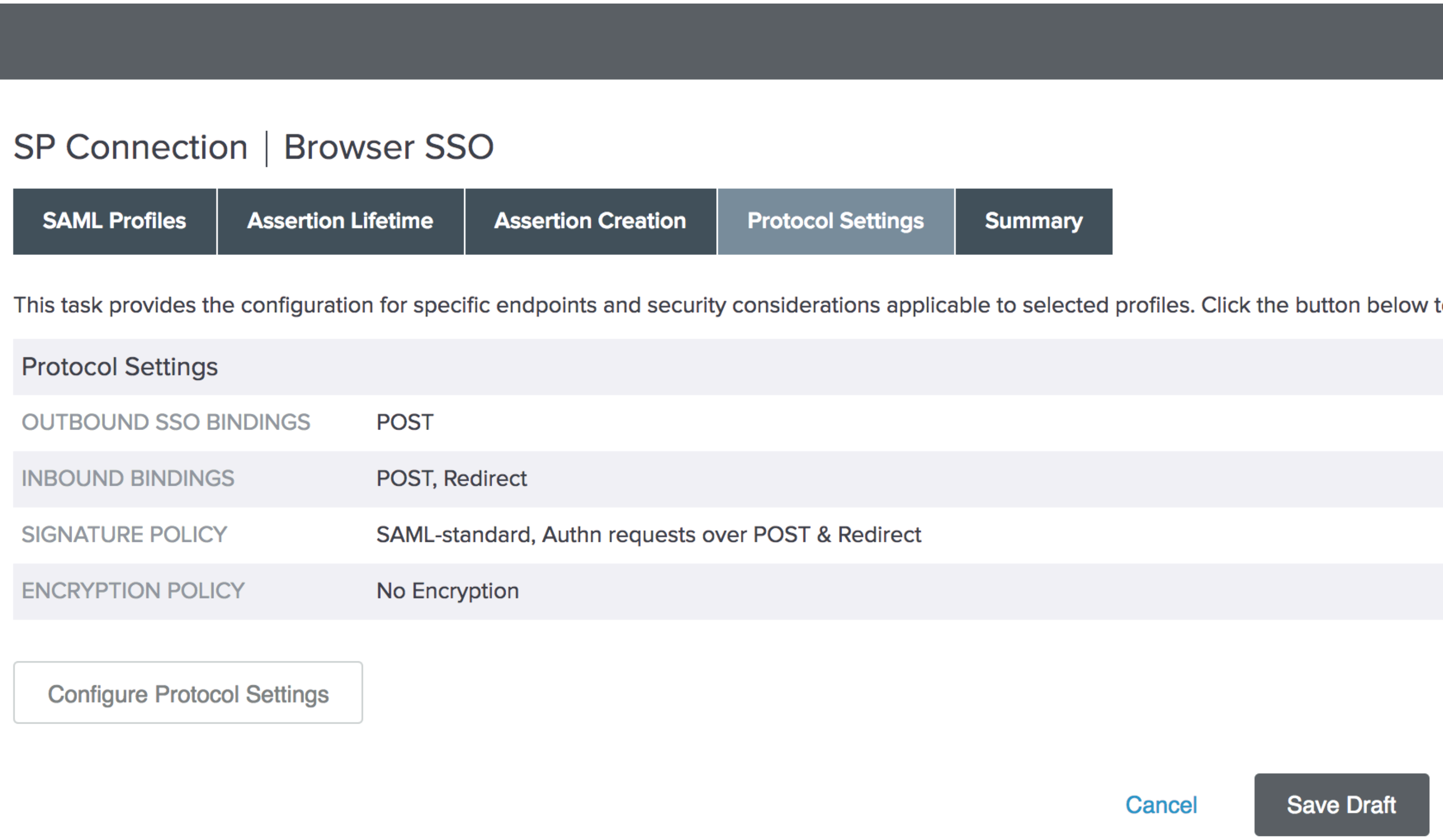
1. Under **Binding**, select **POST**.
2. Under **Endpoint URL**, enter **https://<FQDN_OF_YOUR_TWISTLOCK_CONSOLE>:8083/api/v1/authenticate**.



STEP 9 | In **IdP Configuration > SP Connection > Browser SSO > Protocol Settings > Signature Policy**, leave both values unchecked.



STEP 10 | In **IdP Configuration > SP Connection > Browser SSO > Protocol Settings**, review the protocol settings.



STEP 11 | Click **Done**.

STEP 12 | Copy the PingFederate SAML token signing X.509 certificate as Base64 in **Server Configuration**. This certificate will be imported into Prisma Cloud Console.

Configure Prisma Cloud Console

Configure Prisma Cloud Console.

STEP 1 | Login to the Prisma Cloud Console as an administrator.

STEP 2 | Go to **Manage > Authentication > Identity Providers > SAML**.

STEP 3 | Set **Integrate SAML users and groups with Prisma Cloud** to **Enabled**.

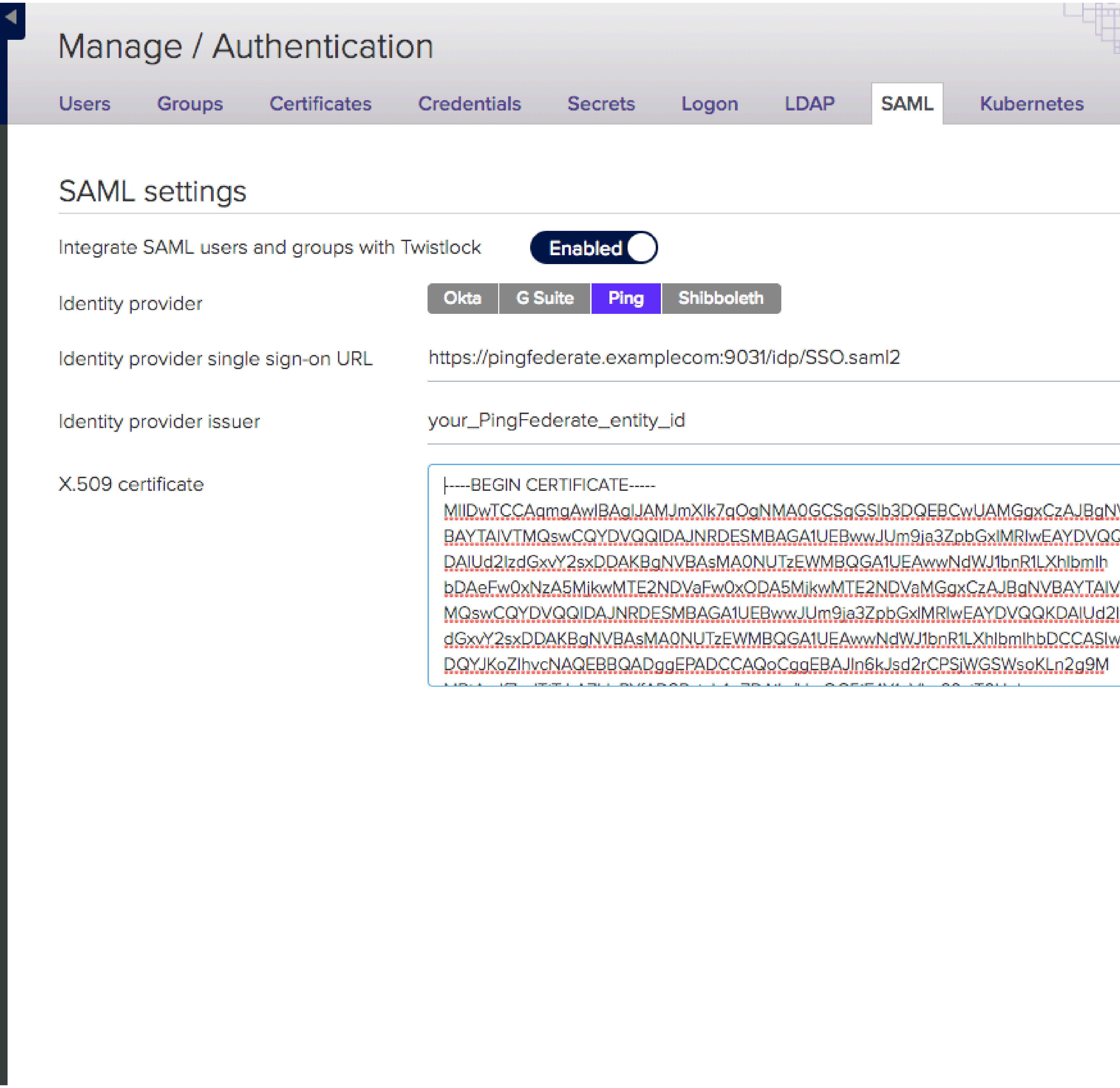
STEP 4 | Set **Identity Provider** to **Ping**.

STEP 5 | In **Identity provider single sign-on URL**, enter your PingFederate IdP endpoint.

STEP 6 | In **Identity provider issuer**, enter your PingFederate Entity ID.

STEP 7 | In **Audience**, enter **twistlock** (default) or the value you set for Partner's Entity ID in PingFederate.

1. In **X.509 certificate**, paste your PingFederate X.509 **Signing Certificate Base64**.



STEP 8 | Click **Save**.

User account name matching

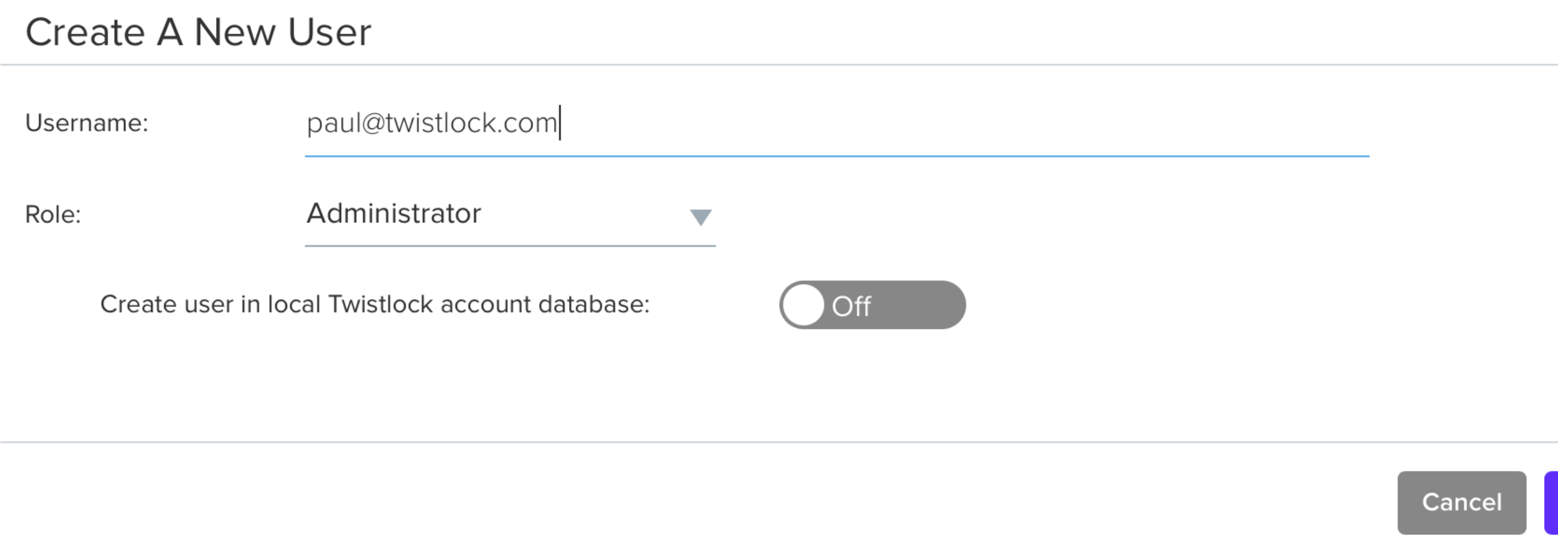
User account name matching.

STEP 1 | Go to **Manage > Authentication > Users**.

STEP 2 | Click **Add user**.

STEP 3 | Create a new user:

1. In **Username**, enter the value returned within the SAML_SUBJECT attribute *IdP user's email address*.
2. In **Role**, select the appropriate role.
3. Set **Create user in local Prisma Cloud account database** to **Off**.



STEP 4 | Click **Save**.

STEP 5 | Test login into the Prisma Cloud Console via PingFederate SAML federation.

Leave your existing session logged onto the Prisma Cloud Console in case you encounter issues. Open a new incognito browser window and go to **https://<CONSOLE>:8083**.

Group name matching

Group name matching.

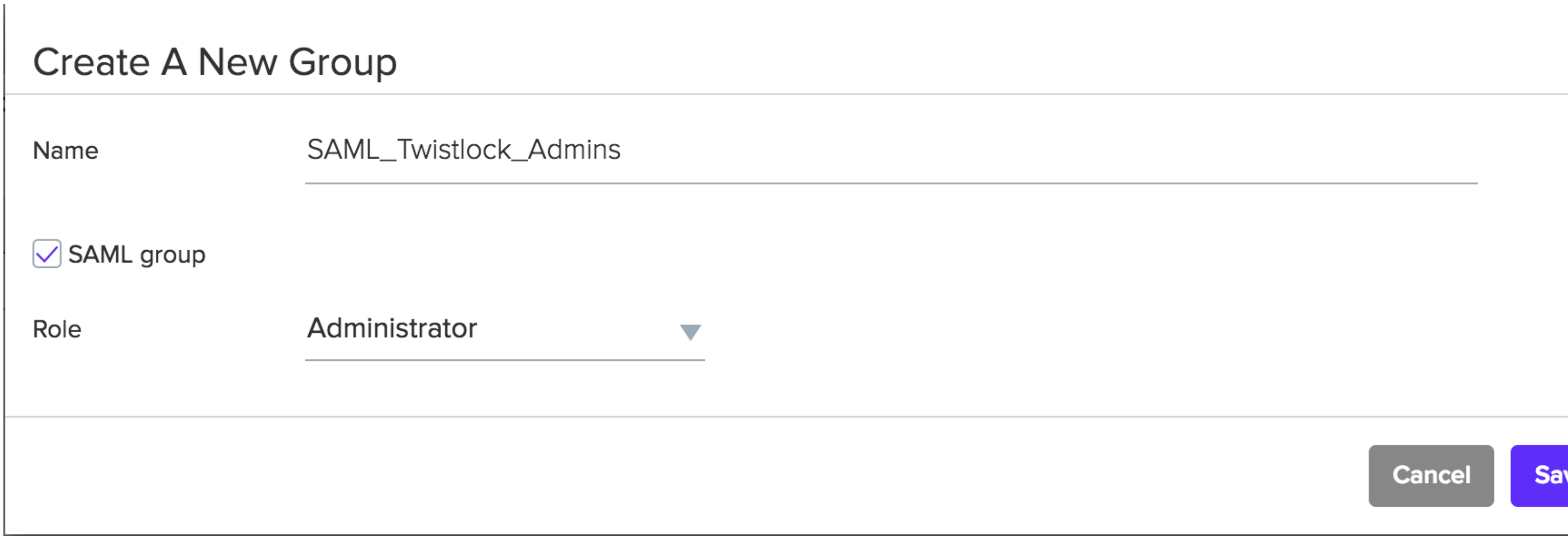
STEP 1 | Go to **Manage > Authentication > Groups**.

STEP 2 | Click the **+Add Group** button.

STEP 3 | In the **Name** field, enter a group name.

> *The group name must exactly match the group name in the SAML IDP. Console does not verify if that the value entered matches a group name in the SAML IDP.*

STEP 4 | Select the **SAML group** checkbox.



STEP 5 | Click **Save**

STEP 6 | Test login into the Prisma Cloud Console via PingFederate SAML federation.

Leave your existing session logged onto the Prisma Cloud Console in case you encounter issues. Open a new incognito browser window and go to **https://<CONSOLE>:8083**.

Integrate with Windows Server 2016 & 2012r2 Active Directory Federation Services (ADFS) via SAML 2.0 federation

Edit on GitHub

Many organizations use SAML to authenticate users for web services. Prisma Cloud supports the SAML 2.0 federation protocol for access to the Prisma Cloud Console. When SAML support is enabled, users can log into Console with their federated credentials. This article provides detailed steps for federating your Prisma Cloud Console with your Active Directory Federation Service (ADFS) Identity Provider (IdP).

Prisma Cloud supports SAML 2.0 federation with Windows Server 2016 and Windows Server 2012r2 Active Directory Federation Services via the SAML protocol. The federation flow works as follows:

1. Users browse to Prisma Cloud Console.
2. Their browsers are redirected to the ADFS SAML 2.0 endpoint.
3. Users authenticate either with Windows Integrated Authentication or Forms Based Authentication. Multi-factor authentication can be enforced at this step.
4. An ADFS SAML token is returned to Prisma Cloud Console.
5. Prisma Cloud Console validates the SAML token's signature and associates the user to their Prisma Cloud account via user identity mapping or group membership.

Prisma Cloud Console is integrated with ADFS as a federated SAML Relying Party Trust.

- Configure Active Directory Federation Services
- Configure the Prisma Cloud Console

The Relying Party trust workflows may differ slightly between Windows Server 2016 and Windows Server 2012r2 ADFS, but the concepts are the same.

Configure Active Directory Federation Services

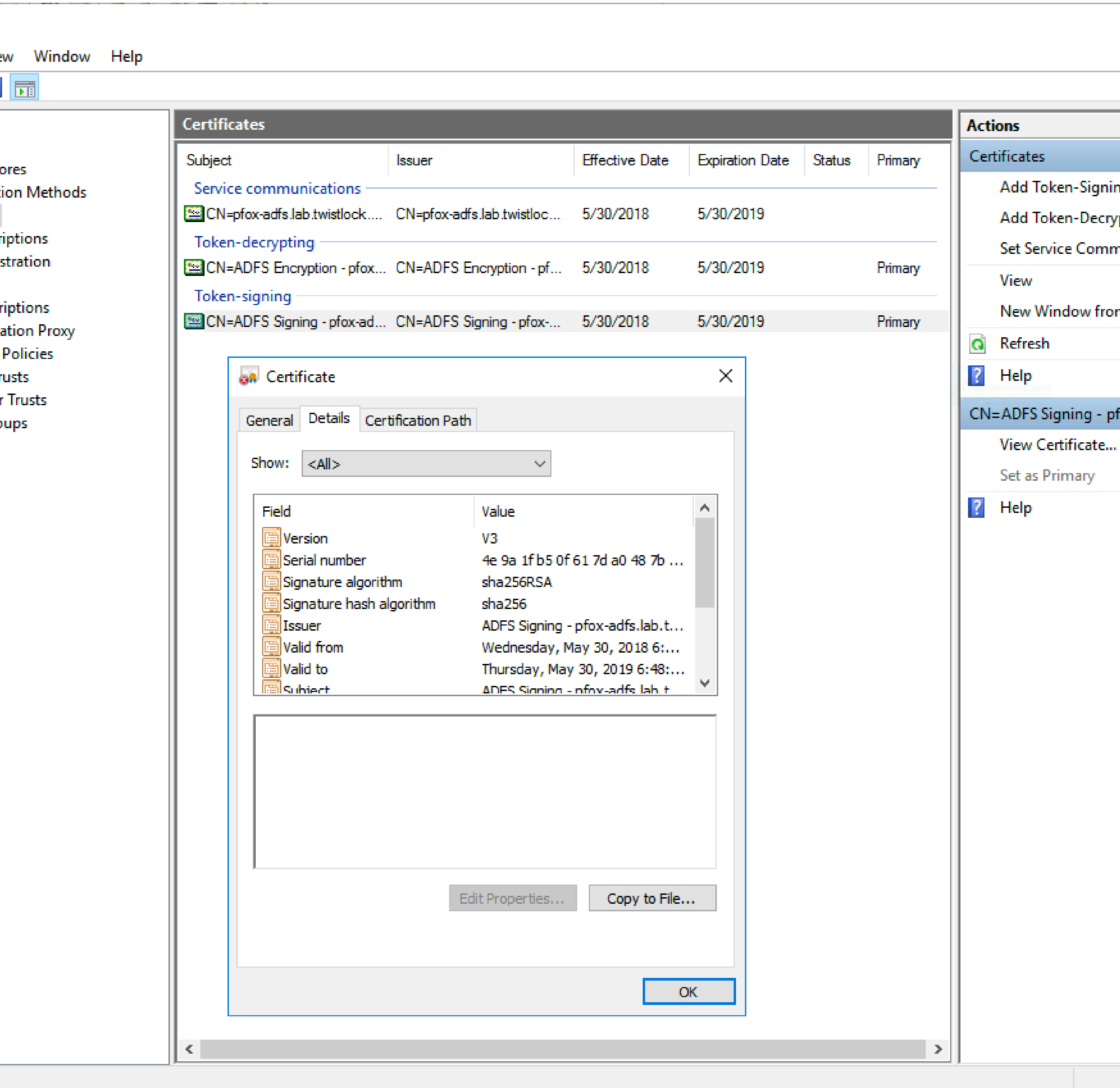
This guide assumes you have already deployed Active Directory Federation Services, and Active Directory is the claims provider for the service.

STEP 1 | Log onto your Active Directory Federation Services server.

STEP 2 | Go to **Server Manager > Tools > AD FS Management** to start the ADFS snap-in.

STEP 3 | Go to **AD FS > Service > Certificates** and click on the **Primary Token-signing** certificate.

STEP 4 | Select the Details tab, and click **Copy to File...**.

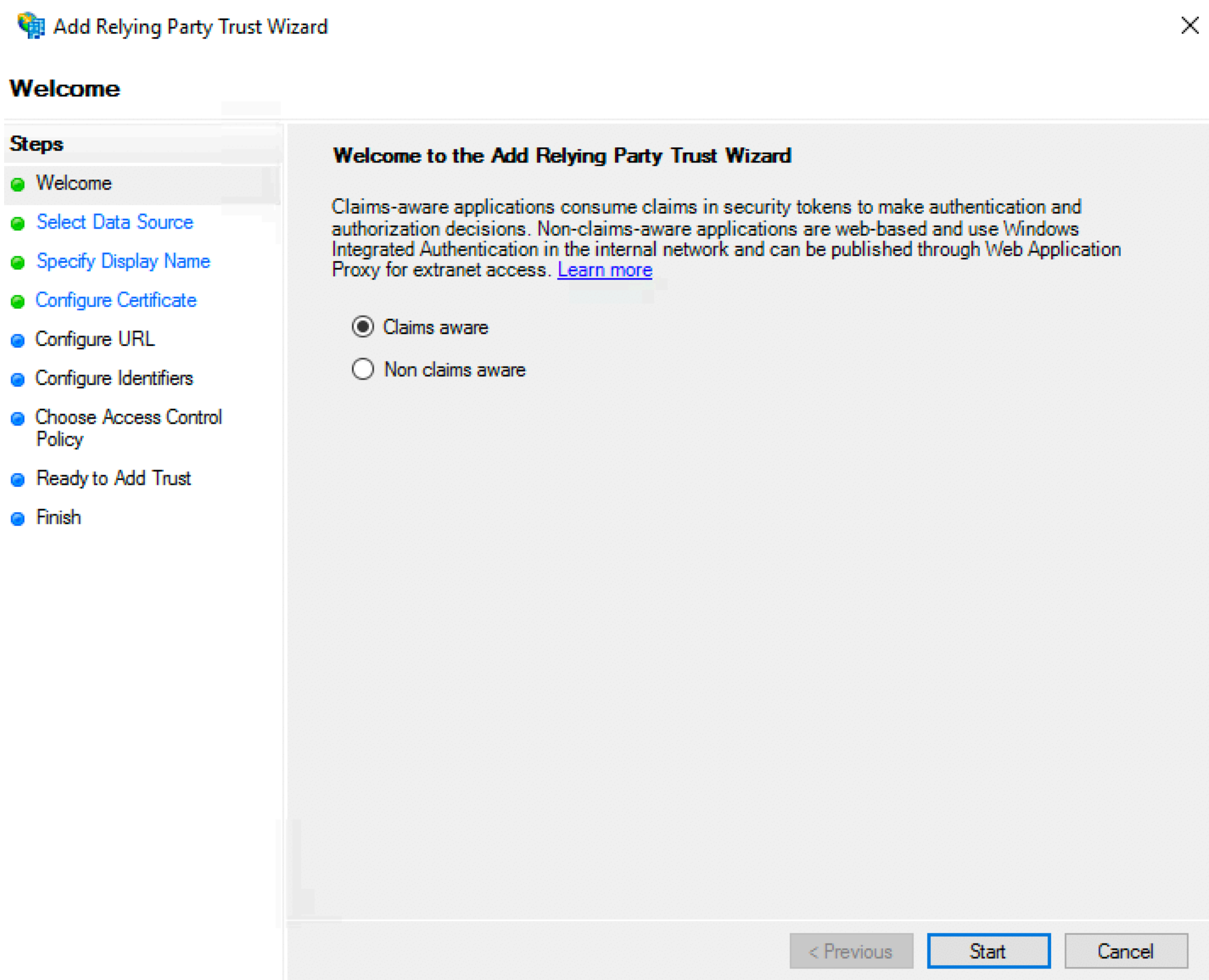


STEP 5 | Save the certificate as a Base-64 encoded X.509 (.CER) file. You will upload this certificate into the Prisma Cloud console in a later step.

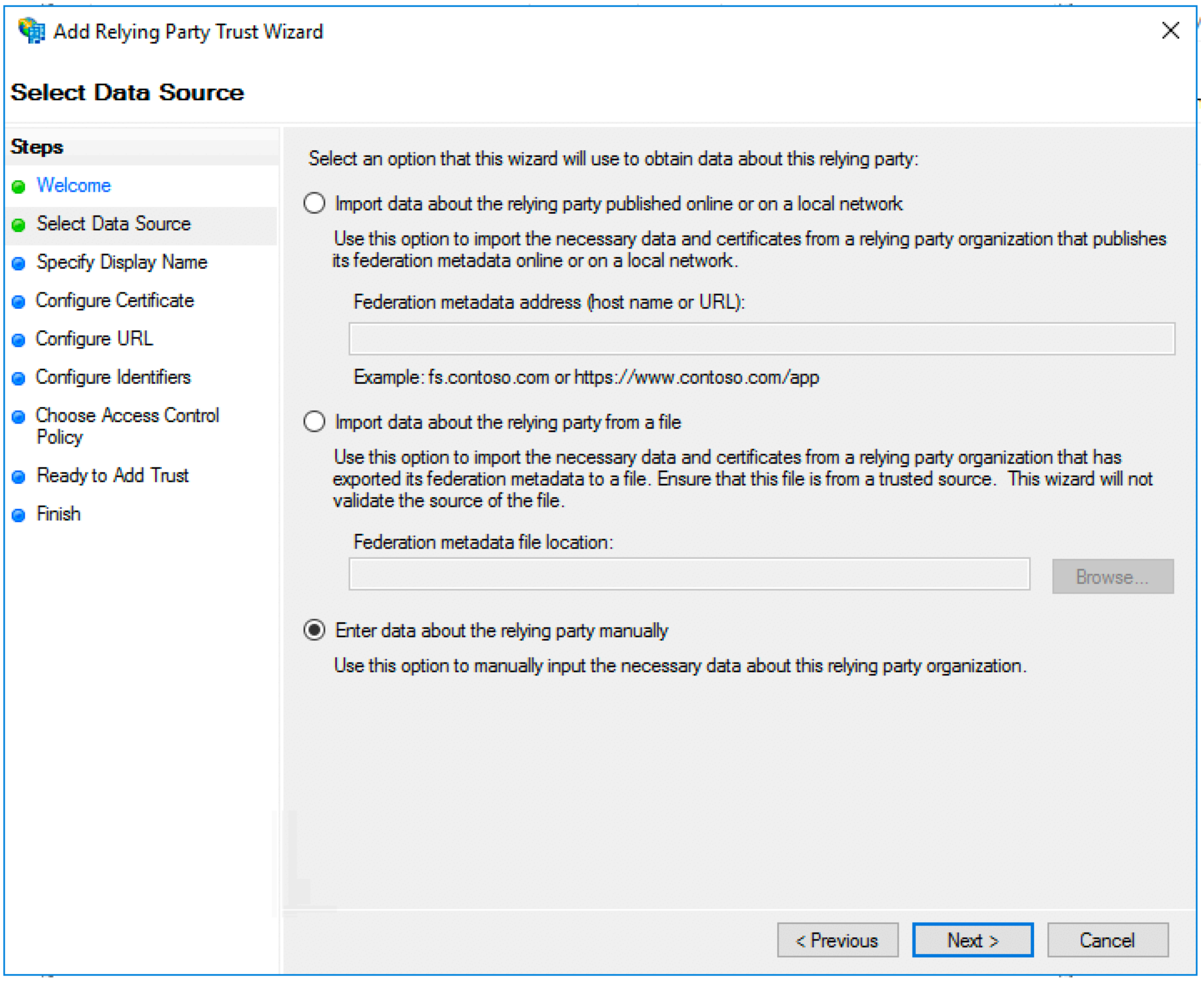
STEP 6 | Go to **AD FS > Relying Party Trusts**.

STEP 7 | Click **Add Relying Party Trust** from the **Actions** menu.

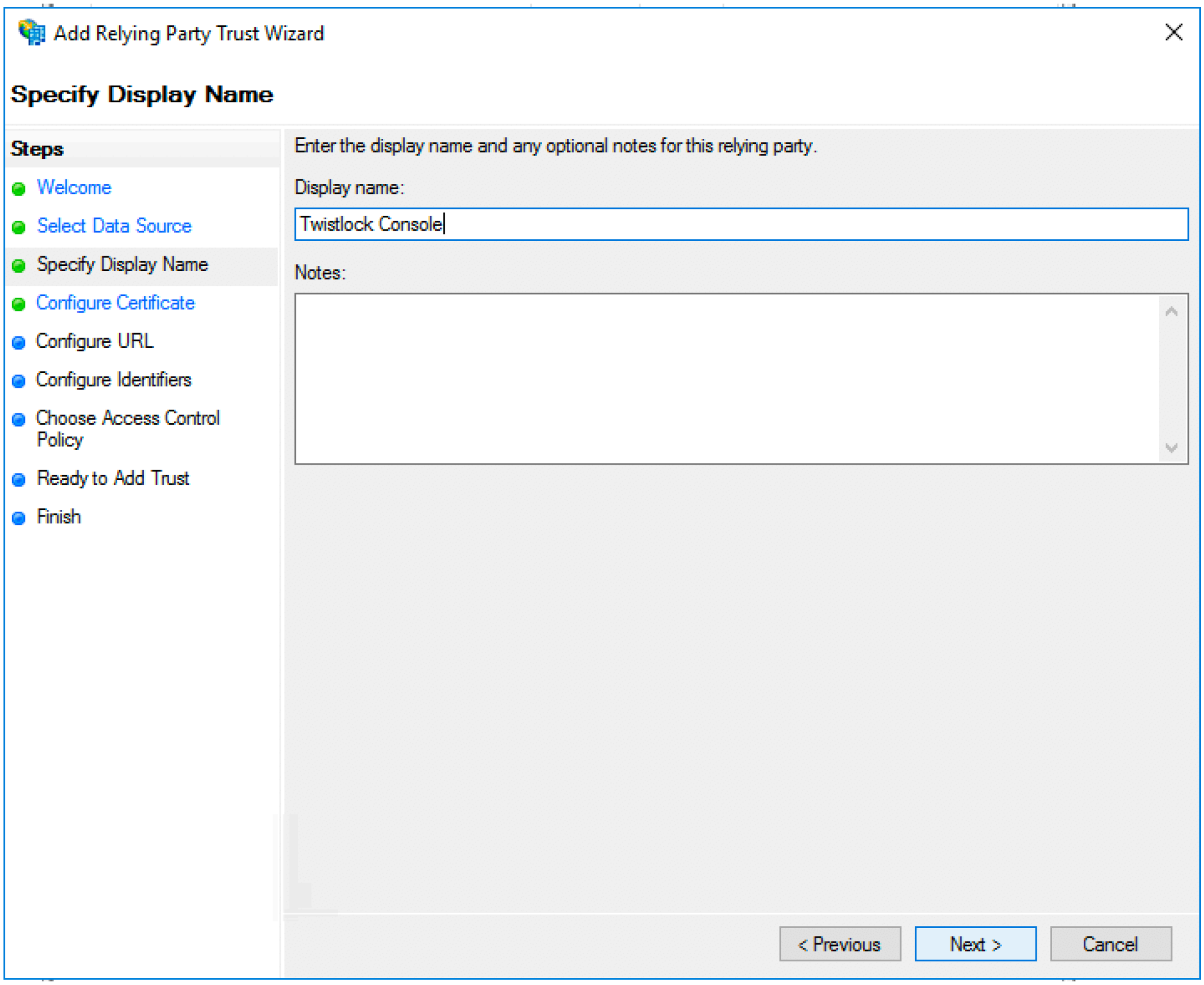
1. Step Welcome: select **Claims aware**.



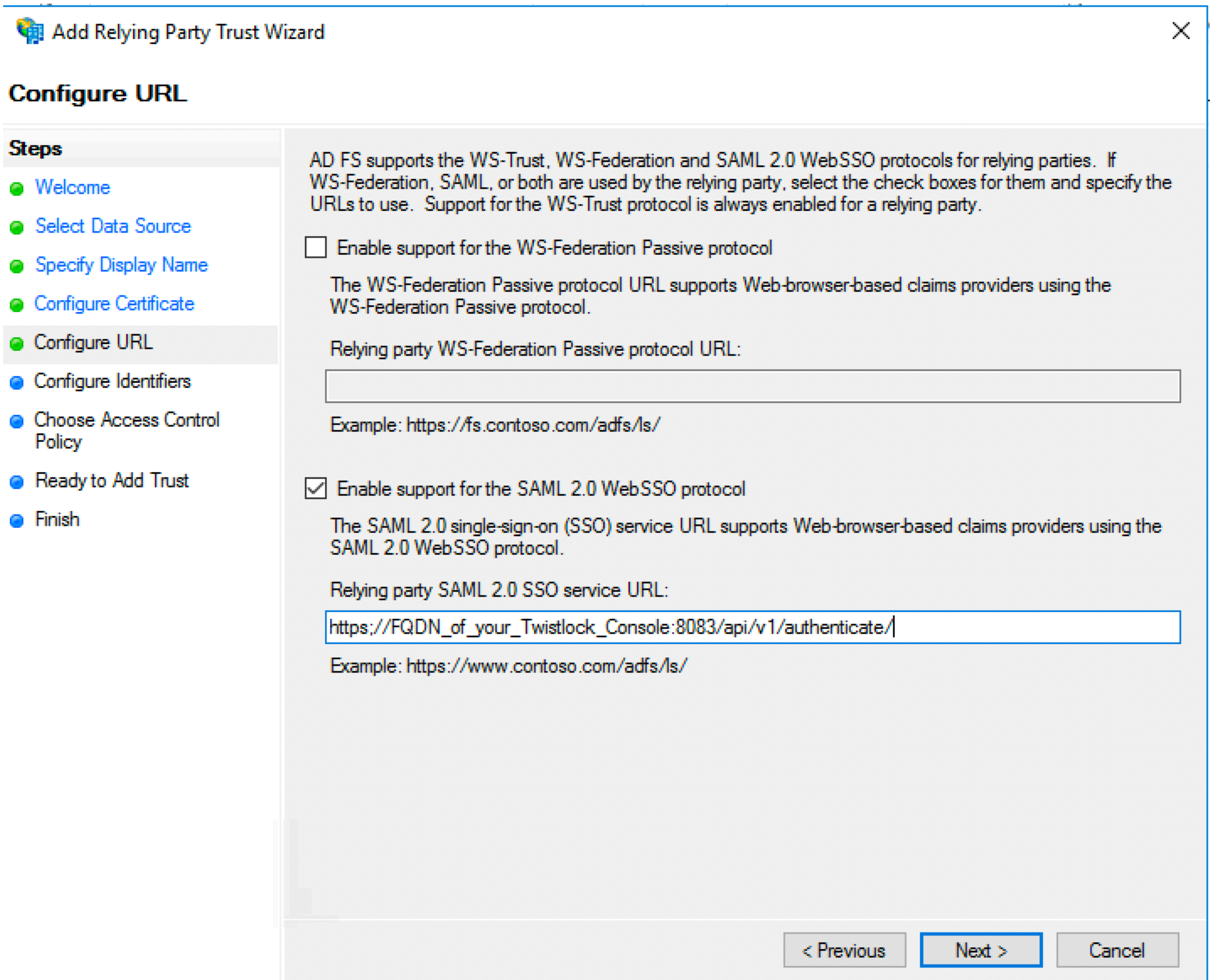
2. Step Select Data Source: select **Enter data about the relying party manually**.



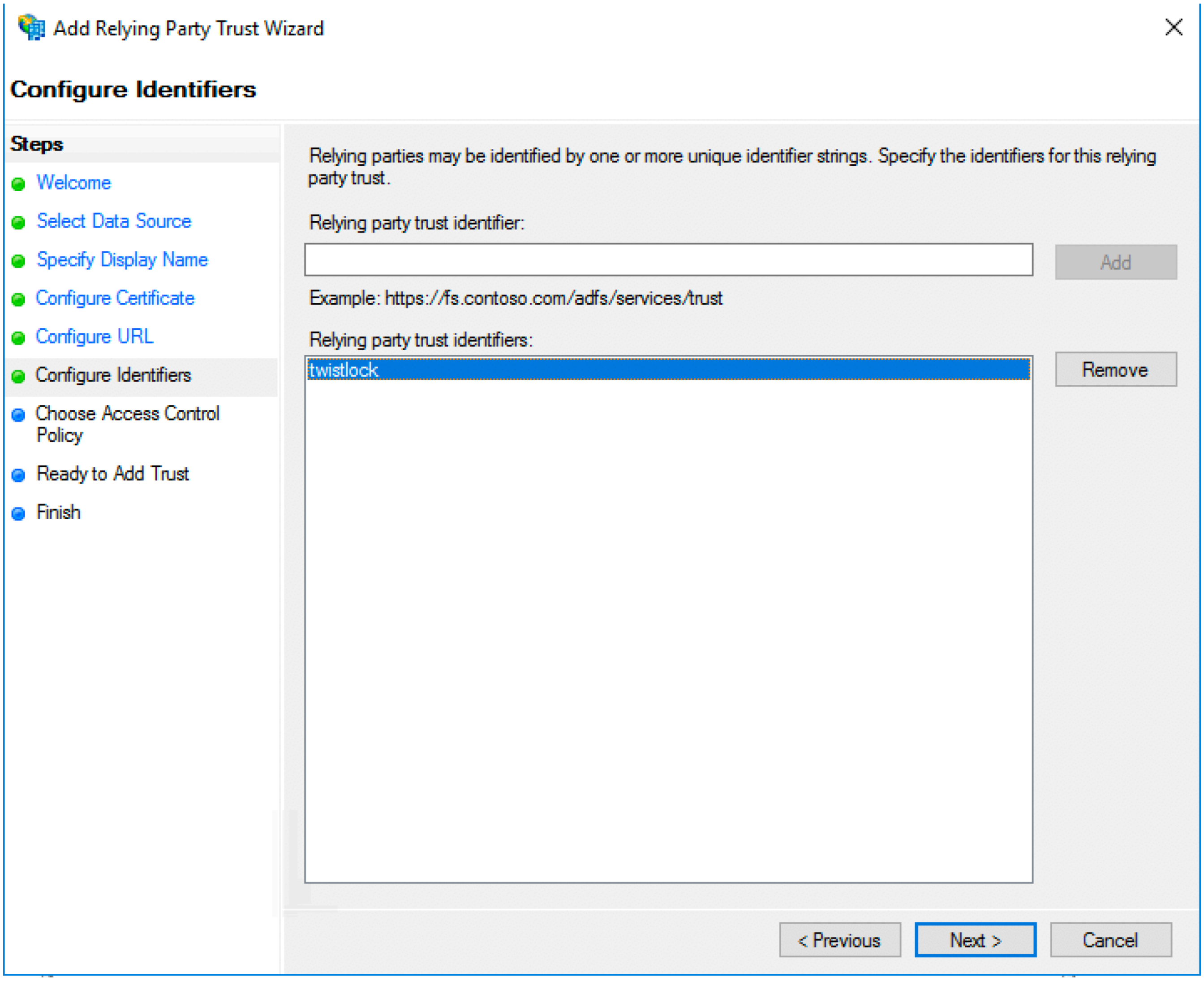
3. Step Specify Display Name: In **Display Name**, enter **twistlock Console**.



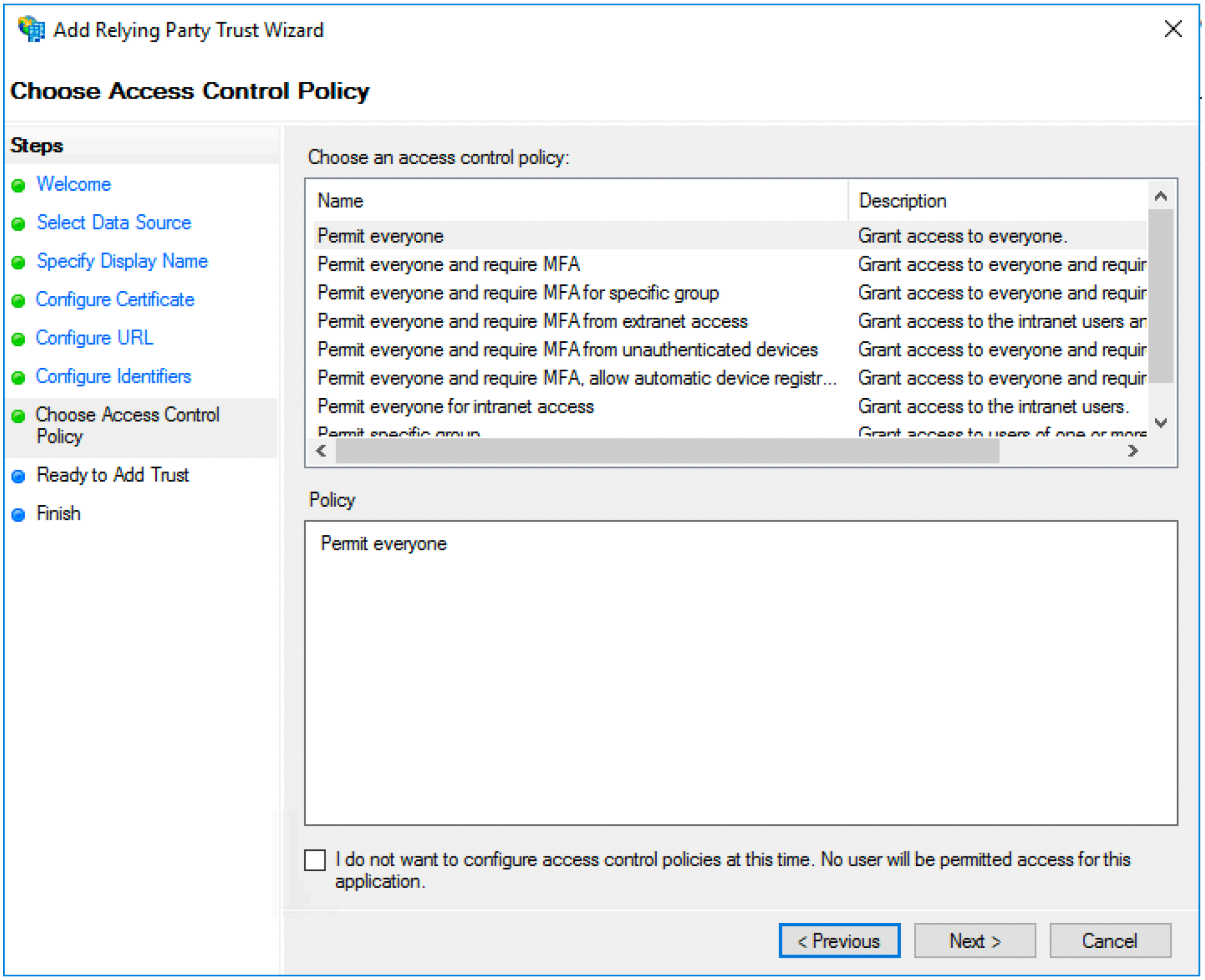
4. Step Configure Certificate: leave blank.
5. Step Configure URL: select **Enable support for the SAML 2.0 WebSSO protocol**. Enter the URL for your Prisma Cloud Console **https://<FQDN_TWISTLOCK_CONSOLE>:8083/api/v1/authenticate/**.



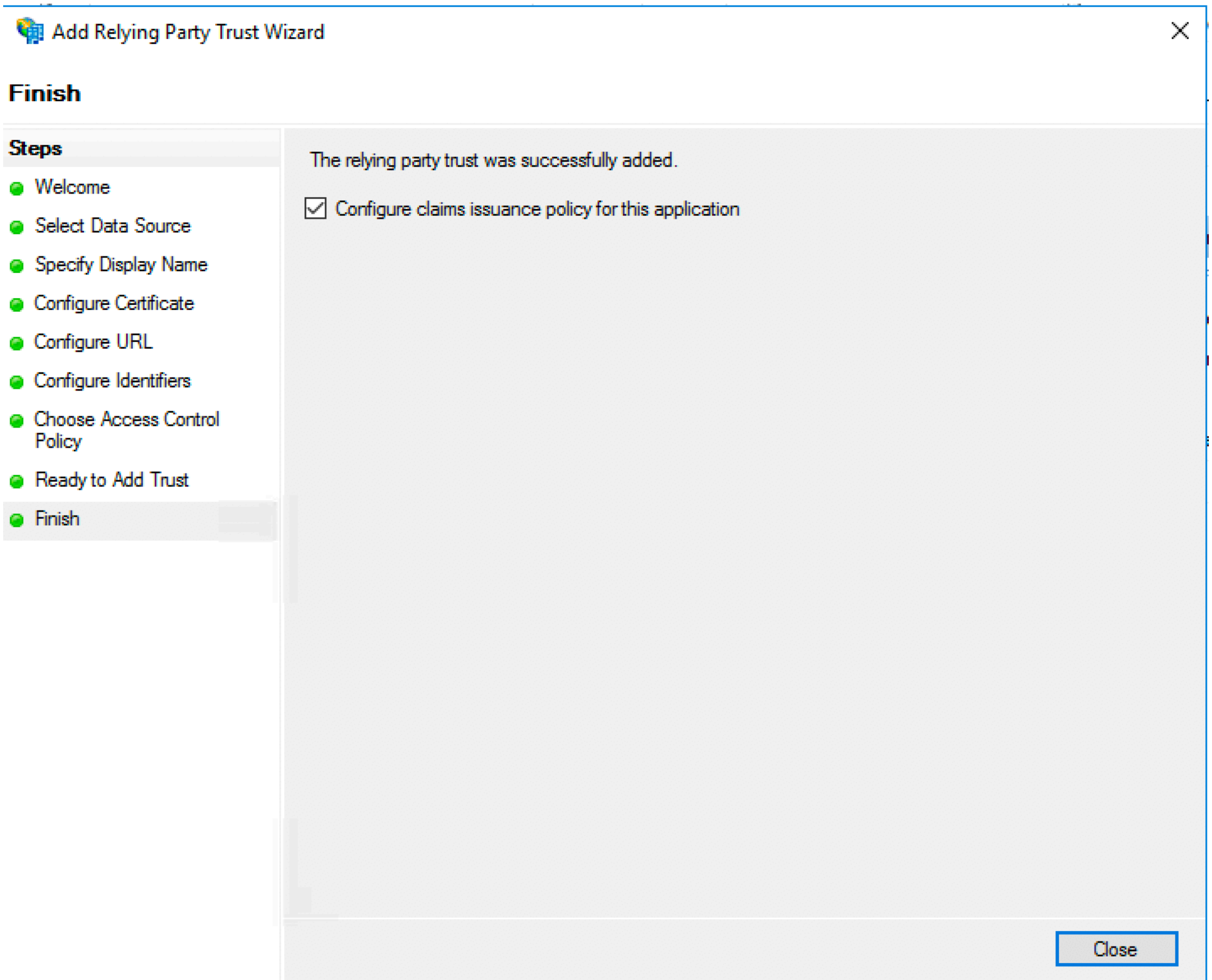
6. Step Configure Identifiers: for example enter **twistlock** all lower case and click **Add**.



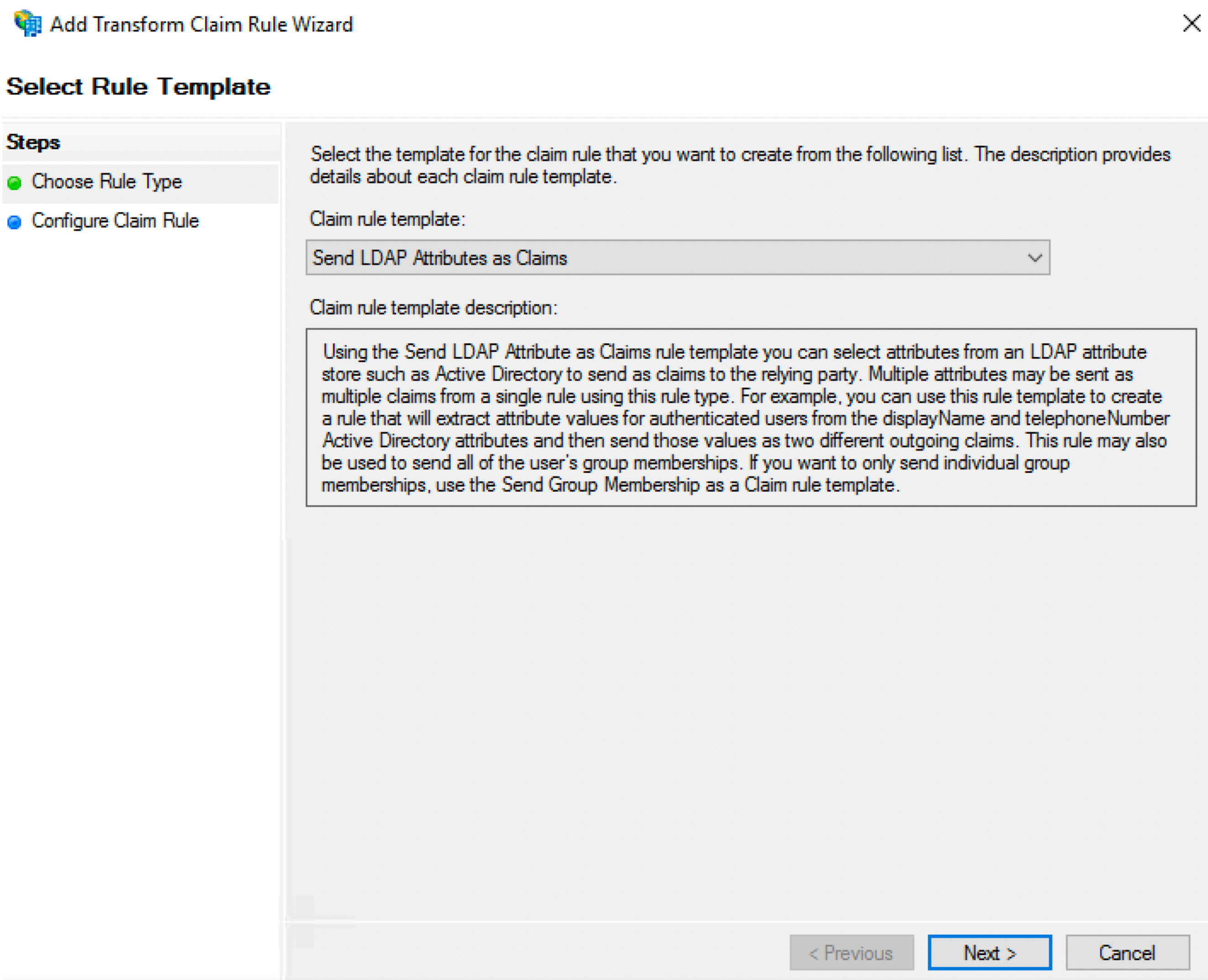
7. Step Choose Access Control Policy: this is where you can enforce multi-factor authentication for Prisma Cloud Console access. For this example, select **Permit everyone**.



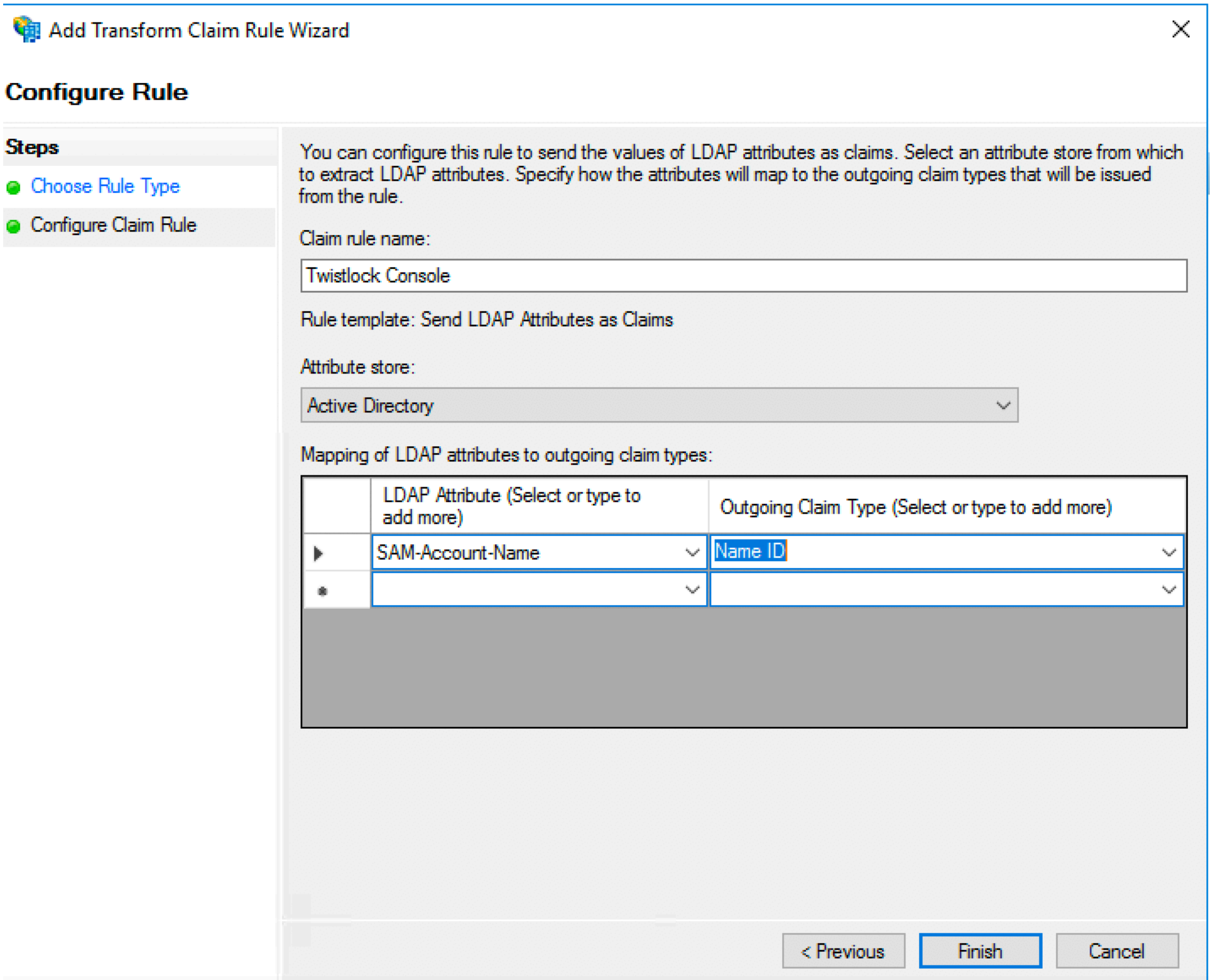
8. Step Ready to Add Trust: no changes, click **Next**.
9. Step Finish: select **Configure claims issuance policy for this application** then click **Close**.



10. In the Edit Claim Issuance Policy for Prisma Cloud Console click **Add Rule**.
11. Step Choose Rule Type: In **Claim rule template**, select **Send LDAP Attributes as Claims**.



12. Step Configure Claim Rule:
    - Set **Claim rule name** to **Prisma Cloud Console**
    - Set **Attribute Store** to **Active Directory**
    - In **Mapping of LDAP attributes to outgoing claim types**, set the **LDAP Attribute** to **SAM-Account-Name** and **Outgoing claim type** to **Name ID**.



The user's Active Directory attribute returned in the claim must match the Prisma Cloud user's name. In this example we are using the samAccountName attribute.

13. Click **Finish**.

STEP 8 | Configure ADFS to either sign the SAML response (*-SamlResponseSignature MessageOnly*) or the SAML response and assertion (*-SamlResponseSignature MessageAndAssertion*) for the Prisma Cloud Console relying party trust. For example to configure the ADFS to only sign the response, start an administrative PowerShell session and run the following command:

```
set-adfsrelyingpartytrust -TargetName "Prisma Cloud Console" -
SamlResponseSignature MessageOnly
```

Active Directory group membership within SAML response

You can use Active Directory group membership to assign users to Prisma Cloud roles. When a user's group membership is sent in the SAML response, Prisma Cloud attempts to associate the user's group to a Prisma Cloud role. If there is no group association, Prisma Cloud matches the user to an identity based on the NameID to Prisma Cloud username mapping. The SAML group to Prisma Cloud role association *does not require* the creation of a Prisma Cloud user. Therefore simplify the identity management required for your implementation of Prisma Cloud.

STEP 1 | In **Relying Party Trusts**, select the **Prisma Cloud Console** trust.

STEP 2 | Click **Edit Claim Issuance Policy** in the right hand **Actions** pane.

STEP 3 | Click **Add Rule**.

STEP 4 | *Claim rule template*: **Send Claims Using a Custom Rule**.

STEP 5 | Click **Next**.

STEP 6 | *Claim rule name:* **Prisma Cloud Groups**.

STEP 7 | Paste the following claim rule into the *Custom rule* field:

```
c:[Type == "http://schemas.microsoft.com/ws/2008/06/identity/
claims/windowsaccountname", Issuer == "AD AUTHORITY"] =>
 issue(store = "Active Directory", types = ("groups"), query =
 ";tokenGroups;{0}", param = c.Value);
```

Configure the Prisma Cloud Console

Configure the Prisma Cloud Console.

STEP 1 | Login to the Prisma Cloud Console as an administrator.

STEP 2 | Go to **Manage** > **Authentication** > **Identity Providers** > **SAML**.

STEP 3 | Set **Integrate SAML users and groups with Prisma Cloud** to **Enabled**.

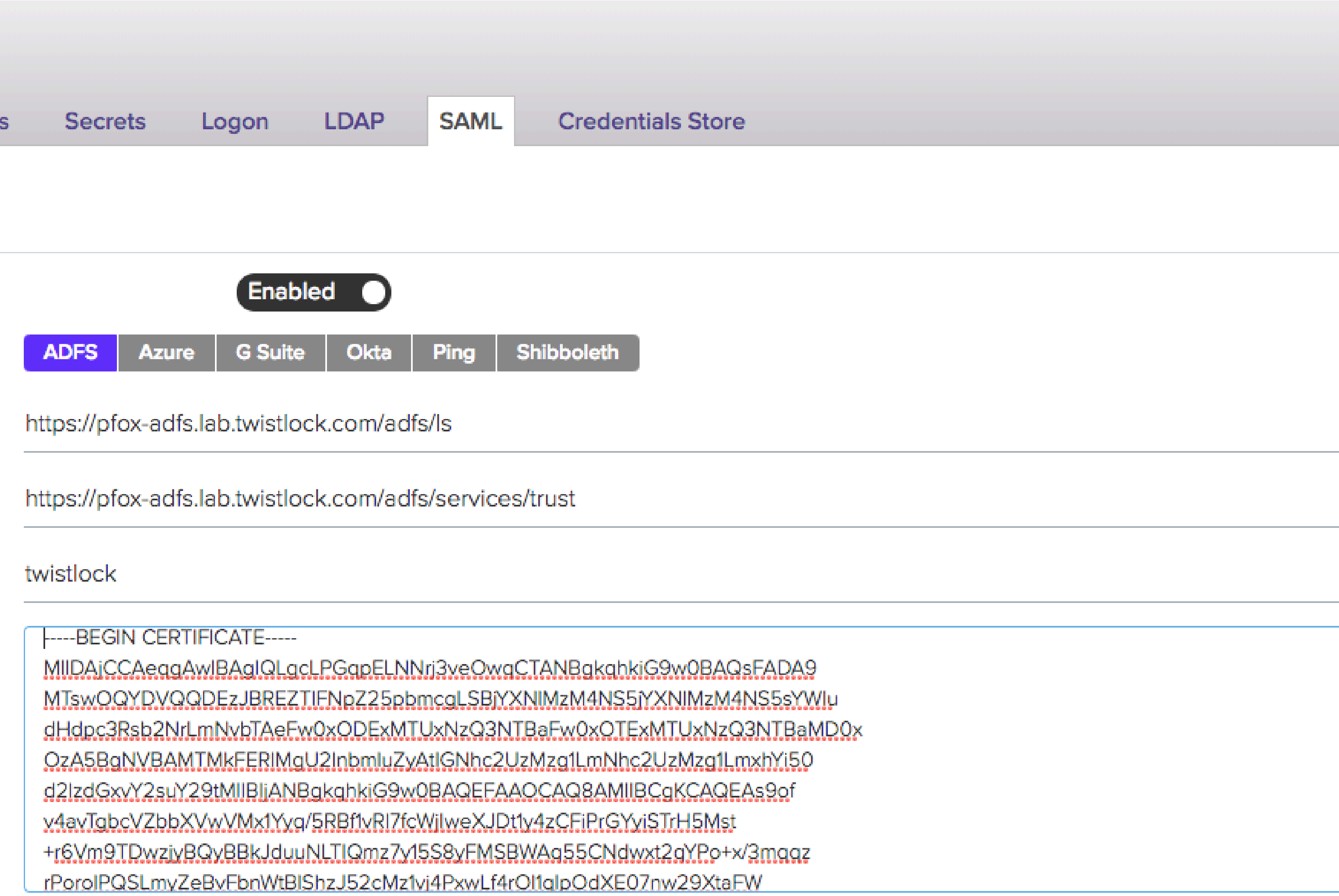
STEP 4 | Set **Identity Provider** to **ADFS**.

STEP 5 | In **Identity provider single sign-on URL**, enter your SAML Single Sign-On Service URL. For example **https://FQDN_of_your_adfs/adfs/ls**.

STEP 6 | In **Identity provider issuer**, enter your SAML Entity ID, which can be retrieved from **ADFS > Service > Federation Service Properties : Federation Service Identifier**.

STEP 7 | In **Audience**, enter the ADFS Relying Party identifier **twistlock**

STEP 8 | In **X.509 certificate**, paste the ADFS **Token Signing Certificate Base64** into this field.



STEP 9 | Click **Save**.

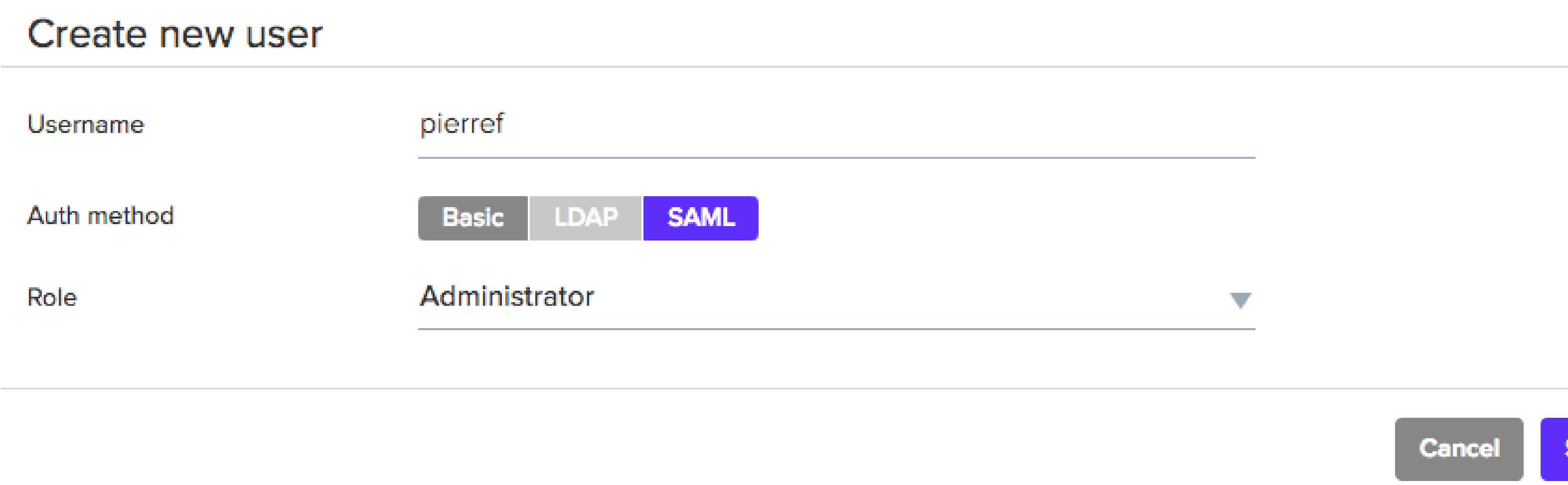
STEP 10 | Go to **Manage > Authentication > Users**.

STEP 11 | Click **Add user**.

1. **Username**: Active Directory *samAccountName* must match the value returned in SAML token's Name ID attribute.

   When federating with ADFS Prisma Cloud usernames are case insensitive. All other federation IdPs are case sensitive.

2. **Auth method**: set to **SAML**.



3. **Role**: select an appropriate role.

STEP 12 | Click **Save**.

Active Directory group membership mapping to Prisma Cloud role

Associate a user's Active Directory group membership to a Prisma Cloud role.

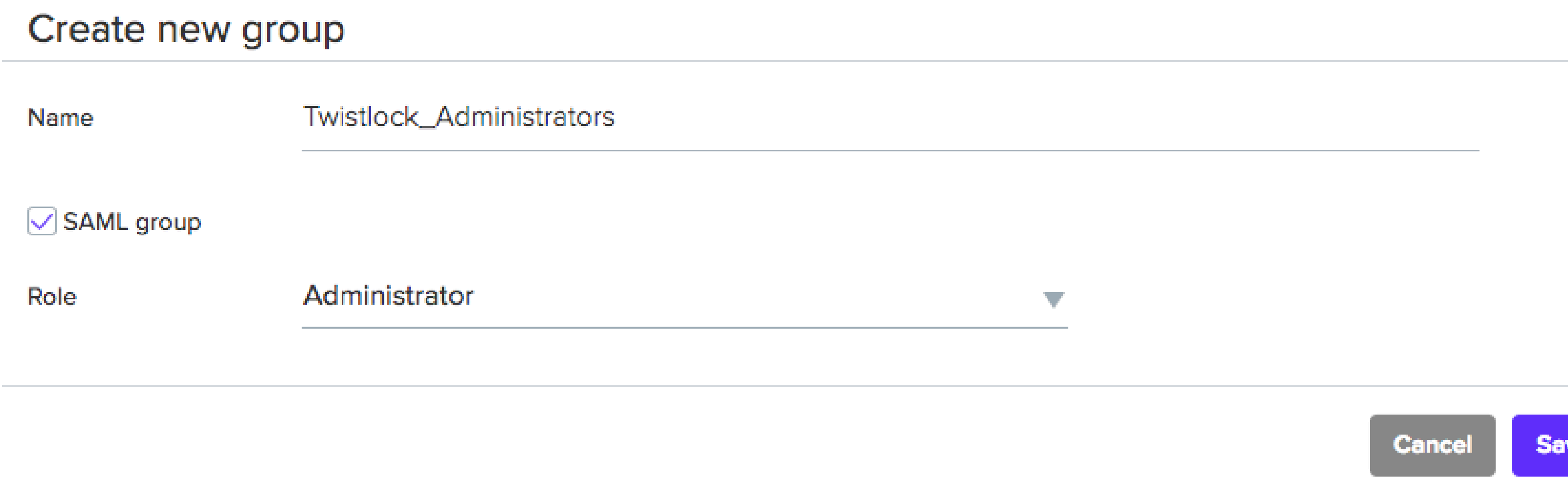
STEP 1 | Go to **Manage > Authentication > Groups**.

STEP 2 | Click **Add group**.

STEP 3 | *Group Name* matches the **Active Directory group name**.

STEP 4 | Select the **SAML group** radio button.

STEP 5 | Assign the **Role**.



The SAML group to Prisma Cloud role association does not require the creation of a Prisma Cloud user.

STEP 6 | Test login into the Prisma Cloud Console via ADFS SAML federation.

Leave your existing session logged onto the Prisma Cloud Console in case you encounter issues. Open a new incognito browser window and go to https://<CONSOLE>:8083.

Integrate Prisma Cloud with GitHub

Edit on GitHub

Prisma Cloud supports OAuth 2.0 as an authentication mechanism. GitHub users can log into Prisma Cloud Console using GitHub as an OAuth 2.0 provider.

Prisma Cloud supports the authorization code flow only.

Configure Github as an OAuth provider

Create an OAuth App in your GitHub organization so that users in the organization can log into Prisma Cloud using GitHub as an OAuth 2.0 provider.

STEP 1 | Log into GitHub as the organization owner.

STEP 2 | Go to **Settings > Developer Settings > OAuth Apps**, and click **New OAuth App** (or **Register an application** if this is your first app).

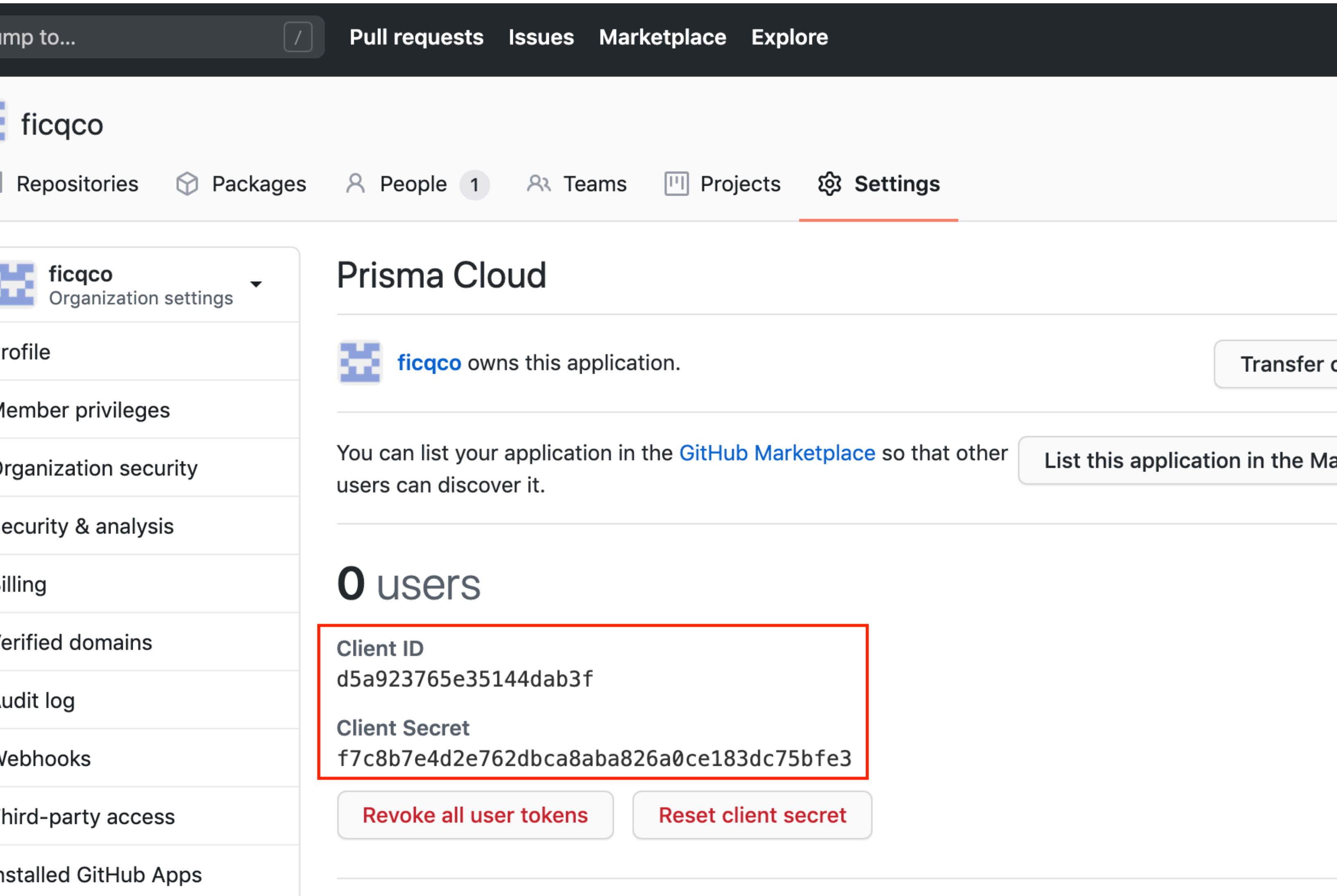
STEP 3 | In **Application name**, enter **Prisma Cloud**.

STEP 4 | In **Homepage URL**, enter the URL for Prisma Cloud Console in the format https://<CONSOLE>:<PORT>.

STEP 5 | In **Authorization callback URL**, enter https://<CONSOLE>:<PORT>/api/v1/authenticate/callback/oauth.

STEP 6 | Click **Register application**.

STEP 7 | Copy the **Client ID** and **Client Secret**, and set them aside setting up the integration with Prisma Cloud.



Integrate Prisma Cloud with GitHub

Set up the integration so that GitHub users from your organization can log into Prisma Cloud.

STEP 1 | Log into Prisma Cloud Console.

STEP 2 | Go to **Manage > Authentication > Identity Providers > OAuth 2.0**.

STEP 3 | Set **Integrate Oauth 2.0 users and groups with Prisma Cloud** to **Enabled**.

STEP 4 | Set **Identity provider** to **GitHub**.

STEP 5 | Set **Client ID** and **Client secret** to the values you copied from GitHub.

STEP 6 | Set **Auth URL** to **https://github.com/login/oauth/authorize**.

STEP 7 | Set **Token URL** to **https://github.com/login/oauth/access_token**.

STEP 8 | Click **Save**.

Prisma Cloud to GitHub user identity mappings

Create a Prisma Cloud user for each GitHub user that should have access to Prisma Cloud.

After the user is authenticated, Prisma Cloud uses the access token to query GitHub for the user's information (user name, email). The user information returned from GitHub is compared against the information in the Prisma Cloud Console database to determine if the user is authorized. If so, a JWT token is returned.

STEP 1 | Go to **Manage > Authentication > Users**.

STEP 2 | Click **Add User**.

STEP 3 | Set **Username** to the GitHub user name.

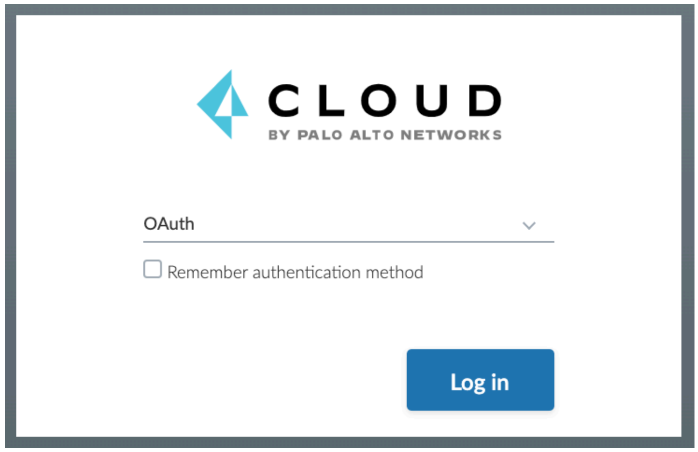
STEP 4 | Set **Auth method** to **OAuth**.

STEP 5 | Select a role for the user.

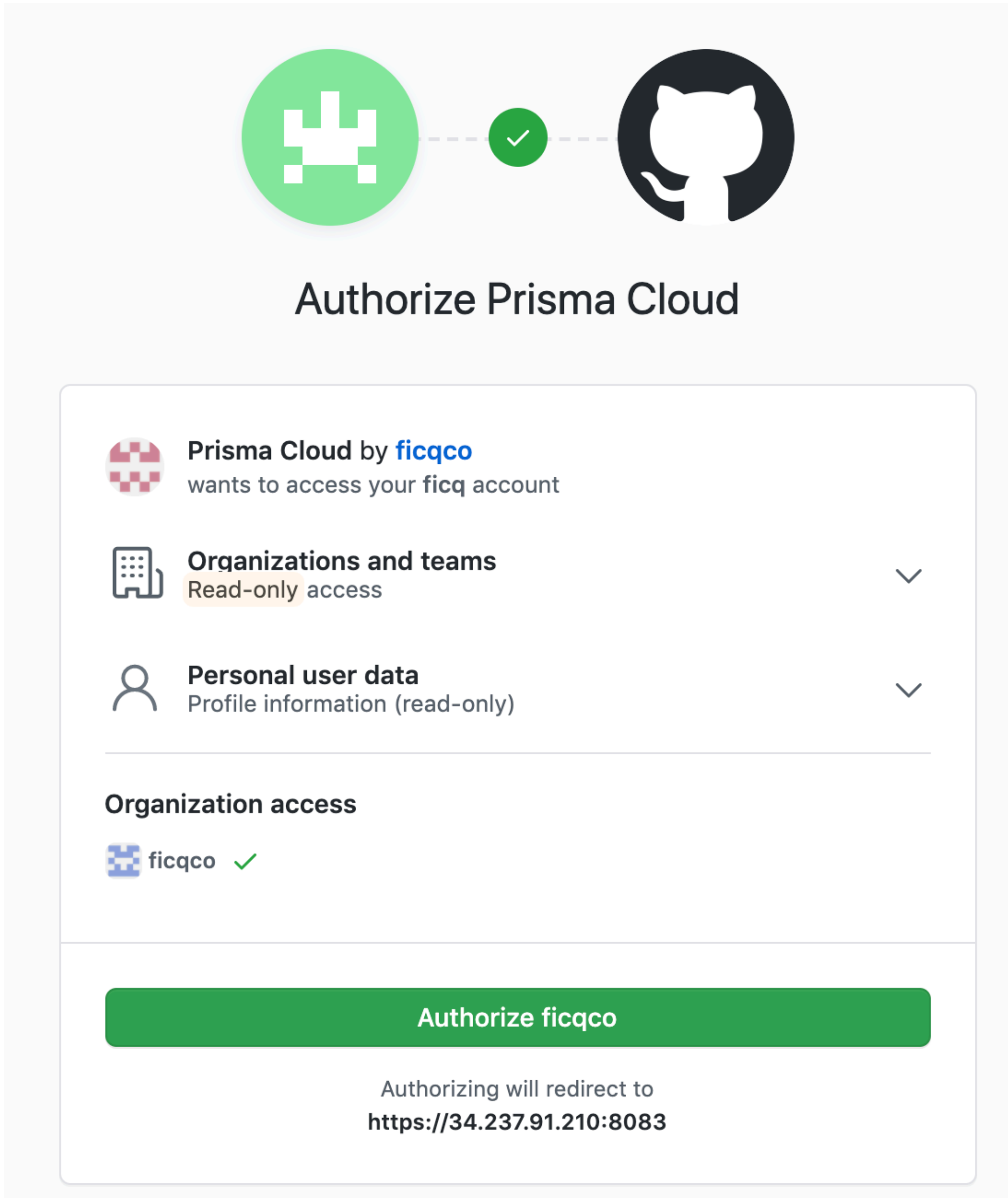
STEP 6 | Click **Save**.

STEP 7 | Test logging into Prisma Cloud Console.

1. Logout of Prisma Cloud.
2. On the login page, select **OAuth**, and then click **Login**.



3. Authorize the Prisma Cloud OAuth App to sign you in.



Prisma Cloud group to GitHub organization mappings

Use groups to streamline how Prisma Cloud roles are assigned to users. When you use groups to assign roles, you don't have to create individual Prisma Cloud accounts for each user.

Groups can be associated and authenticated with by multiple identity providers.

STEP 1 | Go to **Manage > Authentication > Groups**.

STEP 2 | Click **Add Group**.

STEP 3 | In **Name**, enter the the GitHub organization.

STEP 4 | In **Authentication method**, select **External Providers**.

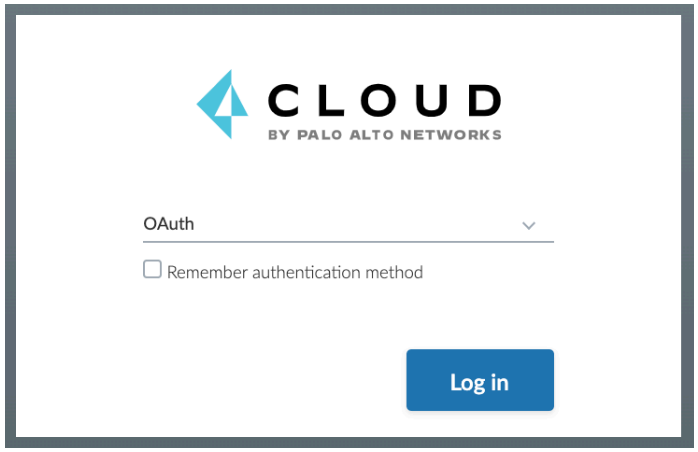
STEP 5 | In **Authentication Providers**, select **OAuth group**.

STEP 6 | Select a role for the members of the organization.

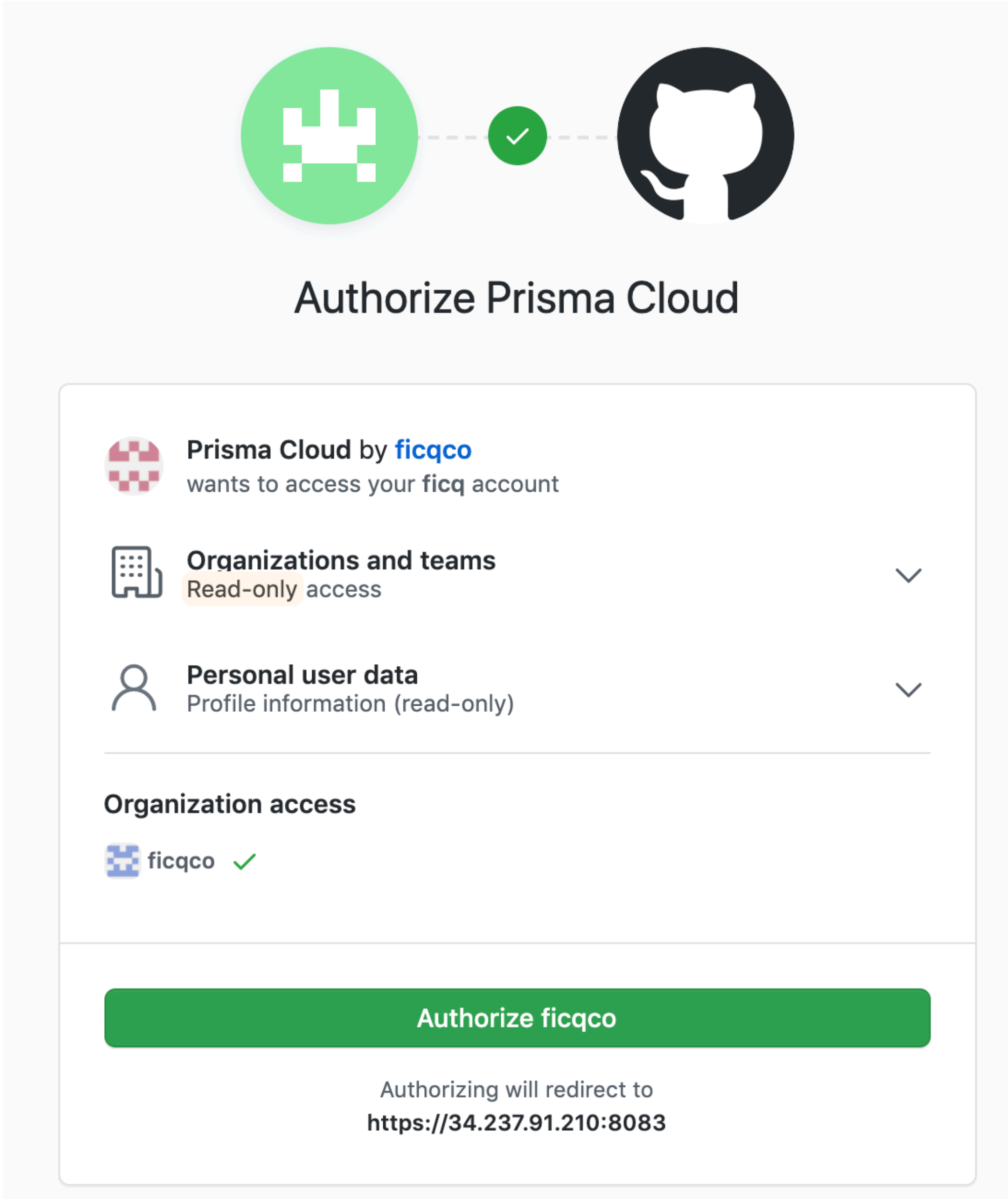
STEP 7 | Click **Save**.

STEP 8 | Test logging into Prisma Cloud Console.

1. Logout of Prisma Cloud.
2. On the login page, select **OAuth**, and then click **Login**.



3. Authorize the Prisma Cloud OAuth App to sign you in.


Authorize Prisma Cloud
Prisma Cloud by ficqco
wants to access your ficq account
Organizations and teams
Read-only access
Personal user data
Profile information (read-only)
Organization access
ficqco
Authorize ficqco
Authorizing will redirect to
https://34.237.91.210:8083

Integrate Prisma Cloud with OpenShift

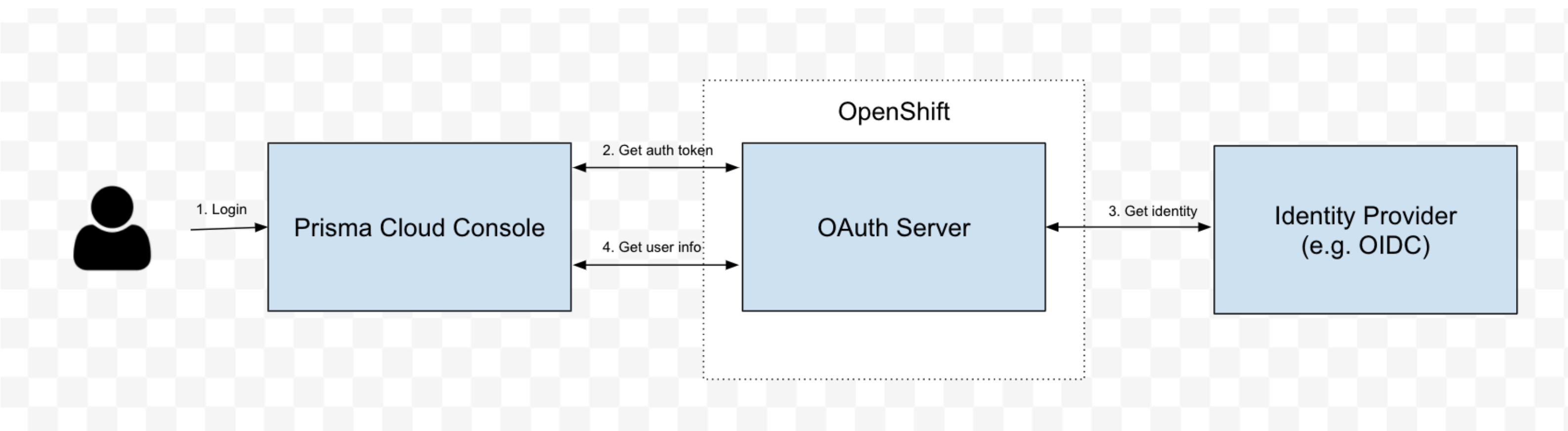
Edit on GitHub

OpenShift users can log into Prisma Cloud Console using OpenShift as an OAuth 2.0 provider.

Prisma Cloud currently supports OpenShift Platform versions 4.5 and older as an OAuth 2.0 provider. We are working to add support for OpenShift versions 4.6 and later.

The OpenShift master includes a built-in OAuth server. You can integrate OpenShift authentication into Prisma Cloud. When users attempt to access Prisma Cloud, which is a protected resource, they are redirected to authenticate with OpenShift. After authenticating successfully, they are redirected back to Prisma Cloud Console with an OAuth token. This token scopes what the user can do in OpenShift. Prisma Cloud only needs the auth token to get the user’s info (e.g. user name, email), and check the Prisma Cloud database to see if this user is authorized. If so, Prisma Cloud creates a JWT token, with a role claim, to complete the authentication process to Console. Roles are assigned based on users and group information specified in Console.

The following diagram shows the login flow when the auth provider is LDAP. With LDAP, users enter their credentials in Prisma Cloud Console, and Prisma authenticates with the LDAP server on the user’s behalf. With all other auth providers, Prisma isn’t part of verifying the user credentials Instead Prisma redirects the client to the auth provider for authentication. Once the user successfully authenticates via the authentication provider, the client is redirected back to Prisma Cloud Console with an object (SAML assertion for SAML, JWT token for OIDC, Access token for OAuth 2.0) that proves a successful login or, in the OAuth 2.0 case, gives us access to the application to verify the user identity.



Prisma Cloud supports the authorization code flow only.

Integrate Prisma Cloud with OpenShift

Configure Prisma Cloud so that OpenShift users can log into Prisma Cloud with the same identity.

STEP 1 | In OpenShift, register Prisma Cloud as an OAuth client. Set the redirect URL to:

https://<CONSOLE>:<PORT>/api/v1/authenticate/callback/oauth.

STEP 2 | Log into Prisma Cloud Console.

STEP 3 | Go to **Manage > Authentication > Identity Providers > OAuth 2.0**.

STEP 4 | Set **Integrate Oauth 2.0 users and groups with Prisma Cloud** to **Enabled**.

STEP 5 | Set **Identity provider** to **OpenShift**.

STEP 6 | Set **Client ID** to the **name** of the OAuth client you set up in OpenShift.

STEP 7 | Set **Client secret** to the **secret** in the OAuth client you set up in OpenShift.

STEP 8 | Set **Auth URL** to **https://github.com/login/oauth/authorize**.

STEP 9 | Set **Token URL** to **https://github.com/login/oauth/access_token**.

STEP 10 | In **User Info API URL**, enter the TCP endpoint for the OpenShift API server. For example, **https://openshift.default.svc.cluster.local**.

STEP 11 | Click **Save**.

Prisma Cloud to OpenShift user identity mappings

Create a Prisma Cloud user for every OpenShift user that should have access to Prisma Cloud.

After the user is authenticated, Prisma Cloud uses the access token to query OpenShift for the user's information (user name, email). The user information returned from OpenShift is compared against the Prisma Cloud Console database to determine if the user is authorized. If so, a JWT token is returned.

STEP 1 | Go to **Manage > Authentication > Users**.

STEP 2 | Click **Add User**.

STEP 3 | Set **Username** to the OpenShift user name.

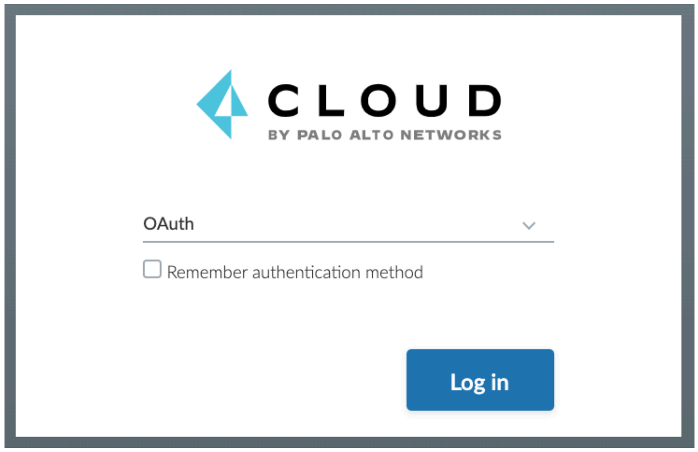
STEP 4 | Set **Auth method** to **OAuth**.

STEP 5 | Select a role for the user.

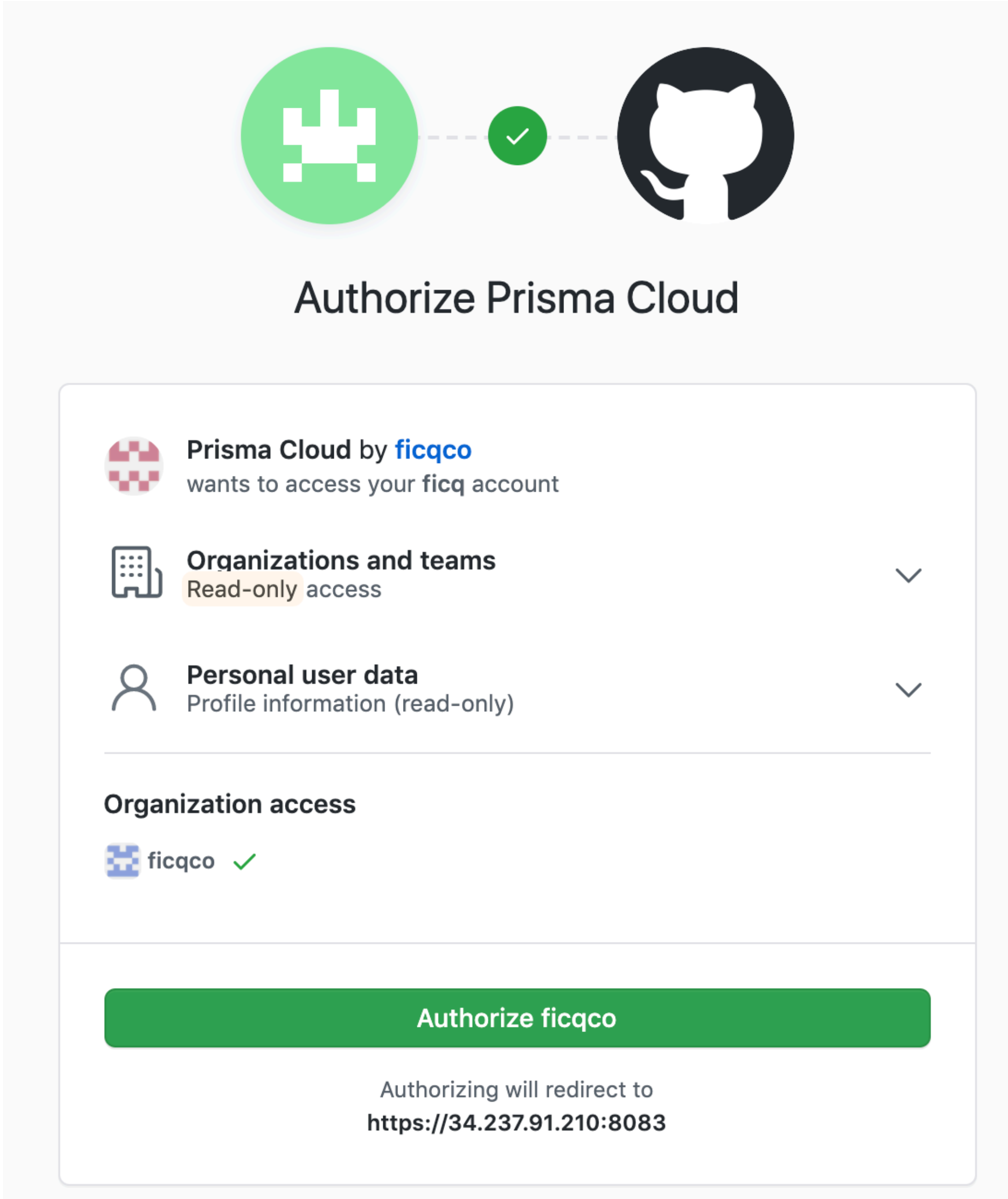
STEP 6 | Click **Save**.

STEP 7 | Test logging into Prisma Cloud Console.

1. Logout of Prisma Cloud.
2. On the login page, select **OAuth**, and then click **Login**.



3. Authorize the Prisma Cloud OAuth App to sign you in.



Prisma Cloud to OpenShift group mappings

Use groups to streamline how Prisma Cloud roles are assigned to users. When you use groups to assign roles, you don't have to create individual Prisma Cloud accounts for each user.

Groups can be associated and authenticated with by multiple identity providers.

STEP 1 | Go to **Manage > Authentication > Groups**.

STEP 2 | Click **Add Group**.

STEP 3 | In **Name**, enter an OpenShift group name.

STEP 4 | In **Authentication method**, select **External Providers**.

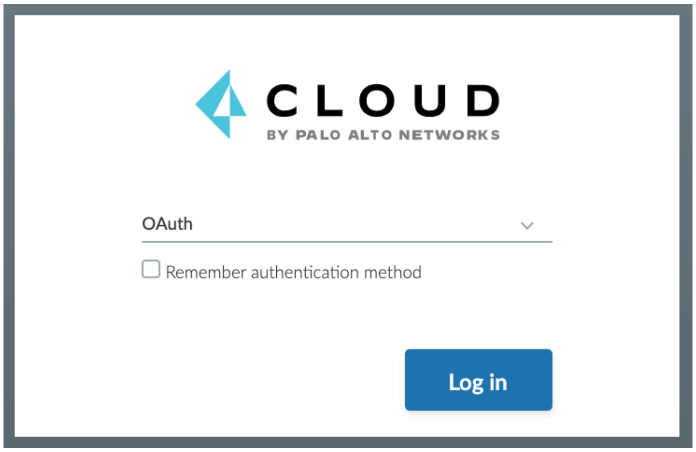
STEP 5 | In **Authentication Providers**, select **OAuth group**.

STEP 6 | Select a role for the members of the group.

STEP 7 | Click **Save**.

STEP 8 | Test logging into Prisma Cloud Console.

1. Logout of Prisma Cloud.
2. On the login page, select **OAuth**, and then click **Login**.



3. Authorize the Prisma Cloud OAuth App to sign you in.

Non-default UPN suffixes

Edit on GitHub

Active Directory allows administrators to specify custom UPN suffixes that can be applied to user accounts. The default UPN suffix for a user account is the Domain Name System (DNS) domain name of the domain that contains the user account. Microsoft refers to this as the implicit UPN. Administrators may choose to add additional suffixes to shorten user names or provide consistent UPNs across a forest composed of multiple domains; these are known as explicit UPNs.

For example, if a domain is named domain.directory.company.com, the default UPN suffix would be domain.directory.company.com and users could logon with username@domain.directory.company.com. However, an admin may want to simplify this and provide an alternative UPN suffix like @company.com that would apply to all users across a forest. Users could then logon with this explicit UPN of username@company.com instead.

Within the directory service, the userPrincipalName attribute is updated to reflect whatever username + UPN suffix the administrator applies to a given account. In Windows systems, the implicit UPN can be used in addition to whatever explicit UPN may be set. However, for non-Windows LDAP systems, **the explicit UPN is the only valid UPN that can be used with the user object**.

Thus, understanding the UPN assigned to a user account is critical to Prisma Cloud integration with Active Directory. Even if the domain name and the search path may use one set of names (such as dc=domain,dc=directory,dc=company,dc=com in our above example), the actual (explicit) UPN must be used for all actions within Prisma Cloud, such as adding users to the system or logging on. From our above example, this means that if the user in Active Directory has a UPN of username@domain.directory.company.com set on their account, this UPN must be used with Prisma Cloud. Alternatively, if an Active Directory admin has set another UPN, such as username@company.com, that UPN must be used instead.

Any attempts to use a UPN not directly found in the userPrincipalName field on the user object will result in 'user not found' errors.

Compute user roles

Edit on GitHub

You can assign roles to users to control their level of access to Prisma Cloud. Roles determine what a user can do and see in Console, and the APIs he or she can access. Roles are enforced the same way for both the Prisma Cloud UI and API.

Prisma Cloud provides several pre-defined system roles you can assign to users and groups, as well as allows you to create your own customized roles.

Summary of system roles

The following table summarizes the system roles available in Prisma Cloud.

Role	Access level	Typical use case(s)
Administrator	Full read-write access to all Prisma Cloud settings and data.	Security administrators.
Operator	Read-write access to all rules and data. Read-only access to user and group management, role assignments, and the global settings under **Manage > System**.	Security operations teams.
Auditor	Read-only access to all Prisma Cloud rules and data.	Auditors and compliance staff that need to verify settings and monitor compliance.
DevSecOps User	Read-only access to all results under **Radar** and **Monitor**, but no access to change policy or settings. Read-only access to Utilities.	DevSecOps personnel.
Vulnerability Manager	Define policy and monitor vulnerabilities and compliance.	DevOps users that also need to define policy and monitor vulnerabilities and compliance.
DevOps User	Read-only access to the Prisma Cloud CI vulnerability, compliance scan reports, and Utilities.	Developer, Operations, and DevOps personnel that need to know about and/or address the vulnerabilities in your environment.
Defender Manager	Install, manage, and remove Defenders from your environment.	DevOps team members that need to manage Defender deployments without sysadmin privileges.

Role	Access level	Typical use case(s)
		Note: The permission groups you assign here only restrict access to what the user can do and see on the administrative Console. Defenders will collect and share information without differentiating which user deployed them.
Access User	Basic API routes only IMPORTANT: Access User role has permissions to some basic API routes only, and no access to the Console UI. This user role will be deprecated in the next release of Prisma Cloud Compute.	Developers (and others) that use the nodes that Prisma Cloud protects.
CI User	Run the Continuous Integration plugin only.	CI Users can only run the plugin and have no other access to configure Prisma Cloud.

Let's look at how two roles at the opposite end of the spectrum differ: Administrator and User. Administrators set the security policy. They decide who can run what Docker commands, and where they can be run. Users need to run Docker commands to do their job. Testers, for example, run Docker commands in the staging environment to test containers under development. Testers, however, have no business starting containers in the production environment. Administrators set a policy to assign testers the user role that lets testers run Docker commands in staging, but restricts their access to production.

System roles

This section describes the system roles Prisma Cloud supports.

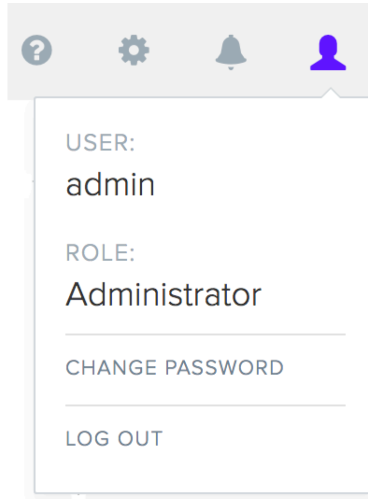
Administrator

The Administrator can manage all aspects of your Prisma Cloud installation. They have full read-write access to all Prisma Cloud settings and data.

Administrators can:

- Create and update security policies.
- Create and update access control policies.
- Create and update the list of users and groups that can access Prisma Cloud.
- Assign roles to users and groups.
- The Admin role is reserved for security administrators.

When Administrators log into Console, they have access to the full dashboard. If you click on the profile button on the top right of the dashboard, you get the details of the currently logged in user (admin) and associated role (Administrator).



Operator

Operators can create and update all Prisma Cloud settings. This role lets you view audit data and manage the rules that define your policies.

Operators cannot:

- Create, update, or delete users or groups.
- Assign or reassign roles to any user or group.
- Change the global settings under **Manage > System**.

The Operator role is designed for members of your Security Operations team.

Auditor

Auditors get read-only access to all Prisma Cloud data, settings, and logs.

Auditors are typically members of your compliance team. They verify that your Prisma Cloud setup meets your organization's security requirements. To verify compliance, they must be able to see your settings, but they do not need to make changes to them.

Auditors have access to the utilities page (**Manage > System > Utilities**).

DevSecOps User

DevSecOps Users get access to all views under **Radar** and **Monitor**. Access to the **Actions** menu in these views is disabled. The **Actions** menu lets you do things such as relearn models, protect services found by Cloud Discovery, and so on.

DevSecOps Users get read only access to vulnerabilities and compliance policies under **Defend**.

Under **Manage**, they only get access to **Manage > System > Utilities**. This page lets you download various Prisma Cloud components. DevSecOps Users can download all files, except Defender images, which are disabled for this role.

Vulnerability Manager

Vulnerability Managers define and monitor vulnerabilities and compliance policy. Vulnerability Managers gain the following permissions:

- Read-write access to **Defend > Vulnerabilities** and **Defend > Compliance**.
- Read-write access to **Monitor > Vulnerabilities**, **Monitor > Compliance** and **Monitor > Events > Trust Audits**.
- Read-only access to **Manage > System > Utilities**. The **Utilities** page lets you download various Prisma Cloud components. Vulnerability Managers can download all files, except Defender images, which are disabled for this role.

DevOps User

DevOps Users get read-only access to the **Jenkins Jobs** and **Twistcli Scans** tabs under **Monitor > Vulnerabilities** and **Monitor > Compliance**. Each tab contains scan reports for images and serverless functions scanned using these tools. DevOps Users can use Prisma Cloud scan reports and tools, for example, to determine why the CI/CD pipeline is stalled.

DevOps Users get read only access to vulnerabilities and compliance policies under **Defend**.

Under **Manage**, they only get access to **Manage > System > Utilities**. This page lets you download various Prisma Cloud components. DevOps Users can download all files, except Defender images, which are disabled for this role.

Defender Manager

Defender Managers get read-write access to **Manage > Defenders** and **Manage > System > Utilities**.

Defender Managers can install, manage, and remove Defenders from your environment. The Defender Manager role was designed to let members of your DevOps team manage the hosts that Prisma Cloud protects without requiring Administrator-level privileges. To help debug Defender deployment issues, Defender Managers get read-only access to Prisma Cloud settings and log files.

Defender Managers are typically members of your DevOps team. They need to manage the hosts that Prisma Cloud protects, but they never need to alter any security policies.

Defender Managers are also used to automate Defender deployment. If you use twistcli to deploy Defenders in your environment, create a service account with the Defender Manager role for the program that calls twistcli.

This role can see view the secrets that Defenders use to do their job, such as cloud credentials for registry scanning.

Access User

Access User role has permissions to some basic API routes only, and no access to the Console UI. This user role will be deprecated in the next release of Prisma Cloud Compute.

Users work with Docker containers. They run Docker client commands on the hosts that are protected by the Defender. The commands they run include:

- Pulling an image from a registry.
- Starting a container on a host.
- Stopping a container.

Users are typically members of your engineering team. For example, all members of your test team would be assigned the User role.

CI User

The CI user role can be assigned to users that should only be able to run the plugin but have no other access to configure Prisma Cloud or view the data that we have. It is designed to only provide the minimal amount of access required to run the plugins.

A CI user cannot log into the Console or even view the UI Dashboard.

Custom roles

Prisma Cloud Compute allows you to create customized user roles to fit the needs of your organization. When creating a role, you will be able to select which sections of the product the role will have access to and with what permissions - Read-Only or Read-Write.

The permissions you grant for a role will apply to both the Prisma Cloud UI and API.

Read permission will grant the role with access to all GET APIs for fetching data. Write permission will grant the role with access to all other APIs (POST, PUT, DELETE, etc.) for saving data and performing actions, in addition to all GET APIs.

If a role allows access to policies, users with this role will be able to see all rules and all collections that scope rules under the Defend section, even if the user's view of the environment is restricted by assigned collections.

Create custom roles

Create a new custom role under **Manage > Authentication > Roles**.

STEP 1 | In **Manage > Authentication > Roles**, click **Add role**.

You can also use the **Clone** action on an existing role, which copies its permissions and saves you the need to set them from scratch.

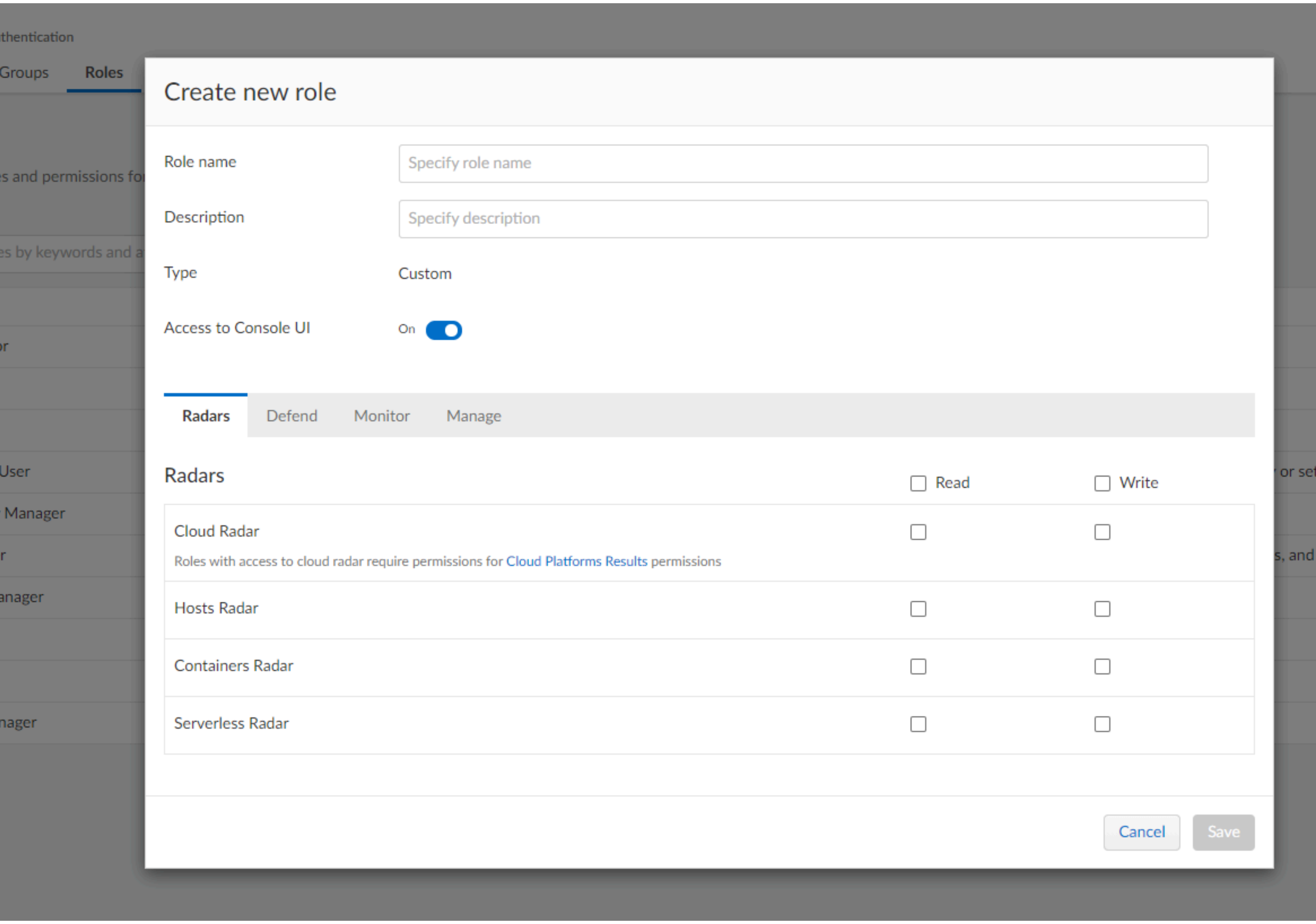
STEP 2 | Enter a name and a description for your custom role.

STEP 3 | Use the **Access to Console UI** toggle to configure whether the role will have access to Prisma Cloud UI. Setting the toggle to off means that the role will only have access to the API (according to the permissions granted to it).

STEP 4 | Select the role's permissions under **Radars**, **Defend**, **Monitor**, and **Manage**. For each permission you can choose granting Read or Write access.

STEP 5 | Click **Save**.

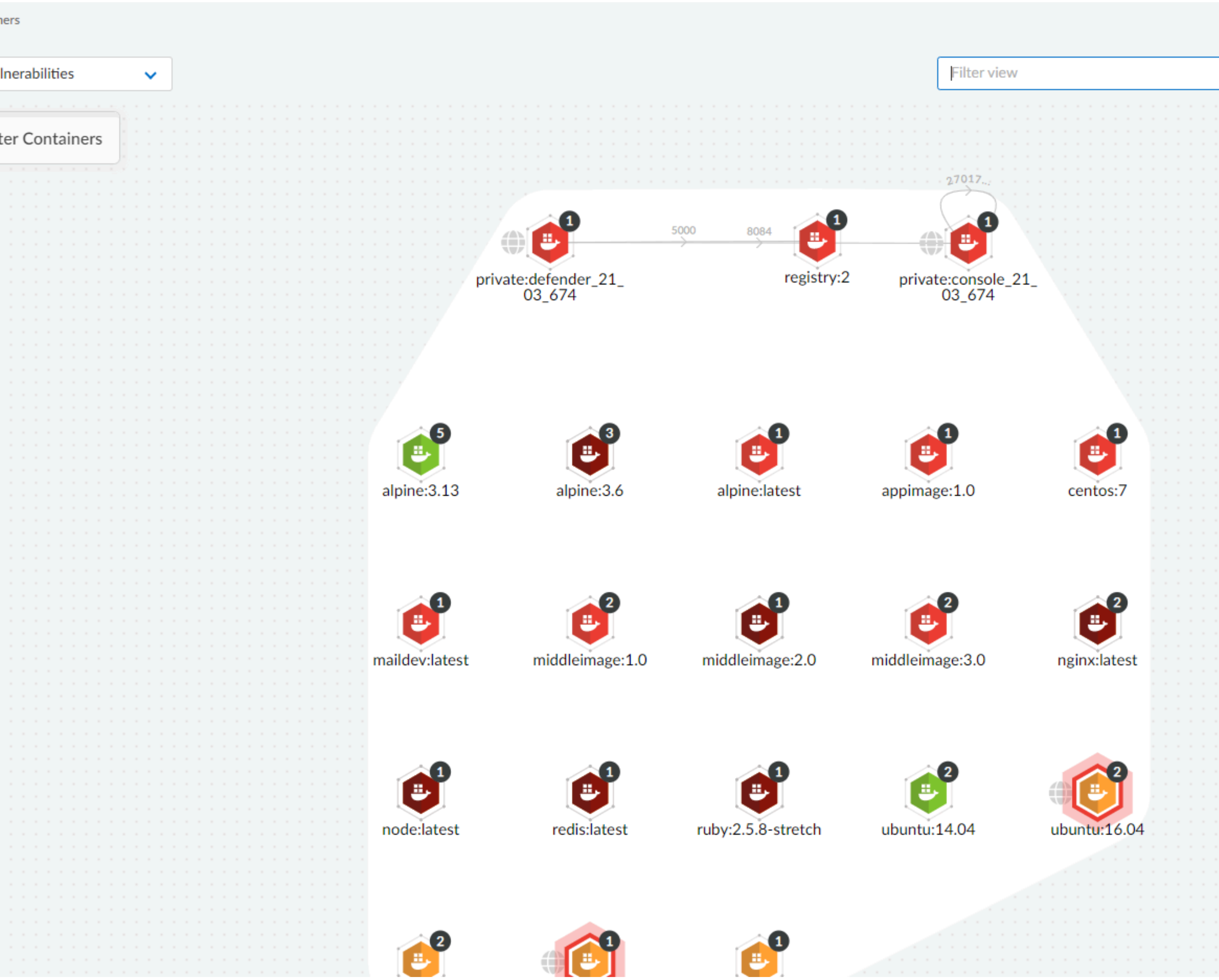
Changes to role permissions while users are logged into Prisma Cloud Console only apply after users re-login.



Unique permissions

- Several permissions require other permissions in order to work properly. For example, roles that access policies typically require permissions for collections. These dependencies are highlighted when setting role permissions.

  If a role is missing permissions, the logged-in user will get a suitable message on the relevant page. Components to which he is missing permissions will be hidden or disabled.



- Some pages do not include write actions (e.g. Containers Radar), however you will still have the option to grant write permission to them. This will have no effect on the UI components and API calls the role has access to.

- **Data updates pushed to client browsers** permission is required in order to control access to sensitive information used to populate views in the UI. This data flows over the connection from the Console to client browsers and includes new audits, scan progress updates, etc. Granting no access to this permission will cause these updates to not be exposed in the UI until an active refresh of the browser.

Assign roles

To learn how to assign roles to users and groups, see Assign roles.

Assign roles

Edit on GitHub

After creating a user or group, you can assign a role to it. Roles determine the level of access to Prisma Cloud's data and settings.

Prisma Cloud supports two types of users and groups:

- Centrally managed users and groups, defined in your organization's directory service. With directory services such as Active Directory, OpenLDAP, and SAML providers, you can re-use the identities set up in these systems.
- Prisma Cloud users and groups, created and managed from Console. For centrally managed users groups, roles can be assigned after you integrate your directory service with Prisma Cloud. Roles can be assigned to individual users or to groups. When you assign a role to a group, all members of the group inherit the role. Managing role assignments at the group level is considered a best practice. Groups provide an easier way to manage a large user base, and simpler foundation for building your access control policies.

For Prisma Cloud users and groups, roles are assigned at the user level when the user is created. When you create a Prisma Cloud group, you add Prisma Cloud users to it. Users in this type of group always retain the role they were assigned when they were created.

Assigning roles to Prisma Cloud users

If you do not have a directory service, such as Active Directory (AD) or Lightweight Directory Access Protocol (LDAP), Prisma Cloud lets you create and manage your own users and groups. When you create a Prisma Cloud user, you can assign it a role, which determines its level of access.

To create a user and assign it a role:

STEP 1 | Open Console, and log in with your admin credentials.

STEP 2 | Go to **Manage > Authentication > Users**.

STEP 3 | Click **Add user**.

1. Enter a username.
2. Enter a password.
3. Assign a role.
4. Click **Save**.

Assigning roles to Prisma Cloud groups

Collecting users into groups makes it easier to manage your access control rules.

Each user in the group retains his own role to prevent erroneous privilege escalation.

To create a Prisma Cloud group and add users to it:

STEP 1 | Open Console and log in with your admin credentials.

STEP 2 | Go to **Manage > Authentication > Groups**.

STEP 3 | Click **Add group**.

1. Enter a name for your group.
2. In the drop down list, select a user.
3. Click **+**.
4. Repeat steps b to c until your group contains all the members you want.
5. Click *Save:

Assigning roles to AD/OpenLDAP/SAML users

By default, AD/OpenLDAP/SAML users have the very basic Access User role. You can grant users a different level of access to Console by assigning them roles.

If a user is a part of an AD, OpenLDAP, or SAML group, and you have assigned a role to the group, the user inherits the group's role.

Prerequisites: You have integrated Prisma Cloud with Active Directory, OpenLDAP, or SAML.

STEP 1 | Open Console.

STEP 2 | Log in with your admin credentials.

STEP 3 | Go to **Manage > Authentication > Users**.

STEP 4 | Click **Add user**.

1. Enter the username for the user whose role you want to set. For example, if you have integrated Prisma Cloud with Active Directory, enter a UPN.
2. In the **Role** drop-down menu, select a role.
3. Click **Save**.

Assigning roles to AD/OpenLDAP/SAML groups

You can assign an AD/OpenLDAP/SAML group a role. Members of the group inherit the group's role. When a user from a group tries to access a resource protected by Prisma Cloud, Prisma Cloud resolves the member's role on the fly.

If a user is assigned multiple system roles, either directly or through group inheritance, then the user is granted the rights of the highest role. If a user is assigned both system and custom roles, then the user will be randomly granted the rights of one of the roles.

For example, assume Bruce is part of GroupA and GroupB in Active Directory. In Console, you assign the Administrator role to GroupA and the Auditor role to GroupB. When Bruce logs into Prisma Cloud, he will have Administrator rights.

The following procedure shows you how to assign a role to an existing AD/OpenLDAP/SAML group:

Prerequisites: You have integrated Prisma Cloud with Active Directory, OpenLDAP, or SAML.

STEP 1 | Open Console, and log in with your admin credentials.

STEP 2 | Go to **Manage > Authentication > Groups**.

STEP 3 | Click **Add group**.

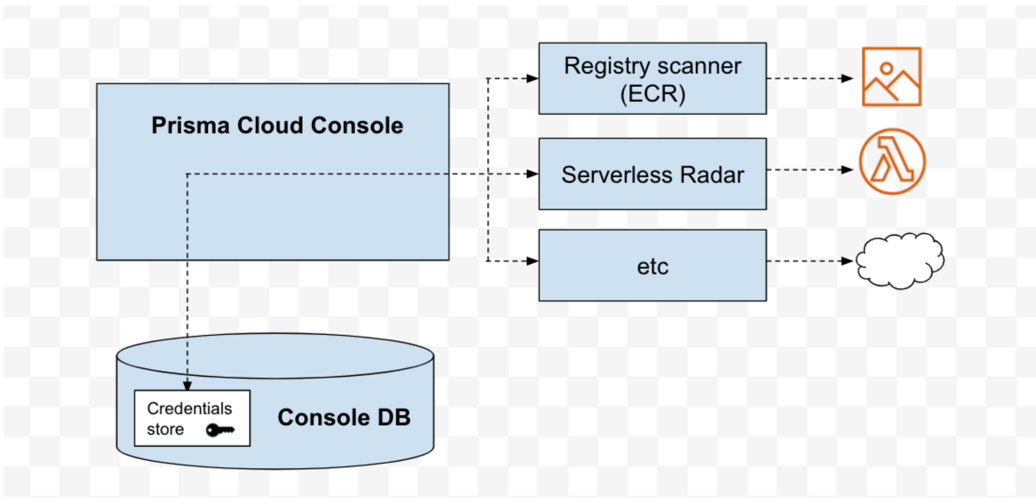
1. Specify the name of the group. It should match the group name specified in your directory service.
2. Check LDAP group.
3. Select a role.
4. Click **Save**.

Credentials store

Edit on GitHub

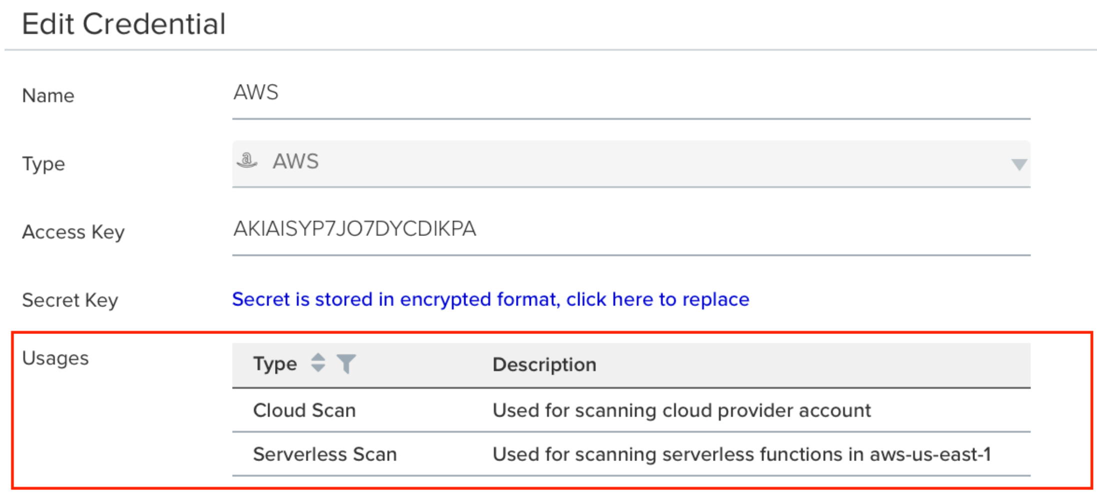
Container environments tend to utilize many third party services across multiple cloud providers. To improve accessibility and reusability, Prisma Cloud manages all credentials in a central encrypted store. Credentials are used when setting up the following integrations:

- Scanning (container registries, serverless functions, agentless scanning, etc).
- Alerting in third party services (email, Slack, ServiceNow, etc).
- Deploying and managing Defender DaemonSets from the Console UI.
- Injecting secrets from secret stores into containers at runtime.



The credential store can be found under **Manage > Authentication > Credentials Store**. Credentials cannot be deleted if they are currently in use. To see all the places where a credentials is being used, click on an entry in the credentials store table, and review the **Usages** list.

If a credential is being used by an integration, and you edit its parameters (e.g. username, password, etc), the new values are automatically propagated to the right places in the product. You don't need to delete and set up the integration again to refresh a credential's values.



AWS

Prisma Cloud lets you authenticate with AWS the following ways:

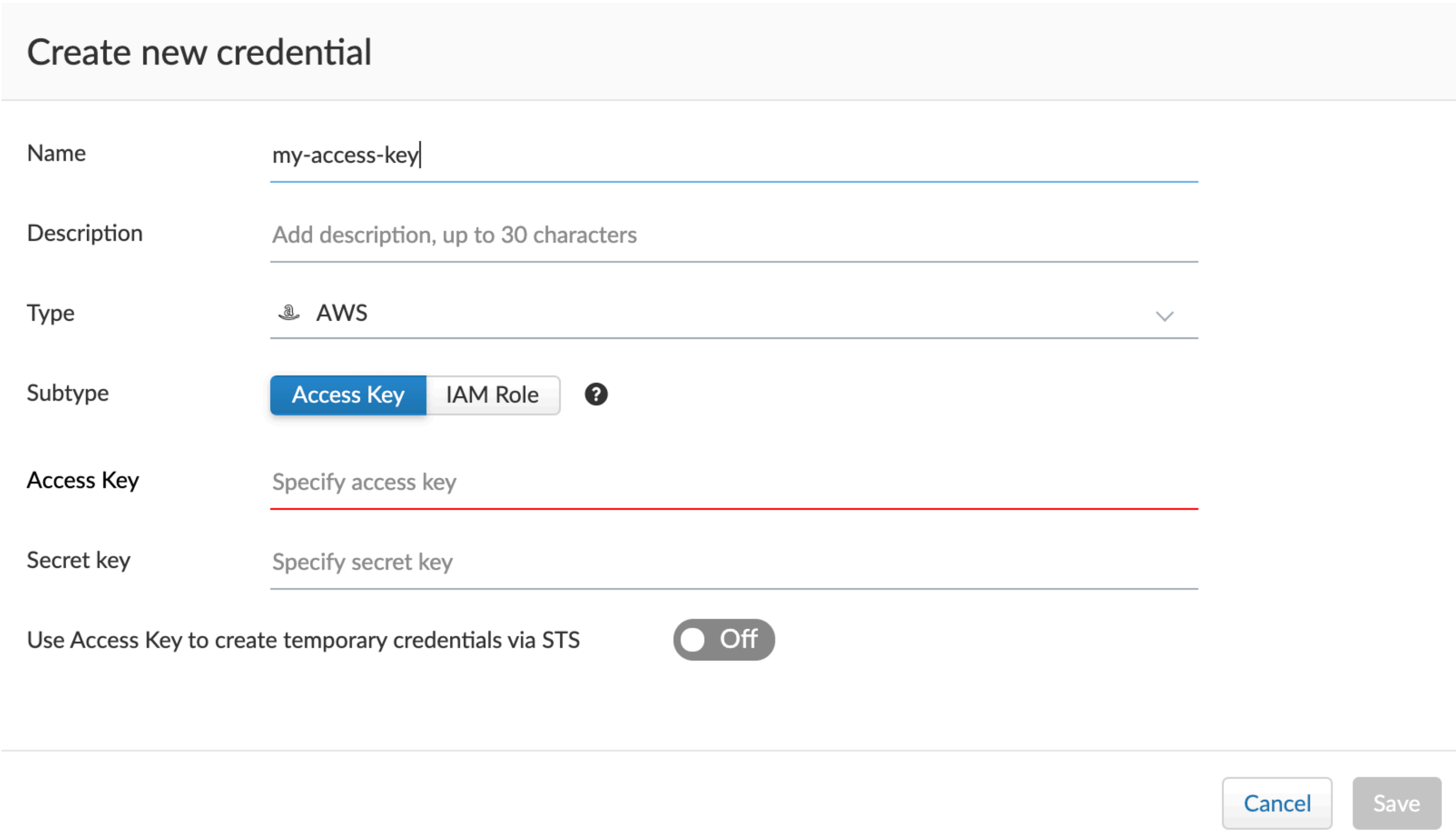
- IAM users (access keys).
- IAM roles.
- Security Token Service (STS) (Recommended when using IAM users).

AWS IAM users

An IAM user is an identity that you create in AWS. It represents a person or service that uses the IAM user to interact with AWS.

Access keys are long-term credentials for IAM users. Access keys consist of two parts: an access key ID and a secret access key. Like a username and password, you must use both the access key ID and secret access key together to authenticate requests with AWS.

The credentials store in Prisma Cloud lets you save access keys. When creating a new credential, select the **AWS** type and **Access Key** subtype, and then enter an access key ID and secret access key.



NOTE

As per AWS best practices, it is recommended to rotate your keys every 90 days. Prisma Cloud will raise an Alert if the age of the credentials added is >90 days. If you use this option, ensure to rotate your keys at least every 90 days.

AWS IAM roles

In many cases, you can take advantage of IAM roles and their temporary security credentials rather than the long-term credentials associated with IAM users.

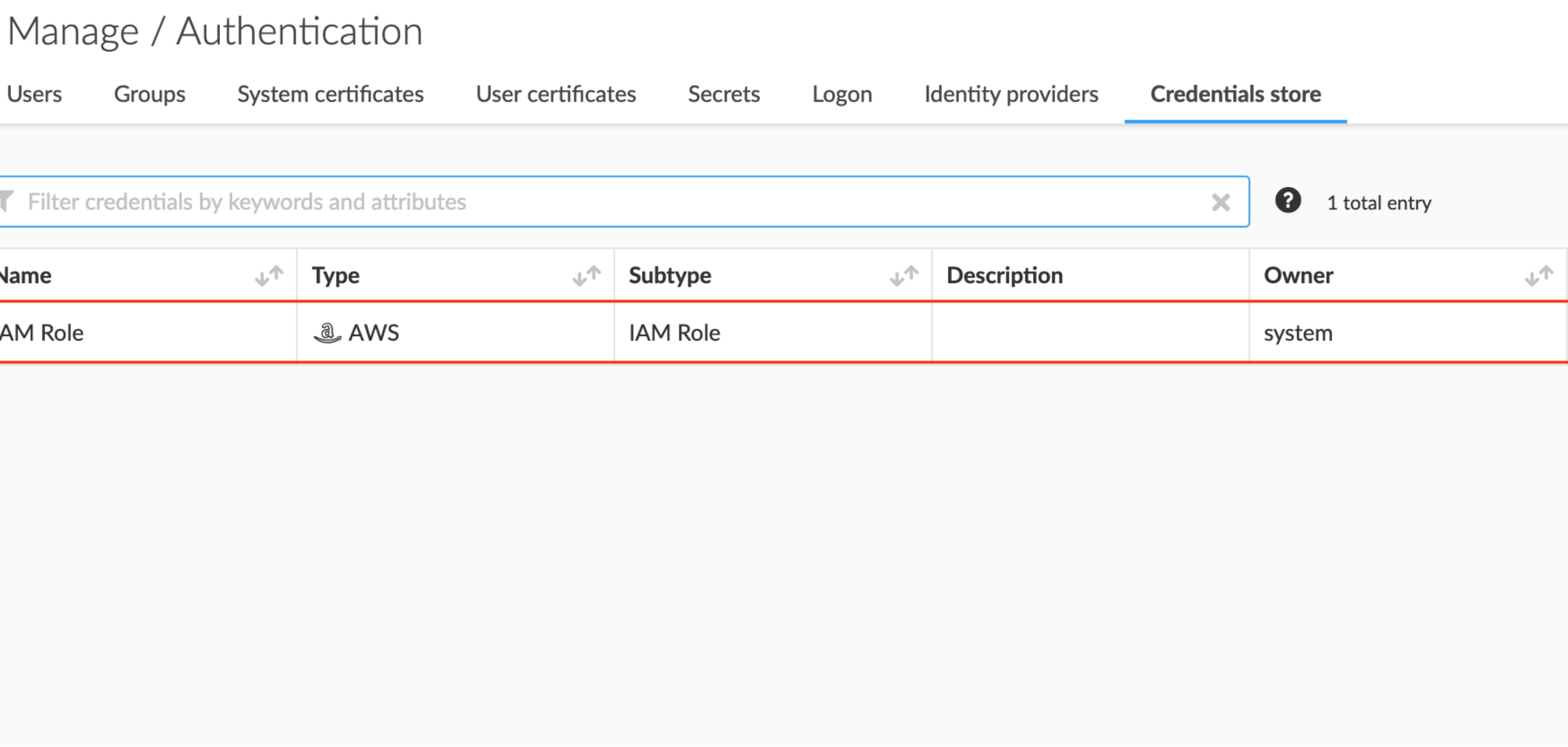
IAM roles are similar to IAM users. Both are identities with permission policies. The permission policy determines what an identity can (and cannot) do in AWS. However, roles don't have any associated credentials (e.g. access keys). Instead of being uniquely associated with one person, roles are assumable by anyone who needs them. IAM users can assume a role to temporarily acquire the permissions needed to carry out a specific task.

IAM roles solve the problem of how to securely manage and distribute credentials. For example, how do you distribute credentials to new EC2 instances created by an auto scaling group? How do you rotate credentials on EC2 instances in a cluster? Instead of creating and distributing credentials, you can delegate permission to call the AWS API as follows:

1. Create an IAM role.
2. Specify the AWS service (e.g. EC2) that can assume the role.
3. Specify the API actions and resources Prisma Cloud can use after assuming the role.
4. Specify the role when you launch the service.

5. Prisma Cloud retrieves a set of temporary credentials and uses them as needed.

Prisma Cloud ships with a default credential called **IAM Role**. Assuming you've created an IAM role in AWS, configured trust (who can use the role), permission policy (what the role can do), and launched the service with the role, Prisma Cloud can acquire the temporary credentials it needs to carry out its work. Each feature in Prisma Cloud has documentation which describes permission policy it requires.



How Prisma Cloud Accesses IAM Role Credentials

Roles provide a way to grant credentials to applications that run on EC2 instances to access other AWS services, such as ECR. IAM dynamically provides temporary credentials to the EC2 instances, and these credentials are automatically rotated for you.

This section shows how Prisma Cloud Defender gets credentials to scan the ECR registry when its running on an EC2 instance with a correctly configured IAM role. The mechanism is similar for other services where Prisma Cloud might run.

When you create an EC2 instance, you can assign it a role. When the instance is started, the AWS instance metadata service (IMDS) attaches your credentials to the running EC2 instance. You can access this metadata from within the instance using the following command:

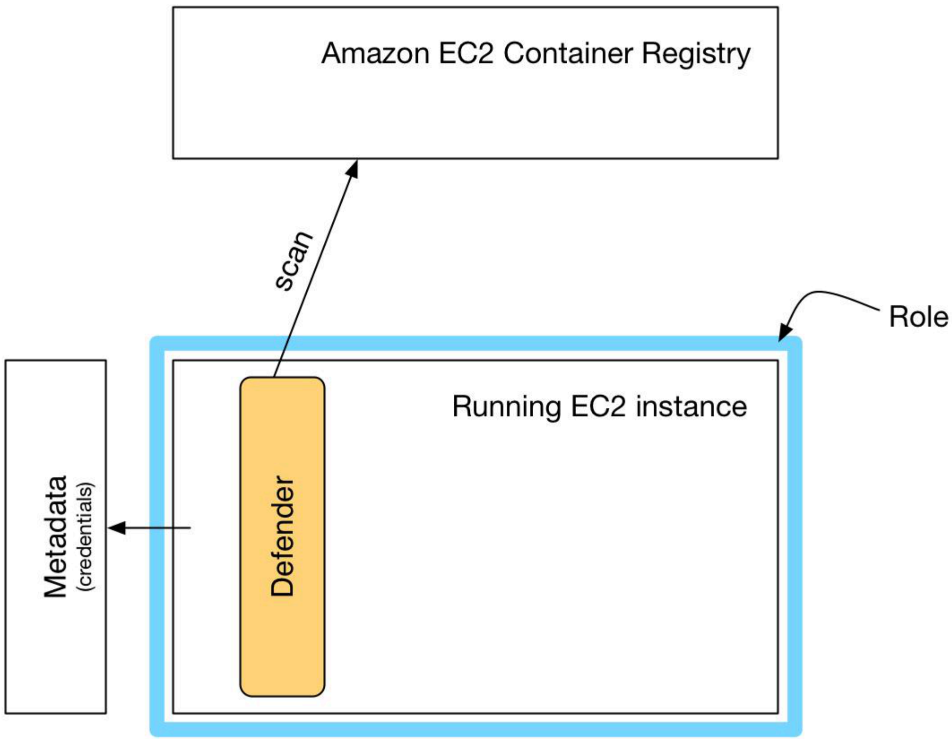
```
curl http://169.254.169.254/latest/meta-data/iam/security-
credentials/<POLICY_NAME>
{
"Code" : "Success",
"LastUpdated" : "2017-06-29T06:12:29Z",
"Type" : "AWS-HMAC",
"AccessKeyId" : "ASIA...",
```



```
"SecretAccessKey" : "3VI...",
"Token" : "dzE...",
"Expiration" : "2017-06-29T12:16:54Z"
}
```

Where *<POLICY_NAME>* is assigned to the EC2 instance when it is created or at some point during its life.

The following diagram shows all the pieces. Defender retrieves the credentials from the metadata service, then uses those credentials to retrieve and scan the container images in ECR.



AWS Security Token Service (STS)

AWS Security Token Service (STS) lets you request temporary, limited-privilege credentials for AWS IAM users or users that you authenticate (federated users).

Per the AWS Well-Architected Framework, this method is a recommended best practice when using IAM users. With STS, you don't have to distribute long-term AWS credentials (access keys) to places like the Prisma Cloud credentials store. Also, the temporary credentials have a limited life span, so you don't have to rotate or revoke them when they're no longer needed.

When you configure integration with an AWS resource, you can pick an AWS credential from the central store, then use STS to change the role of the account. AWS STS lets you have a few number of IAM identities that can be used across many AWS accounts. For example, if you were setting up Prisma Cloud to scan an AWS ECR registry, you would select the AWS credentials from the central store. Then you would enable **Use AWS STS**, and enter the name of the STS role to assume in the target account.

When using AWS STS, ensure the following:

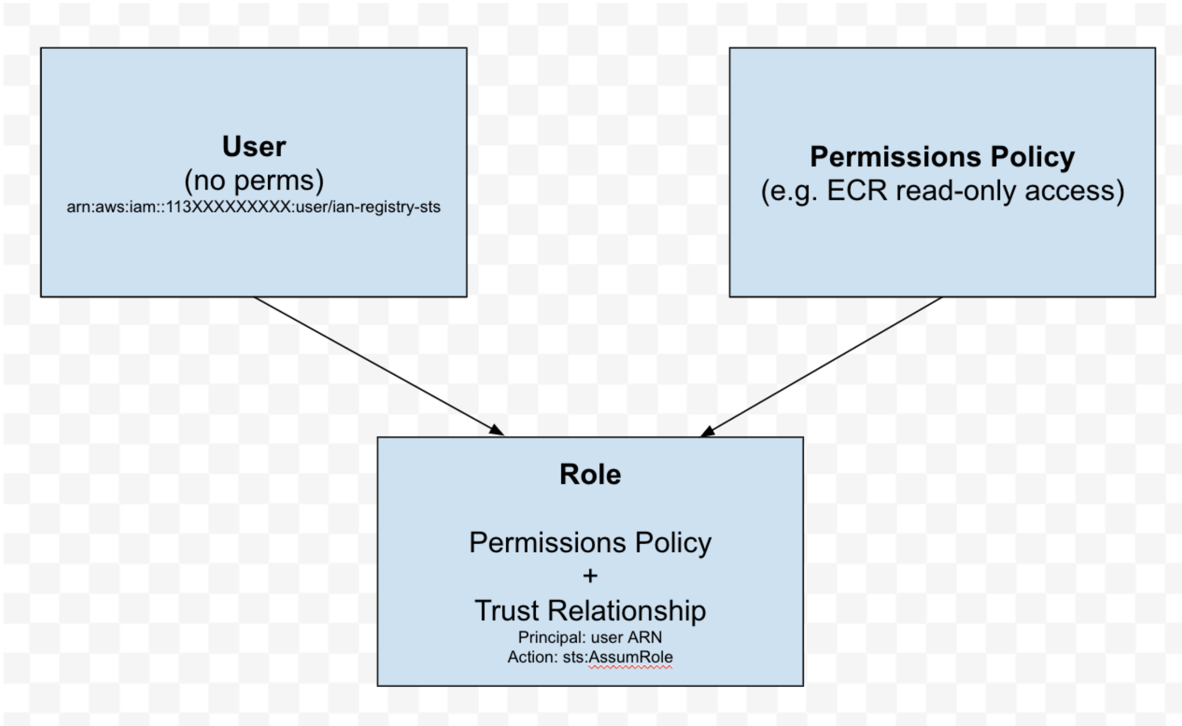
- The policy of the IAM user you use as credentials has **sts:AssumeRole** permission on the IAM role you're going to assume. Sample policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "sts:AssumeRole",
            "Resource": "arn:aws:iam::123456789123:role/stsIAMrole"
        }
    ]
}
```

- The IAM role you're going to assume has the IAM user mentioned above configured as a trusted entity. Sample trusted entity policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::123456789123:user/prismaUser"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

The following diagram shows the relationship between an IAM user, a permissions policy, and an assumed role. By default, the IAM user has no permissions. The permissions policy allows ready-only access to the ECR registry. The role brings everything together. It specifies the trust relationship (who is allowed to assume the role, also known as the principal), it grants to ability for the principal to assume roles (sts:AssumeRole), and it declares what the role can do when it assumed by a principal (permission policy).



Azure

This section discusses Azure credentials.

Creating an Azure Service Principal

Create an Azure Service Principal so that Prisma Cloud Console can scan your Azure tenant for microservices. To get a service key:

STEP 1 | Download and install the Azure CLI.

STEP 2 | Create a service principal and configure its access to Azure resources.

```
$ az ad sp create-for-rbac \
  --name <user>-twistlock-azure-cloud-discovery-<contributor|
reader> \
  --role <reader|contributor> \
  --scopes /subscriptions/<yourSubscriptionID> \
  --sdk-auth
```

The **--role** value depends upon the type of scanning:

- contributor = Cloud Discovery + Azure Container Registry Scanning + Azure Function Apps Scanning
- reader = Cloud Discovery + Azure Container Registry Scanning

STEP 3 | Copy the output of the command and save it to a text file. You will use the output as the **Service Key** when creating an Azure credential.

```
{
```

```
  "clientId": "bc968c1e-67g3-4ba5-8d05-f807abb54a57",
  "clientSecret": "5ce0f4ec-5291-42f8-gbe3-90bb3f42ba14",
  "subscriptionId": "ae01981e-e1bf-49ec-ad81-80rf157a944e",
  "tenantId": "d189c61b-6c27-41d3-9749-ca5c9cc4a622",
  "activeDirectoryEndpointUrl": "https://
login.microsoftonline.com",
  "resourceManagerEndpointUrl": "https://management.azure.com/",
  "activeDirectoryGraphResourceId": "https://graph.windows.net/",
  "sqlManagementEndpointUrl": "https://
management.core.windows.net:8443/",
  "galleryEndpointUrl": "https://gallery.azure.com/",
  "managementEndpointUrl": "https://management.core.windows.net/"
}
```


Storing the Credential in Prisma Cloud

Store the service principal's credentials in Console so that Prisma Cloud can authenticate with Azure for scanning.

STEP 1 | Open Console, and go to **Manage > Authentication > Credentials Store**.

STEP 2 | Click **Add credential**, and enter the following values:

1. Enter a descriptive **Name** for the credential.
2. In the **Type** field, select **Azure**.
3. Enter the **Service Key**.

   Copy and paste the contents of the text file you saved earlier when you created the service principal.
4. **Save** your changes.

Google Cloud Platform (GCP)

Accessing GCP to scan resources can be done in one of two ways. You can make use of a service account and create a key for that account or you can use an API Key. Google recommends that you use a service account with a key and we document that here. More information is available here https://cloud.google.com/docs/authentication/api-keys.

Creating a service account

Create a service account that Prisma Cloud can use to scan your resources in GCP. The permissions the service account should have depend on the features you plan to use the credentials for. For the specific permissions, see the dedicated article of each feature (e.g. Google Container Registry scanning).

STEP 1 | Google provide a comprehensive guide for creating a service account - https://cloud.google.com/iam/docs/creating-managing-service-accounts

STEP 2 | Create a key for this service account. The format of this key should be JSON. Google have a guide for this - https://cloud.google.com/iam/docs/creating-managing-service-account-keys

STEP 3 | Copy the contents of the downloaded key, here is an example:


```
{
```

```
  "type": "service_account",
  "project_id": "mycompany-project",
  "private_key_id": "abe29475a09fb22e709fdc622306a714e17cqd1c",
  "private_key": "-----BEGIN PRIVATE KEY-----
\nMIIEvgIBADANBgkqhkiG9w0BAQEFBBSCBKgwggSkAgEAAoIBAQCyBJgPechqsXAK
\nTaz1y77AGqei47IbgWegRq8JqqoQGERhBX8X41otaRNUIn7fpTdH/JjRfJ0wyduz
\nn6TLmeMz+d/yIZBtztujJ4KoGTS0yTybtcKWKg254upri6RIcMS3ArNXsNtSwLQx
\nicVDCI3uDKLuNyawmLf1BiHLwWZK8bOUe5thd3J9UXc+B+dL9JRYyz1Iq+X/Nz1w
\n7D3TPXfy54Hg39rDRrx0bK0E+AIRMA5vPFmGrWlYn6HyltOxCU/D5NUrExRo3aug
\nsIvQgGE3QYLU4a5n9jsWPbjSGI+EH/+zZ1fze5pk6rprlvKtbvygJSFGpdsjS4EX
\nbFgPVYGJAgMBAAECggEABu9fIaY1yLNIKTyYrvwsnpUDQBkk/
oWSbQfn6IVfqfAa\nFNoHFx4GLLP12u6bmPiZoFoutWWIhaatgpBG9iAU/
fi/cNI+K0r2W0MuJ8CQoTTg\nQbQpZBp4Daxxg7ZNVH2hKjyGklVbW/
xSIMZsWwXNqq8PF17qaNGQRBFEtoh+pM94\nJ23ZKIW5muF+5Svz4wLLS7VMtbl/
XrM9eCepVQNzQ701A67VQa1Z8KIot5IeQ0d2\njnHJn6XgaDe/IKuP8ClXnCUwo/
GbChCtctP0BpTeaTSOUrc1O7ntgFBGYxiSmgZ0\n13x43XkuYaycyZEycKEOaa9U
+k1KrcFZ/CDQdv+AUQKBgQD25mVWFxQdhFqF12vu\nEhz0jRjLbre
+V3UgOYJObmKMdM6iH+NQ9CNeFIn8qgHgOK7pHEgjcLvAv4ZgECin
\n1XtMNAFGRREGuzovvzQKwBGAEz8PovI2gkITqmcSQ7xzcGyY1Xm1mthpqkoWFe5c
\nk253fYhMjuITTXisYv8LBl5XGQKBgQC4lER2AmTSvLe+4sulTeDEocMsP+G4j/
A1\neS1mG5e5YGUtuWgIdfNKUn1YG5uX3ERZVeCdRO7B/osQ4uAeJ1SIS3Zvw5QVtS/
s\nFOJa1UJ/nxGAA8vApjRgJkLyRbf/yoxsRlCQkQJcRd1SO9DRlCSWdSW1CpIpauiN
\nfZZW3iD78QKBgQCWW7Lk3cMjQqH6FjmlTySRDYhHA1MkuI1fFga0Cuc7EDtyYicF
\n+te7CJkL5OClkv95+P45jwLYHAsSX2TDE3o16wnHqHH4/nYt86wWy+ccbxwdQqds
\n6KCi50hDyDpwtst7u62WGgmnN8xMbOivOh2w6SLjNyQ0ix5tJRCavzMeqQKBgQCu
\nYvajf/
N93urDIEdC8Gcxn5tkTR6XXvaVrt0joWIhtF8jag5OIBIx3+m55rywJ100\nAhzquVvSUQlWdONF2
pTZ4iUUPi3dfPhbBS
\nA8TOMRLH1wIZVYYe3BYNSLTNbSVWmDkKpOLLQ6ZqIQKBgA0rkqfzz0MIij58OugB
\nFyv8UWvy+hYR15EvIFOl5jXomVl199x+XHQGiwV6cXGmGcii7eC7vXSmnjxILMEA
\nD4Odwi9vmyJXOtIT1WlVj/
faLrpKfunZEphYnrtRASuDzzU4cTbeElhfLOqkJEA4\nK4CCBhjL3UX8Z9FbJJz7mYoX
\n-----END PRIVATE KEY-----\n",
  "client_email": "mycompany-service-svc@mycompany-
project.iam.gserviceaccount.com",
  "client_id": "120957099362691824155",
  "auth_uri": "https://accounts.google.com/o/oauth2/auth",
  "token_uri": "https://oauth2.googleapis.com/token",
  "auth_provider_x509_cert_url": "https://www.googleapis.com/
oauth2/v1/certs",
  "client_x509_cert_url": "https://www.googleapis.com/
robot/v1/metadata/x509/mycompany-service-svc%40mycompany-
project.iam.gserviceaccount.com"
}
```

Storing the Credential on Prisma Cloud

Store your GCP credential in Prisma Cloud.

STEP 1 | Open Console, and go to **Manage > Authentication > Credentials Store**.

STEP 2 | Click **Add credential**, and enter the following values:

1. In the **Name** field, enter a label to identify the credential.
2. In the **Type** field, select **GCP**.
3. In the **Credential level** field, select **Project** or **Organization**. When using project-level credentials, Prisma Cloud scans the resources of a single project. With organization-level credentials, Prisma Cloud will find and scan all the projects in the organization.

   The service account permissions should fit the credentials level. Use **Project** with service accounts that have permissions to a single project. Use **Organization** with service accounts that have permissions to an organization or multiple projects.

   When using organization-level credentials, the service account you use should be defined for a single organization. Credentials for multiple organizations with the same service account are not supported.

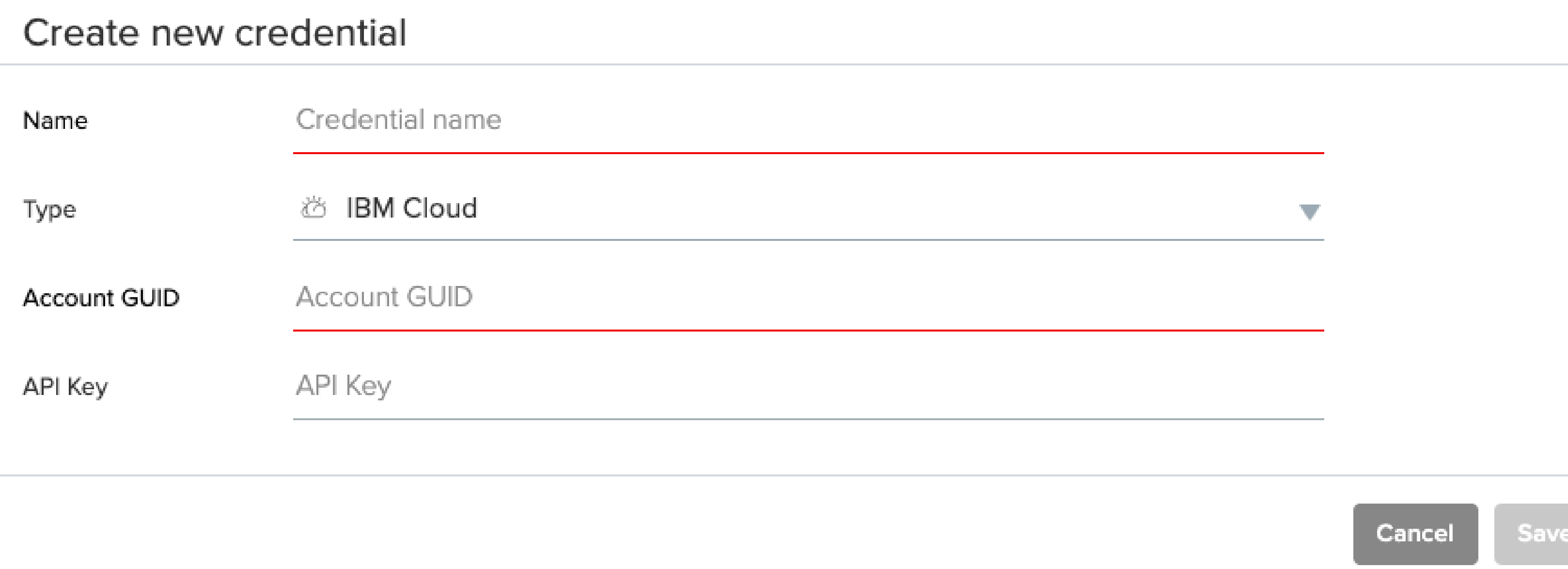
   Organization-level credentials are not supported for Serverless auto-defend, Host auto-defend*, and* Google Cloud Pub/Sub alerts.

   When using organization-level credentials, the performance of several features may be affected if the organization includes a large number of projects. For specific information and guidelines see:

   - Google Container Registry scanning
   - Managed Defender DaemonSet deployment for GKE.
   - Cloud Discovery
4. In the **Service Account** field, copy and paste the entire JSON key that you downloaded.
5. Leave the **API token** blank
6. Click **Save**.

IBM Cloud

Prisma Cloud integrates with IBM Cloud Security Advisor. To enable the integration, you must provide credentials, which consist of an Account GUID and API Key.



Kubeconfig

Kubernetes stores cluster authentication information in a YAML file known as kubeconfig. The kubeconfig file grants access to clients, such as kubectl, to run commands against the cluster. By default, kubeconfig is stored in *$HOME/.kube/config*.

Prisma Cloud uses the kubeconfig credential to deploy and upgrade Defender DaemonSets directly from the Console UI. If you plan to manage DaemonSets from the command line with kubectl, you don't need to create this type of credential.

The user or service account in your kubeconfig must have permissions to create and delete the following resources:

- ClusterRole
- ClusterRoleBinding
- DaemonSet
- Secret
- ServiceAccount

Prisma Cloud doesn't currently support kubeconfig credentials for Google Kubernetes Engine (GKE) or AWS Elastic Kubernetes Service(EKS). The kubeconfig for these clusters require an external binary for authentication (specifically the Google Cloud SDK and aws-iam-authenticator, respectively), and Prisma Cloud Console doesn't ship with these binaries.

STEP 1 | Open Console, and go to **Manage > Authentication > Credentials Store**.

STEP 2 | Click **Add credential**, and enter the following values:

1. In **Name**, enter a label to identify the credential.
2. In **Type** , select **Kubeconfig**.
3. In **Kubeconfig**, paste the contents of your *kubeconfig* file.

Cloud accounts

Edit on GitHub

Credentials for cloud accounts are managed in **Manage > Cloud accounts**. Other types of credentials are managed in the credentials store in **Manage > Authentication > Credentials store**.

Authenticate with Azure using a certificate

You can authenticate with Azure using a certificate as a secret. As with password authentication, the certificate is stored with the Azure service principal. For more information, see the Microsoft docs here.

STEP 1 | Log into Compute Console.

STEP 2 | Go to **Manage > Cloud accounts**

STEP 3 | Click **Add account**.

STEP 4 | In **Select cloud provider**, choose **Azure**.

STEP 5 | Enter a name for the credential.

STEP 6 | In **Subtype**, select **Certificate**.

STEP 7 | In **Certificate**, enter your service principal's certificate in PEM format.

The certificate must include the private key. Concatenate public cert with private key (e.g., cat client-cert.pem client-key.pem).

STEP 8 | Enter a tenant ID.

STEP 9 | Enter a client ID.

STEP 10 | Enter a subscription ID.

STEP 11 | Click **Next**.

STEP 12 | In **Scan account**, disable **Agentless scanning**.

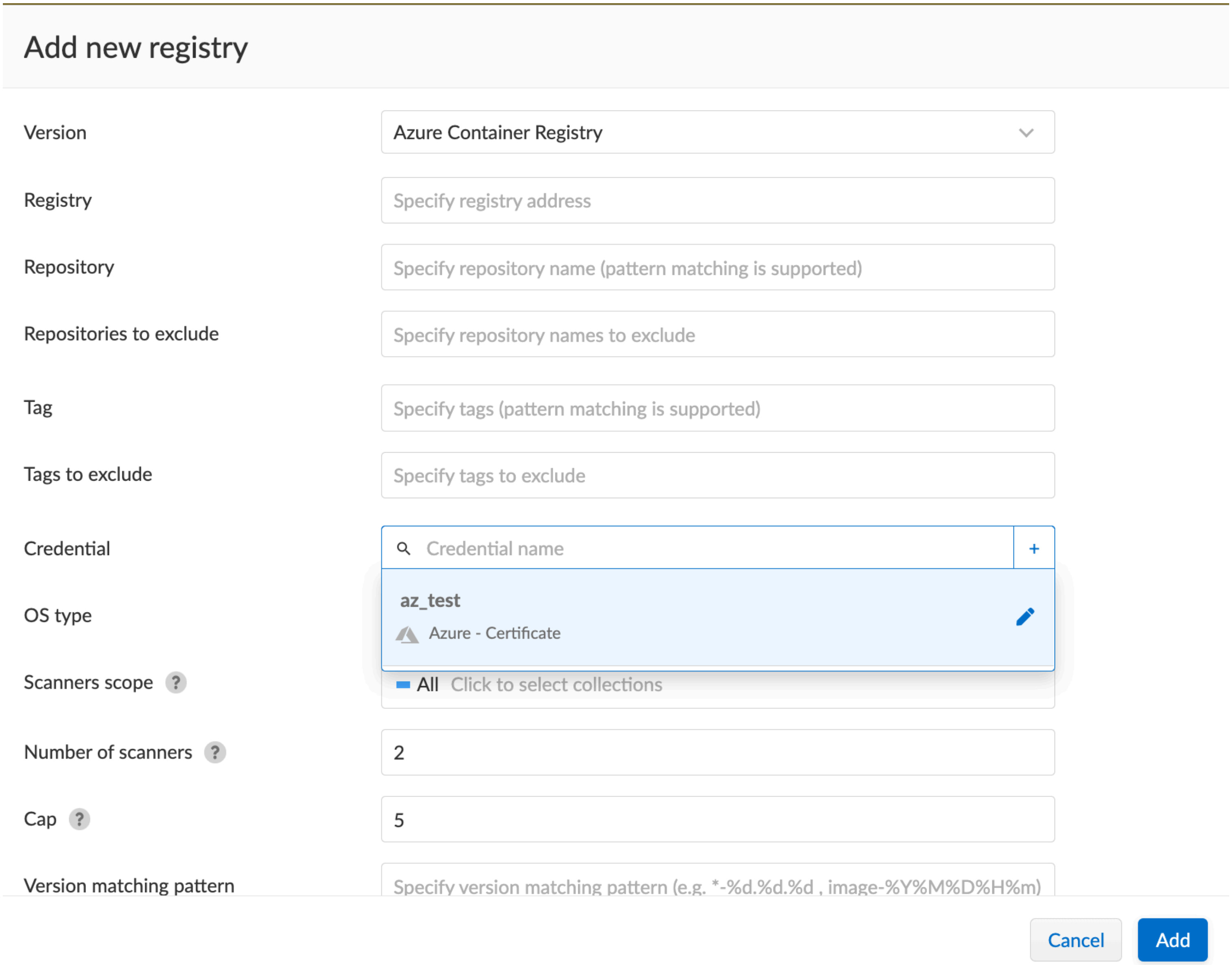
STEP 13 | Click **Next**.

STEP 14 | Click **Add account**.

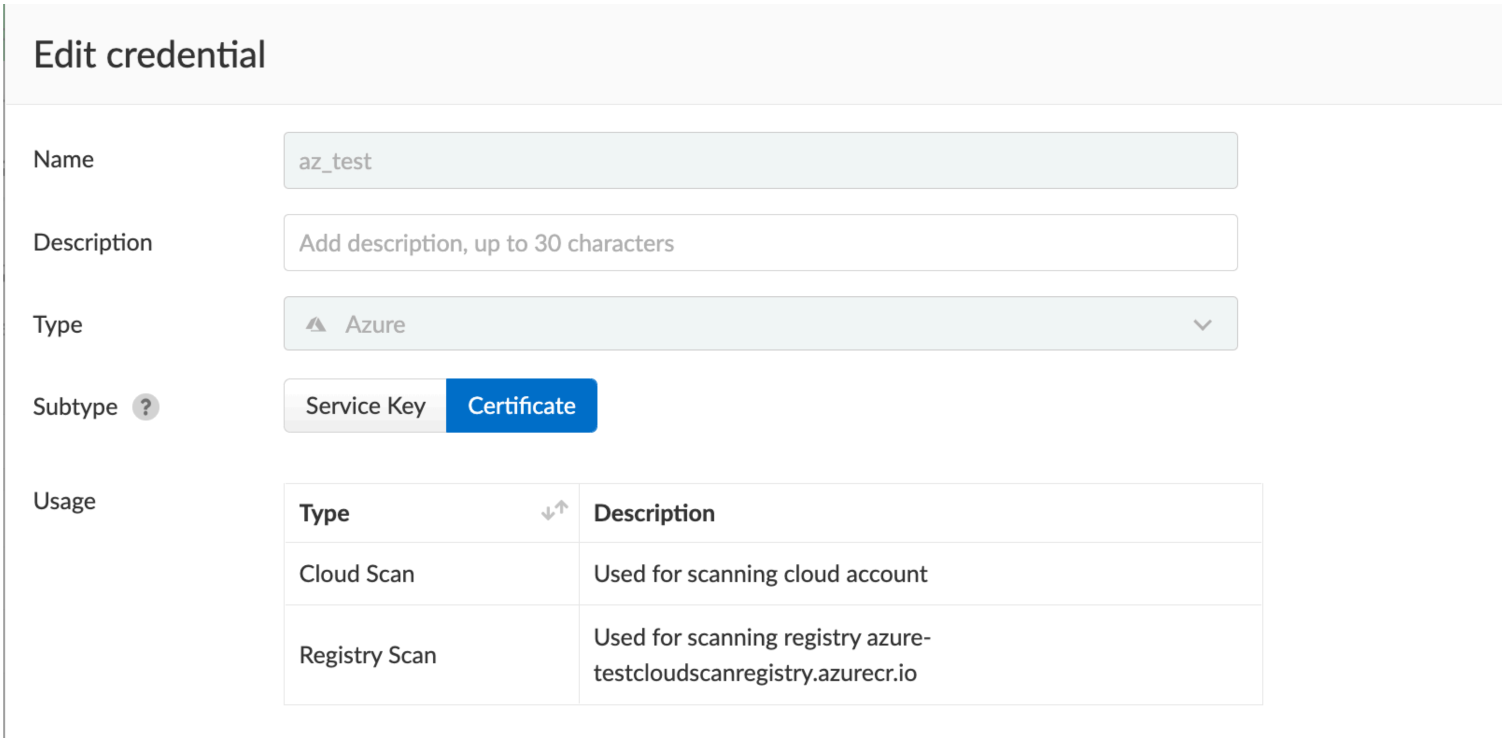
STEP 15 | Validate the credential.

Your Azure credential is now available to be used in the various integration points in the product, including registry scanning, serverless function scanning, and so on. If authentication with a certificate is supported, it's shown in the credential drop-down in the setup dialog.

For example, the following screenshot shows the setup dialog for scanning Azure Container Registry:



After setting up your integrations, you can review how and where the credential is being used by going to **Manage > Authentication > Credentials store** and clicking on the credential.


Edit credential
Name
az_test
Description
Add description, up to 30 characters
Type
Azure
Subtype
Service Key
Certificate
Usage
Type
Description
Cloud Scan
Used for scanning cloud account
Registry Scan
Used for scanning registry azure-testcloudscanregistry.azurecr.io

Vulnerability management

Edit on GitHub

Identify and prevent vulnerabilities across the entire application lifecycle while prioritizing risk for your cloud native environments. Integrate vulnerability management into any CI process, while continuously monitoring, identifying, and preventing risks to all the hosts, images, and functions in your environment. Prisma Cloud combines vulnerability detection with an always up-to-date threat feed and knowledge about your runtime deployments to prioritize risks specifically for your environment.

- Prisma Cloud vulnerability feed
- Vulnerability Explorer
- Vulnerability management rules
- Search CVEs
- Scan reports
- Scanning procedure
- Customize image scanning
- Configure Registry Scans
- Registry scanning
- Base images
- Configure VM image scanning
- Configure code repository scanning
- Agentless scanning
- Malware scanning
- Vulnerability risk tree
- Vulnerabilities Detection
- CVSS scoring
- Windows container image scanning
- Serverless function scanning
- VMware Tanzu blobstore scanning
- Scan App-Embedded workloads
- Troubleshoot vulnerability detection

Prisma Cloud vulnerability feed

Edit on GitHub

Information on threat intelligence and vulnerability data on Prisma Cloud is available through the Prisma Cloud Intelligence Stream(IS) feed.

Pre-filled CVEs

On Prisma Cloud, you may find vulnerabilities with a CVE identifier that neither MITRE nor NVD is reporting or is actively analyzing. A pre-filled CVE is the result of an analysis conducted by Palo Alto Networks Unit 42 researchers. The researchers manually review the details of each vulnerability, identify the correct range of affected releases and deliver the data to IS.

Many vulnerabilities in open-source software are assigned with a CVE ID and promptly analyzed by NVD and Linux distribution vendors. However, some vulnerabilities take a long time to be analyzed, sometimes weeks or even months. Having a CVE but no analysis means users have no information on the severity, affected releases, or description of the vulnerability and thereby making it impossible to defend against these vulnerabilities.

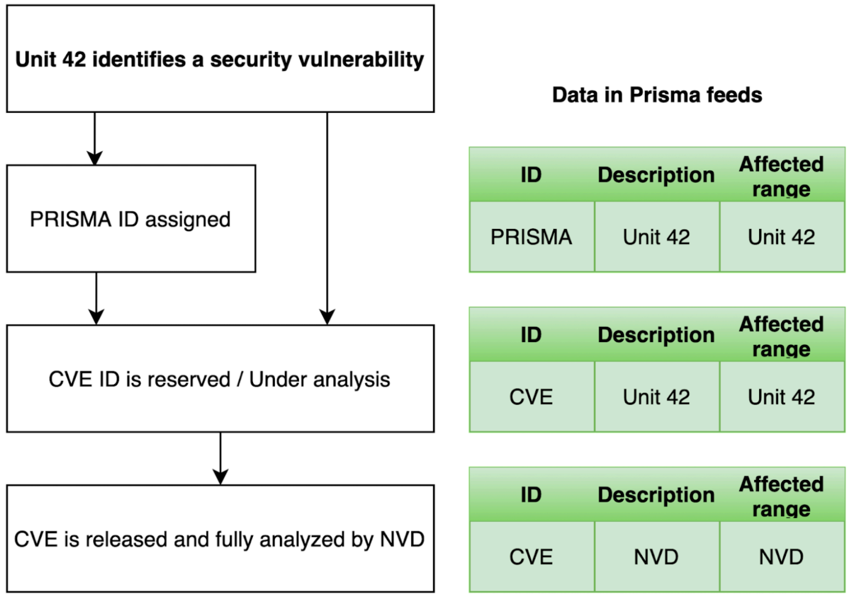
Let's examine an example scenario. Security researchers find a vulnerability in an open source project. The vulnerability details are publicly discussed in the project's bug tracker, e.g. in a GitHub issue. Following the discussion, the issue is fixed and a CVE ID is assigned to the issue. At this stage, NVD analysis takes place, and it may take multiple days for the NVD site to be updated with description and the affected releases range (CPE). Instead of waiting for the official analysis to complete, our researchers evaluate the vulnerability and insert the data into Prisma Cloud feeds quickly, preventing any delay in remediation of the vulnerability. When the NVD entry is fully updated, Prisma Cloud uses the official data from NVD.

PRISMA-* IDs

You may also find vulnerabilities marked with a PRISMA-* identifier. These vulnerabilities lack a CVE ID. Many vulnerabilities are publicly discussed or patched without a CVE ever being assigned to them. While monitoring open source vulnerabilities, our team identifies vulnerabilities you need to be aware of and assigns PRISMA IDs to them whenever applicable.

For example, let's review PRISMA-2021-0020. A user found a bug in the Python package click and opened an issue through its open source repository on GitHub. Our research team found this issue and determined it explains a valid security vulnerability. Although no CVE was assigned to this vulnerability, our team promptly assigned it a PRISMA identifier and analyzed the correct range of affected releases. Affected customers were alerted to this vulnerability despite the lack of any public vulnerability identifier. If a CVE is ever assigned to the same vulnerability that has a Prisma ID, the CVE takes over and the PRISMA ID entry is fully replaced. Read more about the correlation between PRISMA IDs and CVEs in this blog post.

The following diagram shows the PRISMA ID and Pre-filled CVEs assignment flow:



PRISMA-* ID Syntax

PRISMA ID syntax consists of the PRISMA prefix, year of release, and a sequence of four digits. For example, "PRISMA-2020-1234". This format is intentionally similar to that used by CVE IDs. There is absolutely no correlation between the sequence used for PRISMA IDs to that of CVEs released the same year. There is also no grouping of PRISMA IDs. That is, there is no correlation between adjacent PRISMA ID sequences.

Investigating PRISMA-* Vulnerabilities

The vulnerability description includes the necessary information required to understand the vulnerability. The severity is carefully determined by our team based on CVSS scoring. You may also access the ID link to find the original source that resulted in the assignment of the PRISMA ID. This will likely be an external advisory, a GitHub (or another bug tracker) issue, or it may directly lead you to the fix commit (pull request) when there is no correlating informational page.

Prisma ID FAQs

- **Why use PRISMA-IDs?**

  We are committed to ensuring that the Prisma Cloud Intelligence Stream provides the most accurate and up-to-date vulnerability information.

  Through the Intelligence Stream, Prisma Cloud should be able to alert on any relevant vulnerabilities that exist in scanned environments, regardless of having a CVE or not. Our researchers monitor open-source code repositories continuously to detect publicly discussed but undisclosed vulnerabilities that are not tracked under a CVE record. Upon finding such a vulnerability, the researchers complete a full analysis of the vulnerability including assessing its severity and describing its impact, and finally assign a PRISMA ID. The Intelligence Stream is shortly thereafter updated with the new entry, and users immediately benefit from the detection of the vulnerability by Prisma Cloud.

This process allows Prisma Cloud users to be better informed and secure from vulnerabilities that are otherwise not detected by regular vulnerability management tools.

- **Why not wait for a CVE-ID?**

  Although most vulnerabilities in open-source are assigned CVEs quickly after being discovered, some vulnerabilities are not assigned a CVE for a variety of reasons. In some cases, maintainers are unaware of the process to assign CVEs, ignorant to the importance of having a CVE, or may even refuse to have CVE IDs assigned to their projects.

  Prisma Cloud researchers actively encourage all maintainers to assign CVE IDs to security vulnerabilities in their projects. We partner with NVD and MITRE to ensure that information regarding known vulnerabilities is public and available to everyone in the industry. PRISMA IDs are not meant to be a replacement for CVEs – PRISMA IDs are assigned to ensure our users are protected from any known threat regardless of whether a CVE was assigned to it or not.

  Palo Alto Networks is a CVE Numbering Authorities (CNA); we assign CVE IDs to any zero day vulnerability that we discover. The purpose of PRISMA IDs is to track vulnerabilities that were already public knowledge at the time we identified them, but were not tracked under a CVE ID.

- **Why not all PRISMA-IDs get assigned with a CVE ID?**

  As mentioned above, although we do encourage all maintainers to assign CVEs to the vulnerabilities found in their project, we keep seeing a lot of undisclosed vulnerabilities that are publicly discussed. We would be happy to see all PRISMA IDs be replaced with a CVE ID, however, we do have limited resources - and simply cannot assign a CVE for each vulnerability. For zero days found by our research team, we follow the responsible disclosure process and ask the vendor to assign a CVE or offer the assistance of Palo Alto Networks as a CNA.

- **Can PRISMA-IDs be found on NVD or MITRE?**

  Public vulnerabilities identified by our researchers, before a CVE is associated with them, are assigned a PRISMA-* identifier. You may access the reference link to get more information about the source through which our researchers discovered the vulnerability.

- **Do you have a way to correlate PRISMA-ID to CVE when it is assigned a CVE?**

  Through an ongoing maintenance process, PRISMA-IDs are replaced with a corresponding CVE ID when it is created.

- **PRISMA-XXXX disappeared, what happened?**

  When a vulnerability with a Prisma ID is assigned a CVE ID, the PRISMA ID is replaced with the new CVE. Findings will display the official CVE ID instead.

- **What is the “Published Date” in Console?**

  The Published date is the date that the CVE was published by the feed source or by NVD. This information is taken from the relevant feed - either the vendor feed or NVD.

  The date a CVE is published in NVD is not the date it was analyzed. The CVE can be published in NVD and only later updated with the analysis.

- **Why do I see a newly added CVE with an old published/fixed date?**

  The Published Date of the CVE is the date when the vendor published it first. The CVE may have been added to the IS after the published date because the feed is constantly updated.

  > *When a PRISMA ID or a Pre-Filled CVE is replaced with a CVE entry from NVD or a vendor's feed, the* ***Published Date*** *of the CVE will reflect what was published in the official CVE.*

- **I have set a grace period and my builds were passing. Now "all of a sudden" they fail on a CVE/PRISMA ID that wasn't there before. What happened?**

  See the answer above.

- **The severity assigned to a vulnerability is different between the IS and NVD, how is that possible?**

  For known vulnerabilities with a CVE, we rely on the most authoritative source. For OS packages (packages that are maintained by the OS vendor, marked as type "package" in Compute), the CVE details are from the specific vendor feed. For other CVEs, the information is from official sources like NVD and vendor-specific security advisories. If the affected package is maintained by an OS vendor, the severity as indicated by the vendor is used and not the severity determined by NVD. Furthermore, for new vulnerabilities missing analysis, or undocumented vulnerabilities (such as PRISMA-IDs), we rely on severity determined by our researchers.

- **How do I check if my Intelligence Stream is up to date?**

  1. Navigate to **Manage > System > Intelligence**.
  2. Verify that the Status is **Connected**.
  3. Check the **Last streams update**.

- **How can I know which OS releases are supported?**

  Prisma Cloud can protect containers built on nearly any base layer operating system. We update our feed with the vendor's data only for supported versions. CVE information is provided for the base layers detailed in the system requirements for all versions except EOL versions. While our feed could still contain vulnerability data for EOL versions, it is not complete and is potentially inaccurate because of missing details on the vulnerability. If there are no vulnerabilities in our feed for a specific distro release, the version will be tagged with the following message: **OS not supported and may be missing vulnerability data. Please use a supported version of the OS.**

- **Does the Intelligence Stream include CVE information for EOL versions?**

  See the answer above.

- **I have seen an open CVE/PRISMA vulnerability that I believe has a fix. What should I do?**

  The IS uses the automated maintenance process for any updates to existing vulnerabilities. If you believe new information regarding a vulnerability is missing from our feed, please report it through the support channels.

- **Where can I find more information on troubleshooting?**

  See troubleshooting.

Vulnerability Explorer

Edit on GitHub

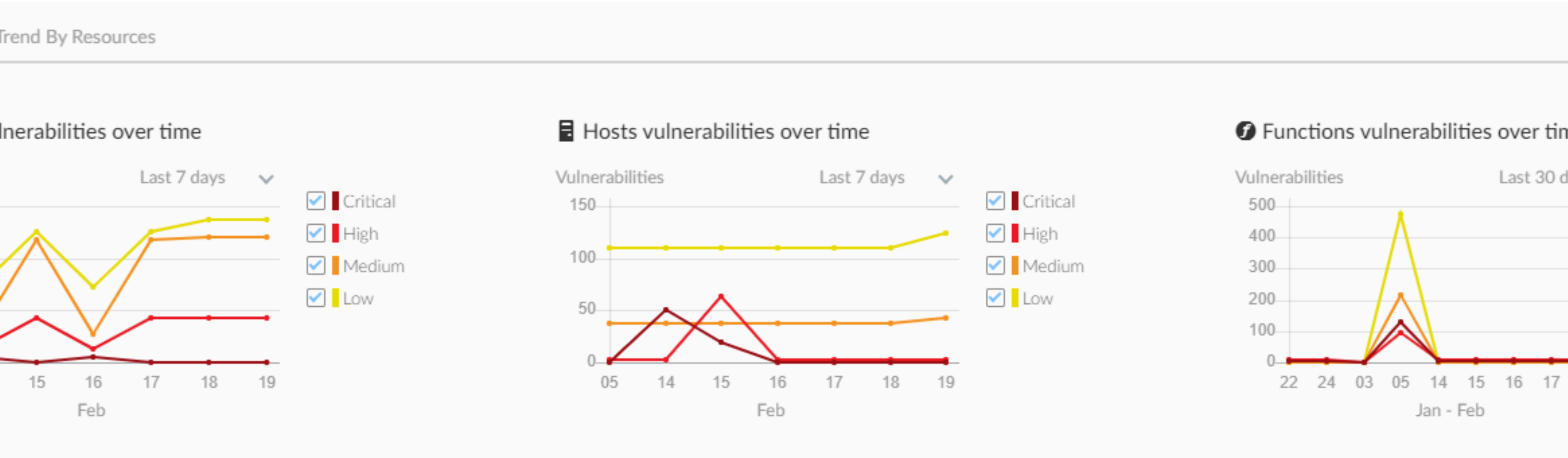
Most scanners find and list vulnerabilities, but Vulnerability Explorer takes it a step further by analyzing the data within the context of your environment. Because Prisma Cloud can see how the containers run in your environment, we can identify the biggest risks and prioritize them for remediation.

To view Vulnerability Explorer, open Console, then go to **Monitor > Vulnerabilities > Vulnerability Explorer**.

Roll-ups

The charts at the top of the Vulnerability Explorer helps you answer two questions:

1. How many CVEs do you have?



For each object type (image, host, function), the chart reports a count of vulnerabilities in each object class in your environment as a function of time. Consider an environment that has just a single image, where that image has three vulnerabilities: one high, one medium, and one low. Then at time=today on the **Images vulnerabilities** chart, you could read the following values:

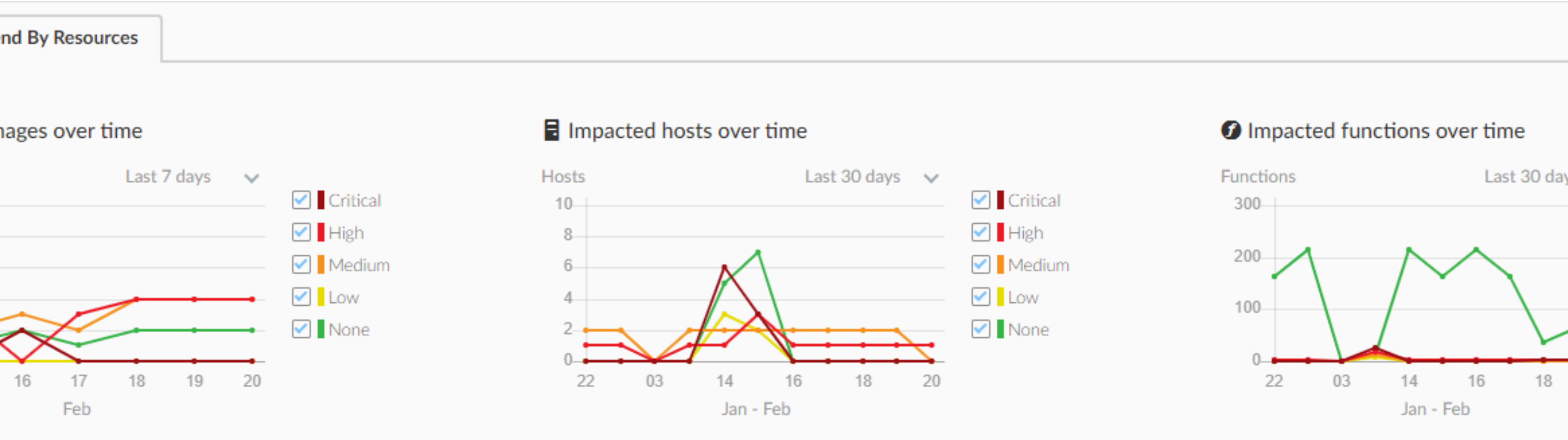
Critical - 0

High - 1

Medium - 1

Low - 1

2. How many images do you need to fix?



For each object type (image, host, function), the chart reports a count of the highest severity vulnerability in each object class in your environment as a function of time. Consider an environment that has just a single image, where that image has three vulnerabilities: one high, one medium, and one low. Then at time=today on the **Impacted images** chart, you could read the following values:

Critical - 0

High - 1

Medium - 0

Low - 0

Let's look at it another way with a different set of data. Assume the reading at t=today reports the following values, where t is some point on the x-axis of the chart.

Critical - 1

High - 1

Medium - 0

Low - 2

If your policy calls for addressing all critical vulnerabilities, then the chart tells you that there is precisely one image in your environment that has at least one critical vulnerability. Therefore, your work for today is to fix one image. That image might also have two high vulnerabilities and twenty low vulnerabilities, which you will see when you open the image's scan report, but this chart is not designed to give you a count of total number of vulnerabilities.

Filter tool

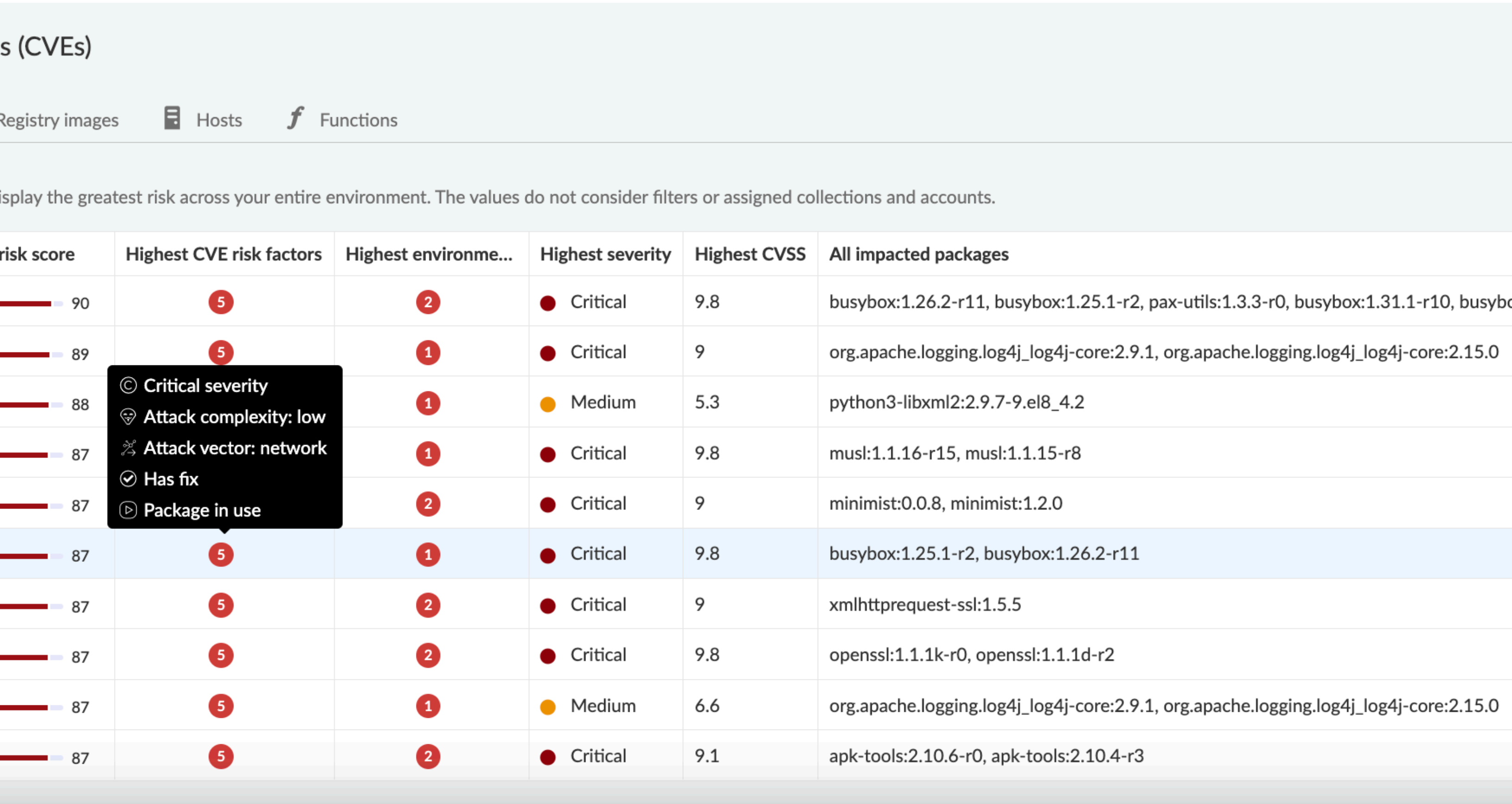
The filter tool at the top of the page allows you to search for a CVE ID in order to determine if any image, function, or host in your environment is impacted by a specific vulnerability (whether it is in the critical vulnerabilties list or not).

The filter tool also allows you to filter vulnerabilities based on CVSS threshold, Severity threshold, or Collections in your environment. For example, the CVE matches the filter if its highest severity is equal to or higher than the severity specified.

Vulnerabilities (CVE) results

Vulnerability Explorer gives you a ranked list of the most critical vulnerabilities in your environment based on the risk score. The ranked list consists of CVEs that are affecting the environment. Each CVE includes data about its risk factors, severity, CVSS, impacted packages, and impacted resources.

There are separate top ten lists for the container images, registry images, hosts, and functions in your environment.

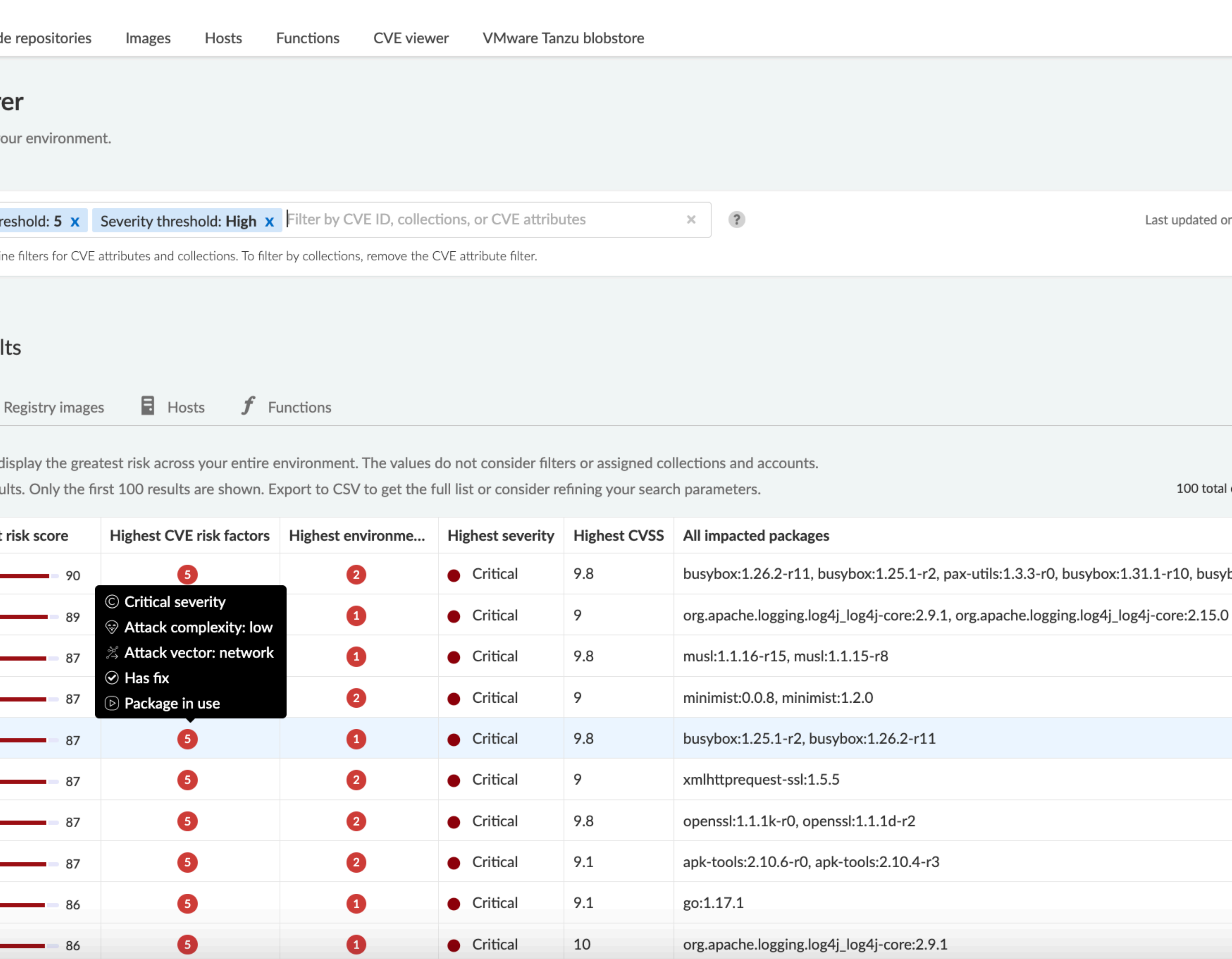


You can export the full list of CVEs affecting your environment in a CSV format. You can also download a detailed CSV report on impacted resources for a CVE ID from the **Actions** column.

The most important factor in the risk score is the vulnerability's severity. But additional factors are taken into account, such as:

- Is a fix available from the vendor?
- Is the container exposed to the Internet?
- Are ingress ports open?
- Is the container privileged?
- Is an exploit available?

The underlying goal of the risk score is to make it actionable (should you address the vulnerability, and with what urgency). Factors that contribute to the risk score are shown in the Highest risk factor columns.



Running containers can introduce additional environmental factors that increase the calculated score for a vulnerability. For example, when the container runs as root, it could exacerbate the problem. A list of container traits that heighten the risk are listed in the detailed information dialog when you click on a row in the top ten table.

Consider the following guidelines:

- The data for each CVE ID that consists of highest risk score, highest CVE risk factors, highest environmental risk factors, highest severity, and highest CVSS for all impacted packages display the highest value for the CVE based on your entire environment. This is irrespective of the applied filters, collections, or accounts that you are assigned to.
- The vulnerability (CVE) results hide the **impacted resources** if you use a filter or have an assigned collection or account as the percentage refers to the entire environment.

- The exported CSV displays an empty column of **impacted resources** if you use a filter or have an assigned collection or account as this percentage refers to the entire environment.
- If a filter returns more than 100 results, only the top 100 results are shown. You can download the full data in a CSV format.
- You cannot combine the filters **CVSS threshold** and **Severity threshold** with **Collections**. Also, filter by **CVSS threshold** and **Severity threshold** is not supported for users with assigned collections or accounts.
- The vulnerability (CVE) results display vulnerabilities based on a set filter threshold or higher.

CVE ID details

The vulnerability explorer CVE dialog appears when you click on a row in the Vulnerabilities (CVE) results.

The vulnerability explorer CVE dialog displays the following:

- CVE description and its impacted packages.
- A list of all impacted resources such as deployed images, registry images, hosts, and functions filtered based on the severity threshold, CVSS threshold, or collections if specified in the **Filter tool** of the vulnerability explorer.
- The highest risk profile for a CVE ID based on the highest risk in an environment.

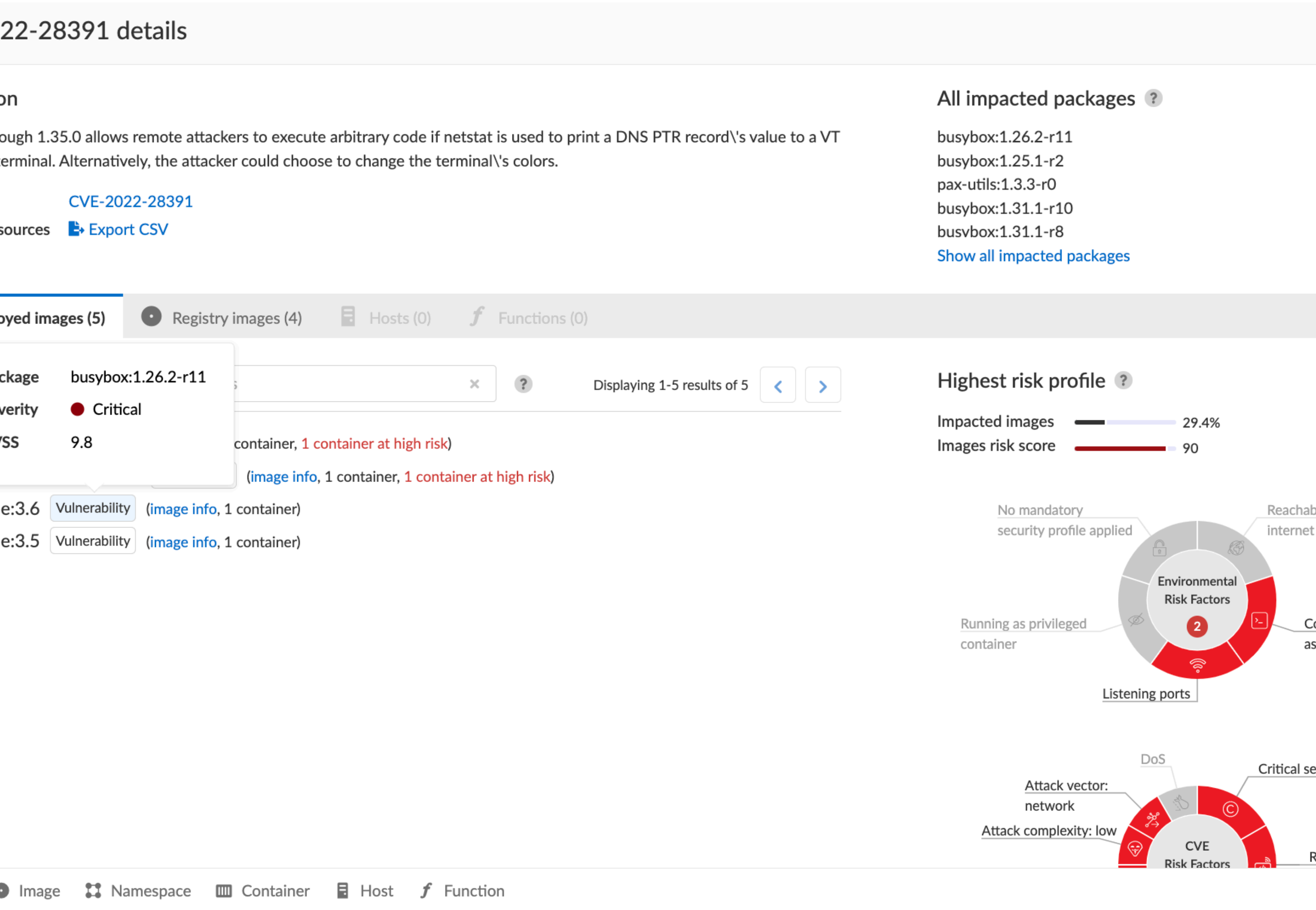
  For each resource type, the highest risk profile includes the risk score and risk factors found in the entire environment and is regardless of the filters and assigned collections or accounts.

In the risk profile section, you can see the impacted resources percentage along with the risk score.

+ The **impacted resources percentage** is not displayed if you use a filter or have assigned collections or accounts as it reflects the value based on the entire environment.

You can export a list of impacted resources in a CSV format from here or from the **Actions** column as described earlier.

For each impacted resource, you can hover over the **Vulnerability** tag next to the resource name to see the specific package, severity, and CVSS of the CVE for a resource.

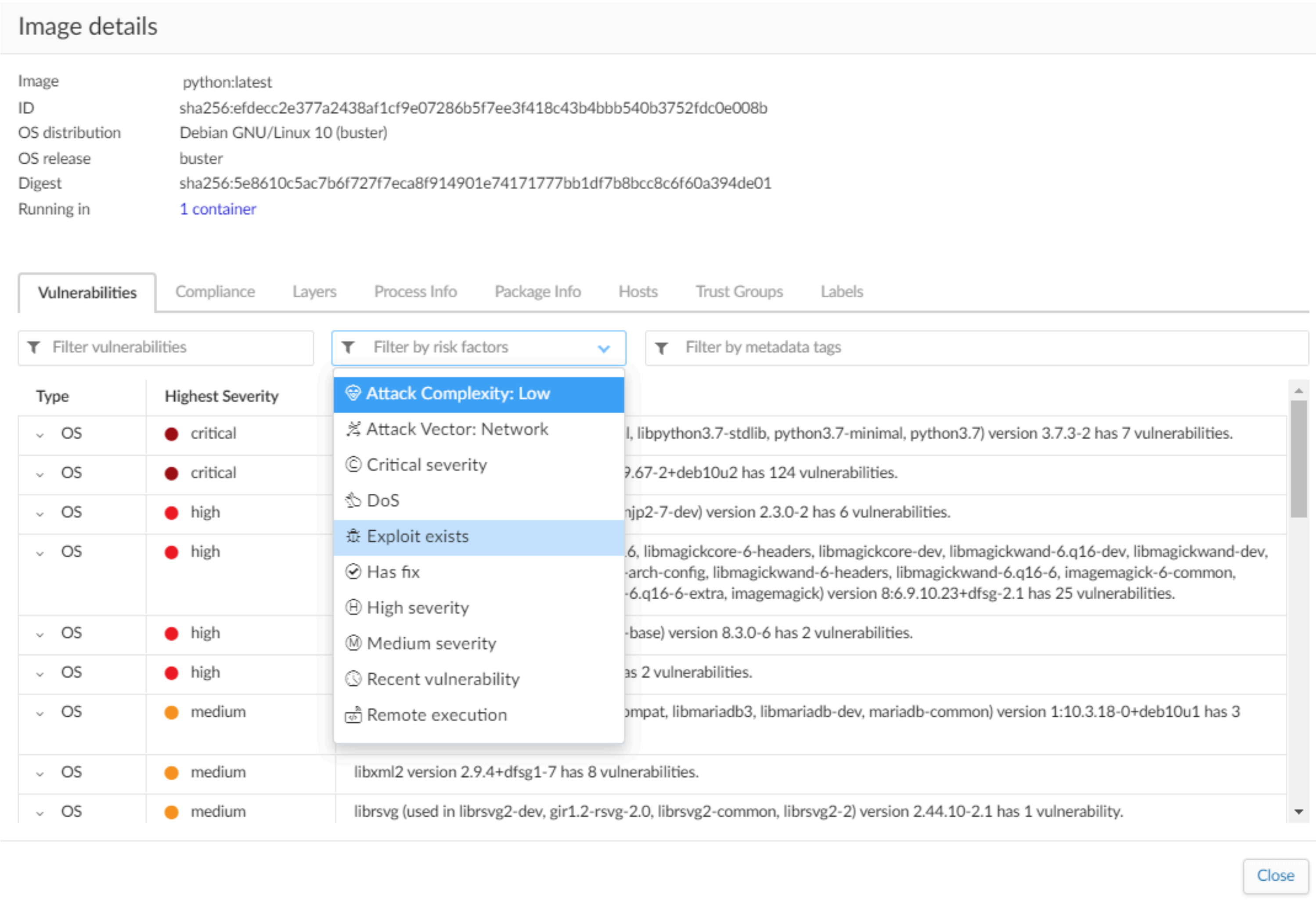


Risk factors

Risk factors are combined to determine a vulnerability's risk score. Vulnerabilities with the highest risk scores are surfaced in the top ten lists.

Risk factors can also be used to prioritize individual vulnerabilities for mitigation. For example, if your cluster runs containers from disparate business groups, a major concern might be container breakouts. DoS vulnerabilities would likely be much less important than remote code execution vulnerabilities, particularly if exploit code were available, you were running as root, and you didn't have AppArmor or SELinux applied.

To filter vulnerabilities based on risk factors: open the image, host, or function scan report; open the **Vulnerabilities** tab; and select one or more risk factors.



Prisma Cloud supports the following risk factors:

- **{Critical | High | Medium} severity** — Vulnerability severity.
- **Has fix** — Fix is available from the distro, vendor, or package maintainer.
- **Remote execution** — Vulnerability can be exploited to run arbitrary code.
- **DoS** — Component is vulnerable to denial of service attacks, such as buffer overflow attacks, ICMP floods, and so on.
- **Recent vulnerability** — Vulnerability was reported in the current or previous year.
- **Exploit exists** — Code and procedures to exploit the vulnerability are publicly available.
- **Attack complexity: low** — Vulnerability is easily exploited.
- **Attack vector: network** — Vulnerability is remotely exploitable. The vulnerable component is bound to the network, and the attacker's path is through the network.
- **Reachable from the internet** — Vulnerability exists in a container exposed to the internet. The detection of this risk factor requires that CNNS will be enabled and network objects will be defined for external sources under **Radar > Settings.** Then, if a connection is established between the defined external source and the container, the container is identified as reachable from the internet.

- **Listening ports** — Vulnerability exists in a container that is listening on network ports.
- **Container is running as root** — Vulnerability exists in a container running with elevated privileges.
- **No mandatory security profile applied** — Vulnerability exists in a container running with no security profile.
- **Running as privileged container** — Vulnerability exists in a container running with --privileged flag.
- **Package in use** — Vulnerability exists in a component that is actually running. For example, if Redis is running in a container or on a host as a service, then all the following (hypothetical) vulnerabilities could be surfaced by filtering on this risk factor:

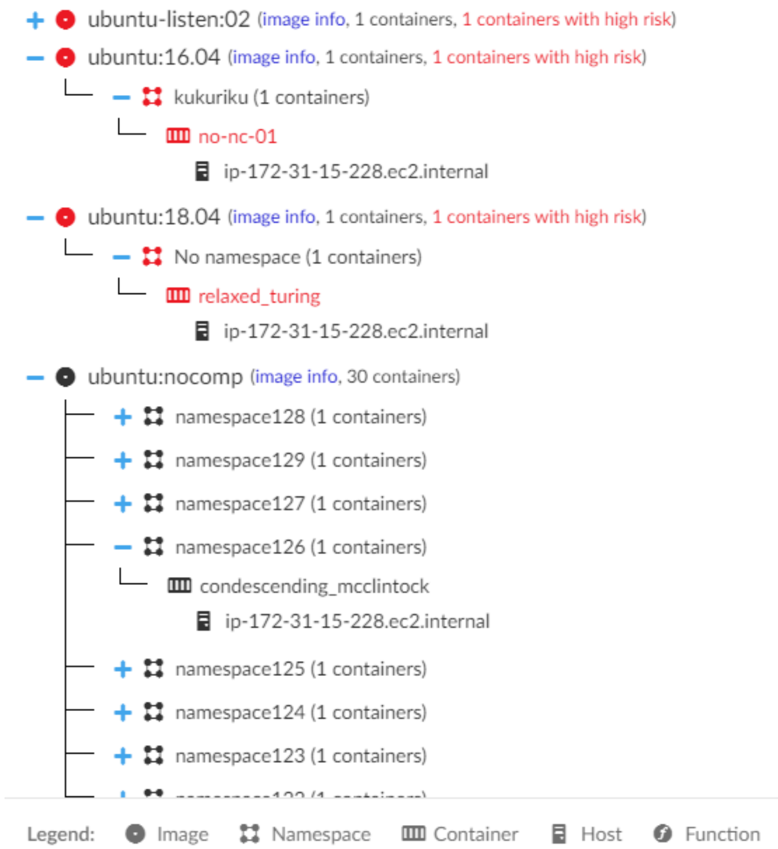
  ```
  redis (main process) CVE-XXX, CVE-XXX
  |- libssl (dependent package) CVE-XXX, CVE-XXX
  |- libzip (dependent package) CVE-XXX, CVE-XXX
  ```

  For more details, see scan reports.

Risk trees

Risk trees lists all the images, namespaces, containers, and hosts that are vulnerable to a specific CVE. Risk trees are useful because they show you how you are exposed to a given vulnerability. Because Prisma Cloud already knows which vulnerabilities impact which packages, which packages are in which images, which containers are derived from which images, which containers run in which namespaces, and which hosts run which containers, we can show you the full scope of your exposure to a vulnerability across all objects in your environment.

For each top ten vulnerability, Prisma Cloud shows you a vulnerability risk tree. To see the vulnerability tree for a given CVE, click on the corresponding row in the top ten table to open a detailed CVE assessment dialog.



You can also generate a risk tree for any arbitrary CVE in your environment by entering the CVE ID into the search bar at the top of the page, then clicking on the result in the table to open a detailed CVE assessment dialog.

Recalculating statistics

Statistical data is calculated every 24 hours. You can force Console to recalculate the statistics for the current day with the latest data by clicking the **Refresh** button in the top right of Vulnerability Explorer. You must rescan each resource such as deployed images, registries, hosts, and functions before a refresh. The **Refresh** button has a red marker when new data is available to be crunched.

The Vulnerability Explorer can not be refreshed when filters are applied. To continue with the ***Refresh*** *option, you need to remove the filters.*

Vulnerability management rules

Edit on GitHub

Vulnerability policies are composed of discrete rules. Rules declare the actions to take when vulnerabilities are found in the resources in your environment. They also control the data surfaced in Prisma Cloud Console, including scan reports and Radar visualizations.

Rules let you target segments of your environment and specify actions to take when vulnerabilities of a given type are found. For example:

Block images with critical severity vulnerabilities from being deployed to prod environment hosts

When there is no matching rule for vulnerability scanning on specific resources such as an image or a function, Prisma Cloud generates alerts on all vulnerabilities that are found.

There are separate vulnerability policies for containers, hosts, and serverless functions. Host and serverless rules offer a subset of the capabilities of container rules, the big difference being that container rules support blocking.

Creating vulnerability rules

Prisma Cloud ships with a simple default vulnerability policy for containers, hosts, and serverless functions. These policies have a rule named *Default - alert all components*, which sets the alert threshold to low. With this rule, all vulnerabilities in images, hosts, and functions are reported.

As you build out your policy, you'll create rules that filter out insignificant information, such as low severity vulnerabilities, and surface vital information, such as critical vulnerabilities.

Rule order is important. Prisma Cloud evaluates the rule list from top to bottom until it finds a match based on the object filters.

By default, Prisma Cloud optimizes resource usage by only scanning images with running containers. Therefore, you might not see a scan report for an image when it's first pulled into your environment unless it's been run. Note that images for short-lived containers are not scanned. To scan all images on the hosts in your environment, go to **Manage > System > Scan**, set **Only scan images with running containers** to **Off**, and click **Save**.

To create a vulnerability rule:

STEP 1 | Open Console.

STEP 2 | Go to **Defend > Vulnerabilities > {Images | Hosts | Functions}**.

STEP 3 | Click **Add rule**.

STEP 4 | Enter a rule name and configure the rule. Configuration options are discussed in the following sections.

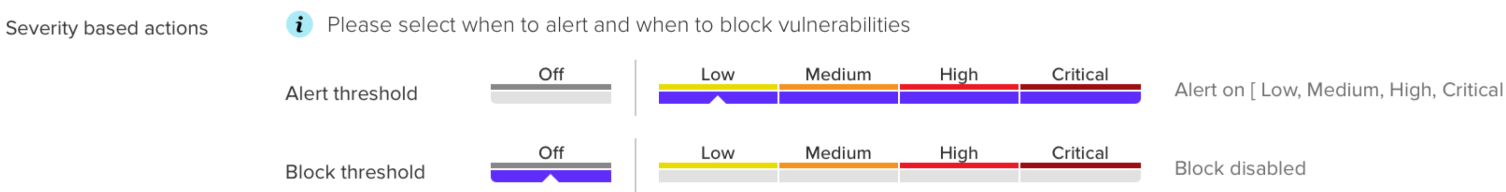
STEP 5 | Click **Save**.

STEP 6 | View the impact of your rule. Go to **Monitor > Vulnerabilities** to view the scan reports.

Severity-based actions

Vulnerability rules let you specify trigger thresholds for alerting and blocking. Alert and block actions let you establish quality gates in the CD segment of your continuous integration (CI) continuous deployment (CD) pipeline.

Alert and block thresholds can be set to different values. The block threshold, however, must always be equal to or greater than the alert threshold.

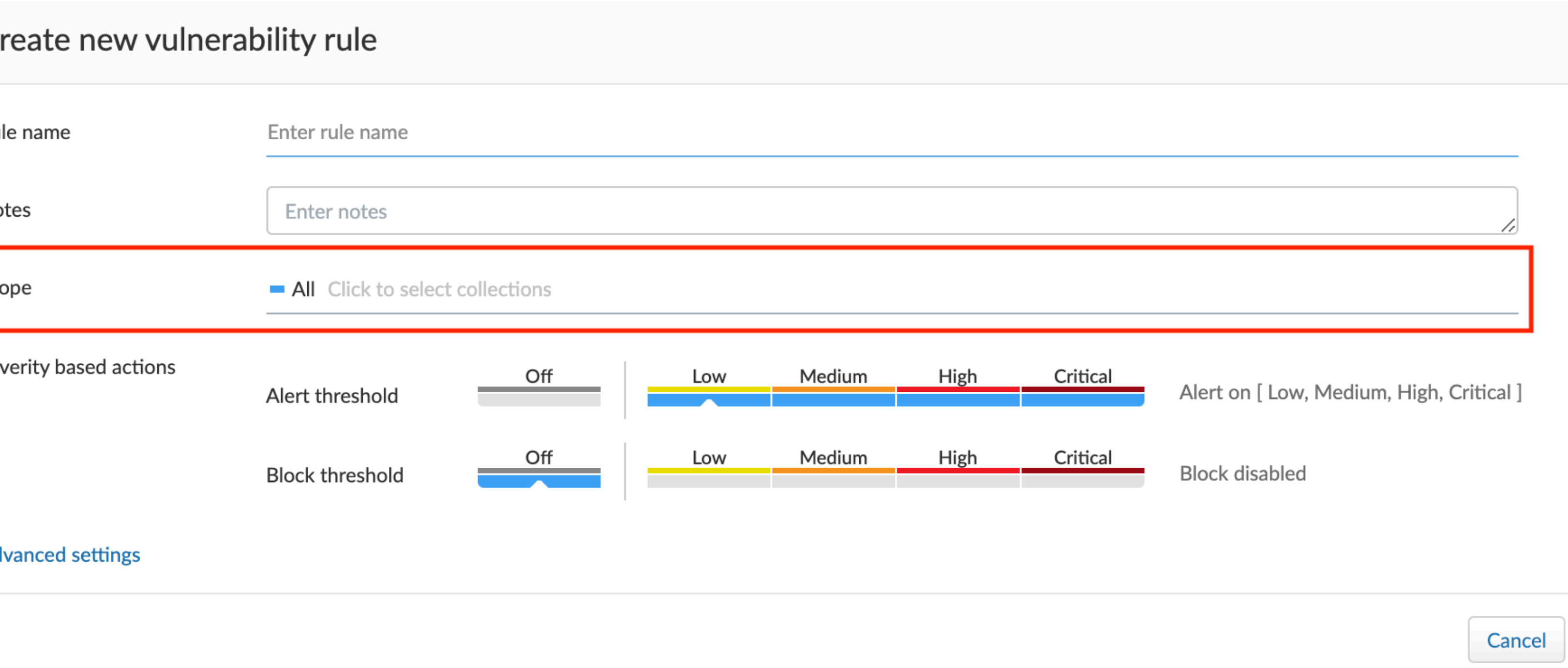


Setting the alert threshold to off allows all vulnerabilities for the resources in scope (as defined by your filters). Practically, resource nodes in Radar turn green (no issues to report), and scan reports are empty (no issues to report).

When you create a blocking rule, Defender automatically installs itself as the final arbiter of all container lifecycle commands. This way, the Defender can assess a Docker command, your current policy, and the status of an image before either forwarding the command to runC for execution, or blocking it all together.

Scope

The scope field lets you target rule to specific resources in your environment. The scope of a rule is defined by referencing one or more collections. By default, the scope is set to the **All** collection, which applies the rule globally. For more information about creating and managing collections, see here.



Vendor fixes

Rules can be applied conditionally depending on whether vendor fixes are available. For example, you could tune your policy to block the deployment of containers with a critical vulnerability **only if** the vulnerable package has an update that resolves the issue. Otherwise, the deployment would be allowed to proceed.

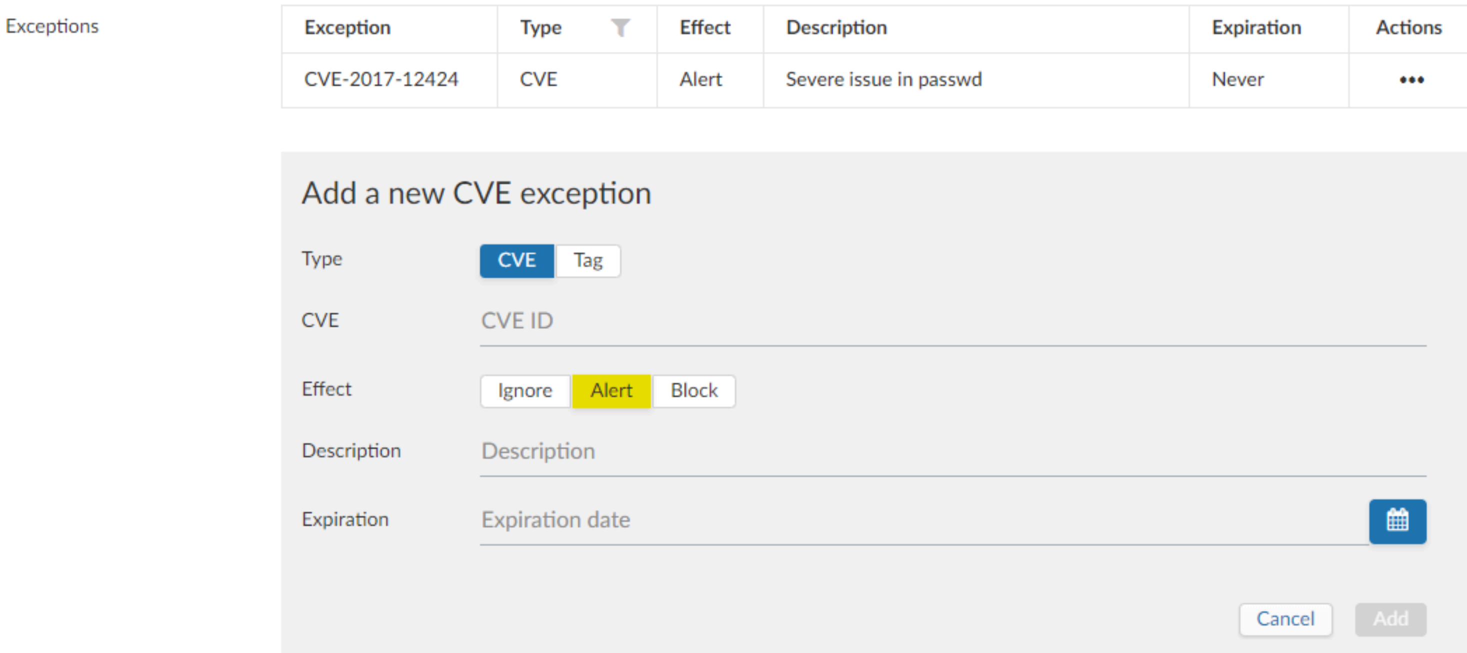
Some vulnerabilities have a vendor status of "Will not fix". This status is applied when vendors don’t intend to resolve a vulnerability because it poses no significant risk to your environment.

Rule exceptions

You can configure Prisma Cloud to:

- Alert or block on specific CVEs or tags (deny).
- Ignore specific CVEs or tags (allow).

Under **Advanced settings**, create a list of vulnerabilities and tags, and specify how the scanner should handle them. Leaving the expiration date blank enforces the action until the CVE or tag is removed from the list. If you set an expiration date, and the current date is later than the expiration date, the scanner ignores the directive. The CVE or tag remains in the list even if it’s expired. It must be manually removed. Notice that for tag exceptions, in case of a conflict (a vulnerability with two tags or more that have different actions in the rule exceptions) there’s no guarantee what action will apply.



Custom terminal output

Prisma Cloud lets you create rules that block access to resources or block the deployment of vulnerable containers. For example, you might create a rule that blocks the deployment of any image that has critical severity vulnerabilities. By default, when you try to run a vulnerable image, Prisma Cloud returns a terse response:

```
$ docker run -it ubuntu:14.04 sh
docker: Error response from daemon: [Prisma Cloud] Image operation
 blocked by policy: (sdf), has 44 vulnerabilities, [low:25
 medium:19].
```

To help the operator better understand how to handle a blocked action, you can enhance Prisma Cloud's default response by:

- Appending a custom message to the default message. For example, you could tell operators where to go to open a ticket.
- Configuring Prisma Cloud to return an itemized list of compliance issues rather than just a summary. This way, the operator does not need to contact the security team to determine which issues are preventing deployment. They are explicitly listed in the response.

When terminal output verbosity is set to **Detailed**, the response looks as follows:

```
$ docker run -it ubuntu:14.04 sh
docker: Error response from daemon: [Prisma Cloud] Image operation
 blocked by policy: (sdf), has 44 vulnerabilities, [low:25
 medium:19].
Image          ID        CVE              Package    Version
 Severity    Status
=====          ==        ===              =======    =======
 ========    ======
```



```
ubuntu:14.04   4333f1   CVE-2017-2518   sqlite3   3.8.2-1ubuntu2.1
 medium     deferred
ubuntu:14.04   4333f1   CVE-2017-6512   perl      5.18.2-2ubuntu1.1
 medium     needed
.
.
.
```

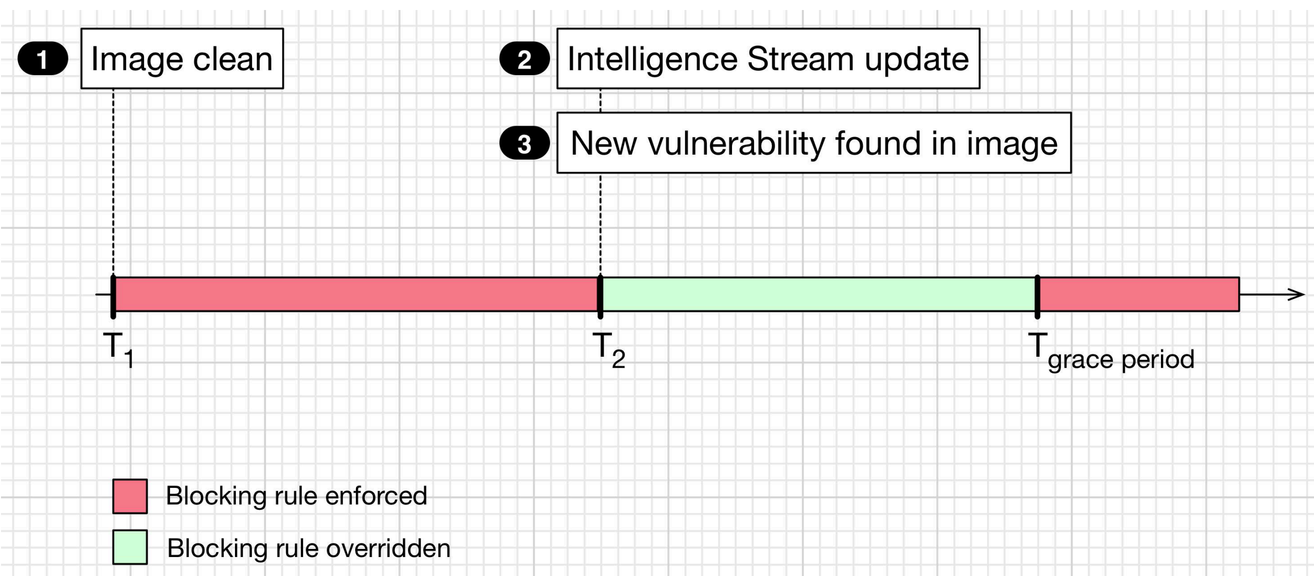
Grace period

Grace periods temporarily override the blocking action of a rule when new vulnerabilities are found. Grace periods give you time to address a vulnerability without compromising the availability of your app. You can configure a uniform grace period for all severities or provide different settings for each severity.

When grace periods are configured, alerts trigger as normal, notifying you that a vulnerability exists in your environment. The block action is suppressed for the number of days specified, giving you time to mitigate the vulnerability.

The start time for the grace period is the date the vulnerability was fixed. The end time is the fixed date plus the number of days configured for the grace period. If there is no fix for the vulnerability, then the start time for the grace period is the date the vulnerability was published/ disclosed.

The following diagram shows how Prisma Cloud Defender responds to a vulnerability discovered in your environment. Assume you have a vulnerability rule that blocks the deployment of any image with critical vulnerabilities, and the grace period is 30 days.



- T_1 — The image has passed the security gates in your CI pipeline. It has no critical vulnerabilities, so it's pushed to the registry.
- T_1 - T_2 — The orchestrator runs the image in your cluster. The image has no critical vulnerabilities, so Defender allows it to run.
- T_2 — Prisma Cloud Intelligence Stream acquires new threat data that identifies a critical vulnerability in the image. The package vendor released a fix as soon as the vulnerability was

disclosed. In the next scan (by default, scans run every 24 hours), Prisma Cloud reports the vulnerability, and raises an alert if alerts are configured in the vulnerability rule.

- T_2 - T_{grace_period} — Prisma Cloud temporarily overrides the block rule, while the dev team addresses the vulnerability. The orchestrator can continue to pull copies of the image into your environment and run it.
- T_{grace_period} — Grace period expires. If the vulnerability has not been fixed yet, Prisma Cloud blocks any new deployments of the image from this time forward.

Grace periods are a policy setting that's available for all components that enforce vulnerability policy, namely Defender, twistcli, and the Jenkins plugin. In order to surface the issue as early as possible in the development lifecycle, you can specify a grace period in the CI pipeline. For example, this control would let you fail image builds that have critical vulnerabilities that were fixed over 30 days ago.

Configure grace period

The following procedure describes how to configure grace periods for blocking actions:

STEP 1 | In Console, go to **Defend > Vulnerabilities > Images > Deployed**.

STEP 2 | Select an existing rule or create a new rule with the **Add rule** button.

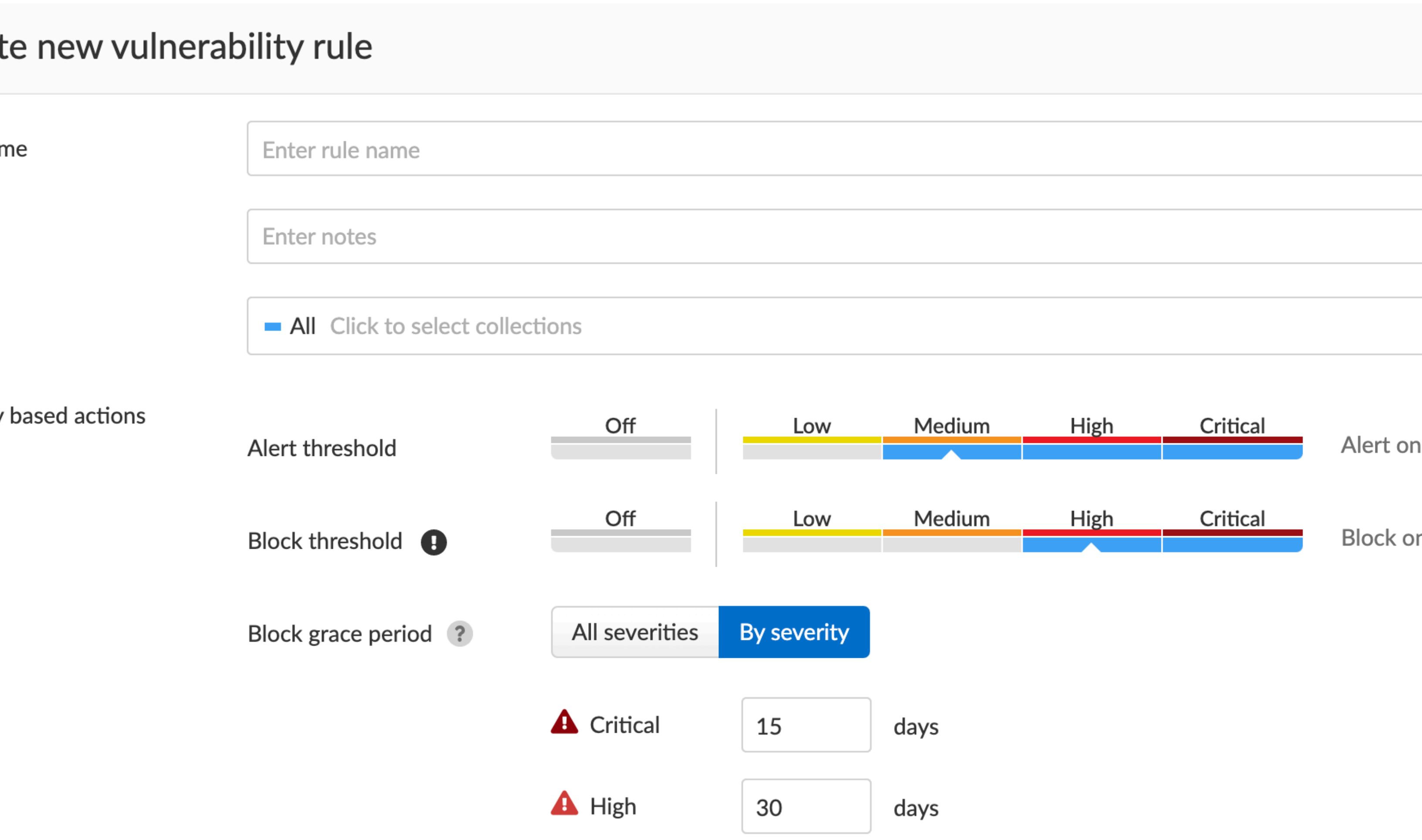
STEP 3 | Enter a rule name, notes, and scope.

STEP 4 | Under **Severity based actions**:

1. Select the desired **Alert threshold**
2. Select the desired **Block threshold**.

   The block threshold must be equal to or greater than the alert threshold. You must define a block threshold in order to configure grace period.
3. Configure the **Block grace period**:

   1. Select whether you would like to define the same grace period for **All severities** or grace period **By severity**.
   2. Specify the number of days. Note that in case of **By severity** grace period you will be able to specify the number of days only for the severities that can be blocked. Values that are not set will be set to 0.



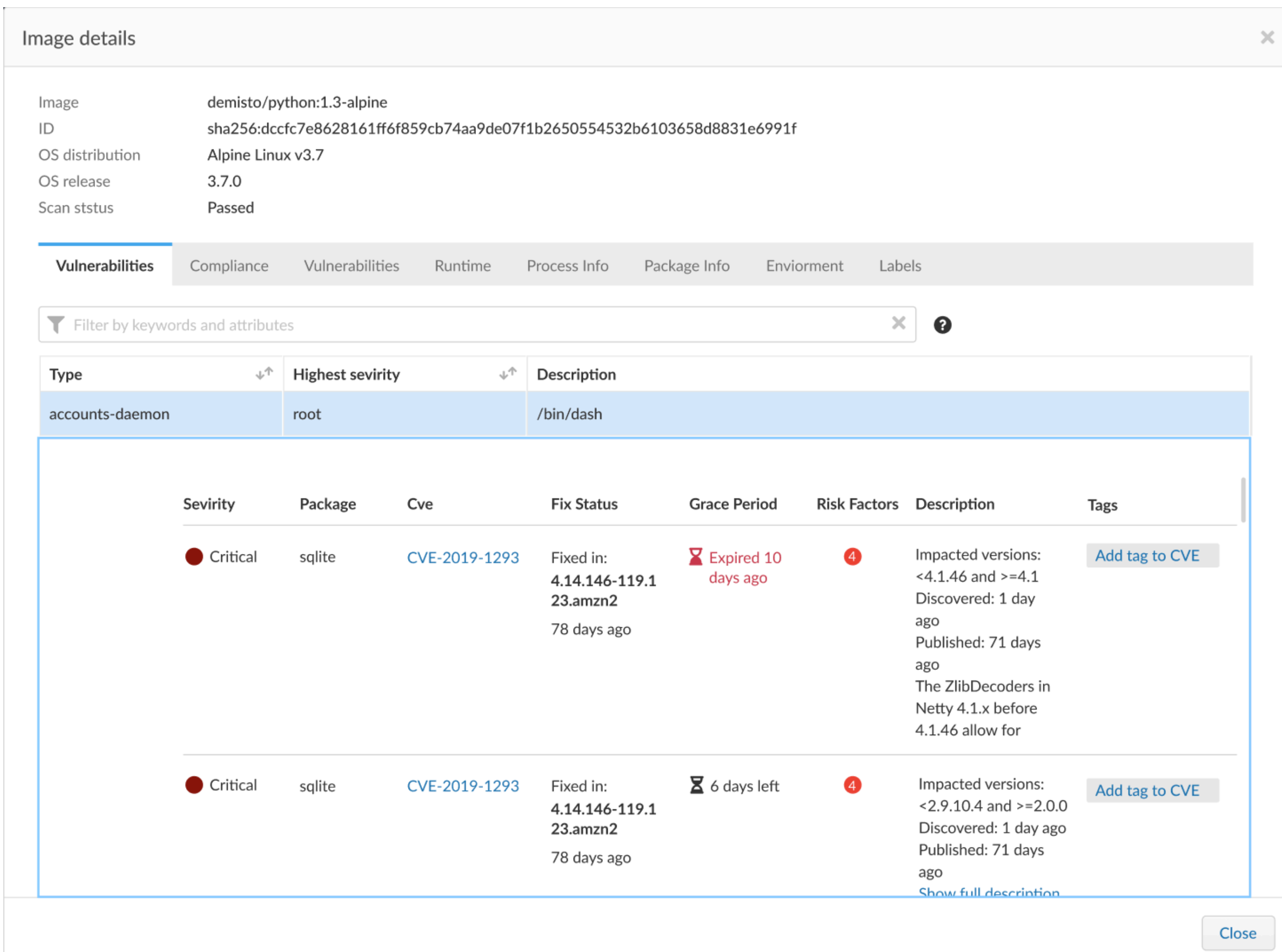
*Use the same procedure to configure grace periods to fail builds in your CI/CD pipeline. To configure CI/CD pipeline vulnerability scanning rules, go to **Defend > Vulnerabilities > Images > CI**.*

Elapsed time

All scan reports show whether a vulnerability has been fixed (fix status) and when it was fixed (fix date), and the time remaining in the grace period. Scan reports are available from the:

- Console UI.
- Console UI as a CSV download.
- API (JSON or CSV).
- Jenkins plugin.
- twistcli.

The following example screenshot shows how the status of grace periods is displayed. Grace periods are either still in force or expired. For grace periods in force, the number of days remaining in the grace period is displayed. For grace periods that have expired, the number of days since they expired is displayed. Scan reports for running images can be retrieved from **Monitor > Vulnerabilities > Images > Deployed**.



The following screenshot shows how the data is represented in the CSV scan report:

L	M	N	O	P	Q	R	S
Packages	Source Package	Package Version	Package License	CVSS	Fix Status	Fix Date	Grace Da
unbound-libs		1.4.20-34.el7	BSD	7.5	fixed in 1.6.6-5.el7_8	2020-06-22 0:00:00	
unbound-libs		1.4.20-34.el7	BSD	7.5	fixed in 1.6.6-4.el7_8	2020-06-08 0:00:00	_2

Blocking based on vulnerability severity

This example shows you how to create and test a rule that blocks the deployment of images with critical or high severity vulnerabilities.

STEP 1 | In Console, go to **Defend > Vulnerabilities > Images**.

STEP 2 | Click **Add rule**.

1. Enter a rule name, such as **my-rule**.
2. In the **Severity based actions** table, set both the **Alert threshold** and **Block threshold** to **High**.
3. Target the rule to a very specific image. In the **Images** filter, delete the wildcard, and enter **nginx***.
4. Click **Save**.

STEP 3 | Validate your policy by pulling down the nginx image and running it.

1. SSH to a host protected by Defender.
2. Pull the nginx:1.14 image.

```
$ docker pull nginx:1.14
```

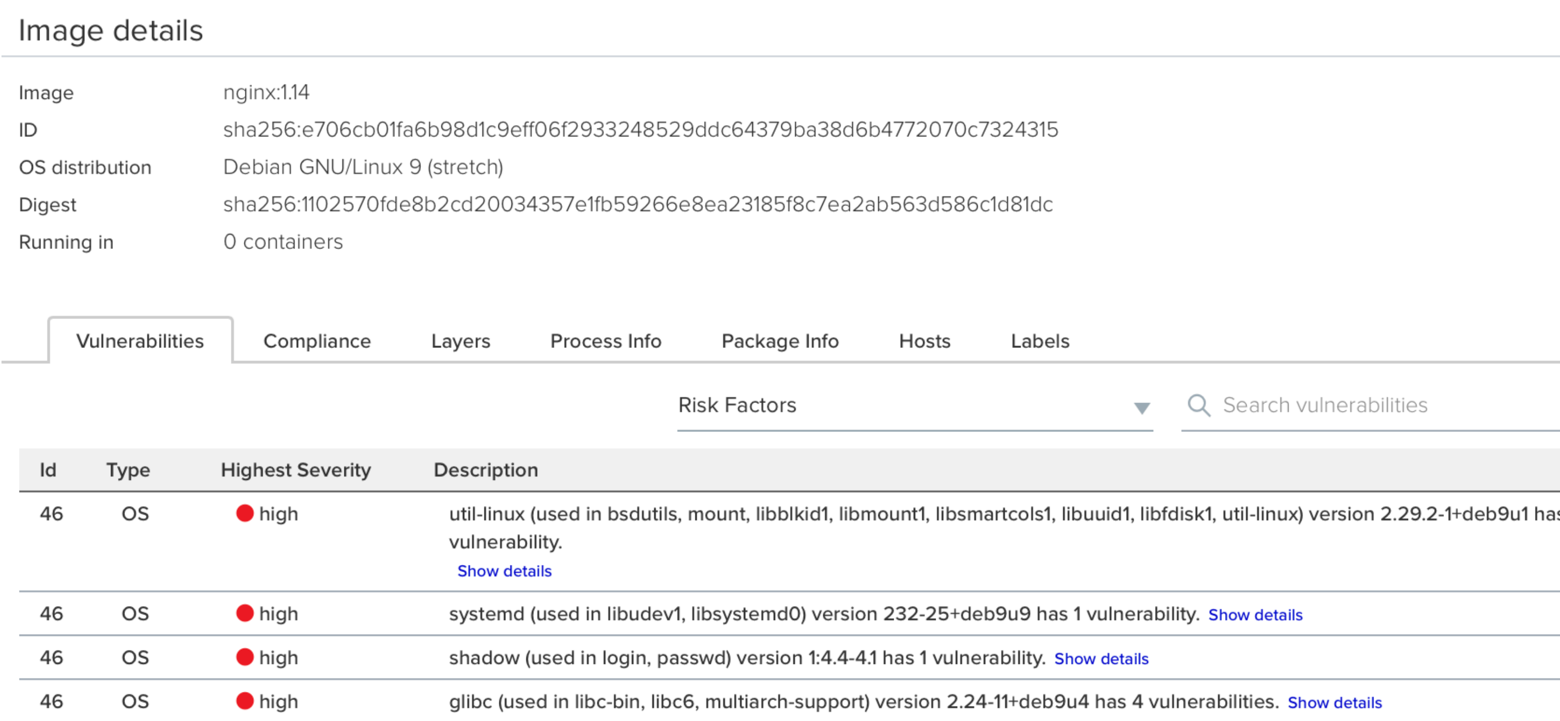
3. Run the nginx image.

```
$ docker run -it nginx:1.14 /bin/sh
```

```
docker: Error response from daemon: oci runtime error: [Prisma
 Cloud] Image operation blocked by policy: my-rule, has 7
 vulnerabilities, [high:7].
```

4. Review the scan report for nginx:1.14. Go to **Monitor > Vulnerabilities > Images**, and click on the entry for nginx:1.14. You'll see a number of high severity vulnerabilities.

   By default, Prisma Cloud optimizes resource usage by only scanning images with running containers. Therefore, you won't see a scan report for ngninx until it's run.



5. Review the audit (alert) for the block action. Go to **Monitor > Events**, then click on **Docker**.

Blocking specific CVEs

This example shows you how to create and test a rule that blocks images with a specific CVE.

STEP 1 | In Console, go to **Defend > Vulnerabilities > Images**.

STEP 2 | Click **Add rule**.

1. Enter a rule name, such as **my-rule2**.
2. Click **Advanced settings**.
3. In **Exceptions**, click **Add Exception**.
4. In **CVE**, enter **CVE-2018-8014**.

   *You can find specific CVE IDs in the image scan reports. Go to **Monitor > Vulnerabilities > Images**, select an image, then click **Show details** in each row.*

5. In **Effect**, select **Block**.
6. Click **Add**.
7. Click **Save**.

STEP 3 | Try running an image with the CVE that you've explicitly denied.

```
$ docker run -it imiell/bad-dockerfile:latest /bin/sh
docker: Error response from daemon: oci runtime error: [Prisma
 Cloud] Image operation blocked by policy: my-rule2, has specific
 CVE CVE-2018-8014
```

Ignoring specific CVEs

Follow the same procedure as above, but set the action to **Ignore** instead of **Block**. This will allow any CVE ID that you've defined in the rule, and lets you run images containing those CVEs in your environment.

Search CVEs

Edit on GitHub

Common Vulnerabilities and Exposures (CVE) is a system for referencing publicly known vulnerabilities by identifiers. The goal of the system is to make it easier to share vulnerability data across stakeholders, including software vendors, tool vendors, security practitioners, and end users.

A *CVE entry* describes a specific known vulnerability. Each CVE entry has an identifier, such as CVE-2020-1234. A *CVE entry* is colloquially known as a *CVE*, and it's security practitioner parlance for a publicly disclosed vulnerability.

Searching for a specific CVE

You can determine if Prisma Cloud offers coverage for a specific CVE by using the search interface in Console. The CVE ID syntax is:

```
CVE-YYYY-NNNN
```

Where:

- **CVE** --

  CVE-ID prefix.
- **YYYY** --

  Calendar year.
- **NNNN** --

  Numeric digits. This field has a variable length, but the minimum length is four digits.

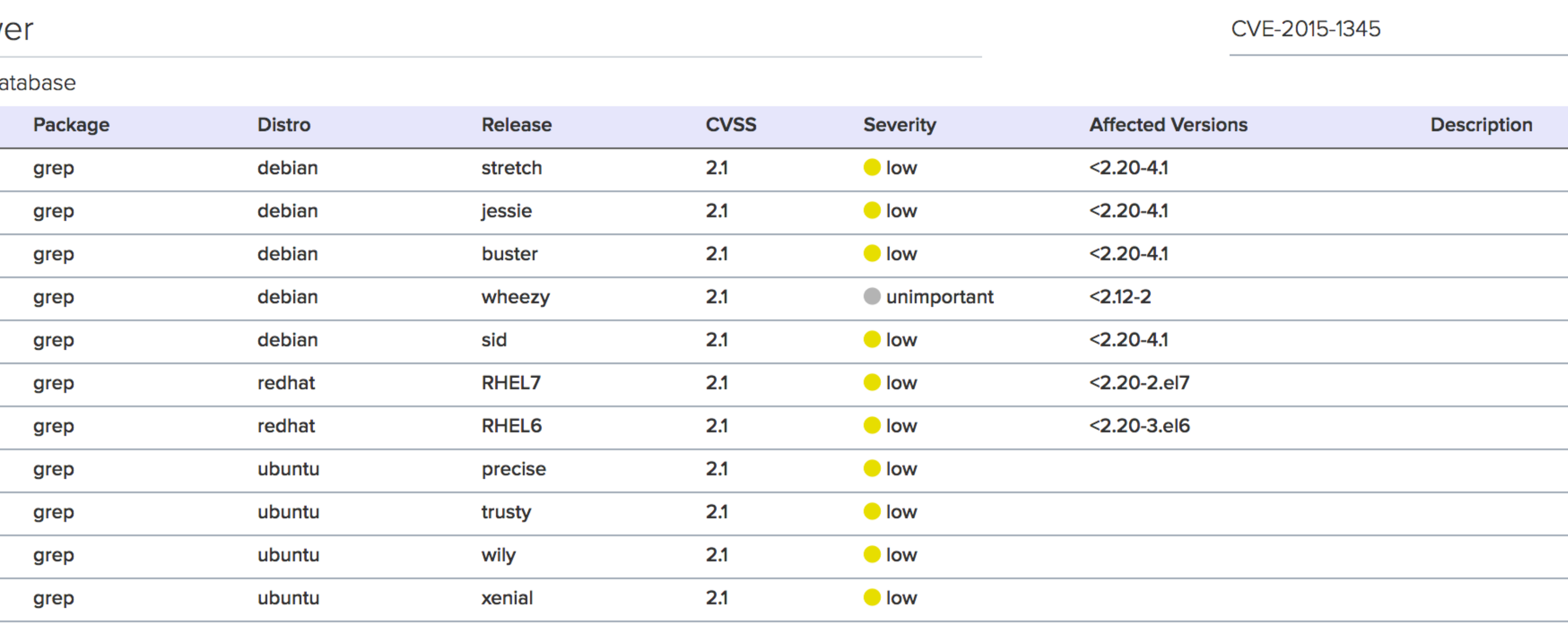
To search for a specific vulnerability:

STEP 1 | Open Console, then go to **Monitor > Vulnerabilities > CVE Viewer**.

STEP 2 | In the query text box in the top right, enter a CVE ID.

For example, enter **CVE-2015-1345**.

If Prisma Cloud has coverage for the queried vulnerability, details are listed in the results table.



Package	Distro	Release	CVSS	Severity	Affected Versions	Description
grep	debian	stretch	2.1	low	<2.20-4.1	
grep	debian	jessie	2.1	low	<2.20-4.1	
grep	debian	buster	2.1	low	<2.20-4.1	
grep	debian	wheezy	2.1	unimportant	<2.12-2	
grep	debian	sid	2.1	low	<2.20-4.1	
grep	redhat	RHEL7	2.1	low	<2.20-2.el7	
grep	redhat	RHEL6	2.1	low	<2.20-3.el6	
grep	ubuntu	precise	2.1	low		
grep	ubuntu	trusty	2.1	low		
grep	ubuntu	wily	2.1	low		
grep	ubuntu	xenial	2.1	low		

Allow a CVE

Allowing CVEs is done directly as a policy.

Scan reports

Edit on GitHub

Prisma Cloud scans all Docker images on all hosts that run Defender. After Defender is installed, it automatically starts scanning images on the host. After the initial scan, subsequent scans are triggered:

- Periodically, according to the scan interval configured in Console. By default, images are scanned every 24 hours.
- When new images are created, pushed, or pulled onto the host.
- When images change.
- When scans are forced with the **Scan** button in Console.

Defender scans Docker images for:

- Published Common Vulnerabilities and Exposures (CVEs).
- Vulnerabilities from misconfigurations.
- Malware
- Zero day vulnerabilities
- Compliance issues
- Secrets

The Prisma Cloud Intelligence Stream keeps Console up to date with the latest vulnerabilities. The data in this feed is distributed to your Defenders, and employed in subsequent scans.

Through Console, Defender can be extended to scan images for custom components. For example, you can configure Defender to scan for an internally developed library named libexample.so, and set a policy to block a container from running if version 1.9.9 or earlier at installed. For more information, see Scanning custom components.

View image scan reports

Review the health of all images in your environment.

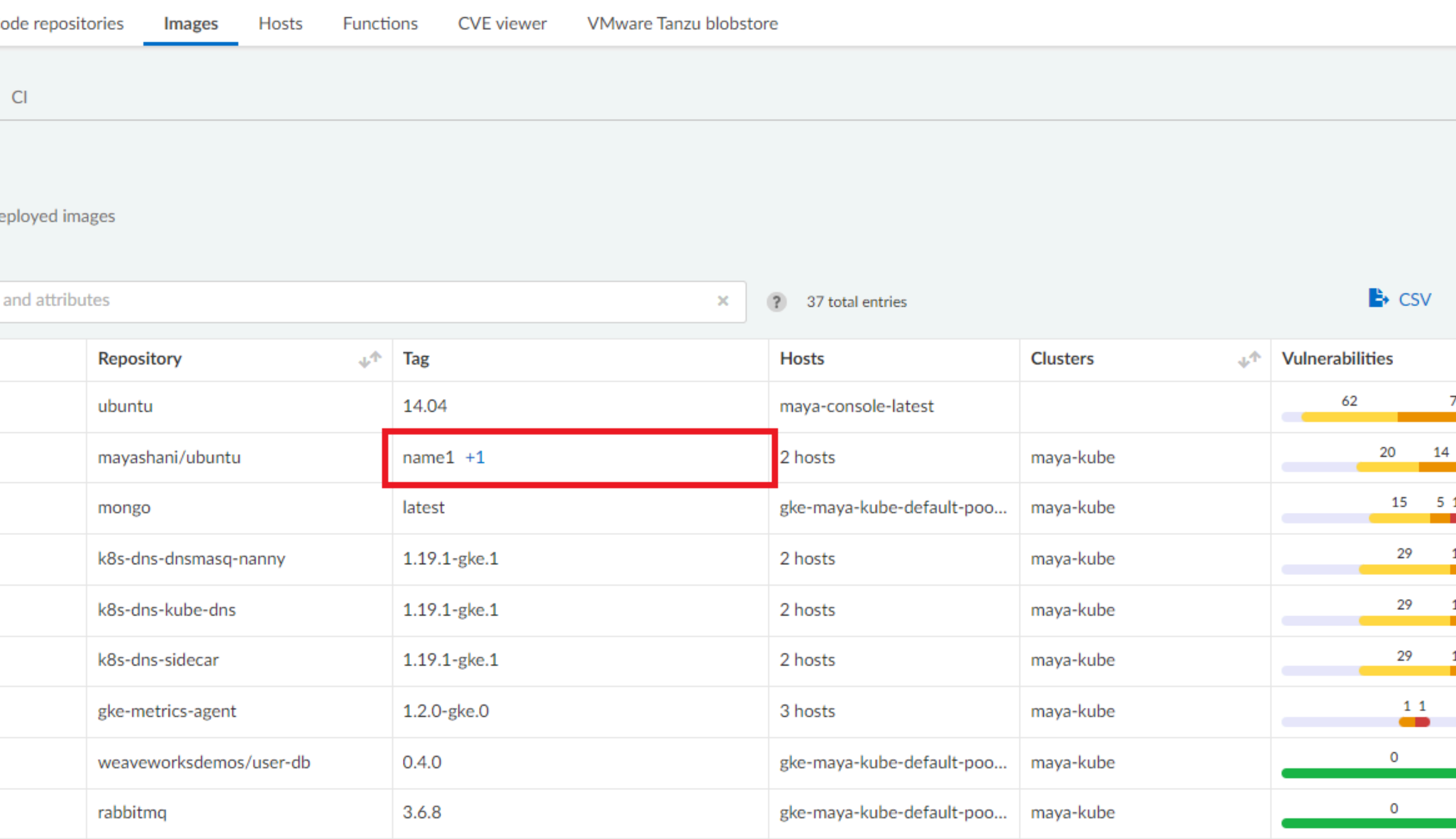
Sorting the table on vulnerability severity as based on data from the last scan. If you update your vulnerability policy with a different alert threshold, rescan your images if you want to be able to sort based on your new settings.

STEP 1 | Open Console, then go to **Monitor > Vulnerabilities > Images**.

The table summarizes the state of each image in your environment.

All vulnerabilities identified in the last image scan can be exported to a CSV file by clicking the **CSV** button in the top left of the page.

In case multiple images share the same image ID, but with different tags on different hosts, then these will be shown using +<Num> in the Tag column, as can be seen in the screenshot below.



STEP 2 | Click on an image report to open a detailed report.

STEP 3 | Click on the **Vulnerabilities** tab to see all CVE issues.

CVE vulnerabilities are accompanied by a brief description. Click **Show details** for more information, including a link to the report on the National Vulnerability Database.

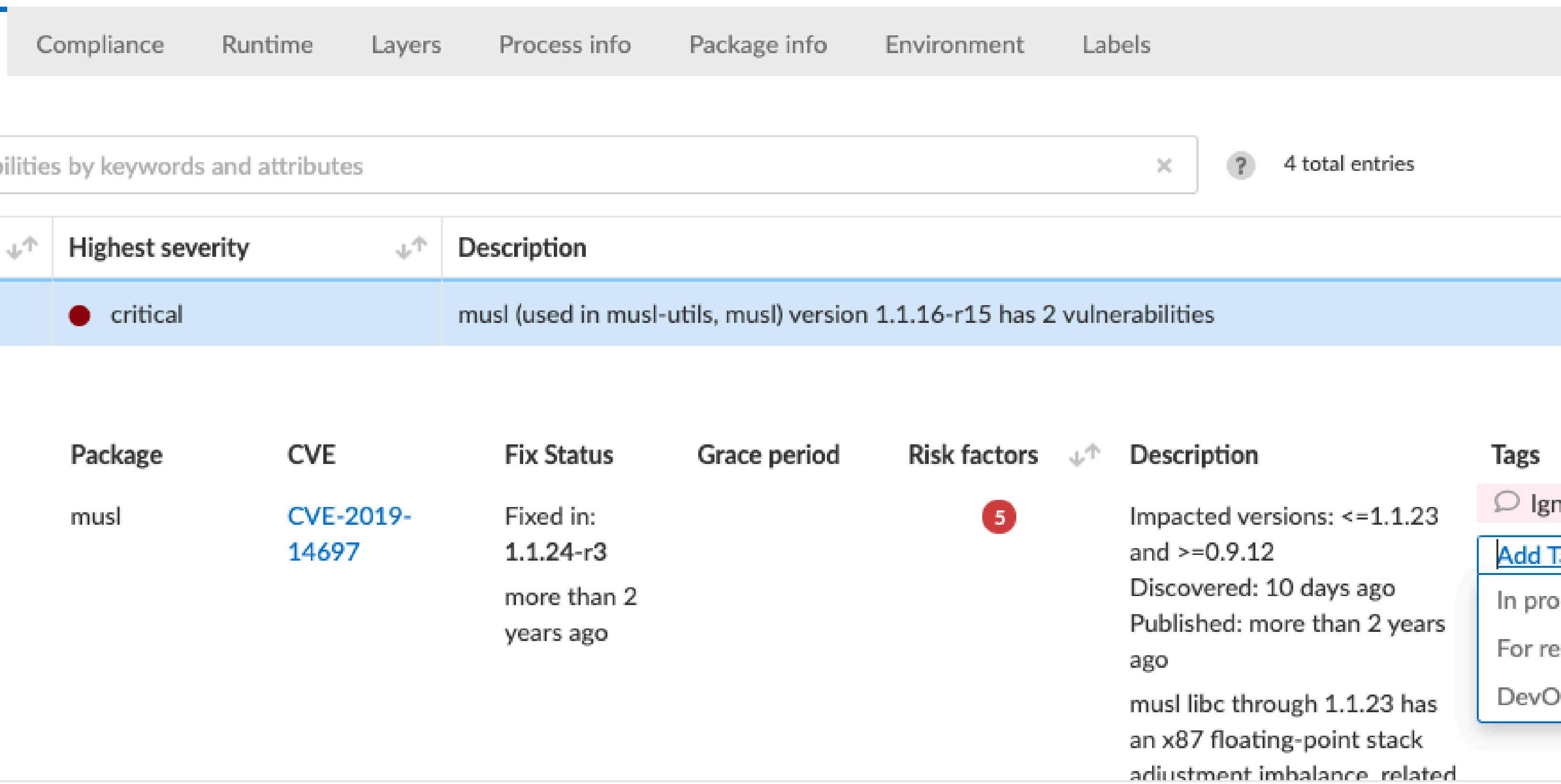
*The **Vendor Status** column contains terms such as 'deferred', 'fixed in…', and 'open'. These strings are imported directly from the vendors' CVE databases. They are not Prisma Cloud-specific.*

Id	Type	Highest Severity	Description
411	Binary	critical	node version 0.10.41 has 18 vulnerabilities. Show details
49	nodejs	critical	tar version 1.0.1 has 1 vulnerability. Hide details

Severity	Package	CVE	Vendor Status	Risk Factors	Description
critical	tar	CVE-2015-8860	fixed in >=2.0.0	4	**Impacted versions: <2.0.0** The tar package before 2.0.0 for Node.js allows remote attackers to write to arbitrary files via a symlink attack in an archive.

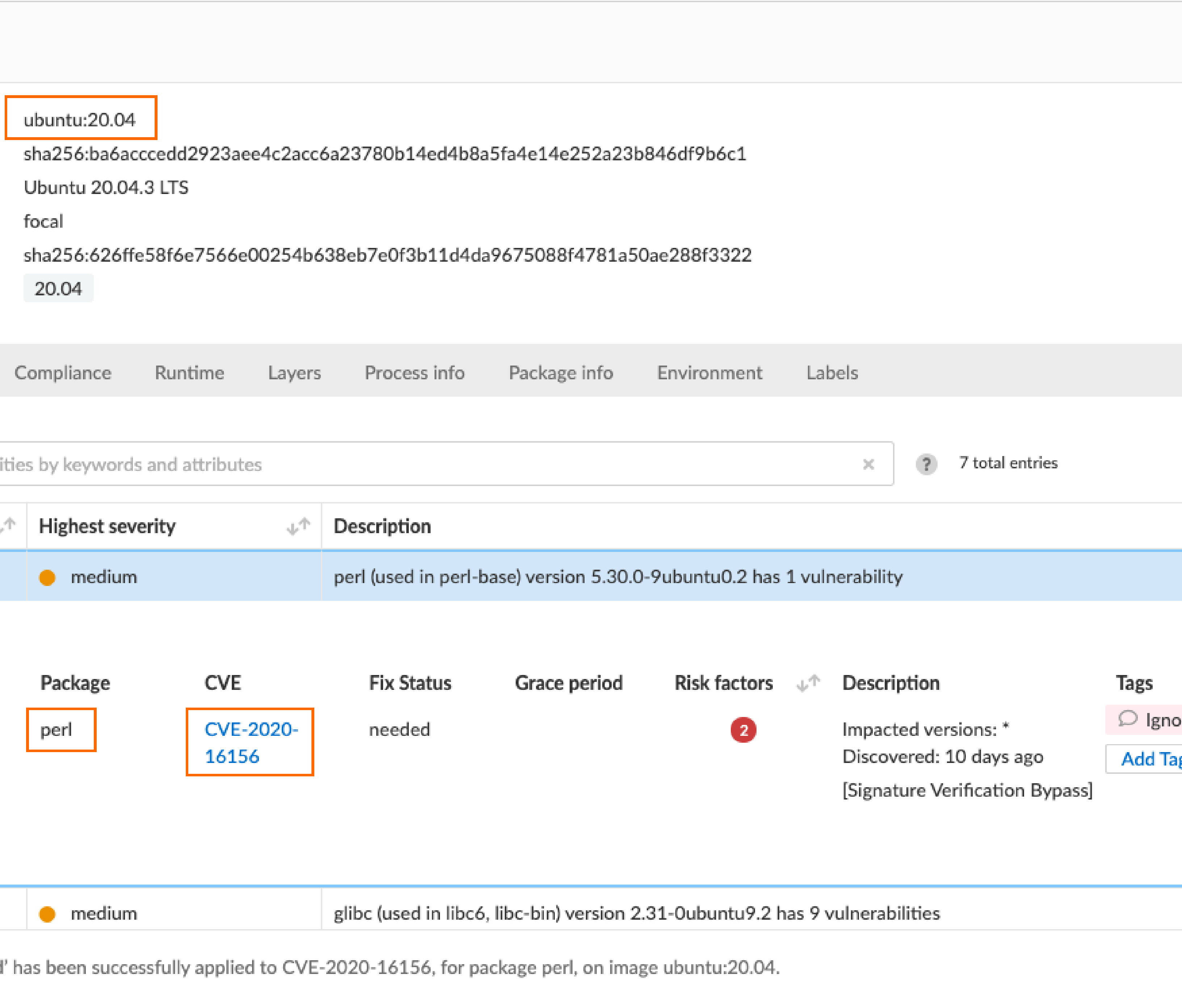
Tagging vulnerabilities

To help you manage and fix the vulnerabilities in your environment, you can assign tags to each vulnerability. The list of available tags is defined under **Manage > Collections and Tags > Tags > Tag definition** (see Tag definition). To assign a tag to a vulnerability, click on the **Add tags to CVE** action in the **Tags** column.



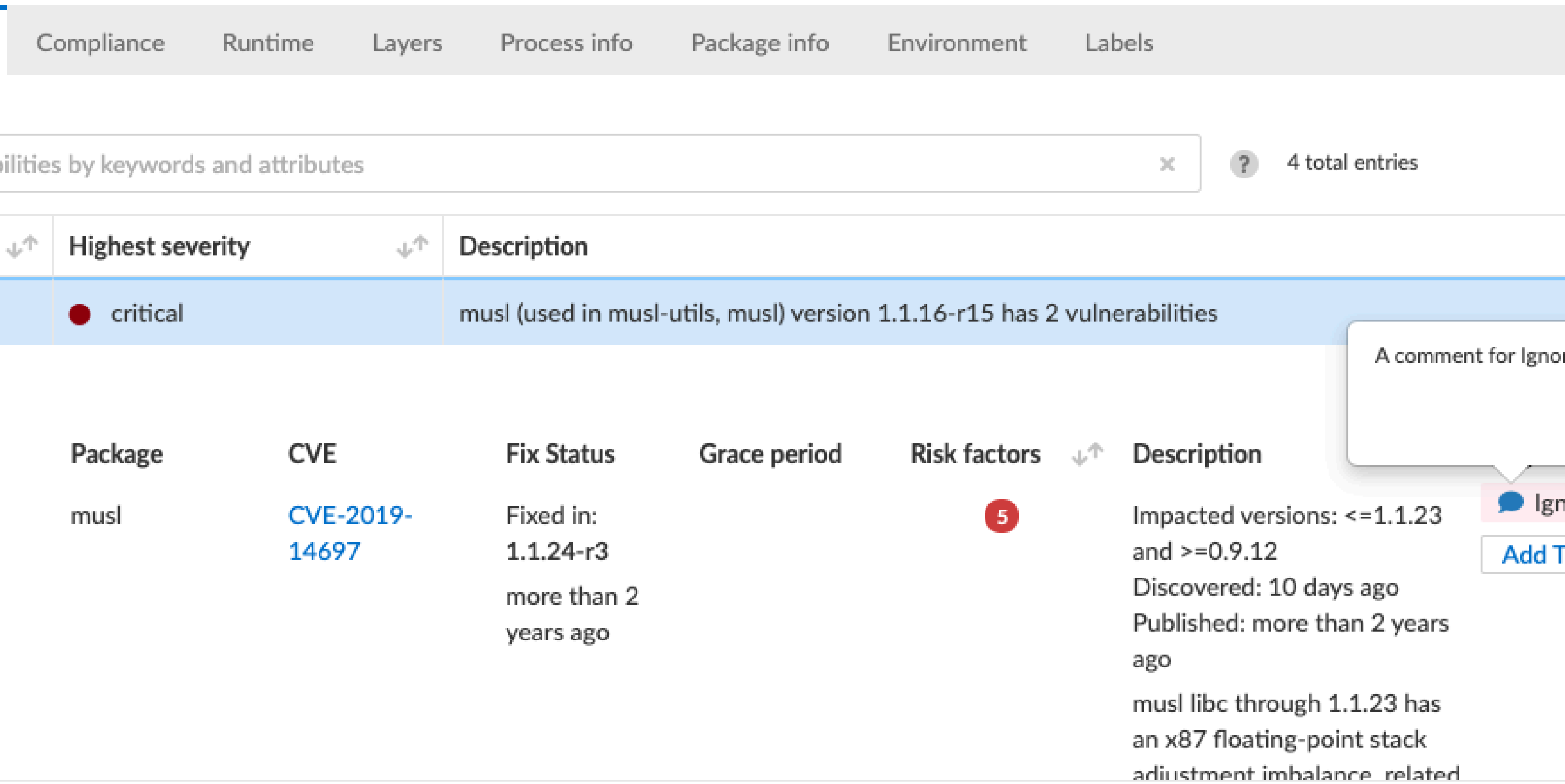
Tagging a vulnerability will apply by default to the CVE ID, package, and resource you assigned the tag from. You can granularly adjust and extend the tag scope under **Manage > Collections and Tags > Tags > Tag assignment** (see Tag assignment).

For example, assigning a tag from the following scan report, will apply to *CVE-2020-16156*, package *perl* and image *ubuntu:20.04*.

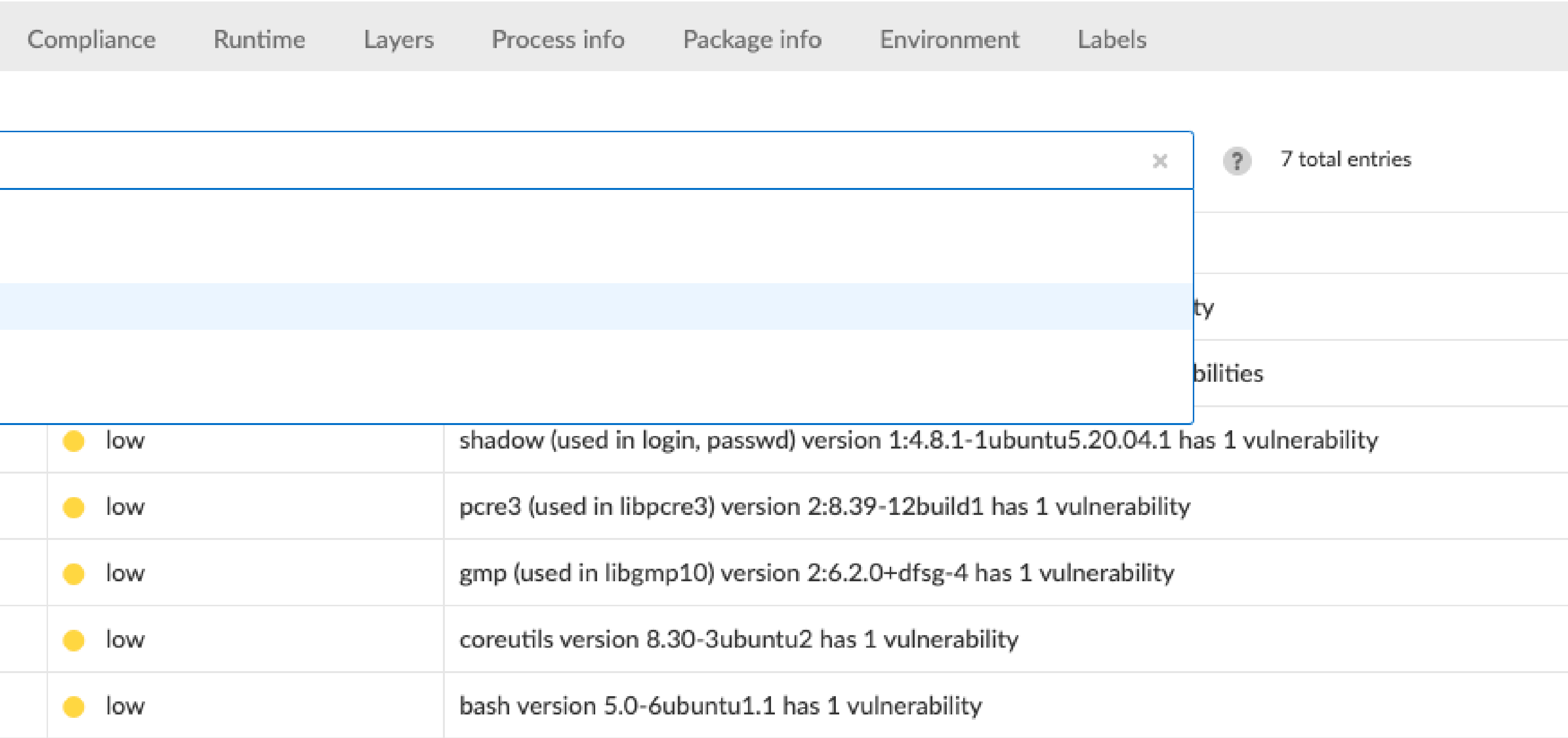


For tags that are not used as policy exceptions, all user roles that can view the scan results and have the Collections and Tags permission, are allowed to assign these tags on CVEs. Assigning tags that are used as policy exceptions is allowed only for Admin, Operator, and Vulnerability Manager user roles. Custom roles aren't allowed to set these tags, regardless of their other permissions.

You can also add comments to each tag assignment, for example, to explain the reason this tag was added. Do it by clicking the comment icon on the left side of the tag.



By default, all vulnerabilities, according to your policy, are listed. However, you can also examine vulnerabilities only with specific tags. Use the drop-down list to filter by tags.



Remove a tag from a vulnerability using the close action available on the tag.

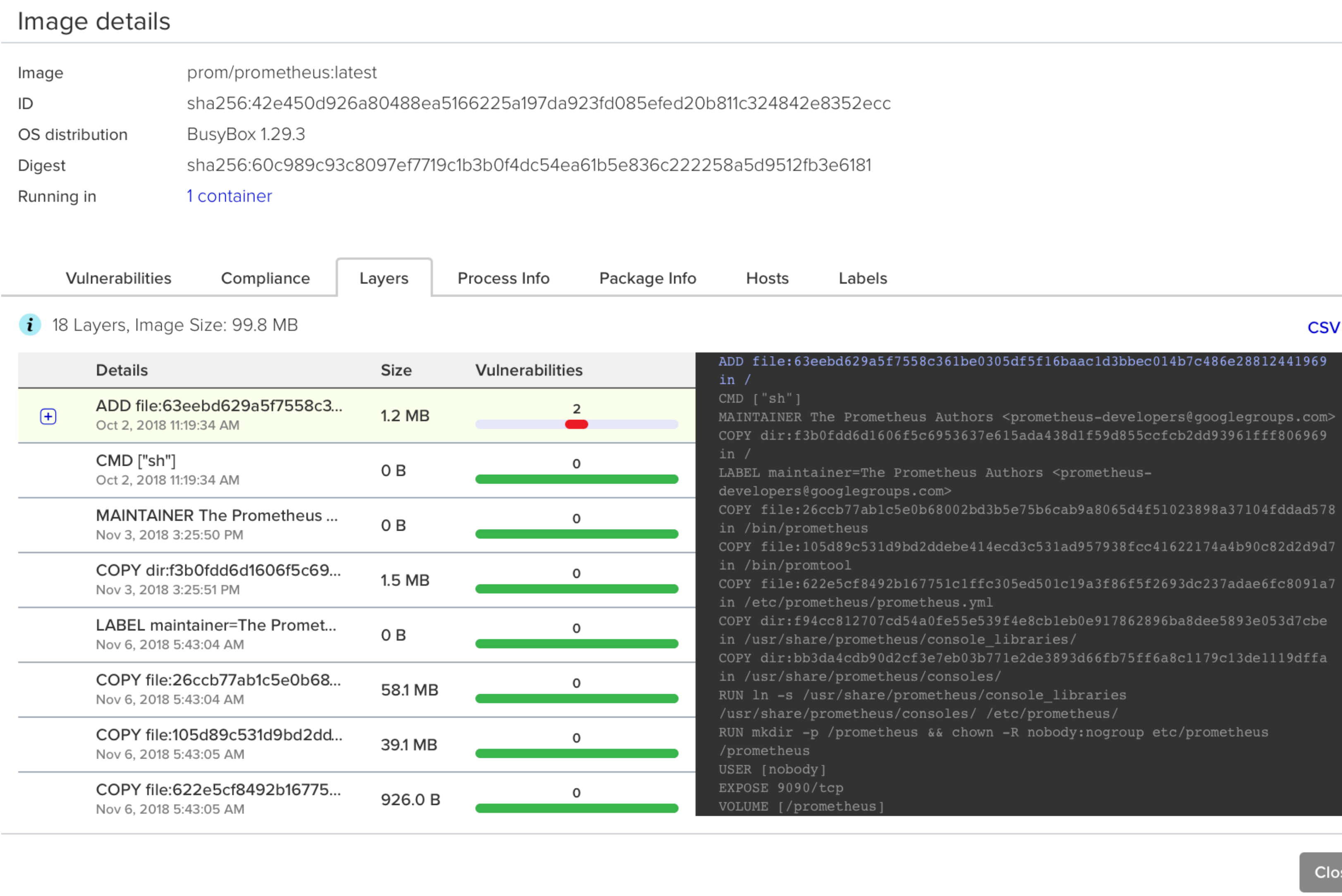
When removing a tag from the scan report, the entire tag assignment is removed, which may be wider than just the single place you remove it from. For example, removing a tag that is applied to image *ubuntu:20.04* by a tag assignment defined for images *ubuntu:**, will remove the entire tag assignment, which means the tag will be removed from all *ubuntu* images.

For more granular tag removal, go to the **Manage > Collections and Tags > Tags > Tag assignment**, and adjust the relevant tag scope.

Per-layer vulnerability analysis

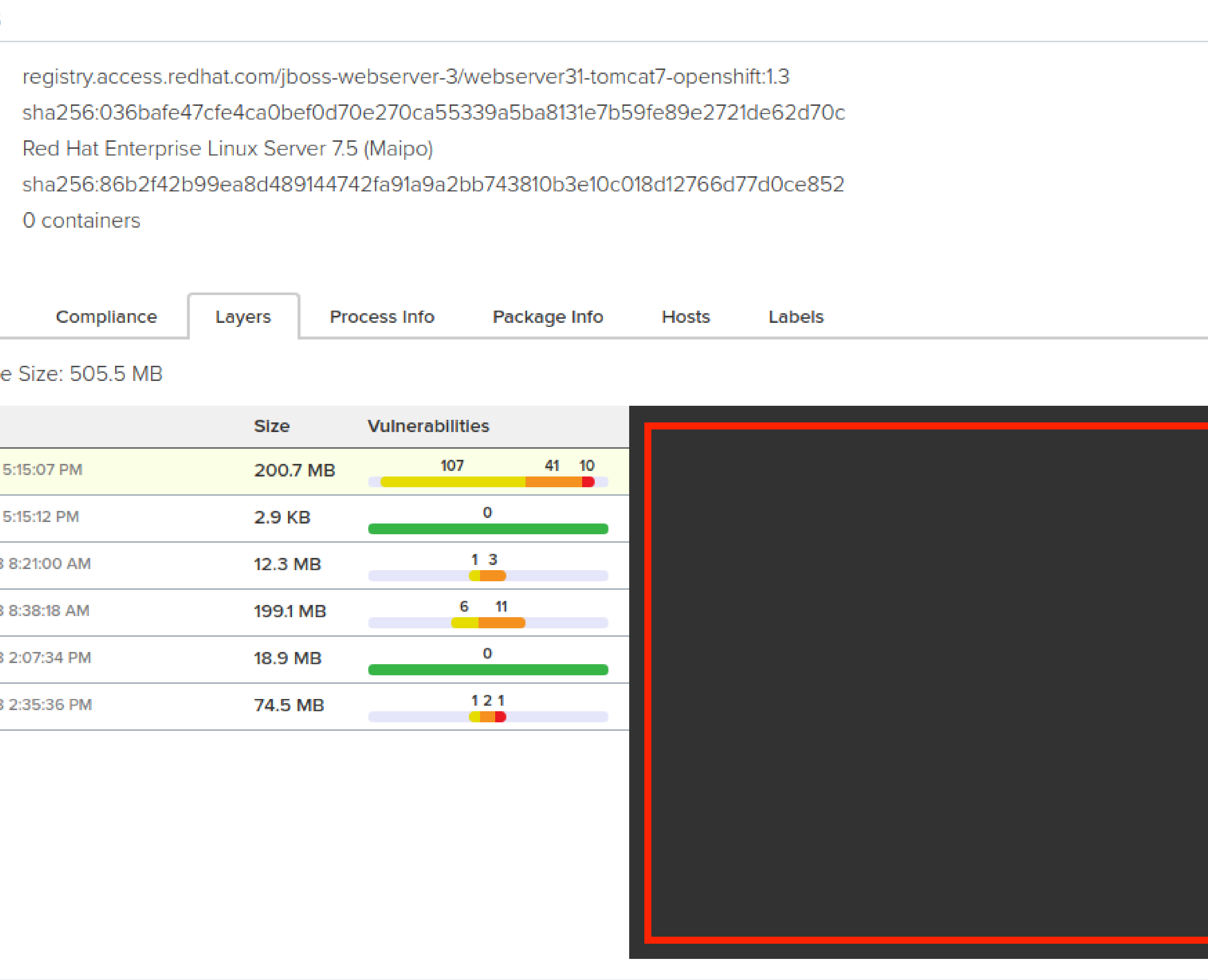
To make it easier to understand how images are constructed and what components have vulnerabilities, Prisma Cloud correlates vulnerabilities to layers. This tool helps you assess how vulnerabilities were introduced into an image, and pick a starting point for remediation.

To see the layer analysis, click on an image to open the scan report, then click the **Layers** tab.



RHEL images

The Prisma Cloud layers tool shows the instructions used to create each layer in an image. RHEL images, however, don’t contain the necessary metadata, so the Prisma Cloud layers tool shows an empty black box.



To validate the required metadata is absent, run *docker history IMAGE-ID* on a non-RHEL image. The *CREATED BY* column is fully populated.

```
mri@omri:/$ docker history 865f6c802d4a
MAGE            CREATED             CREATED BY                                      SIZE                COM
65f6c802d4a     5 months ago        /bin/sh -c echo blah > 4/4                      5B
7f5bfc452c2     5 months ago        /bin/sh -c mkdir /5                             0B
c33f84b4923     5 months ago        /bin/sh -c mkdir /4                             0B
4c497d5c758     9 months ago        /bin/sh -c #(nop)  CMD ["mongod"]               0B
missing>        9 months ago        /bin/sh -c #(nop)  EXPOSE 27017/tcp             0B
missing>        9 months ago        /bin/sh -c #(nop)  ENTRYPOINT ["docker-ent...   0B
missing>        9 months ago        /bin/sh -c #(nop) COPY file:18c5d9b642a89a...   10.4kB
missing>        9 months ago        /bin/sh -c #(nop)  VOLUME [/data/db /data/...   0B
missing>        9 months ago        /bin/sh -c mkdir -p /data/db /data/configd...   0B
missing>        9 months ago        /bin/sh -c set -x  && apt-get update  && a...   278MB
missing>        9 months ago        /bin/sh -c echo "deb http://$MONGO_REPO/ap...   67B
missing>        9 months ago        /bin/sh -c #(nop)  ENV MONGO_VERSION=3.6.4      0B
missing>        9 months ago        /bin/sh -c #(nop)  ENV MONGO_MAJOR=3.6          0B
missing>        9 months ago        /bin/sh -c #(nop)  ENV MONGO_PACKAGE=mongo...   0B
missing>        9 months ago        /bin/sh -c #(nop)  ARG MONGO_REPO=repo.mon...   0B
missing>        9 months ago        /bin/sh -c #(nop)  ARG MONGO_PACKAGE=mongo...   0B
missing>        9 months ago        /bin/sh -c set -ex;  export GNUPGHOME="$(m...   1.16kB
missing>        9 months ago        /bin/sh -c #(nop)  ENV GPG_KEYS=2930ADAE8C...   0B
missing>        9 months ago        /bin/sh -c mkdir /docker-entrypoint-initdb.d    0B
missing>        9 months ago        /bin/sh -c set -ex;   apt-get update;  apt...   2.48MB
missing>        9 months ago        /bin/sh -c #(nop)  ENV JSYAML_VERSION=3.10.0    0B
missing>        9 months ago        /bin/sh -c #(nop)  ENV GOSU_VERSION=1.10        0B
missing>        9 months ago        /bin/sh -c apt-get update  && apt-get inst...   6.42MB
missing>        9 months ago        /bin/sh -c groupadd -r mongodb && useradd ...   330kB
missing>        9 months ago        /bin/sh -c #(nop)  CMD ["bash"]                 0B
missing>        9 months ago        /bin/sh -c #(nop) ADD file:50be6ceb11c382e...   79.2MB
```

Next, run *docker history IMAGE-ID* on a RHEL image. Notice that the *CREATED BY* column is empty.

```
omri@omri:/$ docker history 036bafe47cfe
IMAGE               CREATED             CREATED BY          SIZE                COMMENT
036bafe47cfe        4 months ago                            74.5MB
<missing>           4 months ago                            18.9MB
<missing>           4 months ago                            199MB
<missing>           4 months ago                            12.3MB
<missing>           6 months ago                            2.92kB
<missing>           6 months ago                            201MB               Imported from -
```

Packages in use

Prisma Cloud uses risk scores to calculate the severity of vulnerabilities in your environment. One of the factors in the risk score is called "Package in use", which indicates a package is utilized by running software.

Scan reports have a **Package info** tab, which lists all the packages installed in an image or host. It also shows all active packages, which are packages used by running software.

To see these active packages, open a scan report, open the **Package info** tab, and look at the **Binaries** column (see the **App** column in host scan reports). This column shows what's actually running in the container. For example, the fluent/fluentd:latest container in the following screenshot runs */usr/bin/ruby*. One of the packages utilized by the Ruby runtime is the bigdecimal gem. If you were prioritizing mitigation work, and there was a severe vulnerability in bigdecimal, bigdecimal would be a good candidate to address first.

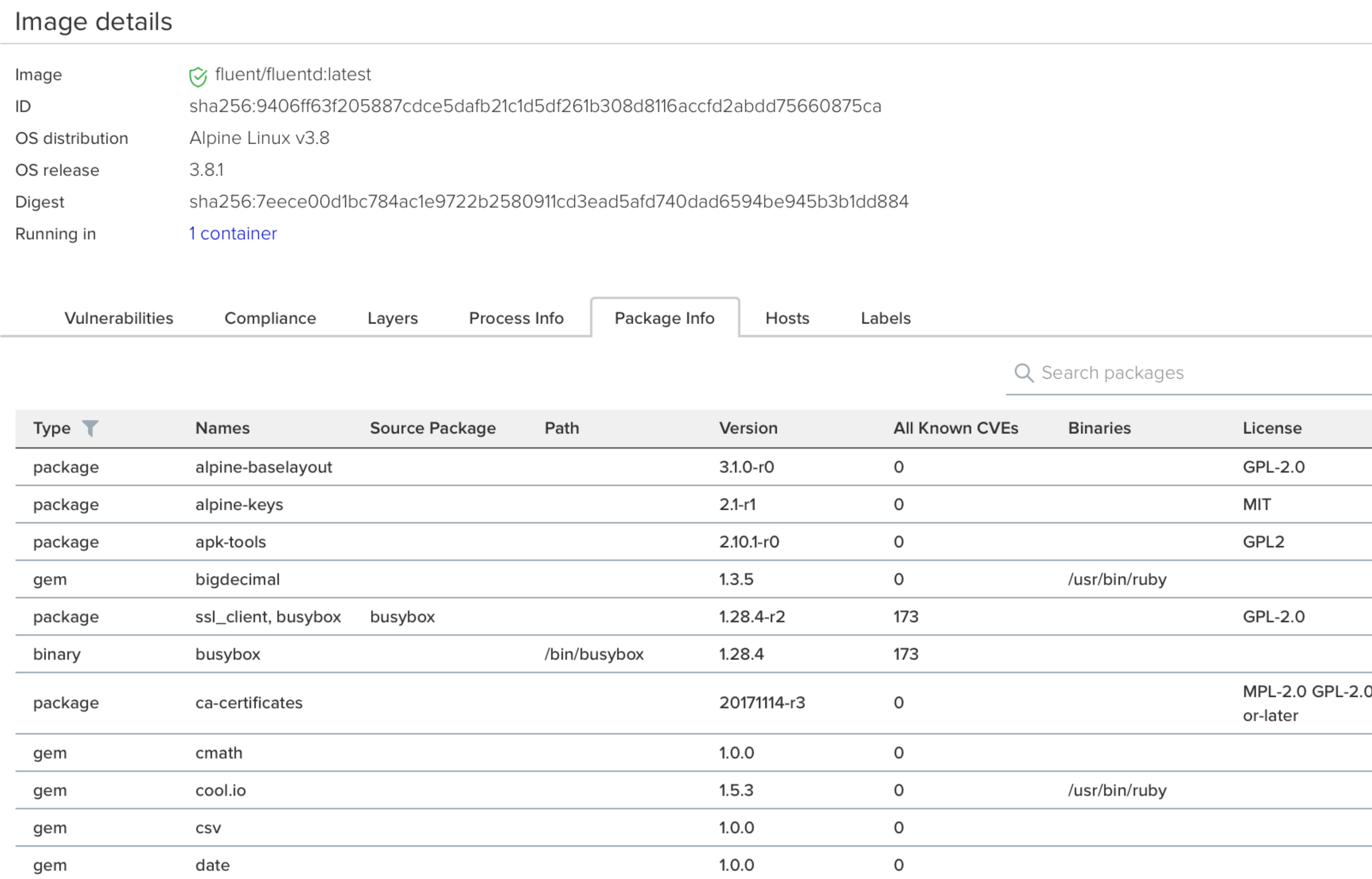


Image details

Image	fluent/fluentd:latest
ID	sha256:9406ff63f205887cdce5dafb21c1d5df261b308d8116accfd2abdd75660875ca
OS distribution	Alpine Linux v3.8
OS release	3.8.1
Digest	sha256:7eece00d1bc784ac1e9722b2580911cd3ead5afd740dad6594be945b3b1dd884
Running in	1 container

Vulnerabilities Compliance Layers Process Info Package Info Hosts Labels

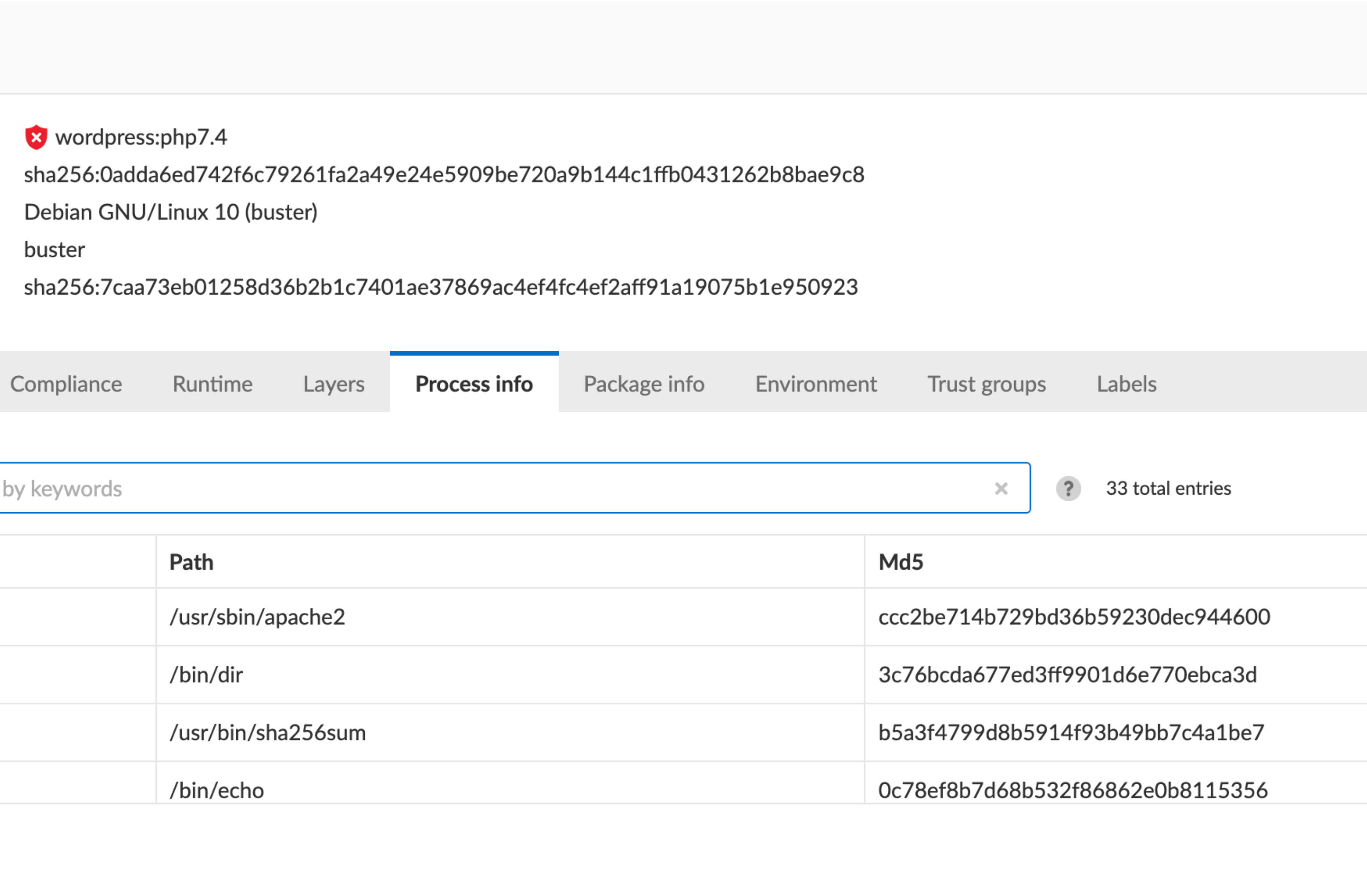
Search packages

Type	Names	Source Package	Path	Version	All Known CVEs	Binaries	License
package	alpine-baselayout			3.1.0-r0	0		GPL-2.0
package	alpine-keys			2.1-r1	0		MIT
package	apk-tools			2.10.1-r0	0		GPL2
gem	bigdecimal			1.3.5	0	/usr/bin/ruby	
package	ssl_client, busybox	busybox		1.28.4-r2	173		GPL-2.0
binary	busybox		/bin/busybox	1.28.4	173		
package	ca-certificates			20171114-r3	0		MPL-2.0 GPL-2.0-or-later
gem	cmath			1.0.0	0		
gem	cool.io			1.5.3	0	/usr/bin/ruby	
gem	csv			1.0.0	0		
gem	date			1.0.0	0		

Process info

Prisma Cloud scan reports provide visibility over the startup processes of the image. To see the image startup processes, open a scan report and go to the **Process info** tab.

The processes list is created by a static analysis of the image, which first parses the image history to get the list of startup binaries. The algorithm then iterates over the image binaries and tries to find these startup binaries on the disk (in the file system). Those which were found are displayed under the **Process info** tab.

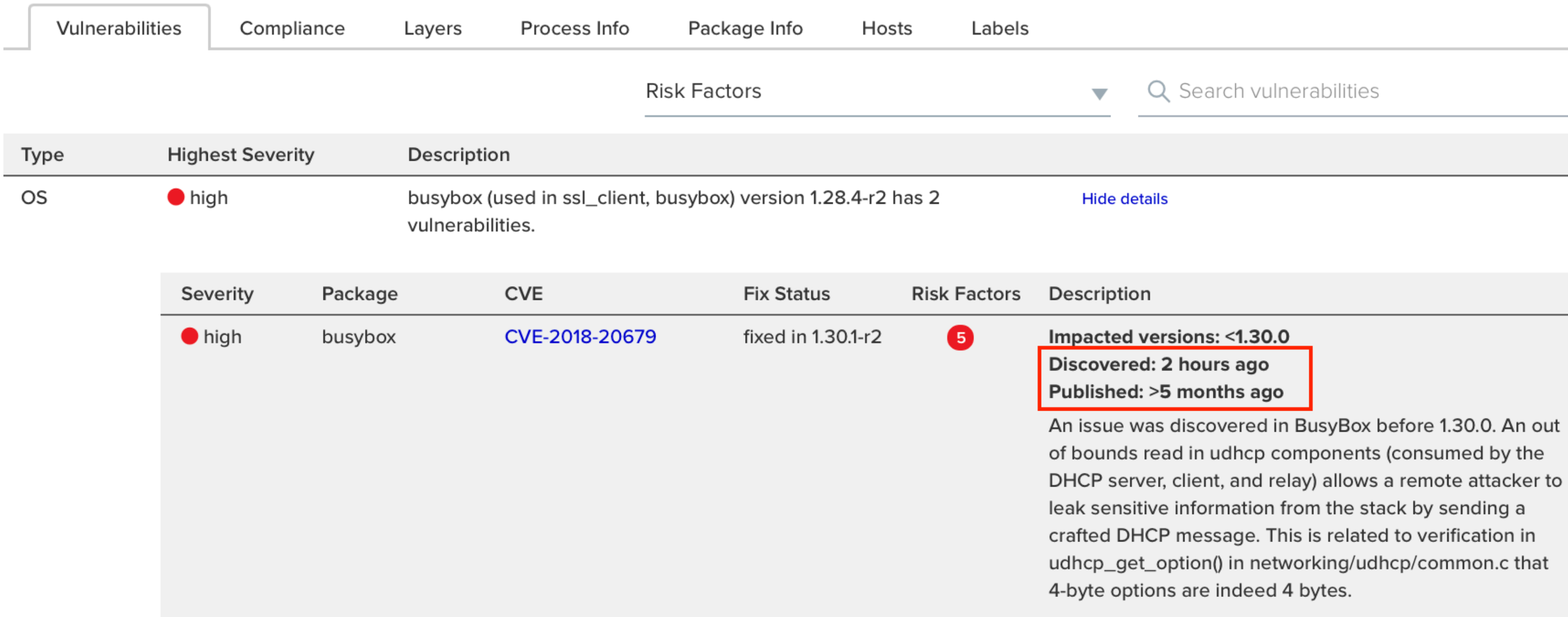


Per-finding timestamps

Prisma Cloud's image scan reports show the following per-vulnerability timestamps:

- Age of the vulnerability based on the discovery date. This is the first date that the Prisma Cloud scanner found the vulnerability.
- Age of the vulnerability based on its published date. This represents the date the vulnerability was announced to the world.

Host scan reports and registry scan reports show the published date only.



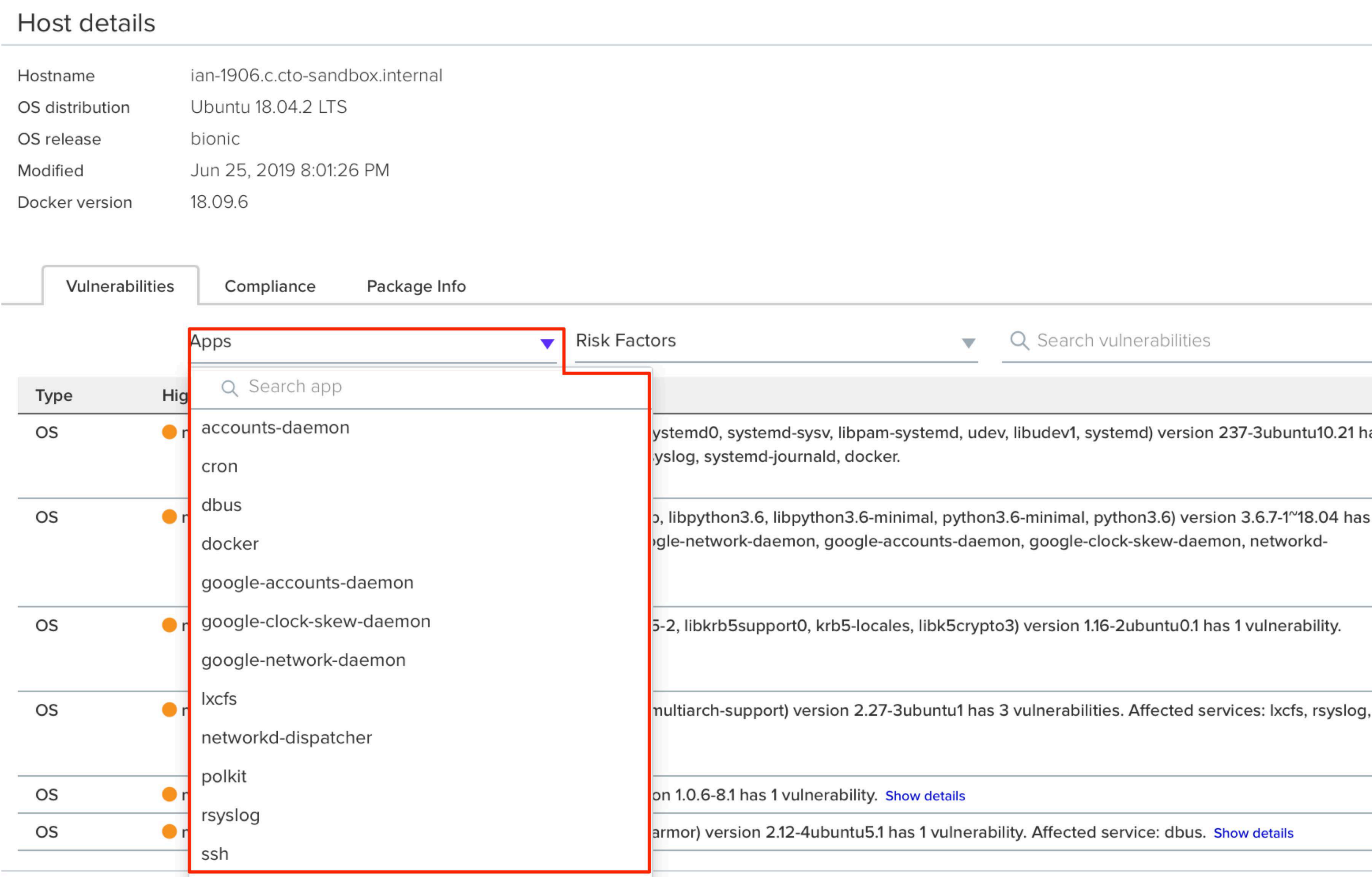
Timestamps are per-image, per-vulnerability. For example, if CVE-2019-1234 was found in image foo/foo:3.1 last week and image bar/bar:7.8 is created from foo/foo:3.1 today, then the scan results for foo show the discovery date for CVE-2019-1234 to be last week and for bar it shows today.

Timestamped findings are useful when you have time-based SLAs for remediating vulnerabilities (e.g. all critical CVEs must be fixed within 30 days). Per-finding timestamp data makes it possible to track compliance with these SLAs.

Host and VM image scanning

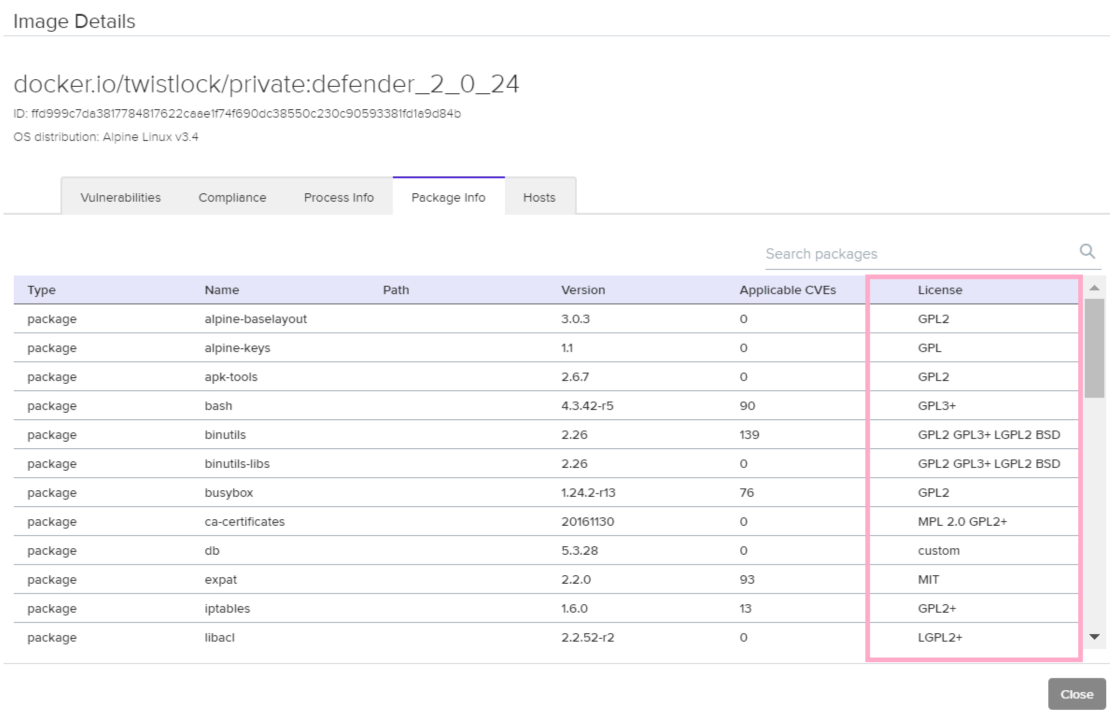
Prisma Cloud also scans your hosts and VM images for vulnerabilities. To see the scan report for your hosts and VM images, go to **Monitor > Vulnerabilities > Hosts**.

By default, all vulnerable packages, according to your policy, are listed. However, you can also examine vulnerabilities specific to an app (systemd service). Use the drop-down list to select an app. Clear the selection to see all vulnerabilities for a host/VM image.



The **Package Info** tab lists all packages installed on the host/VM image. If a package has a component utilized by a running app, the affected running apps are listed in the **Apps** column.

Prisma Cloud also collects and displays package license details. License information is available at all places where package details are displayed, such as **Monitor > Vulnerabilities > Images** (under the **Package Info** tab), **Monitor > Vulnerabilities > Hosts** and **Monitor > Vulnerabilities > Registry**, as well as the corresponding API endpoints.



Licensing compliance is currently supported only for viewing purposes and cannot be included in policies for alert/block capabilities.

Scan status

The initial scan can take substantial time when you have a large number of images. Subsequent scans are much faster.

To see the status of the image scans, go to **Monitor > Vulnerabilities > Images**.

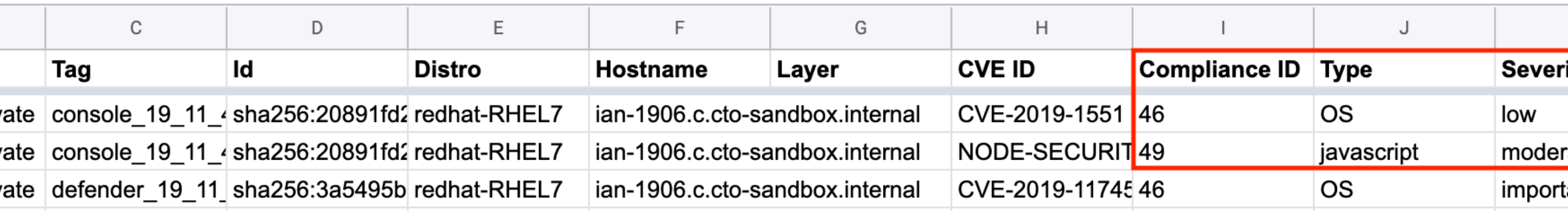
Each row in the table represents an image in your environment.

If an image is being scanned, a progress bar shows the status of the scan. If there is no progress bar, the scan has completed.

Package types

Prisma Cloud uses compliance identification numbers to designate the package type when reporting vulnerabilities in images. Compliance IDs can be found in the CSV export files and API responses.

To download image reports in CSV format, go to **Monitor > Vulnerabilities > Images**, and click the **CSV** button at the top of the table. The **Compliance ID**, **Type**, and **Packages** fields report the package ID, package type, and package name respectively. The API output reports compliance IDs only.



	C	D	E	F	G	H	I	J	
	Tag	Id	Distro	Hostname	Layer	CVE ID	Compliance ID	Type	Severi
ate	console_19_11_	sha256:20891fd2	redhat-RHEL7	ian-1906.c.cto-sandbox.internal		CVE-2019-1551	46	OS	low
ate	console_19_11_	sha256:20891fd2	redhat-RHEL7	ian-1906.c.cto-sandbox.internal		NODE-SECURIT	49	javascript	moder
ate	defender_19_11_	sha256:3a5495b	redhat-RHEL7	ian-1906.c.cto-sandbox.internal		CVE-2019-11745	46	OS	import

The following table shows how compliance IDs map to package type.

Compliance ID number	Package type
46	Operating system/distro packages
47	JAR files
48	Gem files
49	Node.js
410	Python
411	ie. MySgl
412	Custom (set by customer)
415	Nuget
416	Go

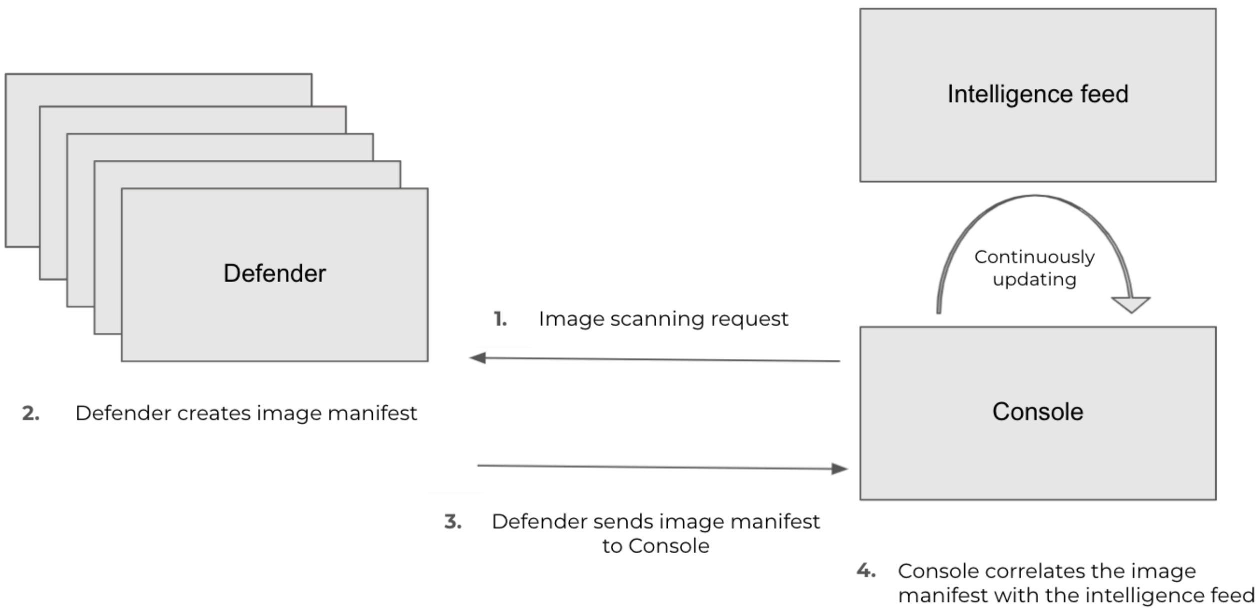
Scanning procedure

Edit on GitHub

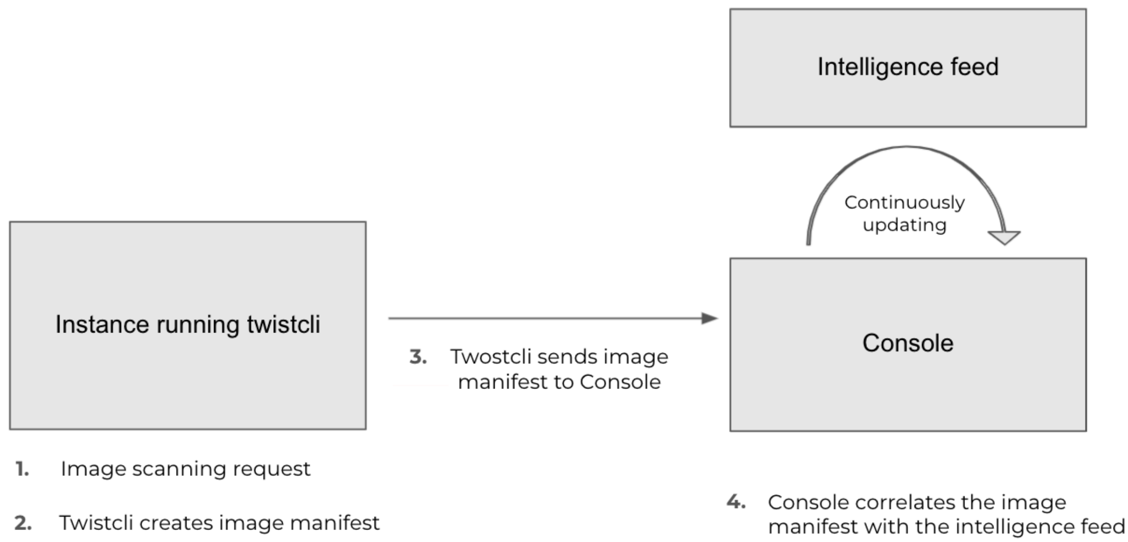
This article describes the vulnerability image scanning flow for deployed containers, registries, and CI. The scanning flow is similar for both Docker and Dockerless images, except for a single difference, described in Scan reports for CRI environments.

Image scanning procedure

The following diagram (Chart 1) shows the Defender scanning flow:



The following diagram (Chart 2) shows the twistcli scanning flow:



The steps in the scanning flow are:

1. An image scanning request is initiated. This can be done by one of the following:
    - Console - generates periodic scan requests (see Chart 1)
    - Defender - when a new container starts (in this step we skip step 1 in Chart 1 since the Defender initiates the scan)
    - twistcli - triggered manually or in the CI pipeline (see Chart 2)
2. Defender harvests the image components versions to create the image manifest by scanning the image, and it:
    1. Looks for component and version details from OS package managers.
    2. Looks at executables (identified by magic signatures on the file system) **not** installed by the package manager:
        1. Selects the executables that are supported by Prisma Cloud.
        2. Identifies each executable's version details from the binary metadata.
3. Based on the information from step 2, Defender generates an **image manifest** and sends it to Console.
4. Console identifies the vulnerabilities in each image by correlating the image manifest with the intelligence stream:
    1. The intelligence CVE stream is composed of per-distro CVEs (Red Hat, Ubuntu, etc.), un-packaged software CVEs (see supported apps), and various open-source library CVEs (nodejs, python, etc.). Console is continually updated by the Intelligence Stream to provide the most up-to-date results.
    2. The correlation results are calculated, stored in DB, and displayed in the Console UI.

Scan reports for CRI environments

Deployed images — The scanning logic is the same for Dockers and Dockerless environments The only difference lies in the scanned object. In Docker environments, Prisma Cloud scans images by running the image with Defender as the entrypoint. Dockerless doesn't support this method, so for Dockerless environments, Prisma Cloud scans the running container. As a result, when scanning deployed images in Dockerless environments, Defender might discover packages that are not in the original image, but installed sometime during the container lifetime (for example, if you exec'd into a running container, and ran *wget … .jar*).

Registry scan — The scanning logic is the same for Docker and Dockerless environments Any Container Defender running on a host with the Docker Engine container runtime or container runtime interface (CRI) can scan a registry. Learn more about registry scanning.

Twistcli scans — Scans conducted by twistcli are similar for Docker and Dockless (CRI). In both environments twistcli scans run from outside the container image. For Dockerless environments, Podman must be installed on the host, to allow scans to run from outside the container image. Learn more in the twistcli scan images document.

Customize image scanning

Edit on GitHub

You can customize how Prisma Cloud scans images and reports data.

Configuring the severity of reported CVEs

By default, Prisma Cloud reports all vulnerabilities. Setting the minimum reported severity lets you clean up the reported vulnerabilities to an actionable set.

To configure a minimum severity, install a new vulnerability rule, which overrides the default rule. Note that Prisma Cloud maps the Common Vulnerability Scoring System (CVSS) to a grading system that ranges from Low to Critical.

STEP 1 | Open Console, and go to **Defend > Vulnerabilities > Images > Deployed > Add rule**.

STEP 2 | Click **Add rule**.

STEP 3 | Give your rule a name.

STEP 4 | In the table of **Severity based actions**, set the **Severity** in each row to an appropriate level. For example, if you want to concentrate on just the most severe issues, set every row to **Critical**.

STEP 5 | Click **Save**.

STEP 6 | View the scan reports for all the entities in your system.

Go to **Monitor > Vulnerabilities**. All reported vulnerabilities match or exceed the severity setting in your custom rule.

Scanning custom components

Prisma Cloud lets you scan for insecure versions of proprietary software components.

First, augment Prisma Cloud's Intelligence Stream with your own custom data that specifies a package type, name, and version number. Then configure Prisma Cloud to take action (alert, block) when the scanner finds this package in an image. By default, Prisma Cloud raises an alert when it detects a vulnerability in a custom component.

Prisma Cloud supports the following package types:

- Distro packages (deb, rpm).
- Binaries.
- Nodejs packages.
- Python packages.
- Ruby gems.
- Java artifacts (JAR files).

For cases where Prisma Cloud does not offer built-in support for a package type, you can specify an MD5 hash for the file.

Defining a custom vulnerability

Define a custom vulnerability.

STEP 1 | Open Console.

STEP 2 | Go to **Manage > System > Custom Feeds**.

STEP 3 | Click on **Custom Vulnerabilities**.

STEP 4 | Click **Add**.

1. Enter a name for your vulnerability..
2. From the drop-down list, select a package type.

   For Debian packages, RPM packages, and shared libraries, select **package**.

   If your package type is not supported, select **binary**.
3. Enter the name of your package/binary.

   Package names must be specific for matching. For example, "containerd" is valid, "containerd*" is not.
4. Specify the range of package versions for which your rule applies.

   The following formats can be used to specify versions:

Rule	Format	Example
Specific version	Enter a single multi-dot number.	1.1
Range of versions: Min and max are known.	Enter two multi-dot numbers, separated by a dash.	5.4-5.5
Range of versions: Only min version is known.	Specify a multi-dot number for the minimum version, followed by a dash, then a wild card.	0.22.4.1-*
Range of versions: Only max version is known.	Specify a wild card (*) for the minimum version, followed by a dash, then a multi-dot number for the maximum version.	*-0.22.4.1

If package type is set to binary, the version fields are not visible. Instead, enter the MD5 hash for your file or binary.

STEP 5 | Click **Save**.

Your custom vulnerability is now available to the scanner.

By default, an alert is logged if an image scan detects a component that you have designated as vulnerable. To see the default rule, go to **Defend > Vulnerabilities > Images**, and click on the **Default - alert all components** rule. To change the default rule, select a different Alert or block threshold. To take a different action, create a new vulnerability rule.

Configure Registry Scans

Edit on GitHub

Prisma Cloud can scan container images in public and private repositories on public and private registries.

A registry is a system that stores and distributes container images. The most well-known public registry is Docker Hub, but you can use other registries from Amazon, Google, and other providers. Your organization can set up its own internal private registries too. Prisma Cloud supports scanning container images on all these registries.

After you configure repository scanning, Prisma Cloud automatically scans images for vulnerabilities. By default, scans occur once every 24 hours, but you can configure periodic scans at specific intervals specified in **Manage > System > Scan**.

Configure Prisma Cloud to Scan a Registry

To scan images in a registry, create a new registry scan rule.

Prerequisites: You have deployed at least one Defender in your environment.

To avoid interrupting an ongoing scan, Prisma Cloud provides the option to save your configuration without restarting the scan. When saving your configuration changes, you are prompted on whether you want to save the changes and start a new scan or to save your changes and wait until the next scheduled scan. If you use the */settings/registry* API to manage registry scanning, you can use the *scanLater* flag when using the *PUT* or *POST* methods to decide whether to initiate a scan after saving or not. By default, Prisma Cloud initiates a scan.

STEP 1 | Open the Prisma Cloud Console.

STEP 2 | Go to **Defend > Vulnerabilities > Images > Registry settings**.

STEP 3 | Review the available settings if the default values don't fit your scenario.

STEP 4 | Click **Add registry**.

Deployment Patterns

Defenders handle registry scanning. When you configure registry scanning, you can select the scope of defenders used to perform the scans.

Any Container Defender running on a host with the Docker Engine container runtime or container runtime interface (CRI) can scan a registry, and any number of them can simultaneously operate as registry scanners. This flexibility gives you a lot of options when trying to determine how to cover disparate environments.

You can use host names or AWS tags to select a collection of defenders to distribute the scanning job between them, and use the **Number of scanners** setting to control how many defenders are included in the collection. When you select the **All** collection, Prisma Cloud automatically distributes the scan job across all available defenders.

Configuring Prisma Cloud to use a large number of defenders reduces operational complexity and improves resiliency. During a scan, Prisma Cloud lists the available defenders based on the configured scope, manages the resource pool, and handles issues such as restarting partially completed jobs. If you explicitly select one or two defenders to handle scanning, the hosts running those defenders become a single point of failure. If that host fails or gets destroyed, you have to reconfigure your scan settings with different defenders.

The type of operating system (OS) scopes registry scanning. Windows defenders only scan Windows images, and Linux defenders only scan Linux images.

When you remove an image from the registry or the registry becomes unavailable, Prisma Cloud maintains the scan results for a specific number of days. You can configure the number of days under **Manage > System > Scan > Registry scan results**. After the specified number of days, the scan results are purged.

Registry Scan Steps

At a high level, defenders scan your registries following these steps.

1. Scan registry settings one by one in sequential order.
2. Discover the repositories based on your registry configuration.
3. Discover the images using tags within each configured repository.
4. Scan the discovered images.

In more detail, defenders scanning your registries follow this sequential flow to collect the metadata.

1. Get a list of all repositories in the registry.
2. For each repository, scanning defenders perform the following tasks.
    - Get a list of all image tags.
    - For each image tag, they get the image manifest containing the date the image was last modified.
3. Once the metadata of all images is discovered, scanning defenders perform the following tasks.
    - Sort the images by the last modified date.
    - Cap the list of images based on the configured value. By default, lists are capped at five.
    - Scan the images.

Registry Scan Settings

You can set the following parameters for each rule has the following parameters, but the parameters can vary between registry types. If you use a specific registry provider, follow the appropriate step-by-step instructions in our guides.

Field	Description
Version	Specify the type of registry to scan. • If you do not find your vendor’s registry in the drop-down list, try **Docker Registry v2**. Most vendors comply with the Docker Registry version 2 API.
Registry	Specify the URL for the registry.

Field	Description
	Docker Hub: leave this field blank. **Harbor**: specify the FQDN of your Harbor registry (https://). **Nexus Registry: <http|https://<nexus_hostname>:<HTTP/HTTPS connector port for the specific Nexus repo>** Example: **https://ec2-100-25-223-135.compute-1.amazonaws.com:18079**
Repository name	Specify the repository to scan. This field supports pattern matching. To scan all repositories, simply leave this field blank or enter a wildcard (*). **Docker Hub:** To specify an official Docker repository, enter library/, followed by the short string used to designate the repo. For example, to scan the images in the official Alpine Linux repository, enter library/alpine. To specify non-official repositories, enter the username or organization name, followed by a slash, followed by the name of the repo. For example, to specify the alpine repository in onescience's account, enter onescience/alpine. To scan all repos from a user or organization, simply enter the user or organization name, followed by a wildcard (*). For example, to scan all repos created by onescience, enter onescience*. **Google Cloud Platform Container Registry:** Enter your project ID and image name in the following format: project-id/image-name. To scan all images, follow the repository name with /*. (e.g. *company-sandbox/**) **Harbor:** Enter the name of the repository, followed by a wildcard (*). For example, to scan the repository library, enter library*. **Any Docker V2 API compliant registry:** Docker Hub, Docker Registry, and Alibaba Container Registry all support the Docker Registry version 2 API. **Nexus Registry:** Leave blank or include a pattern to match the Docker repositories inside the Nexus registry. For example: To scan all the images under a path, include the **path/to** string.
Tag	Specify an image tag. Leave this field blank to scan all tags (limited by the value in Cap).
Credentials	Specify the credentials required to access the registry. If the credentials have already been created in the Prisma Cloud credential store, select it. If not, click **Add New**. **Public repositories on public registries (such as Docker Hub):** Leave this field blank. No credentials are required. **AWS EC2 Container Registry:** Use the IAM access keys for authentication. For more information, see Amazon EC2 Container Registry (ECR). **Google Container Registry:** Use the service account and JSON token. For more information, Google Container Registry (GCR).

Field	Description
	Harbor Registry: Create a **Basic authentication** credential. Credentials for Harbor can be a **Limited Guest**. **Registries that support token authentication (e.g. Quary, GitLab):** Create a **Basic authentication** credential. *Username* is the name of the token and the token value is entered into the *password* field.
OS Type	Specify whether the image is built on a Windows or Linux based OS.
Scanners scope	Select collections of Defenders to scan this registry. Only Linux Defenders can scan Linux container images, and only Windows Defenders can scan Windows container images. App-Embedded Defenders can't be used for registry scanning.
Number of scanners	Number of Defenders from scope across which the scan job can be distributed. Increase the number of Defenders to increase throughput and reduce scan time.
Cap (Capacity)	Specify the maximum number of images to scan in the given repository, sorted according to last modified date. A repository is a collection of different docker images with same name, that have different tags. That is, the most recently modified image in each repository is scanned first, followed by the image next most recently modified, and so on. With a cap of five, scanning defenders fetch the five most recently modified images from each repository in the registry. In other words, for each image in the registry we will include the 5 latest versions. The Docker Registry API does not support directly querying for the most recently updated images. To handle your CAP setting, Prisma Cloud first polls the registry for all tags and manifests in the given repository to discover the last updated dates. This is a low overhead operation because images do not need to be downloaded. Prisma Cloud then sorts the results by date and then scans the most recently updated images in each repository up to the limit specified by CAP. Even when CAP is set to a low number, you might still notice the Prisma Cloud UI polling the registry for data about the images in the repository. To scan all images in a repository, set CAP to 0.
Version matching pattern	Customize sort order by values in the image tag. Specify a pattern from which a version or date can be extracted from the image tag. There are two use cases for specifying version matching patterns: • You want to reduce the total time it takes to complete the scan for very large registries. Rather than fetching the metadata from the registry required to sort images, you specify how the scanner can extract the metadata directly from the image tag. • You want to order and cap the images to be scanned by some value other than last modified date. Specify patterns with strings, wildcards, time/date elements, and integers.

Field	Description
	• *%d* - version number • *%Y* - 4 digit year • *%M* - 2 digit month • *%D* - 2 digit day • *%H* - 2 digit hour • *%m* - 2 digit minute • *%s* - 2 digit second For image tags that match the pattern, the tag is split into its constituent parts. After all image tags are parsed, they're ordered and capped according to the value set in Cap. Ordering is the best-effort. Tags that don't conform to the pattern are ignored. If both date and version are specified in your pattern, the date takes precedence. If the version matching pattern is left unspecified, Prisma Cloud orders images by last modified date.

Registries with a Large Scale

If your registries are very large, optimize your scan configuration to maximize throughput and minimize scan time. Defenders scan registries sequentially following specific steps. The following best practices help you improve your registry scanning speed.

- If you have large registries or need aggressive scan intervals, increase the number of scanners in the scope.

  The number of scanning defenders should increase with regard to the registry size. As the number of images in the registry increases, so does the number of defenders scanning this registry.

- Use the default cap value of five in your registry scan configuration.

  The cap value impacts the duration of the scan. Large cap values lead to longer scan times since more images are scanned.

- Use a version matching pattern in your registry scan configuration. Only use version pattern matching for deployments with very large registries containing tens of thousands of repositories and millions of images.

  If you specify a version matching pattern, the scanner looks to the image tag for sort order. Without a version matching pattern, images are sorted by last modified date. With a version matching pattern, you configure how image tags are sorted. Using semantic versioning in your image names, you can specify the following version pattern:

```
*-%d.%d.%d
```

This optimized flow to collect metadata eliminates the sorting loop and substantially reduces the number of requests. Then, defenders can start scanning the registry sooner. The simplified flow is as follows.

1. Get a list of all repos in the registry.
2. For each repository, scanning defenders perform the following tasks.
    - Get a list of all image tags
3. Once the metadata of all images is discovered, scanning defenders perform the following tasks.
    - Sort the images by last modified date.
    - Cap the list of images based on the configured value. By default, lists are capped at five.
    - Scan the images.

A repository with three images, configured with a cap of 2, and a version pattern of **-%d.%d.%d*, produces the following set of images to be scanned.

```
myimage-3.0.0 <<<--- Image scanned
myimage-2.0.1 <<<--- Image scanned
myimage-2.0.0 (Not scanned)
```

- When you have multiple registries, create multiple collections of defender scanners.

  Each registry should have dedicated Defenders to perform the scanning. If a 1:1 ratio of collections to registries isn't feasible, create as many collections as possible to split the load. Don't reuse the same collection for all registries.

  This best practice prevents the scenario where a single Defender performs too many queries to the registry provider API. If too many queries are made during repository or tag discovery, providers could throttle the Defender.
- Properly dimension the hardware running your defenders.

  Ensure the hardware system requirements for defenders scanning registries are met.
- Colocate scanning defenders in the same region as the registry.

  This best practice minimizes network latency since the defenders run in the same region as your registries.

Additional Scan Settings

You can find additional scan settings under **Manage > System > Scan**, where you can set the registry scan interval.

The **Manage > System > Scan** page has an option called **Only scan images with running containers**. This option does NOT apply to registry scanning. All images included in your registry scanning rule are scanned regardless of the setting to **Only scan images with running containers**.

CRI and containerd-only environments

Prisma Cloud fully supports scanning CRI and containerd-only environments.

Registry Scanning Limitations

When scanning registries, consider the following constraints.

- Defenders only scan the operating system images that match the OS of the system running them.

  For example, a Defender running on a Linux host can only scan Linux images and won’t scan Windows images.
- Defenders running on Linux only scan images suited for the hardware architecture that matches the architecture of the system running them.

  For example, a Defender running on x86_64 architecture with Linux can only scan images for x86_64 systems with Linux. Similarly, a Defender running on ARM64 architecture with Linux can only scan images for ARM64 systems with Linux. You can’t mix Linux ARM64 and Linux x86_64 defenders within the same registry scanning scope.

Registry scanning

Edit on GitHub

Configure Prisma Cloud to scan your registries.

- Scan images in Sonatype Nexus Registry
- Scan images in Alibaba Cloud Container Registry
- Scan images in Amazon EC2 Container Registry (ECR)
- Scan images in Azure Container Registry (ACR)
- Scan images in Docker Registry v2
- Scan images in Google Artifact Registry
- Scan images in Google Container Registry (GCR)
- Scan images in Harbor Registry
- Scan images in IBM Cloud Container Registry
- Scan images in Artifactory Docker Registry
- Scan Images in OpenShift integrated Docker Registry
- Trigger Registry scans with webhooks

Scan Images in Sonatype Nexus Registry

Edit on GitHub

To scan a repository in Sonatype Nexus Registry, create a new registry scan setting.

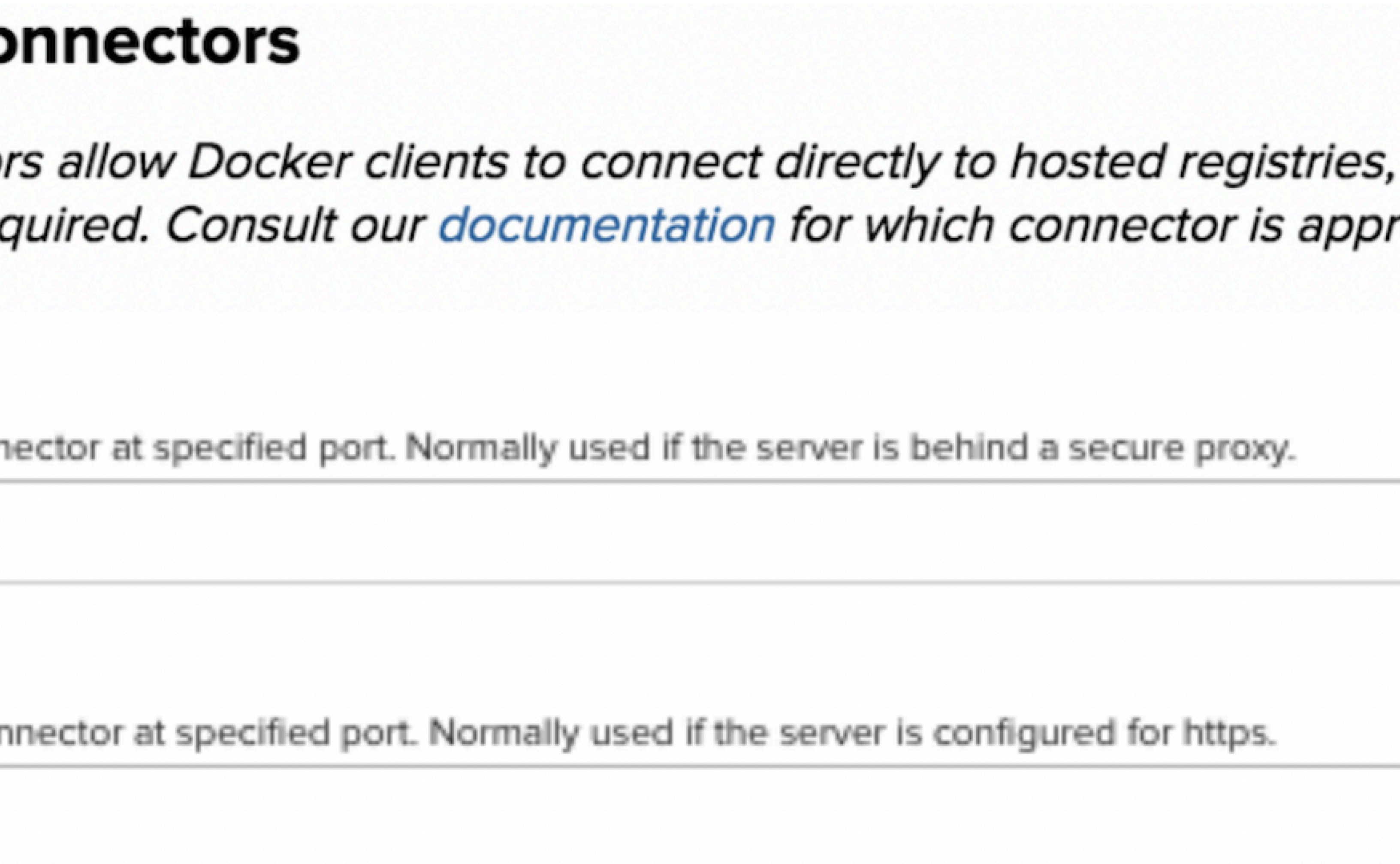
Prerequisites

- You have installed a Container Defender somewhere in your environment.

- Configure the connector port for the Docker engine to connect to the Nexus registry.

  The Docker client needs the Docker registry exposed at the root of the host together with the port that it is accessing. Nexus offers several methods to overcome this limitation, one of which is the connector port method, which Prisma Cloud supports.

  1. From the Nexus web portal, select the Administration screen.
  2. Select the Nexus Repository.
  3. Select **Online** to allow the Nexus repository to accept the incoming requests.
  4. Configure the Docker repository connector to use an HTTP or HTTPS port.



- The Defender can establish a connection with the Nexus registry over the connector port that you configured in the Nexus registry.

  Ensure that the port is open for the image to be accessed successfully.

Add a Nexus registry in Prisma Cloud

STEP 1 | Log in to Console, and select **Compute > Defend > Vulnerabilities > Registry**.

STEP 2 | Select **Add registry**.

STEP 3 | In the **Add New Registry Setting Specification**, enter the following values:

1. In the **Version** drop-down list, select **Sonatype Nexus**.
2. In **Registry**, enter the hostname, or Fully Qualified Domain Name (FQDN), and the connector port for the Nexus registry's login server.

   The format for the FQDN is **<hostname>:<connector_port>**, where **<hostname>** is a unique value specified when the registry was created, and the **<connector_port>** is the one you configured in the Nexus repository administration.

   Example: **<http|https://<nexus_hostname>:<HTTP/HTTPS connector port for the specific Nexus repo>**.

   https://ec2-100-25-223-135.compute-1.amazonaws.com:8083
3. Leave the **Repository** blank or include a pattern to match the Docker repositories inside the Nexus registry.

   For example: To scan all the images under a path, include the **path/to** string.
4. In **Tag**, enter an image tag. Leave this field blank to scan all images, regardless of their tag.
5. In **Credential**, configure how Prisma Cloud authenticates with Nexus registry.

   Select a credential from the drop-down list.

   If there are no credentials in the list, click **Add** to create new credentials and select the Basic authentication.
6. In **OS type**, specify whether the repo holds **Linux** or **Windows** images.
7. In **Scanners scope**, specify the collections of defenders to use for the scan.

   Console selects the available Defenders from the scope to execute the scan job according to the **Number of scanners** setting. For more information, see deployment patterns.
8. In **Number of scanners**, enter the number of Defenders across which scan jobs can be distributed.
9. In **Cap**, limit the number of images to scan.

   Set **Cap** to **5** to scan the five most recent images, or enter a different value to increase or decrease the limit. Set **Cap** to **0** to scan all images.
10. Select **Add**.

STEP 4 | Select **Save and scan**.

View Results

Verify that the images in the repository are being scanned.

STEP 1 | Go to **Monitor > Vulnerabilities > Images > Registries**.

A progress indicator at the top right of the window shows the status of the current scan. As the scan of each image is completed, the finding results display in the table.

STEP 2 | To get details about the vulnerabilities in an image, click on it.

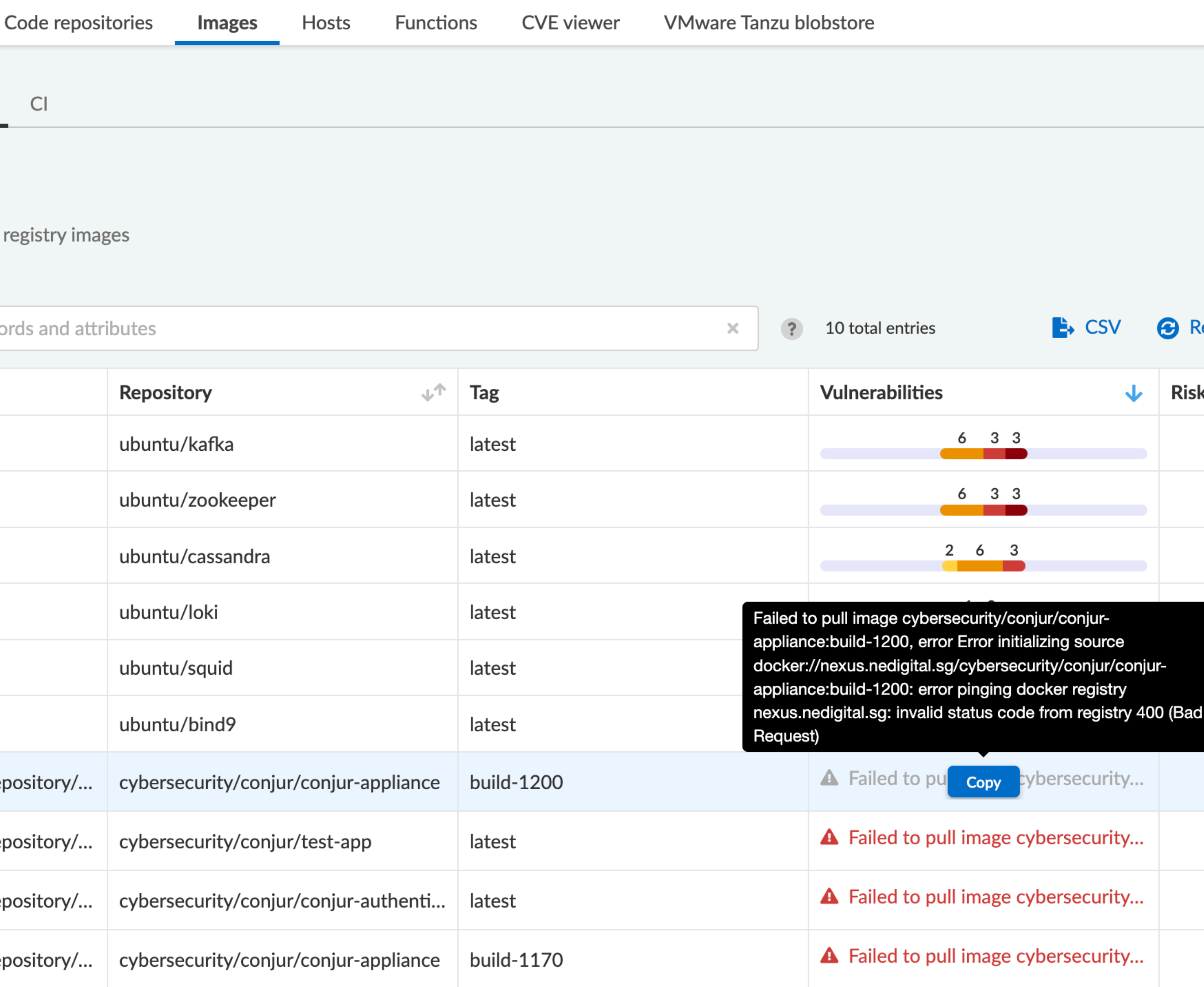
To force a specific repository to be scanned again, select **Scan** from the top right of the results table, then click on the specific registry to rescan.

Troubleshooting

Prisma Cloud failed to pull images from the Nexus repository

Monitor > Vulnerabilities > Images > Registries shows the following error:

ERRO 2022-06-07T06:55:39.046 scanner.go:110 Failed to pull image cybersecurity/conjur/test-app:latest, error Error initializing source docker://nexus.nedigital.sg/cybersecurity/conjur/test-app:latest: error pinging docker registry nexus.nedigital.sg: invalid status code from registry 400 (Bad Request)



STEP 1 | Ensure that you have installed Defender on the host on which the Nexus registry is installed.

STEP 2 | Verify that you can pull the nexus registry using the docker command.

STEP 3 | Create a Nexus repository connector port as mentioned in the Prerequisites.

STEP 4 | Add a Nexus registry in Prisma Cloud using the connector port in the Registry URL.

Scan images in Alibaba Cloud Container Registry

Edit on GitHub

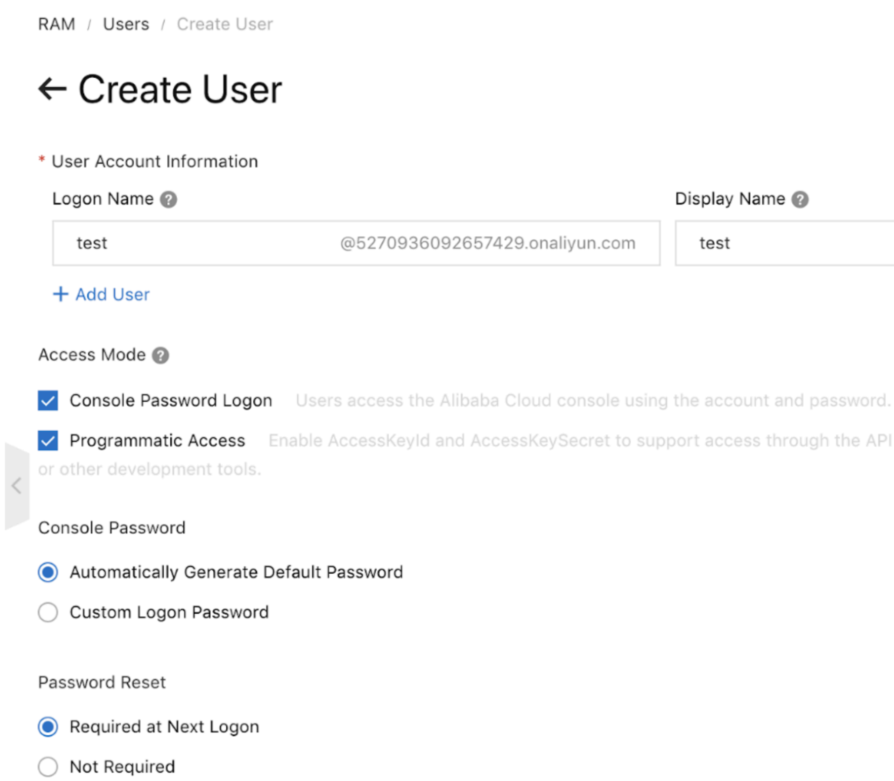
Configure Prisma Cloud to scan your Alibaba Cloud Container Registry. First, create a service account, and then specify the scan parameters.

Create a service account

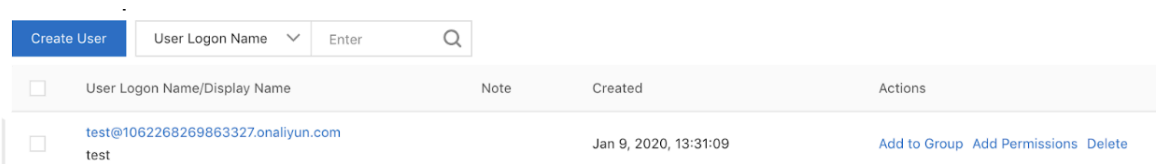
Create a service account so Prisma Cloud can access your registry. Prisma Cloud needs the **AliyunContainerRegistryReadOnly** permission policy to query, download, and scan the images in your registry.

STEP 1 | In Alibaba Cloud, create a RAM account.

Go to **RAM > Users**, and click **Create User**.



STEP 2 | Click **Add Permissions**.



STEP 3 | Search for **registry**, and then select **AliyunContainerRegistryReadOnly**.

Scan images in Alibaba Cloud Container Registry

To scan a repository in Alibaba Cloud Container Registry, create a new registry scan setting.

Prerequisites:

- You've installed a Defender somewhere in your environment.
- You've already created an Alibaba Cloud Container Registry.
- You have the service account credentials.

STEP 1 | Open Console, and go to **Defend > Vulnerabilities > Registry**.

STEP 2 | Click **Add registry**.

STEP 3 | In the **Add New Registry Setting Specification** dialog, enter the following values:

1. In the **Version** drop-down list, select **Docker Registry v2**.
2. In the **Registry** field, enter the Fully Qualified Domain Name (FQDN) for the registry. For example, **registry-intl.cn-hangzhou.aliyuncs.com**.
3. In the **Repository** field, enter the name of the repository to scan. Example: **library/alpine**.
4. In the **Tag** field, enter an image tag. Leave this field blank to scan all images, regardless of their tag.
5. In the **Credential** field, configure how Prisma Cloud authenticates with Alibaba Cloud Container Registry.

   Select a credential from the drop-down list. If there are no credentials in the list, click **Add new**, and create a **Basic authentication** credential with the service account username and password.
6. In the **OS type** field, specify whether the repo holds **Linux** or **Windows** images.
7. In **Scanners scope**, specify the collections of defenders to use for the scan.

   Console selects the available Defenders from the scope to execute the scan job according to the **Number of scanners** setting. For more information, see deployment patterns.
8. In **Number of scanners**, enter the number of Defenders across which scan jobs can be distributed.
9. In **Cap**, limit the number of images to scan.

   Set **Cap** to **5** to scan the five most recent images, or enter another value to increase or decrease the limit. Set **Cap** to **0** to scan all images.
10. Click **Add**.

STEP 4 | Click the **Save** button.

Results

Verify that the images in the repository are being scanned.

STEP 1 | Go to **Monitor > Vulnerabilities > Images > Registries**.

A progress indicator at the top right of the window shows the status of the current scan. As the scan of each image is completed, its findings are added to the results table.

STEP 2 | To get details about the vulnerabilities in an image, click on it.

To force a specific repository to be scanned again, select **Scan** from the top right of the results table, then click on the specific registry to rescan.

Scan images in Amazon EC2 Container Registry (ECR)

Edit on GitHub

To scan a repository, Prisma Cloud has to authenticate with ECR using either an IAM user (service account) or IAM role. The minimum permissions policy required is **AmazonEC2ContainerRegistryReadOnly**. It is a managed, predefined policy. AWS managed policies grant the minimum set of permissions required for common use cases so you don't need to spend a lot of time investigating permissions yourself.

Authenticate Prisma Cloud to ECR

The **AmazonEC2ContainerRegistryReadOnly** permissions policy is currently defined as follows:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "PrismaCloudComputeECRScanning",
            "Effect": "Allow",
            "Action": [
                "ecr:BatchCheckLayerAvailability",
                "ecr:BatchGetImage",
                "ecr:DescribeImages",
                "ecr:DescribeRepositories",
                "ecr:GetAuthorizationToken",
                "ecr:GetDownloadUrlForLayer",
                "ecr:GetRepositoryPolicy",
                "ecr:ListImages"
            ],
            "Resource": "*"
        }
    ]
}
```

Prerequisites: You have installed a Defender somewhere in your environment.

STEP 1 | Open Console and go to **Defend > Vulnerabilities > Registry**.

STEP 2 | Click **Add registry**.

STEP 3 | In the dialog, enter the following information:

1. In **Version**, select **Amazon EC2 Container Registry**.
2. In **Registry**, enter the URL for the registry. This should be of format: **<acct_id>.dkr.ecr.<region>.amazonaws.com**
3. In **Repository**, enter the name of the repository to scan.
4. In the **Tag** field, enter an image tag. Leave this field blank to scan all tags.
5. Configure how Prisma Cloud authenticates with AWS.

   You can use an IAM user, IAM role, or AWS STS.
6. In **OS type**, specify whether the repo holds **Linux** or **Windows** images.
7. In **Scanners scope**, specify the collections of defenders to use for the scan.

   Console selects the available Defenders from the scope to execute the scan job according to the **Number of scanners** setting. For more information, see deployment patterns.
8. In **Number of scanners**, enter the number of Defenders across which scan jobs can be distributed.
9. Set **Cap** to the number of most recent images to scan. Leaving **Cap** set to **5** will scan the 5 most recent images. Setting this field to **0** will scan all images.
10. Click **Add**.

STEP 4 | Click the **Save** button.

Results

Verify that the images in the repository are being scanned.

STEP 1 | Go to **Monitor > Vulnerabilities > Images > Registries**.

A progress indicator at the top right of the window shows the status of the current scan. As the scan of each image is completed, its findings are added to the results table.

STEP 2 | To get details about the vulnerabilities in an image, click on it.

To force a specific repository to be scanned again, select **Scan** from the top right of the results table, then click on the specific registry to rescan.

Scan images in Azure Container Registry (ACR)

Edit on GitHub

To scan a repository in Azure Container Registry (ACR), create a new registry scan setting.

Create a new registry scan

STEP 1 | Prerequisites

1. You have installed a Defender somewhere in your environment.
2. The Defender can establish a connection with the ACR over port 443.

   Ensure that the port is open for the image to be accessed successfully.

STEP 2 | Log in to Console, and select **Defend > Vulnerabilities > Registry**.

STEP 3 | **Add registry**.

STEP 4 | In the **Add New Registry Setting Specification** dialog, enter the following values:

1. In the **Version** drop-down list, select **Azure Container Registry**.
2. In the **Registry** field, enter the Fully Qualified Domain Name (FQDN) for the registry's ACR login server.

   The format for the FQDN is **<REGISTRY_NAME>.azurecr.io**, where **<REGISTRY_NAME>** is a unique value specified when the registry was created. Example: **example.azurecr.io**.
3. In the **Repository** field, enter the name of the repository to scan. Example: **library/alpine**.
4. In the **Tag** field, enter an image tag. Leave this field blank to scan all images, regardless of their tag.
5. In the **Credential** field, configure how Prisma Cloud authenticates with ACR.

   Select a credential from the drop-down list.

   If there are no credentials in the list, click **Add new** to create an Azure credential where the service principal authenticates with a password.

   To authenticate with a certificate, create a cloud account.
6. In the **OS type** field, specify whether the repo holds **Linux** or **Windows** images.
7. In **Scanners scope**, specify the collections of defenders to use for the scan.

   Console selects the available Defenders from the scope to execute the scan job according to the **Number of scanners** setting. For more information, see deployment patterns.
8. In **Number of scanners**, enter the number of Defenders across which scan jobs can be distributed.
9. In **Cap**, limit the number of images to scan.

   Set **Cap** to **5** to scan the five most recent images, or enter a different value to increase or decrease the limit. Set **Cap** to **0** to scan all images.
10. Click **Add**.

STEP 5 | Click the **Save** button.

Results

Verify that the images in the repository are being scanned.

STEP 1 | Go to **Monitor > Vulnerabilities > Images > Registries**.

A progress indicator at the top right of the window shows the status of the current scan. As the scan of each image is completed, its findings are added to the results table.

STEP 2 | To get details about the vulnerabilities in an image, click on it.

To force a specific repository to be scanned again, select **Scan** from the top right of the results table, then click on the specific registry to rescan.

Scan images in Docker Registry v2 (including Docker Hub)

Edit on GitHub

Most vendors' registries comply with the Docker Registry version 2 API, including Docker Hub.

Create a new registry scan

For Docker Hub repositories:

- To specify an official Docker Hub repository, enter library/, followed by the short string used to designate the repo. For example, to scan the images in the official Alpine Linux repository, enter library/alpine.
- To specify non-official repositories, enter the user name or organization name, followed by a slash, followed by the name of the repo. For example, to specify the alpine repository in onescience's account, enter onescience/alpine.
- To scan all repos from a user or organization, simply enter the user or organization name, followed by a wildcard (*). For example, to scan all repos created by onescience, enter onescience*.

Prerequisites: You have installed a Defender somewhere in your environment.

STEP 1 | Open Console, and then go to **Defend > Vulnerabilities > Registry**.

STEP 2 | Click **Add registry settings**.

STEP 3 | In the dialog, enter the following information:

1. In the **Version** drop-down list, select **Docker Registry v2**.
2. Leave the **Registry** field blank. An empty field specifies Docker Hub (hub.docker.com).
3. In **Repository name**, enter the name of the repo to scan. For example, enter **library/alpine** to scan the official Alpine image. If the repo is part of an organization, use the organization/repository format. For example, **bitnami/nginx**.
4. In **Credential**, select the credentials to use.

   If you are scanning a public repository, leave this field blank.

   If you are scanning a private repository, and Console doesn't have your credentials yet, click **Add New**. Select either **Basic authentication** or **Certificate-based authentication**, and fill out the rest of the fields. For certificate-based authentication, provide a client certificate with private key, and an optional CA certificate.
5. In **OS type**, specify whether the repo holds **Linux** or **Windows** images.
6. In **Scanners scope**, specify the collections of defenders to use for the scan.

   Console selects the available Defenders from the scope to execute the scan job according to the **Number of scanners** setting. For more information, see deployment patterns.
7. In **Number of scanners**, enter the number of Defenders across which scan jobs can be distributed.
8. Set **Cap** to the number of most recent images to scan. Leaving **Cap** set to the default value of **5** will scan the most recent 5 images. Setting this field to **0** will scan all images.
9. Click **Add**.

STEP 4 | Click the **Save** button.

Results

Verify that the images in the repository are being scanned.

STEP 1 | Go to **Monitor > Vulnerabilities > Images > Registries**.

A progress indicator at the top right of the window shows the status of the current scan. As the scan of each image is completed, its findings are added to the results table.

STEP 2 | To get details about the vulnerabilities in an image, click on it.

To force a specific repository to be scanned again, select **Scan** from the top right of the results table, then click on the specific registry to rescan.

Scan images in Google Artifact Registry

Edit on GitHub

Although Artifact Registry supports a number of content types (for example, Java, Node.js, and Python language packages), Prisma Cloud only supports discovering and scanning Docker images.

Prisma Cloud doesn't support scanning Helm charts saved as OCI images and stored in Artifact Registry. Helm charts saved as OCI images have a single layer that contains the Helm package. It is only a way to store a Helm chart, but it has no meaning in terms of a container. Therefore, Prisma Cloud can't scan it.

Create a new registry scan

Prerequisites:

- You've deployed a Defender somewhere in your environment.
- You've created GCP credentials (service account) with, at minimum, the Artifact Registry Reader role (*roles/artifactregistry.reader*).
- You've added the service account credentials to the Prisma Cloud Compute Console credentials store under **Manage > Cloud accounts**.

STEP 1 | Open Console, then go to **Defend > Vulnerabilities > Images > Registry settings**.

STEP 2 | Click **Add registry**.

STEP 3 | In **Version**, select **Google Artifact Registry**.

STEP 4 | In **Registry**, enter the registry address.

The format for the address is <GCP-region>-docker.pkg.dev.

For example, europe-north1-docker.pkg.dev

Multi-region registry addresses are also supported, <GCP-multi-region>-docker.pkg.dev. For example, us-docker.pkg.dev, europe-docker.pkg.dev, and asia-docker.pkg.dev.

STEP 5 | In the **Credential** field, select the service account you created in **Manage > Cloud accounts**.

If the credentials haven't been created already, click **+** to create them now. If creating credentials:

1. In the **Cloud accounts onboarding** dialog, select **GCP** for the cloud provider.
2. Enter a credential name.
3. Select the credential level.
4. Paste the JSON token blob from your service account into the **Service Account** field. Leave the **API Token** field blank.
5. Click **Next**.
6. Disable agentless scanning, then click **Next**.
7. Disable cloud discovery, then click **Add account**.

STEP 6 | (Optional) Refine which images Prisma Cloud should scan with the **Repositories**, **Repositories to exclude**, **Tags**, and **Tags to exclude** fields.

Pattern matching is supported.

STEP 7 | In **OS type**, specify whether the repo holds **Linux** or **Windows** images.

STEP 8 | In **Scanners scope**, select the Defenders to use for the scan.

Console selects the available Defenders from this scope to execute the scan job. For more information, see deployment patterns.

STEP 9 | In **Number of scanners**, enter the number of Defenders across which scan jobs can be distributed.

STEP 10 | Set **Cap** to the number of most recent images to scan.

Leaving **Cap** set to **5** will scan the 5 most recent images. Setting this field to **0** will scan all images.

STEP 11 | Click **Add**.

STEP 12 | Click **Save and scan**.

Results

Verify that the images in the repository are being scanned.

STEP 1 | Go to **Monitor > Vulnerabilities > Images > Registries**.

A progress indicator at the top right of the window shows the status of the current scan. As the scan of each image is completed, the findings are added to the results table.

STEP 2 | To get details about the vulnerabilities in an image, click on it.

To force a specific repository to be scanned again, click **Scan** at the top right of the results table, and then click on the specific repository to rescan.

Scan images in Google Container Registry (GCR)

Edit on GitHub

Create a new registry scan

Prerequisites:

- You have installed a Defender somewhere in your environment.
- GCR access is governed by Google's storage permissions. For Prisma Cloud to scan GCR, your service account must have the GCP IAM **Storage View** role.
- You must grant Prisma Cloud access to your registry with a service account JSON key file. Your JSON token blob will look something like this:

```
{
  "type": "service_account",
  "project_id": "my_project_id",
  "private_key_id": "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
  "private_key": "-----BEGIN PRIVATE KEY-----
\nXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
\n-----END PRIVATE KEY-----\n",
  "client_email":
 "XXXXXXXXXXXXXXX@XXXXXXXXXXXXXX.iam.gserviceaccount.com",
  "client_id": "XXXXXXXXXXXXXXXXXXXXXXXXX",
  "auth_uri": "https://accounts.google.com/o/oauth2/auth",
  "token_uri": "https://oauth2.googleapis.com/token",
  "auth_provider_x509_cert_url": "https://www.googleapis.com/
oauth2/v1/certs",
  "client_x509_cert_url": "https://www.googleapis.com/robot/v1/
metadata/x509/
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX.iam.gserviceaccount.com"
}
```

STEP 1 | Open Console, then go to **Defend > Vulnerabilities > Registry**.

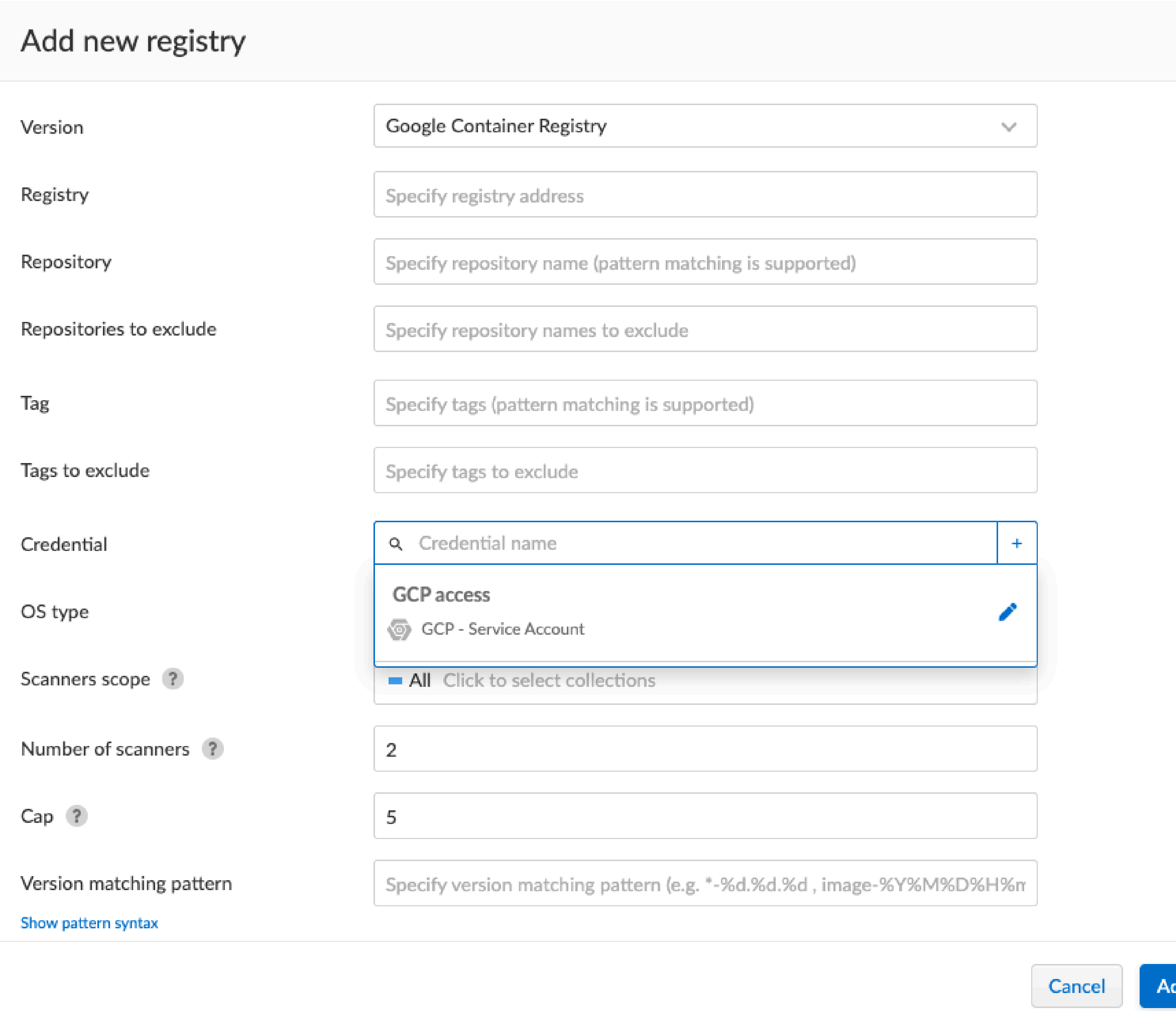
STEP 2 | Click **Add registry**.

STEP 3 | In the **Version** field, select **Google Container Registry**.

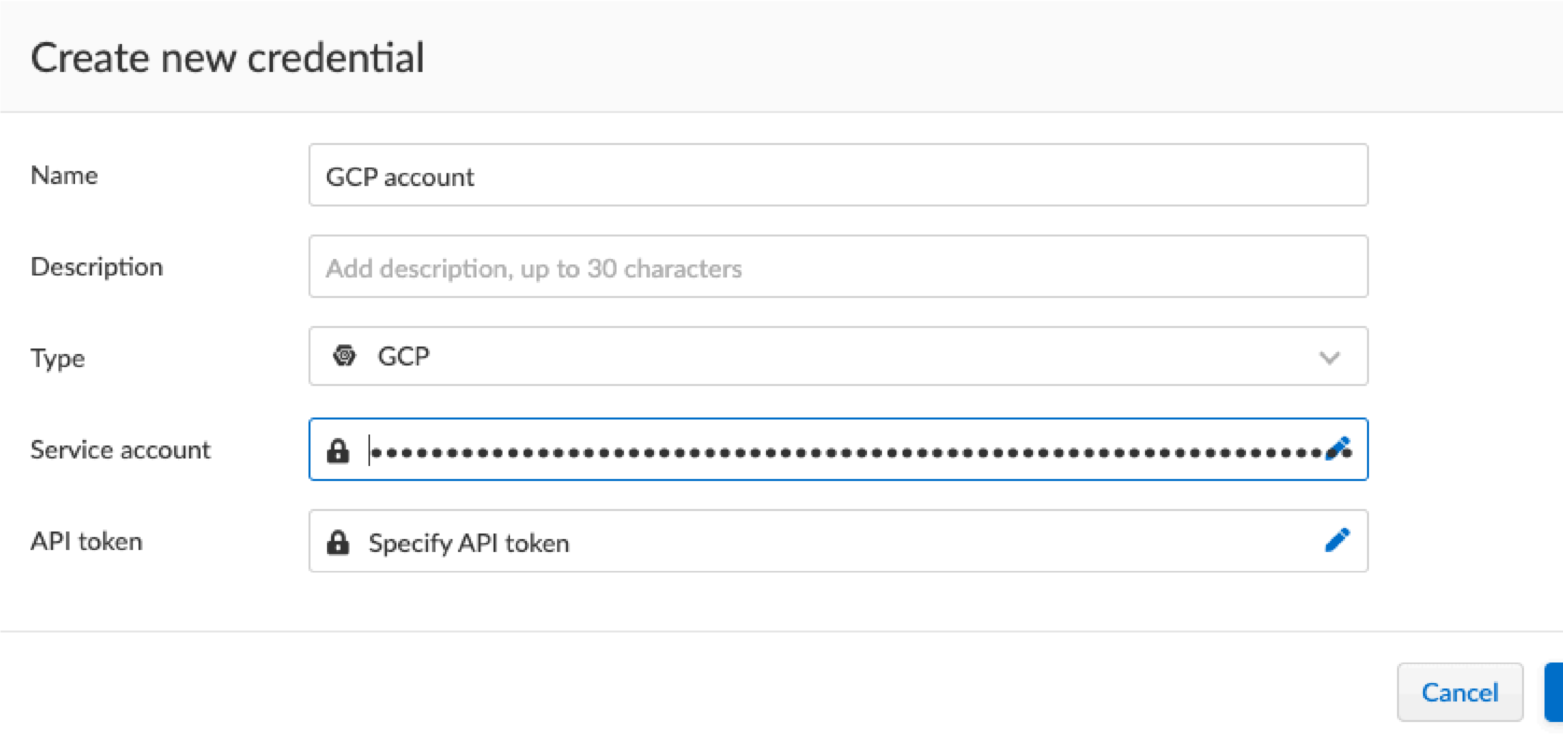
STEP 4 | Enter the registry address in the **Registry** field (e.g. *gcr.io*).

STEP 5 | Enter the repository name followed by /* in the **Repository** field (e.g. *company-sandbox/**).

STEP 6 | In the **Credential** field, enter the credentials required to access the registry. If the credentials have already been created in the Prisma Cloud credential store, select it. If not, click **Add** to create new credentials.



1. Select the **GCP** credential type and credential level, then paste the JSON token blob from your service account into the **Service Account** field. Leave the **API Token** field blank.



For GCP organizations with hundreds of projects, scanning GCR using organization level credentials might affect the scanning performance due to long query time from GCP. Therefore, the best approach to reduce scan time and to avoid potential timeouts, is to divide the projects within your organization into multiple GCP folders. Then, create a service account and credential for each one of them, and use these credentials for GCR scanning.

2. Save your credentials.

STEP 7 | In **OS type**, specify whether the repo holds **Linux** or **Windows** images.

STEP 8 | In **Scanners scope**, specify the collections of defenders to use for the scan.

Console selects the available Defenders from the scope to execute the scan job according to the **Number of scanners** setting. For more information, see deployment patterns.

STEP 9 | In **Number of scanners**, enter the number of Defenders across which scan jobs can be distributed.

STEP 10 | Set **Cap** to the number of most recent images to scan.

Leaving **Cap** set to **5** will scan the 5 most recent images. Setting this field to **0** will scan all images.

STEP 11 | Click **Add**.

STEP 12 | Click the **Save** button.

Results

Verify that the images in the repository are being scanned.

STEP 1 | Go to **Monitor > Vulnerabilities > Images > Registries**.

A progress indicator at the top right of the window shows the status of the current scan. As the scan of each image is completed, its findings are added to the results table.

STEP 2 | To get details about the vulnerabilities in an image, click on it.

To force a specific repository to be scanned again, select **Scan** from the top right of the results table, then click on the specific registry to rescan.

Scan images in Harbor Registry

Edit on GitHub

Configure Prisma Cloud to scan your Harbor registry.

Create a new registry scan

To scan a repository in Harbor, create a new registry scan setting.

STEP 1 | Open Console

STEP 2 | Go to **Defend > Vulnerabilities > Images > Registry Settings**.

STEP 3 | Click **Add Registry**.

STEP 4 | In the dialog, enter the following information:

1. In the **Version** drop-down list, select **Harbor**.
2. In the **Registry** field, enter the FQDN of your Harbor registry (https://).
3. In **Repository**, enter the name of the repository to scan, or leave this blank to scan all repositories.
4. In **Tag**, enter an image tag. Leave this field blank to scan all images, regardless of any tags.
5. In **Credential**, select the credentials to use.

   If Console doesn't have a copy of your credentials yet, click **Add New**. Select **Basic authentication**, and fill out the rest of the fields. The minimum required credentials for each repository is **Limited Guest**.
6. The **Bypass deployment security** toggle is applicable only when using Prisma Cloud Compute pluggable scanner.

   To scan Harbor projects with the deployment security setting enabled, Harbor requires additional permissions that can not be granted with a regular user credentials.

   When the toggle is ON, Prisma Cloud Compute scans the registry using a temporary token provided by Harbor in the scanning request, instead of the credentials provided in the settings. This token has sufficient permissions to bypass the deployment security setting, and it's the mechanism Harbor provides to allow external security scanners to scan these projects.

When the toggle is OFF, Prisma Cloud Compute uses the credentials provided in the setting to scan the registry. It will not be able to scan images in Harbor projects with the deployment security setting enabled.

Harbor's token expiration time must be at least 30 minutes so that it won't expire during the Prisma Cloud scanning process. If you set the expiration period to less than 30 minutes, registry scanning might fail.

7. In **OS type**, specify whether the repo holds **Linux** or **Windows** images.
8. In **Scanners scope**, specify the collections of defenders to use for the scan.

   Console selects the available Defenders from the scope to execute the scan job according to the **Number of scanners** setting. For more information, see deployment patterns.
9. In **Number of scanners**, enter the number of Defenders across which scan jobs can be distributed.
10. Set **Cap** to the number of most recent images to scan. Leaving **Cap** set to the default value of **5** will scan the most recent 5 images. Setting this field to **0** will scan all images.
11. Click **Add**.

STEP 5 | Click the **Save** button.

Results

Verify that the images in the repository are being scanned.

STEP 1 | Go to **Monitor > Vulnerabilities > Images > Registries**.

A progress indicator at the top right of the window shows the status of the current scan. As the scan of each image is completed, its findings are added to the results table.

STEP 2 | To get details about the vulnerabilities in an image, click on it.

To force a specific repository to be scanned again, select **Scan** from the top right of the results table, then click on the specific registry to rescan.

Integrate Compute as pluggable scanner

Configure Compute as a pluggable scanner to view vulnerability scan results in Harbor Console itself, in addition to Compute Console. To add Compute as a vulnerability scanner in Harbor, follow steps outlined above for adding Harbor registry in Compute Console. Thereafter, follow the steps below in Harbor:

STEP 1 | In Harbor, go to the Administration > Interrogation Services page and click New Scanner.

STEP 2 | In the pop up, enter your Compute Console details.

1. In **Name**, provide a name for the scanner.
2. In **HTTP Endpoint**:

   Login to your Compute Console, navigate to Defend > Vulnerabilities > Registry page. Under **Harbor scanner adapter** section, copy the URL from field b: "Use the following URL as the Harbor Scanner endpoint".

   This section only becomes visible after adding Harbor Registry in Compute Console as a registry as per steps outlined in section above.
3. **Authorization: None**:

   Due to a current bug in all Harbor versions other types of authentication methods result in error messages. See https://github.com/goharbor/harbor/issues/12919

STEP 3 | Test Connection and click **Save**.

You can now go to Vulnerability tab under Interrogation services and hit Scan Now for vulnerability scanning reports.

Note that when a scan is revoked from Harbor Console using Compute as a vulnerability scanner, Harbor pulls scan from Compute Console. In order to receive faster results, make sure you scan the registry on Compute Console as well.

Scan images in IBM Cloud Container Registry

Edit on GitHub

To scan a repository on IBM Cloud Container Registry, create a new registry scan setting.

Create a new registry scan

Prerequisites: You have installed a Defender somewhere in your environment.

STEP 1 | Open Console

STEP 2 | Set up credentials so that Prisma Cloud can access the images in your registry.

1. Go to **Manage > Authentication > Credentials Store**.
2. Click **Add credential**.
3. Enter a name.
4. In **Type**, select **IBM Cloud**.
5. In **Account GUID**, enter the GUID for your IBM Cloud account. See the IBM Cloud Docs to learn how to get the GUID of an account
6. In **API Key**, enter your API key. See the IBM Cloud Docs to learn how to create a service ID for Prisma Cloud, and then create an API key for the service ID.
7. Click **Save**.

STEP 3 | Go to **Defend > Vulnerabilities > Images > Registry settings**.

STEP 4 | Click **Add registry**.

STEP 5 | In the dialog, enter the following information:

1. From the **Version** drop-down list, select **IBM Cloud Container Registry**.
2. In **Registry**, enter the registry address for your region.

   For example, if you use the us-south registry, enter **us.icr.io**.
3. In **Namespace**, enter the namespace for your image.

   For images in private registries, this field is mandatory. For images in IBM's public registry, leave this field blank. Wildcards are not supported for this field.

   IBM provides namespaces to help you organize your registries. Namespaces are appended to the registry URL as follows: *registry.<REGION>.icr.io/<NAMESPACE>*
4. In **Repository name**, specify the repository to scan.

   If you leave this field blank or enter a wildcard, Prisma Cloud finds and scans all repositories in the registry.

   If you specify a partial string that ends with a wildcard, Prisma Cloud finds and scans all repositories that start with the partial string.

   If you specify an exact match, Prisma Cloud scans just the specified repository.
5. In **Tag**, enter an image tag.

   If you leave this field blank or enter a wildcard, Prisma Cloud finds and scans all images in the repository.

   If you specify a partial string that ends with a wildcard, Prisma Cloud finds and scans all images that match the partial tag.

   If you specify an exact match, Prisma Cloud scans just the specified image with specified tag.
6. In **Credential**, select the credential you just created.
7. In **OS type**, specify whether the repo holds **Linux** or **Windows** images.
8. In **Scanners scope**, specify the collections of defenders to use for the scan.

   Console selects the available Defenders from the scope to execute the scan job according to the **Number of scanners** setting. For more information, see deployment patterns.
9. In **Number of scanners**, enter the number of Defenders across which scan jobs can be distributed.
10. **Cap** the number of images to scan.

    Specify the maximum number of images to scan in the given repository, sorted according to last modified date. To scan all images in a repository, set **Cap** to 0. For a complete explanation of **Cap**, see the table in registry scan settings.
11. Click **Add**.

STEP 6 | Click the **Save** button.

Results

Verify that the images in the repository are being scanned.

STEP 1 | Go to **Monitor > Vulnerabilities > Images > Registries**.

A progress indicator at the top right of the window shows the status of the current scan. As the scan of each image is completed, its findings are added to the results table.

STEP 2 | To get details about the vulnerabilities in an image, click on it.

To force a specific repository to be scanned again, select **Scan** from the top right of the results table, then click on the specific registry to rescan.

Scan images in Artifactory Docker Registry

Edit on GitHub

Artifactory is a service for hosting and distributing container images. Artifactory lets you segment the service by repository key, so that you can allocate dedicated registries per project, team, or any other facet. Repositories can be accessed with the Docker client. A repository is a collection of related images, versioned by tag.

Artifactory lets you configure how images in the repository are accessed with a setting called the *Docker Access Method*. Prisma Cloud supports the subdomain method and the repository method. The port method is not supported.

In the subdomain model, the repository is accessed through a reverse proxy. Each Docker repository is individually addressed by a unique value, known as the repository key, positioned in subdomain of the registry's URL.

```
$ docker {pull|push} <REPOSITORY_KEY>.art.example.com/<IMAGE>:<TAG>
```

In the repository path model, each repository can be directly addressed. The repository key is part of the path to the image repo.

```
$ docker {pull|push} art.example.com:443/<REPOSITORY_KEY>/
<IMAGE>:<TAG>
```

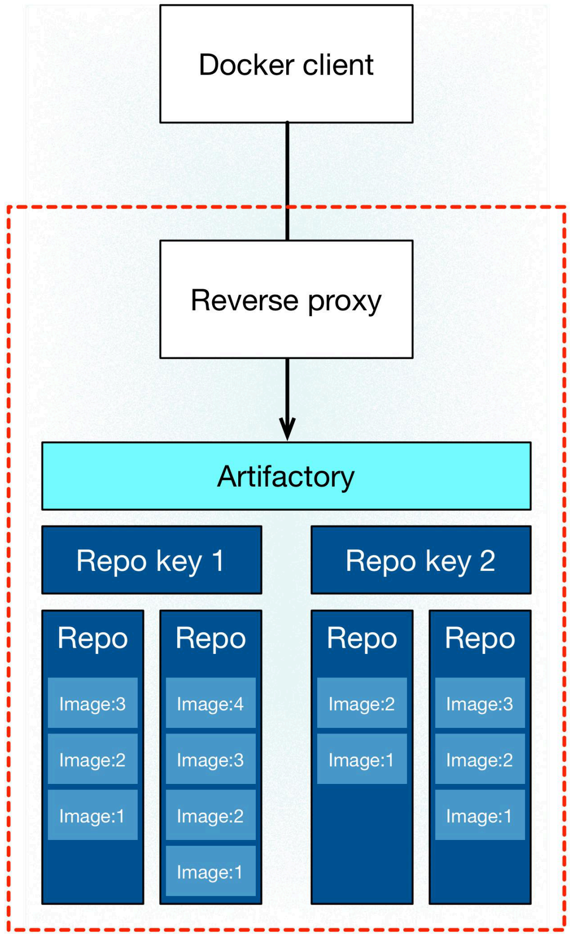
Artifactory recommends that the subdomain method be used for production environments. The repository model is suitable for small test setups and proof of concepts.

Configuring Prisma Cloud to scan images in your registry

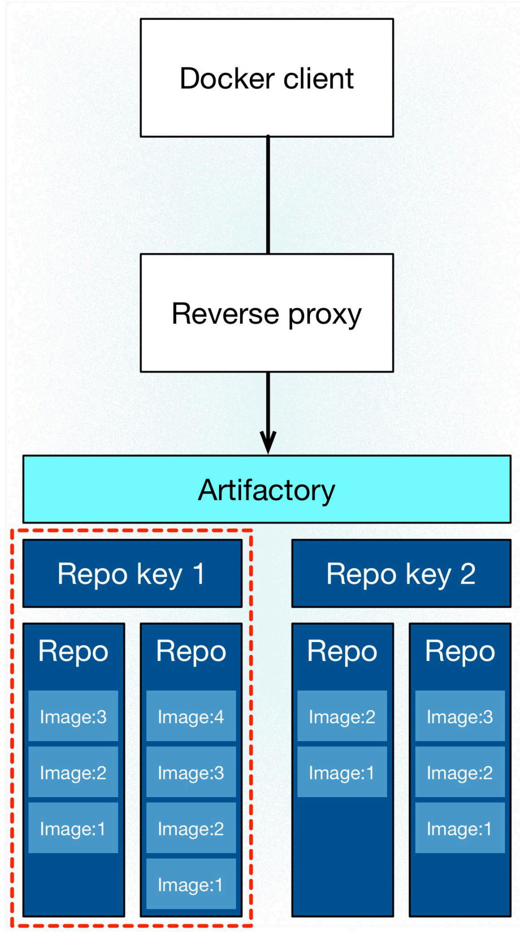
To scan images in a JFrog Artifactory Docker registry (on-prem/self-hosted version only), create a new registry scan setting. You have a couple of options for setting up your scan on Prisma Cloud:

1) Autodiscover and scan all images in all repos across the Artifactory service for versions of Artifactory greater than or equal to 6.2.0. In the registry scan settings, set the version to **JFrog Artifactory** and set the registry address to your reverse proxy.

JFrog Cloud is not supported.



2) Scan all repositories under a repository key for the subdomain method. Repository keys effectively subdivide the Artifactory service into stand-alone fully-compliant Docker v2 registries. In the registry scan settings, set the version to **Docker Registry v2**, and set the registry address to the full path to the "sub-registry". For example: https://<REPOSITORY_KEY>.example.com/.



Prerequisites: You have installed a Defender somewhere in your environment.

Last downloaded date

JFrog Artifactory lets security tools download image artifacts without impacting the value for the **Last Downloaded** date. This is especially important when you depend on artifact metadata for purge/clean-up policies.

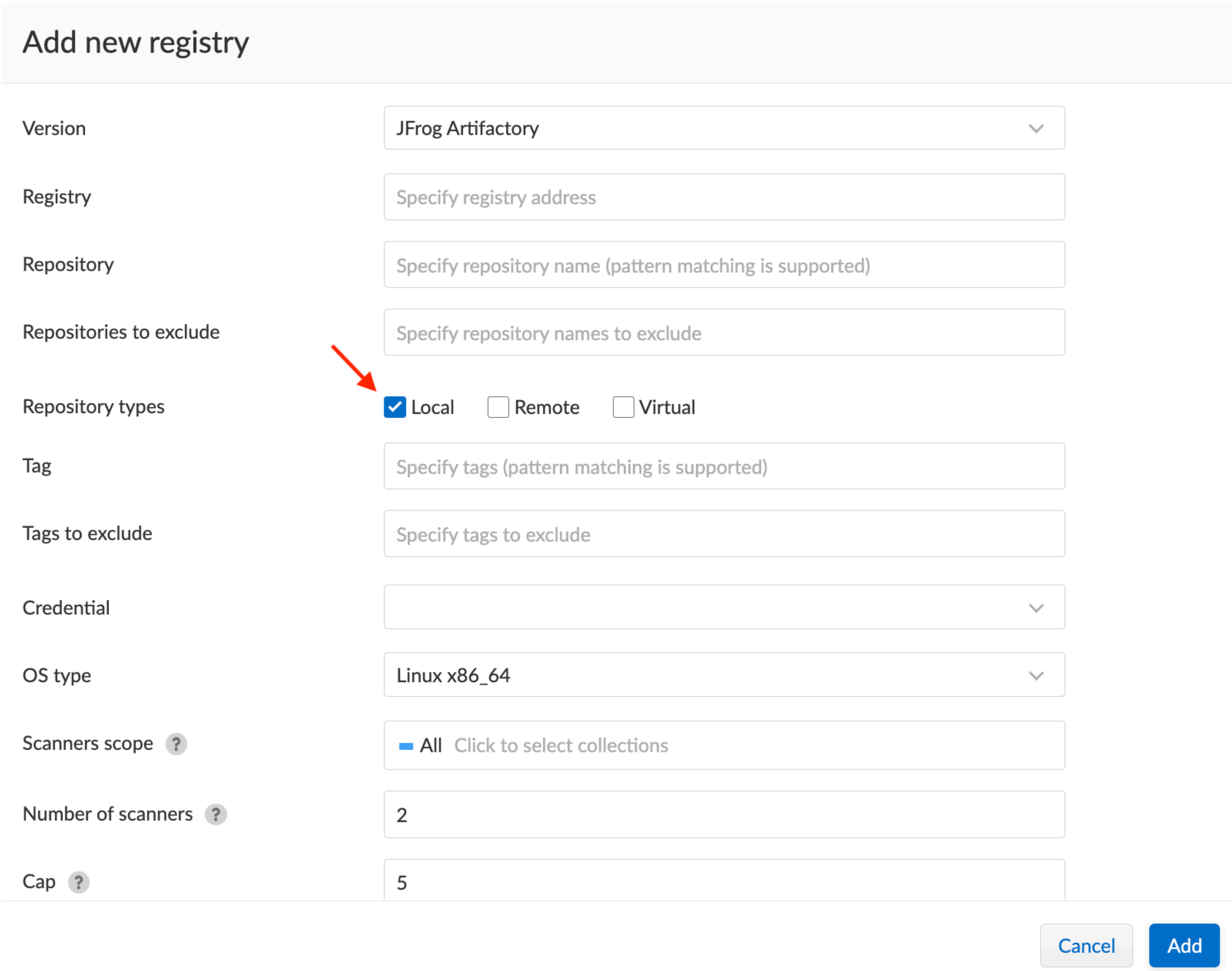
The Prisma Cloud scanning process no longer updates the **Last Downloaded** date for all image and manifest files of all the images in the registry.

Requirements:

- JFrog Artifactory version 7.21.3 and later.
- In your Prisma Cloud registry scan settings, version must be set to **JFrog Artifactory**. If you set version to **Docker V2**, Prisma Cloud uses the Docker API, which doesn't offer the same support.

"Transparent security tool scanning" is **not** supported for anything other than **Local** repositories. If you select anything other than **Local** in your scan configuration, including virtual repos backed by local repos, then Prisma Cloud automatically uses the Docker API to scan all repositories (local, remote, and virtual). When using Docker APIs, the **Last Downloaded** field in local JFrog Artifactory registries will be impacted by scanning.

The following screenshot shows the supported configuration for this capability:



If you've got a mix of local, remote, and virtual repositories, and you want to ensure that the **Last Downloaded** date isn't impacted by Prisma Cloud scanning, then create separate scan configurations for local repositories and remote/virtual repositories.

*The **Last Downloaded** date of the image and manifest files of the images that are eventually pulled for scanning, based on your registry scan policy, will be updated. The scan process first evaluates which images to scan by retrieving all manifest files for all images. In this phase of the scan, the **Last Downloaded** date will no longer be impacted. In the next phase, where Prisma Cloud actually pulls an image to be scanned, the manifest file's **Last Downloaded** date will be updated. Often, the number of images scanned will be a subset of all images in the registry, but that's based on your scan policy.*

*Just because an image has been selected for scanning, doesn't mean that it will actually be pulled. If an image's hash hasn't changed, it won't be pulled for scanning, so the **Last Downloaded** date will be unchanged.*

Grant Prisma Cloud access to your repo

When configuring Prisma Cloud to scan Artifactory as standard Docker v2 registries (i.e. in your scan configuration, you've set **Version** to **Docker registry v2**), Prisma Cloud requires only standard scanning permissions.

When configuring Prisma Cloud to autodiscover and scan all images in all repos across the Artifactory service (i.e. in your scan configuration, you've set **Version** to **JFrog Artifactory**), Prisma Cloud requires an account with Administrator privileges (admin user). This is because some of the Artifactory APIs that Prisma Cloud uses to perform disovery require Administrator privileges.

STEP 1 | Log in Prisma Cloud Console, then go to **Manage > Authentication > Credentials Store**.

STEP 2 | Click **Add credential**.

STEP 3 | Enter a credential name, such as **JFrog Artifactory**.

STEP 4 | In **Type**, select **Basic authentication**.

STEP 5 | In **Username**, enter a username.

STEP 6 | In **Password**, enter a password.

STEP 7 | Click **Save**.

Configure the scan

After you set up your credentials, create a new registry scan setting.

STEP 1 | Open Console, then go to **Defend > Vulnerabilities > Registry**.

STEP 2 | Click **Add registry**.

STEP 3 | In the dialog, enter the following information:

1. From the **Version** drop-down list, select one of:
   - **JFrog Artifactory** — Autodiscover and scan all images in all repos across the Artifactory service. Only JFrog on-prem/self-hosted is supported.
   - **Docker Registry v2** — Scan all images in all repos under a specific repository key.
2. In **Registry**, specify the address to scan.
   - If you selected **JFrog Artifactory**, enter the FQDN of the reverse proxy.
   - If you selected **Docker Registry v2**, enter the FQDN, including subdomain, of the sub-registry.
3. In **Repository**, specify the repository to scan.

   If you leave this field blank or enter a wildcard, Prisma Cloud finds and scans all repositories in the registry.

   If you specify a partial string that ends with a wildcard, Prisma Cloud finds and scans all repositories that start with the partial string.

   If you specify an exact match, Prisma Cloud scans just the specified repository.
4. In **Repository types**, select the repository types that Prisma Cloud should scan.

   This setting is available only when **Version** is set to **JFrog Artifactory**. Specify at least one registry type (local, remote, virtual).
5. Do the same with the **Tag** field.
6. In **Credential**, select the JFrog Artifactory credentials you created.
7. In **OS type**, specify whether the repo holds **Linux** or **Windows** images.
8. In **Scanners scope**, specify the collections of defenders to use for the scan.

   Console selects the available Defenders from the scope to execute the scan job according to the **Number of scanners** setting. For more information, see deployment patterns.
9. In **Number of scanners**, enter the number of Defenders across which scan jobs can be distributed.
10. **Cap** the number of images to scan.

    Cap specifies the maximum number of images to scan in the given repository, sorted according to last modified date. To scan all images in a repository, set **Cap** to 0. For a complete explanation of **Cap**, see the table in registry scan settings.
11. Click **Add**.

STEP 4 | Click the **Save** button.

Results

Verify that the images in the repository are being scanned.

STEP 1 | Go to **Monitor > Vulnerabilities > Images > Registries**.

A progress indicator at the top right of the window shows the status of the current scan. As the scan of each image is completed, its findings are added to the results table.

STEP 2 | To get details about the vulnerabilities in an image, click on it.

To force a specific repository to be scanned again, select **Scan** from the top right of the results table, then click on the specific registry to rescan.

Troubleshooting

If Artifactory is deployed as an insecure registry, Defender cannot pull images for scanning without first configuring an exception in the Docker daemon configuration. Specify the URL of the insecure registry on the machine where the registry scanning Defender runs, then restart the Docker service. For more information, see the Docker documentation.

Scan images in OpenShift integrated Docker registry

Edit on GitHub

To scan an OpenShift integrated registry, create a new registry scan setting.

Create a new registry scan

Prerequisites:

- Installed a Defender within in your OpenShift cluster.
- Service account to authenticate to the internal registry.
  - We recommend you use the existing *twistlock-service* account.
  - The Defender authenticates to the OpenShift registry using this service account.
  - Added the cluster role permission of registry-viewer to the twistlock-service account.

    ```
    oc adm policy add-cluster-role-to-user registry-viewer
     system:serviceaccount:<twistlock_project>:twistlock-service
    ```
  - Obtain the password for the twistlock-service account.
    - Determine the secret used by the service account *oc describe sa twistlock-service -n <twistlock_project>*
    - Use the **Image pull secrets** value (e.g. twistlock-service-dockercfg-64jtt) in the following command, for example:

      ```
      oc get secret twistlock-service-dockercfg-64jtt -n twistlock --
      output=json|grep openshift.io/token-secret.value
      ```
    - Copy the openshift.io/token-secret.value for use later in the workflow.
    - If you use the OpenShift UI to obtain the token, click view-all to see the full token.

STEP 1 | Open Console, then go to **Defend > Vulnerabilities > Registry**.

STEP 2 | Click **Add registry**.

STEP 3 | In **Version**, select **Red Hat OpenShift**.

STEP 4 | Enter the registry address in the **Registry** field.

STEP 5 | In **Repository** , specify the repository to scan.

If you leave this field blank or enter a wildcard, Prisma Cloud finds and scans all repositories in the registry.

If you specify a partial string that ends with a wildcard, Prisma Cloud finds and scans all repositories that start with the partial string.

If you specify an exact match, Prisma Cloud scans just the specified repository.

STEP 6 | Click in the **Credential** field, then click **Add new**.

1. Select the **Basic authentication** credential type
2. In **Username**, enter any arbitrary value.
3. In **Password**, enter the service account token you copied when you completed the prerequisite.
4. Save your credentials.

STEP 7 | In **OS type**, specify whether the repo holds **Linux** or **Windows** images.

STEP 8 | In **Scanners scope**, specify the collections of defenders to use for the scan.

Console selects the available Defenders from the scope to execute the scan job according to the **Number of scanners** setting. For more information, see deployment patterns.

STEP 9 | In **Number of scanners**, enter the number of Defenders across which scan jobs can be distributed.

STEP 10 | Set **Cap** to the number of most recent images to scan. Leaving **Cap** set to **5** will scan the 5 most recent images. Setting this field to **0** will scan all images.

STEP 11 | Click **Add**.

STEP 12 | Click the **Save** button.

Results

Verify that the images in the repository are being scanned.

STEP 1 | Go to **Monitor > Vulnerabilities > Images > Registries**.

A progress indicator at the top right of the window shows the status of the current scan. As the scan of each image is completed, its findings are added to the results table.

STEP 2 | To get details about the vulnerabilities in an image, click on it.

To force a specific repository to be scanned again, select **Scan** from the top right of the results table, then click on the specific registry to rescan.

Trigger registry scans with Webhooks

Edit on GitHub

You can use webhooks to trigger a scan when images in your registry's repositories are added or updated.

Prisma Cloud supports webhooks for:

- Docker Hub
- Docker Registry
- Azure Registry
- Nexus Repository
- JFrog Artifactory

Prisma Cloud requires Docker Registry 2.4 or later.

Google Container Registry and Amazon EC2 Container Registry do not currently support webhooks.

For Docker Hub, you must have Automated Builds enabled for your repository. Docker Hub webhooks are called when an image is built or a new tag is added to your automated build repository.

For Docker Private Registry, webhooks are called when manifests are pushed or pulled, and layers are pushed or pulled. Prisma Cloud scans images in response to layer push events.

For Azure Registry, you can configure webhooks for your container registry that generate events when certain actions are performed against it. See Azure's documentation for more information.

The benefit of webhook-initiated scans is that they are triggered as soon as images change, but support is limited to Docker Hub, Docker Registry, and Azure Registry. Prisma Cloud also supports scheduled registry scans, with support for almost all registry types, including Google Container Registry and Amazon EC2 Container Registry.

Securing Console's management port

Webhooks call the Prisma Cloud API on Console's management ports over either HTTP or HTTPS.

Although it is convenient to test webhooks with HTTP, we strongly recommend that you set up webhooks to call Console over HTTPS. To call webhooks over HTTPS, you must install a certificate trusted by the registry. For more information about securing Console's management port with a custom cert, see certs customization for Console TLS communication.

By default, Prisma Cloud uses self-signed certificates to secure HTTP traffic. Self-signed certificates are not supported (trusted) by Docker Hub, and Docker Registry would require you to configure Prisma Cloud as a trusted CA (not supported, and not recommended). Instead install a certificate signed by a trusted certificate authority (CA), such as Comodo or Symantec.

Setting up webhooks

To set up webhook-initiated scans, configure your registry's webhook with the URL provided in Console. The following procedure shows you how to set up webhooks in Docker Hub.

Prerequisites: Docker Hub, with Automated Builds enabled.

STEP 1 | Open Console.

STEP 2 | Go to **Defend** > **Vulnerabilities** > **Registry**.

STEP 3 | Scroll down to the section **Registry webhooks**, then enter the following information:

1. In the drop down list, select the DNS name or IP address that the registry can use to reach the Console.

   Your selection generates a URL that you will use to configure the registry.
2. Copy the URL.

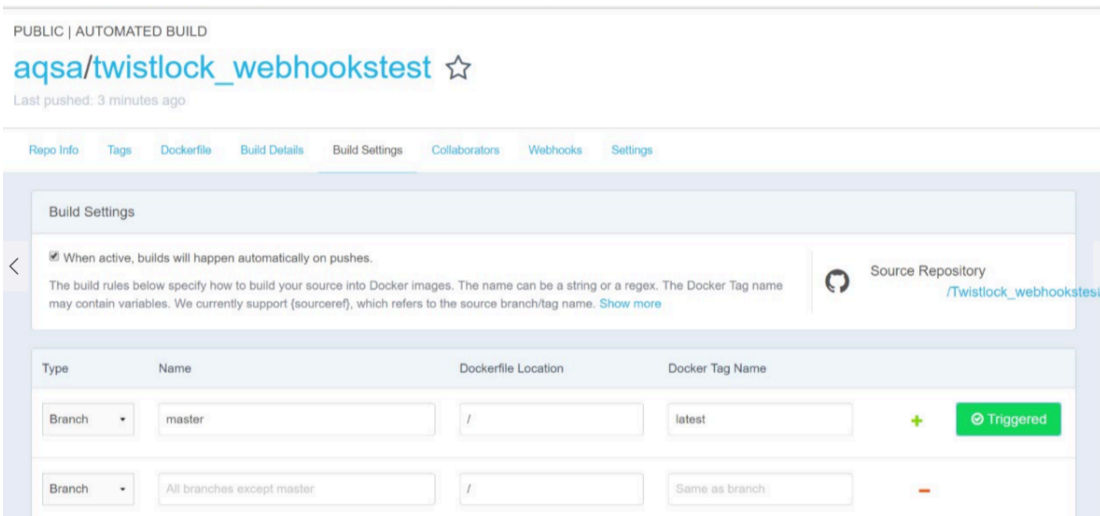
   By default, the generated URL employs HTTP. For HTTPS, replace http:// with https://.

STEP 4 | Configure your repository.

The following sections show how to configure Docker Hub and Nexus Repository. For other repositories, consult the vendor's documentation.

- Docker Hub
- Nexus Repository

STEP 5 | Test the integration by triggering a build.



STEP 6 | Go to **Monitor** > **Vulnerabilities** > **Registry** to view the scan report. Prisma Cloud scans the image as soon as it is built.

Configuring Docker Hub

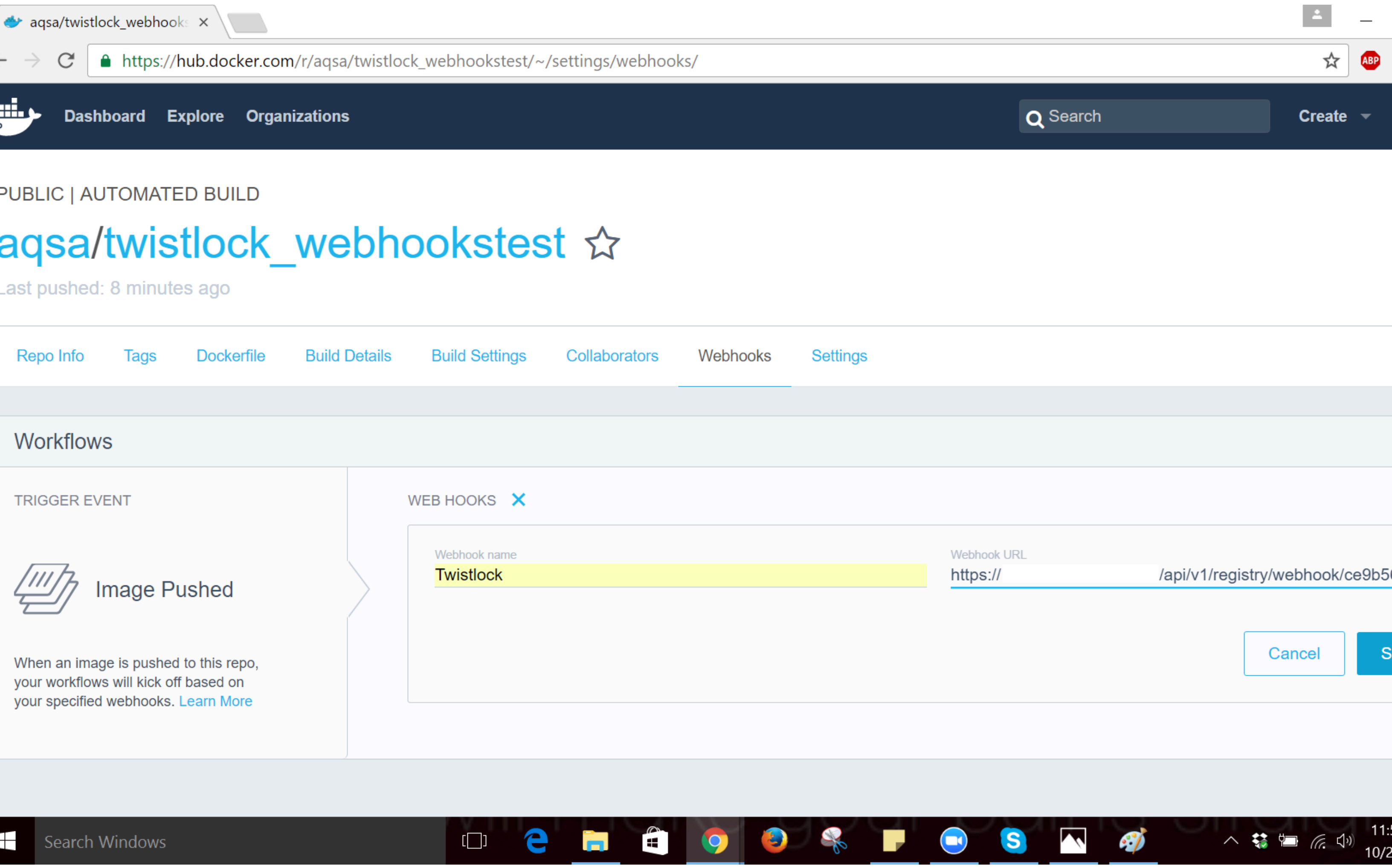
Configure your Docker Hub repository.

STEP 1 | Log into Docker Hub.

STEP 2 | Select a repository, and then click Webhooks.

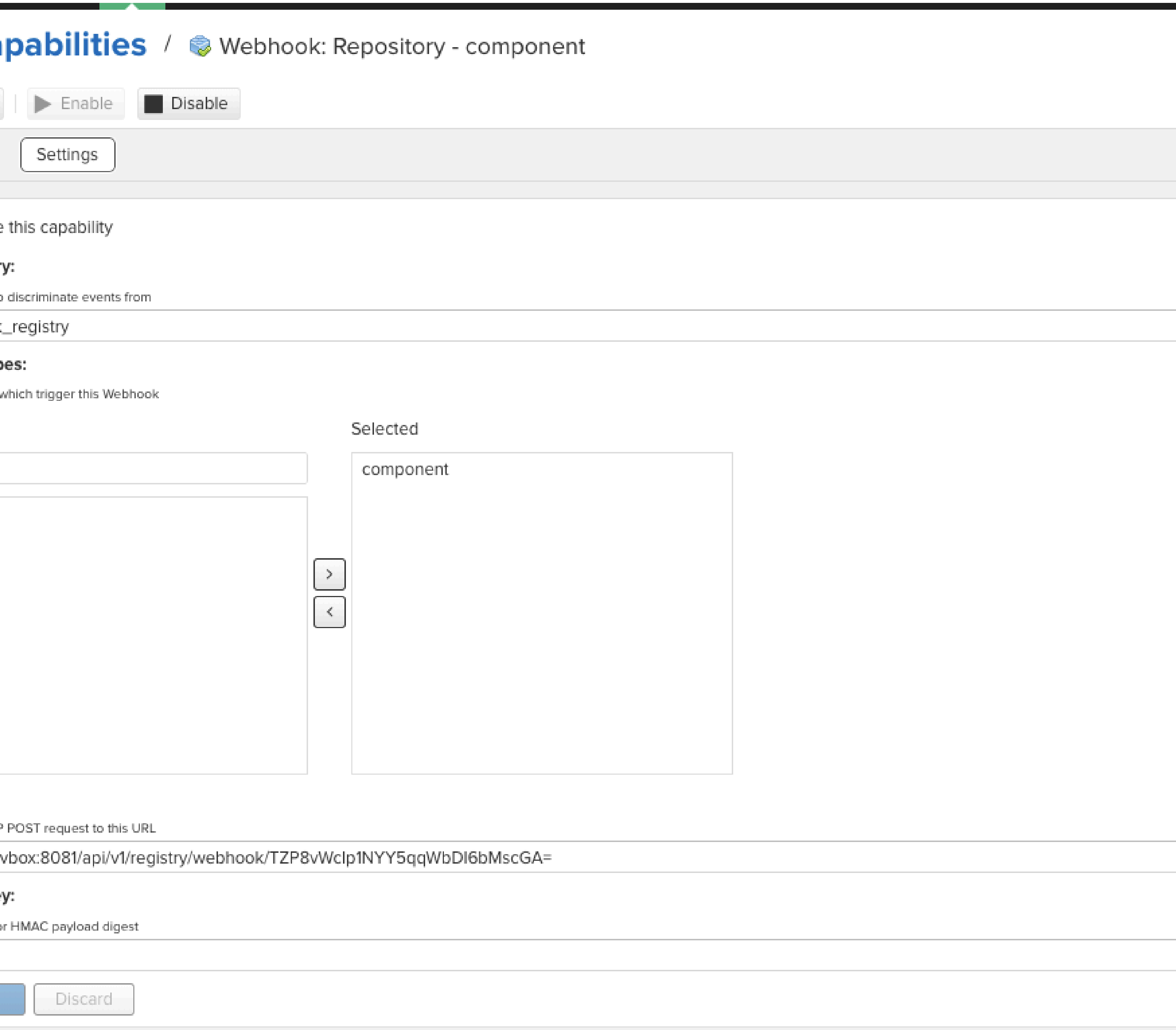
STEP 3 | Create a new webhook. Specify a name, and paste the URL you copied from Console.

STEP 4 | Click Save.



Configuring Nexus Repository

Configure the Nexus Repository. When setting up webhooks in Nexus Repository, select the "component" event type for triggering the webhooks.


pabilities / Webhook: Repository - component
Enable
Disable
Settings
e this capability
ry:
o discriminate events from
_registry
pes:
which trigger this Webhook
Selected
component
P POST request to this URL
vbox:8081/api/v1/registry/webhook/TZP8vWclp1NYY5qqWbDl6bMscGA=
ry:
r HMAC payload digest
Discard

Base images

Edit on GitHub

Prisma Cloud lets you filter out base image vulnerabilities from your scan reports.

A base image is a starting point or an initial step for the image. Dockerfile usually starts from a base image. Filtering out vulnerabilities which their source is the base image can help your teams focus on the vulnerabilities relevant for them to fix.

Excluding base image vulnerabilities is currently not supported for Windows images.

Define base images

For Prisma Cloud to be able to exclude base image vulnerabilities, first identify the base images in your environment.

To define your base images, go to **Defend > Vulnerabilities > Images > Base images**. The base images you define must reside in your registry and they must be scanned in order to exclude their vulnerabilities from scan reports.

STEP 1 | Open Console.

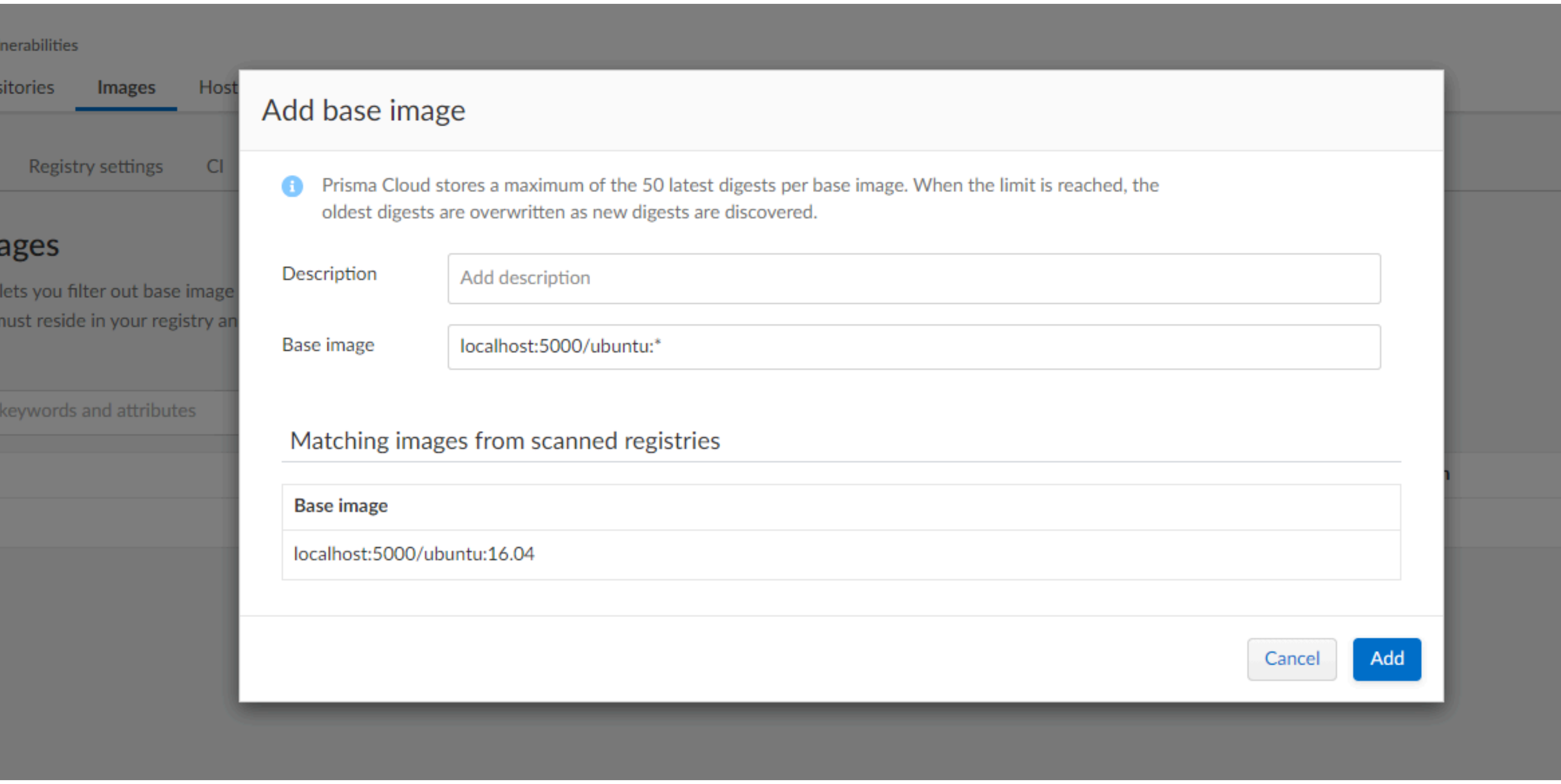
STEP 2 | Go to **Defend > Vulnerabilities > Images > Base images**.

STEP 3 | Click **Add New**.

STEP 4 | Specify the base image and provide a description for it.

The base image should be specified in the following format: *registry/repo:tag*. You can use wildcards for the tag definition.

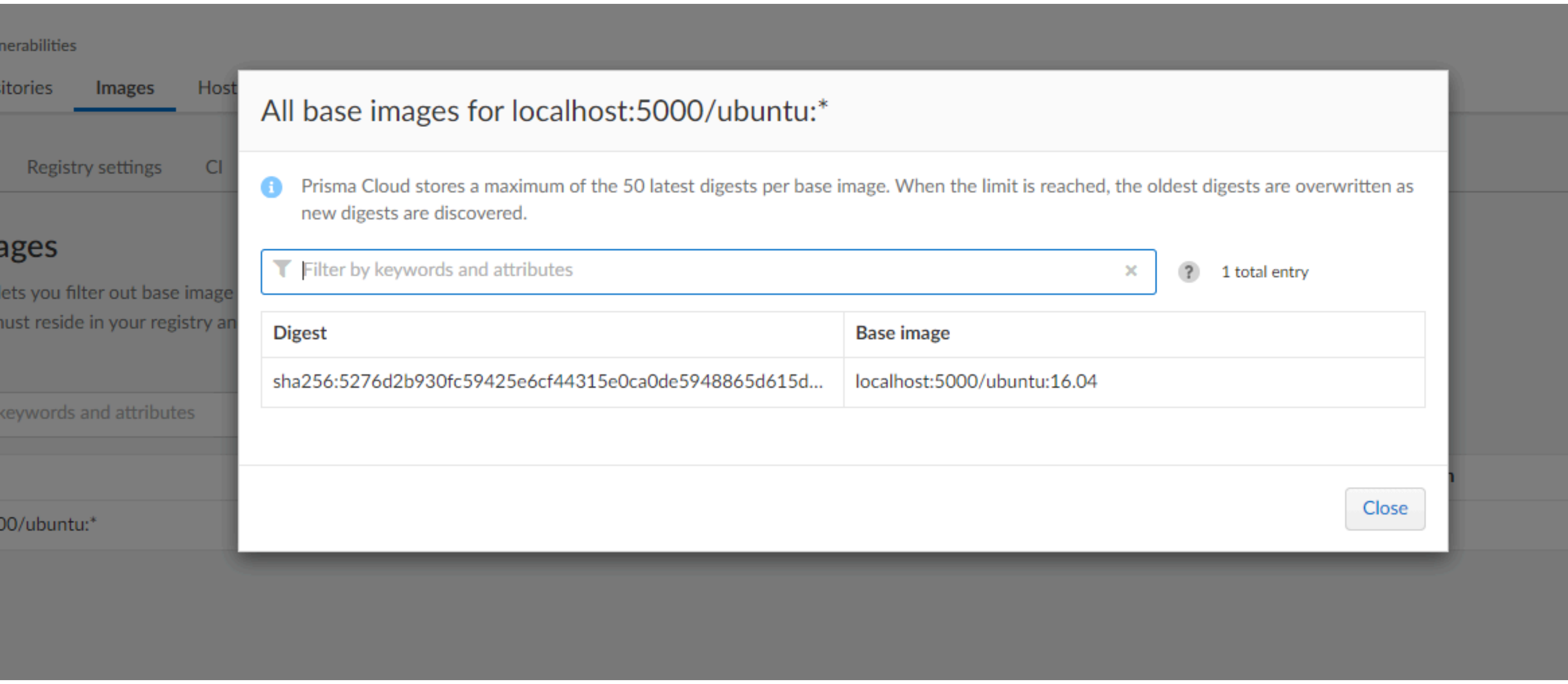
STEP 5 | Click **Save**.



STEP 6 | You can view the specific base image digests of the base image you created using the **View images** action.

These are the digests found in the registry scanning that are matching the base image you defined.

Prisma Cloud stores a maximum of the 50 latest digests per base image. When the limit is reached, the oldest digests are overwritten as new digests are discovered.

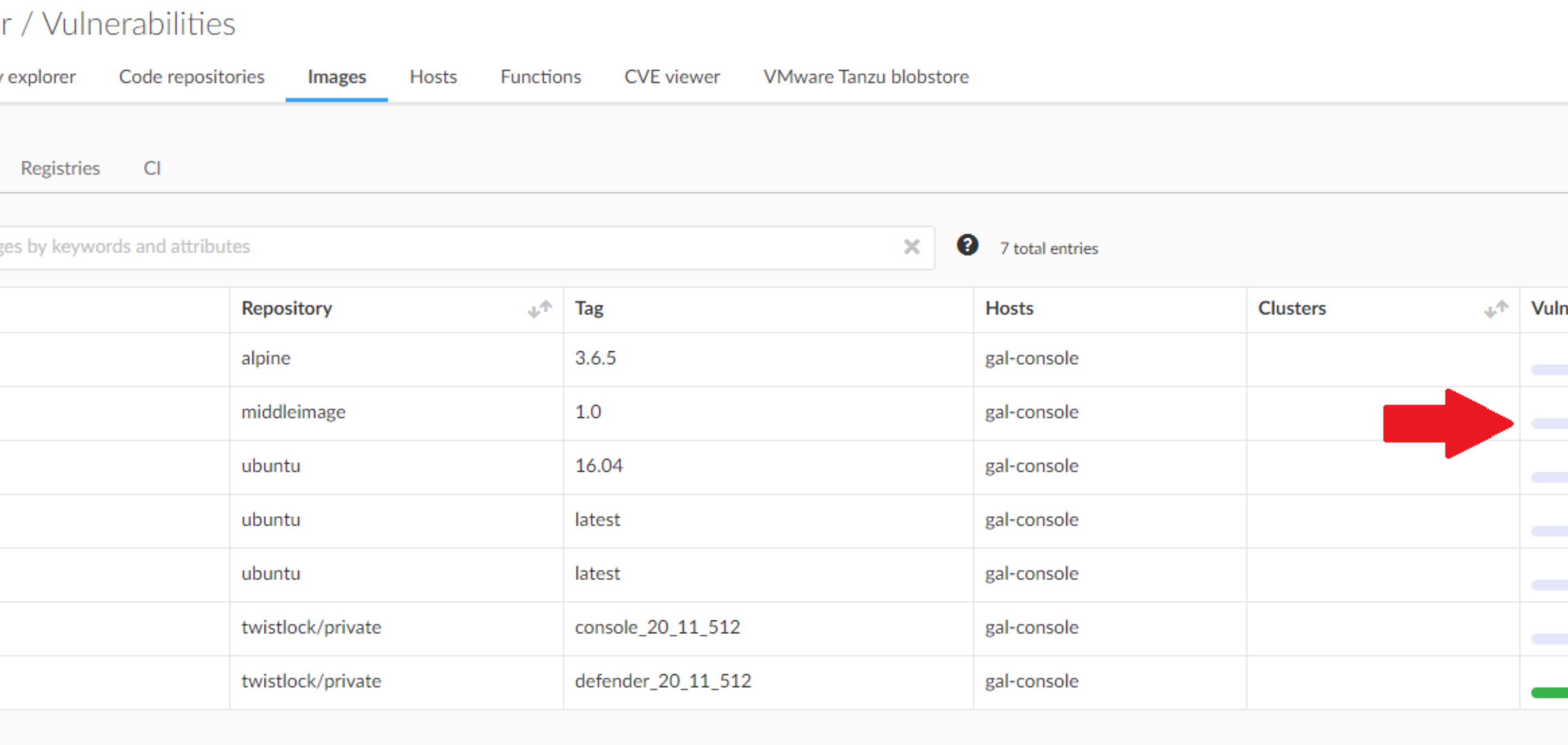


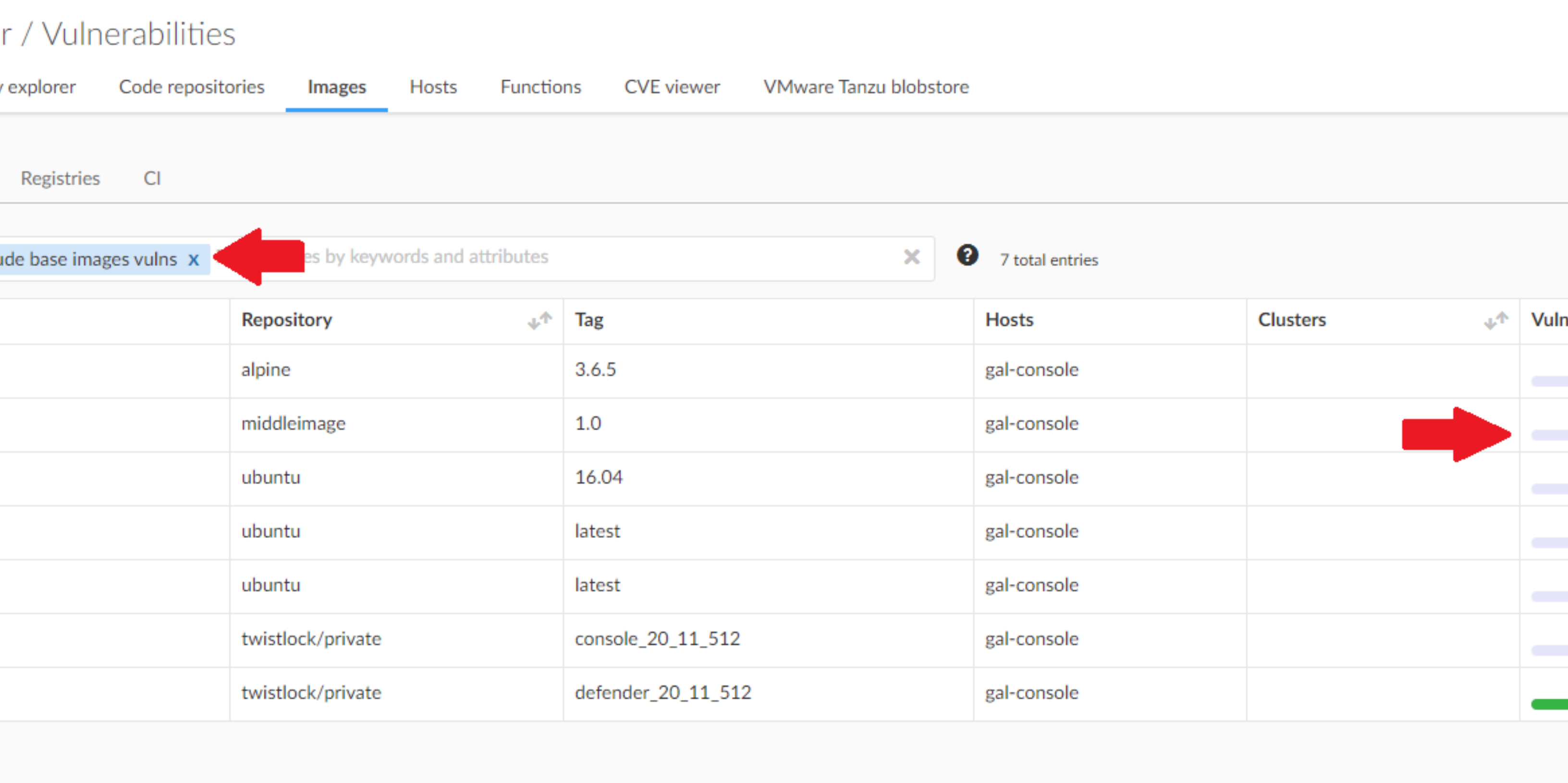
Exclude base images vulnerabilities

When reviewing the health of the images in your environment, whether they are deployed images, registry images, or images scanned in a CI process, you can exclude the base image's vulnerabilities from the scan results.

STEP 1 | Open the Console, then go to **Monitor > Vulnerabilities > Images > Deployed images / Registries / CI**.

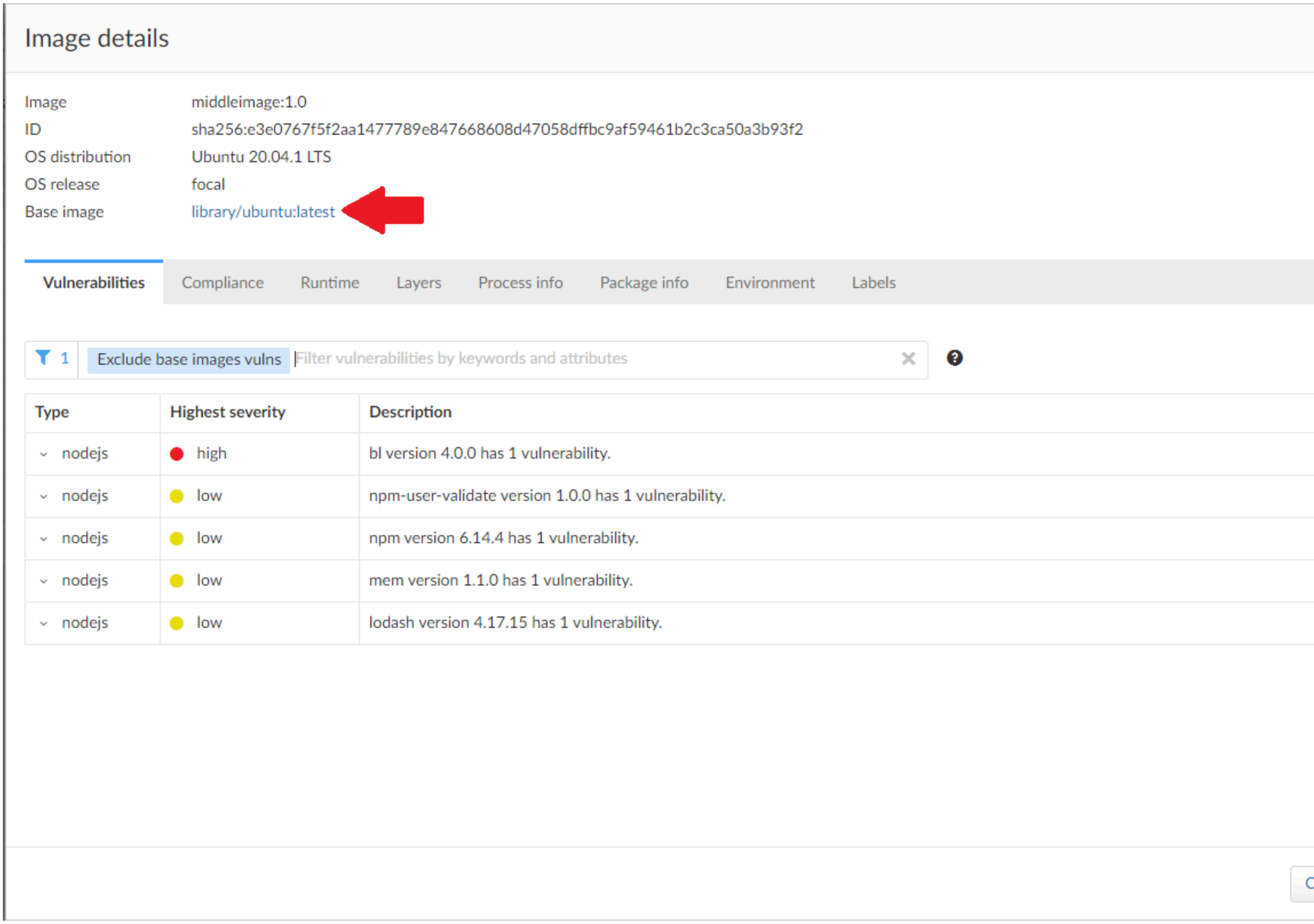
STEP 2 | Use the **Exclude base images vulns** filter to exclude the vulnerabilities coming from base images. You will see the vulnerabilities counters changing.



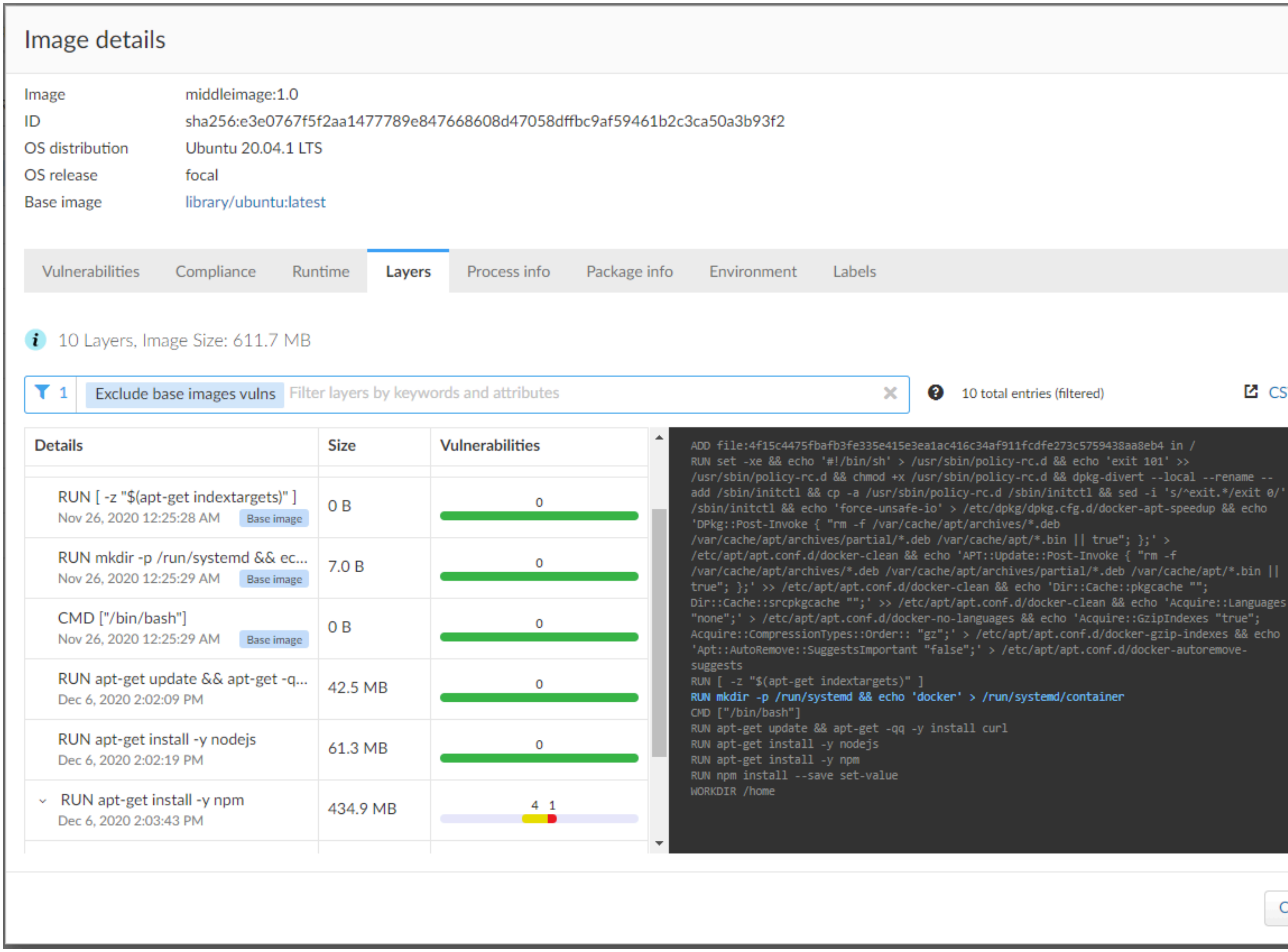


STEP 3 | Click on an image report to open a detailed report.

STEP 4 | Review the filtered vulnerabilities. For reviewing the base image, use the link in the top of the page.



STEP 5 | In the **Layers** tab, the vulnerabilities counters will also exclude base image vulnerabilities, and you'll see an indication for the base image's layers.



Configure VM image scanning

Edit on GitHub

Prisma Cloud supports scanning VM images on AWS, Azure and GCP.

On AWS, Prisma Cloud can scan Linux Amazon Machine Images (AMIs). On Azure, Prisma Cloud supports Managed, Gallery and Marketplace images. On GCP, Prisma Cloud supports Public and Custom images (including Premium images).

AWS

The following AMIs aren't supported:

- Images that don't use cloud-init for bootstrapping, such as Red Hat Enterprise Linux CoreOS (CoreOS for OpenShift). RHCOS uses Ignition.
- Images that use paravirtualization.
- Images that only support old TLS protocols (less than TLS 1.1) for utilities such as curl. For example, Ubuntu 12.10.

Prerequisites

- Access from the VPC to the Prisma Cloud Compute Console.

  For the VMs to send scan results back to the Console, the default port used for communication is 8084. If you use a different port for enabling Defender to Console communication, make sure that the port is allowed access. Note that this port is used for communication although Defenders are not used for VM image scanning.

- The service account Prisma Cloud uses to scan AMIs must have at least the following policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "PrismaCloudComputeAMIScanning",
            "Effect": "Allow",
            "Action": [
                "ec2:AuthorizeSecurityGroupEgress",
                "ec2:AuthorizeSecurityGroupIngress",
                "ec2:CreateSecurityGroup",
                "ec2:CreateTags",
                "ec2:DeleteSecurityGroup",
                "ec2:DescribeImages",
                "ec2:DescribeInstances",
                "ec2:DescribeSecurityGroups",
                "ec2:RevokeSecurityGroupEgress",
                "ec2:RunInstances",
                "ec2:TerminateInstances"
            ],
            "Resource": "*"
        }
    ]
```



```
}
```

- *Prisma Cloud requires the permissions listed above for VM image scanning. To restrict permissions for creating and deleting resources, you can use conditional clauses in AWS IAM policy for the security groups and instances that have the prefix "twistlock-scan".*
- *It is strongly recommended to make sure the images scanned have DeleteOnTermination attribute enabled.*

Azure

Prisma Cloud supports the following image types:

- Marketplace images (publicly available images)
- Managed (custom) images
- Shared image galleries
- Encrypted images

Prisma Cloud doesn’t support the following image types:

- Azure paid images

Prerequisites

- The service account Prisma Cloud uses to scan Azure images must have at least the following policy:

```
Microsoft.Compute/locations/publishers/artifacttypes/offers/skus/
versions/read
Microsoft.Compute/images/read
Microsoft.Compute/galleries/read
Microsoft.Compute/galleries/images/read
Microsoft.Compute/galleries/images/versions/read
Microsoft.Resources/subscriptions/resourceGroups/read
Microsoft.Resources/subscriptions/resourceGroups/write
Microsoft.Resources/subscriptions/resourceGroups/delete
Microsoft.Network/networkSecurityGroups/read
Microsoft.Network/networkSecurityGroups/write
Microsoft.Network/networkSecurityGroups/join/action
Microsoft.Network/networkSecurityGroups/delete
Microsoft.Network/networkInterfaces/read
Microsoft.Network/networkInterfaces/write
Microsoft.Network/networkInterfaces/join/action
Microsoft.Network/networkInterfaces/delete
Microsoft.Compute/disks/write
Microsoft.Compute/disks/delete
Microsoft.Network/virtualNetworks/subnets/read
Microsoft.Network/virtualNetworks/subnets/join/action
Microsoft.Compute/virtualMachines/read
Microsoft.Compute/virtualMachines/write
Microsoft.Compute/virtualMachines/start/action
Microsoft.Compute/virtualMachines/delete
Microsoft.KeyVault/vaults/keys/read
```



```
Microsoft.KeyVault/vaults/keys/wrap/action
Microsoft.KeyVault/vaults/keys/unwrap/action
```

Use Azure's *Key Vault Crypto Service Encryption User* built-in role to scan encrypted images.

If you have managed and gallery images limited to specific regions, Prisma Cloud skips the scan when the region defined in the scope doesn't match region defined for the image.

GCP

Prisma Cloud supports the following image types:

- Public images (including Premium images)
- Custom images
- Encrypted images

Prerequisites

You can only scan encrypted images that use a customer-managed encryption key (CMEK). Customer-supplied encryption keys (CSEK) are not supported.

- The service account Prisma Cloud uses to scan GCP VM images must have at least the following policy:

  ```
  compute.disks.create
  compute.images.get
  compute.images.list
  compute.images.useReadOnly
  compute.instances.create
  compute.instances.delete
  compute.instances.get
  compute.instances.list
  compute.instances.setMetadata
  compute.instances.setTags
  compute.networks.updatePolicy
  compute.networks.use
  compute.networks.useExternalIp
  compute.subnetworks.use
  compute.subnetworks.useExternalIp
  ```

- Verify that the Compute Engine Service Agent service account in the target image project has the *Cloud KMS CryptoKey Decrypter* role or equivalent.
- If you use a shared VPC, verify that the service account in the target image project has the *compute.subnetworks.use* permission in the project containing the subnetwork. For a shared VPC, the project containing the shared VPC is the host project.

  This built-in service account ends with *compute-system.iam.gserviceaccount.com*. The service agent has these permissions by default since it used these permissions to encrypt the images.

Deployment

VM image scanning is handled by the Console and it does not require Defenders. The Prisma Cloud Console scans a VM image by creating a VM instance which is running the VM image to be scanned.

The VM instances created for scanning VM Images come with default tags as: Key - Name, Value - prismacloud-scan-*

When you configure Prisma Cloud to scan VM images, you can define the number of scanners to use. Defining more than one scanner means that the Console will create a number of VM instances to scan multiple VM images simultaneously. For scanning large numbers of VM images, increase the number of scanners to improve throughput and reduce scan time.

If you remove a VM image, or it becomes unavailable, Prisma Cloud maintains the scan results for 30 days. After 30 days, the scan results are purged.

VM images scan settings

STEP 1 | Open Console.

STEP 2 | Go to **Defend > Vulnerabilities/Compliance > Hosts > VM Images**.

STEP 3 | Click **Add Scope**.

Each scope has the following parameters:

Field	Description
Provider	Specify the cloud provider. The current supported providers are AWS, Azure, and GCP.
Credential	Specify the credential required to access the VM images. If the credential has already been created in the Prisma Cloud credential store, select it. If not, click **Add New**. *For Azure: if you create a credential in the credentials store (**Manage > Authentication > Credentials store**), your service principal authenticates with a password. To authenticate with a certificate,* create a cloud account.
Project ID (only GCP)	If unspecified, the project ID where the service account was created is used.
Image type (only Azure)	Specify the relevant image type. Prisma Cloud supports three image types: Managed, Gallery and Marketplace.
Images	Specify the the VM images to scan. *When the image field contains a string and a wildcard (e.g. Amazo*), only private AMIs are scanned. When using explicit image names, AWS Marketplace and community AMIs are scanned as well.* *As of the Joule release, only one wildcard can be used at this time.*

Field	Description
	Only the AMI names are permitted in the image field. AMI IDs are not supported. Use the label field in the referenced collection to restrain the scan by AWS tag. Use the key-value pattern 'key:value'. All supported resource fields support pattern matching.
Excluded VM images	Specify VM images to exclude from the scan. This field supports pattern matching.
Region	Specify the region to scan.
Console address	Specify the Console URL for the scanner VM instance to use.
API communication port	If your Console listens on a port other than the default port, specify the port number. By default, Console listens on port 8083.
Zone (only GCP)	Specify the Zone where scan instances will be deployed.
Number of scanners	Number of VM images to concurrently scan. Increase the number of scanners to increase throughput and reduce scan time.
Cap	Specify the maximum number of VM images to scan, sorted according to the 'Creation Date'. The most recently created VM images are scanned first, followed by the image next most recently created image, and so on. In the case of Azure Marketplace and Managed images, the images are scanned according to their resource ID, in descending lexicographic order (i.e., ID3, then ID2, then ID1). To scan all VM images, set CAP to 0.
VPC Name (only GCP)	If you want a custom VPC for the scanner VM instance, specify the VPC name.
VPC ID (only AWS)	If you want a custom VPC for the scanner VM instance, specify the VPC id to use (e.g., vpc-xxxxx).
Subnet Name (only GCP)	If you want a custom subnet for the scanner VM instance, specify the subnet name.

Field	Description
Subnet ID (only AWS)	If you want a custom subnet for the scanner VM instance, specify the subnet id to use (e.g., subnet-xxxxx).
Subnet Resource ID (only Azure)	Specify the Resource ID of the subnet where scan instances should be deployed.
Instance Type	The default size is m4.large, if you want a custom instance size for the scanner VM instance, specify the desired instance type. Recommend not to choose nano types, as they can increase the scan time.

VPC and subnet scope mapping are 1:1. You can only scope one VPC and subnet per unique rule created.

VM images rules

To define which VM images to scan, create a new VM images scan rule.

STEP 1 | Open Console.

STEP 2 | Go to **Defend > Vulnerabilities/Compliance > Hosts > VM Images**.

STEP 3 | Click **Add Rule**.

STEP 4 | Fill out your policy.

STEP 5 | Click **Save**.

Additional scan settings

Additional scan settings can be found under **Manage > System > Scan**, where you can set the VM images scan interval.

General Notes

- VM image scanning results older than 30 days are automatically deleted.
- On upgrade, VM image scanning results are deleted.
- When a scan is cancelled, it might take a few minutes for the scan to stop completely.

Configure code repository scanning

Edit on GitHub

Prisma Cloud can scan GitHub repositories and identify vulnerabilities in your software's dependencies. Modern apps are increasingly composed of external, open source dependencies, so it's important to give developers tools to assess those components early in the development lifecycle. Repository scanning gives you early insight into the software as it's being developed, and long before apps are packaged (e.g. as a container) and deployed by CI/CD pipelines.

Currently, Prisma Cloud supports Python, Java, and JavaScript (Node.js).

Prerequisites

Prisma Cloud authenticates with the GitHub API using user-generated API tokens. The following scopes are required for scanning private repos. Prisma Cloud doesn't modify or write to your repos.

- repo — Full control of private repositories
  - repo:status — Access commit status
  - repo_deployment — Access deployment status
  - public_repo — Access public repositories
  - repo:invite — Access repository invitations
  - security_events — Read and write security events

If you're scanning public repos only, select just the public_repo scope. The benefit of creating an access token for scanning public repos is that GitHub grants you a higher rate limit to their API, which Prisma Cloud utilizes for scanning.

Deployment

Prisma Cloud selects the repositories to scan according to a user-defined *scope*. For example, you might want to scan all repositories in your organization or just a subset of them. For each repo in scope, Prisma Cloud searches for well-known package manifest files, and enumerates the dependencies listed in them. Those dependencies are assessed against the latest threat data in the Intelligence Stream.

Code repository scans is handled by Console.

The following table lists the manifest files known to the scanner.

Package manager	File name
Go	go.sum
Java (Gradle)	build.gradle, build.gradle.kts, gradle.properties
Java (Maven)	pom.xml

Package manager	File name
JavaScript (NPM)	package.json, package-lock.json, npm-shrinkwrap.json, bower.json
Python (pip)	req*.txt

Finally, Prisma Cloud can continuously monitor your code repositories for vulnerabilities by rescanning on every push event. Prisma Cloud integrates with GitHub using webhooks, which notify the scanner when there are changes in the repository.

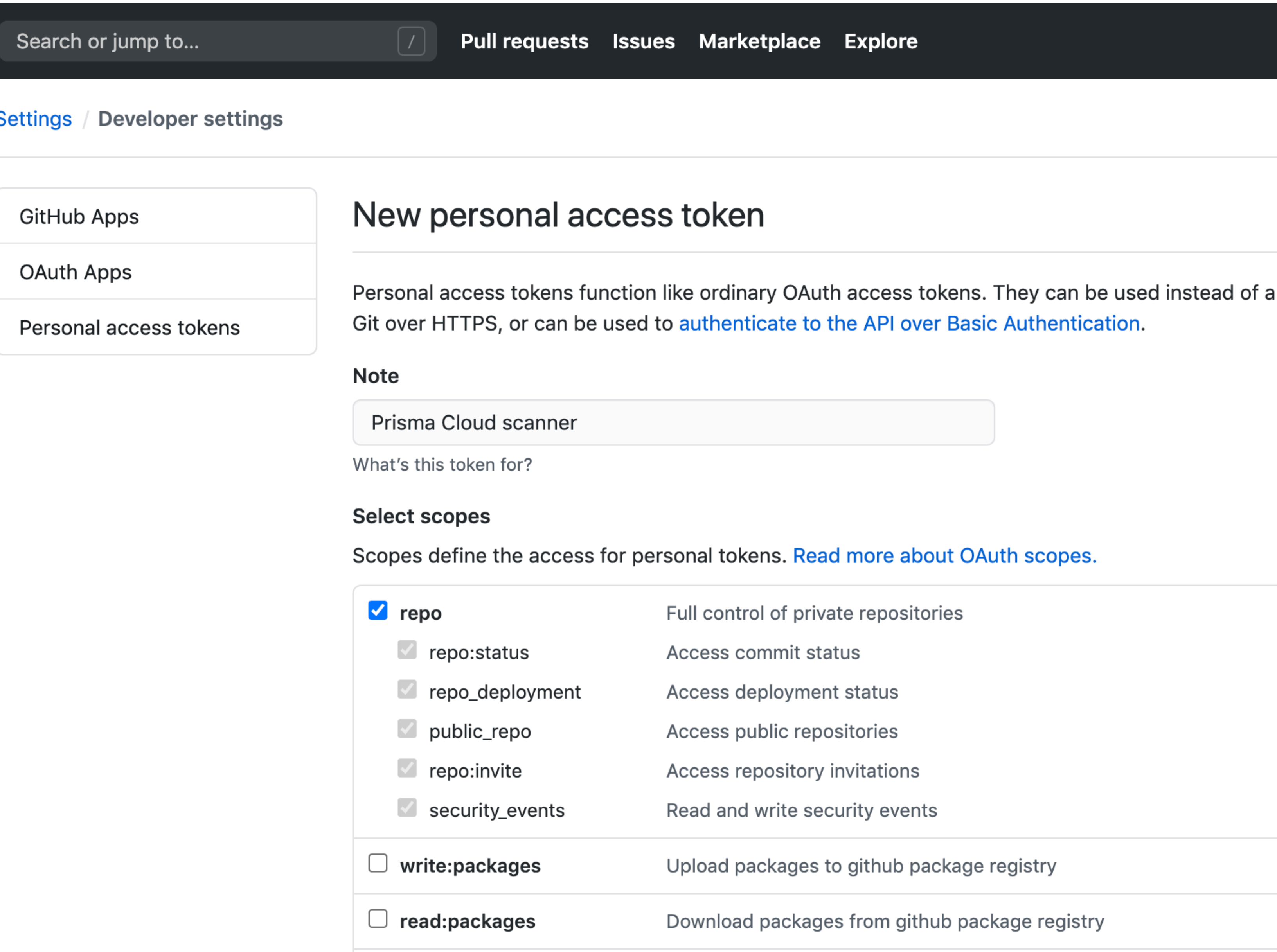
Prisma Cloud uses the GitHub API. The GitHub API is rate-limited. For unauthenticated requests, which can be used to scan public repositories, the cap is very low (60 requests/hour). Here the rate limit is gauged by IP address. For authenticated requests, which can scan either public or private repositories, the cap is 5000 requests/hour. Here the rate limit is gauged per account.

Set up your credentials

Generate a personal access token in GitHub, and then save it in the Prisma Cloud Credentials Store so that the scanner can access your repositories for scanning.

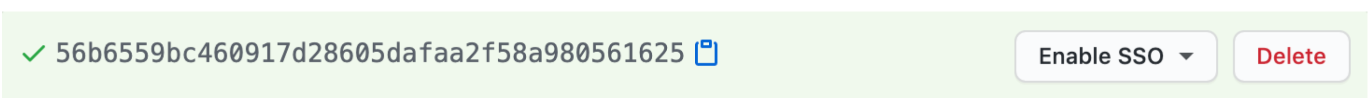
STEP 1 | Generate a GitHub access token.

1. Log into your GitHub account.
2. Go to **Settings > Developer Settings > Personal access tokens**.
3. Click **Generate new token**.
4. Set the scope to **repo**.



If you're scanning public repos only, select just the **public_repo** scope.

5. Click **Generate token**. If your account requires SSO, enable it.
6. Copy the generated token.



STEP 2 | Save the token in Prisma Cloud's credentials store.

1. Log into Prisma Cloud Console.
2. Go to **Manage > Authentication > Credentials Store**.
3. Click **Add Credential**.
4. Enter a **Name** for the credential.
5. In **Type**, select 'GitHub Cloud' or 'GitHub Enterprise Server' access token.

   For GitHub Enterprise Server, specify the Server URL. If you use a self-signed certificate, specify it, or choose 'Skip Verify' to skip certificate validation.
6. In **Access Token**, paste the access token you generated in GitHub.
7. Click **Save**.

Configure the repos to scan

Specify the repositories to scan. If your repository specifies dependencies in non-standard package manifest files, specify them here so the scanner can parse them. If there are manifests the scanner should ignore, specify them here as well.

STEP 1 | Open Console.

STEP 2 | Go to **Defend > Vulnerabilities > Code Repositories**.

STEP 3 | Click **Add Scope**. If this is your first repository, click **Add the first item**.

Each scope spec has the following parameters:

Field	Description
Provider	Select the appropriate GitHub deployment. GitHub Cloud and GitHub Enterprise are currently the only supported providers. *For other Git repositories, use twistcli's coderepo scan option*
Type	To scan all repos in an organization, including both public and private repos, set the type to **Private**. You'll need to set up an access token, so that Prisma Cloud can access your repos. To scan public repositories not related to your account or organization, set the type to **Public**. When type is **Public**, credentials are not required, although API access to GitHub is capped to a very low value. Even if you're only scanning public repos, we recommend that you set up an access token for authenticated access.
Credential	Specify credentials for the repository owner. If the credentials have already been created in the Prisma Cloud credentials store, select it. If not, click **Add New**.

Field	Description
Repositories	Specify the repositories to scan in the format: owner/name When you've selected a credential, the drop-down lists all repositories in the owner's account. Wildcards are supported when the repo type is **Private**. They aren't supported when the type is **Public**.
Excluded manifest paths	Specify paths to be excluded for analysis. Wildcards are supported.
Advanced settings > Explicit manifest names	Supported for Python only. Specify any additional file names that should be included for analysis. If you have a custom naming scheme for your manifest files, specify them here so that the scanner can find and parse them.
Advanced settings > Python version	For a more accurate analysis of your app's dependencies, specify the version of Python you deploy in production. Otherwise, the scanner assumes the latest available version of Python.

STEP 4 | Click **Add**.

STEP 5 | Click **Save**.

Scan repos on push events

Configure GitHub webhooks to rescan your repositories on push events.

STEP 1 | Open Console.

STEP 2 | Go to **Defend > Vulnerabilities > Code Repositories**.

STEP 3 | In **Webhook settings**, select the publicly accessible name or IP address GitHub will use to notify Prisma Cloud that a push event occurred.

STEP 4 | Copy the URL.

STEP 5 | Configure GitHub.

1. Log into GitHub, select a repo, and go to **Settings > Webhooks**.
2. Click **Add webhook**.
3. In **Payload URL**, paste the URL you copied from Prisma Cloud Console.
4. In **Content type**, select **application/json**.
5. Select **Disable SSL verification**.

   For Compute Edition, you can enable SSL verification if your Console runs under a domain with a valid certificate signed by a known authority.

   For Prisma Cloud Enterprise Edition, select **Enable SSL verification**.
6. Leave all other settings in their default state.
7. Click **Add webhook**.
8. Verify that the ping webhook was delivered successfully.

Policy

Prisma Cloud ships with a default rule that alerts on vulnerabilities. In **Defend > Vulnerabilities > Code Repositories**, create vulnerability rules to tailor what's reported.

Additional scan settings can be found under **Manage > System > Scan**, where you can set the scan interval. By default, it's 24 hours.

Agentless scanning

Edit on GitHub

Agentless scanning lets you inspect the risks and vulnerabilities of a virtual machine without having to install an agent or affecting the execution of the instance. Prisma Cloud gives you the flexibility to choose between agentless and agent-based security using Defenders. Currently, Prisma Cloud supports agentless scanning on AWS, GCP and Azure hosts for vulnerabilities and compliance. Agentless capabilities will continue to be enhanced over future releases in parallel with Defender capabilities.

See scanning modes to review the scanning options and to configure agentless scanning.

Vulnerability Scan

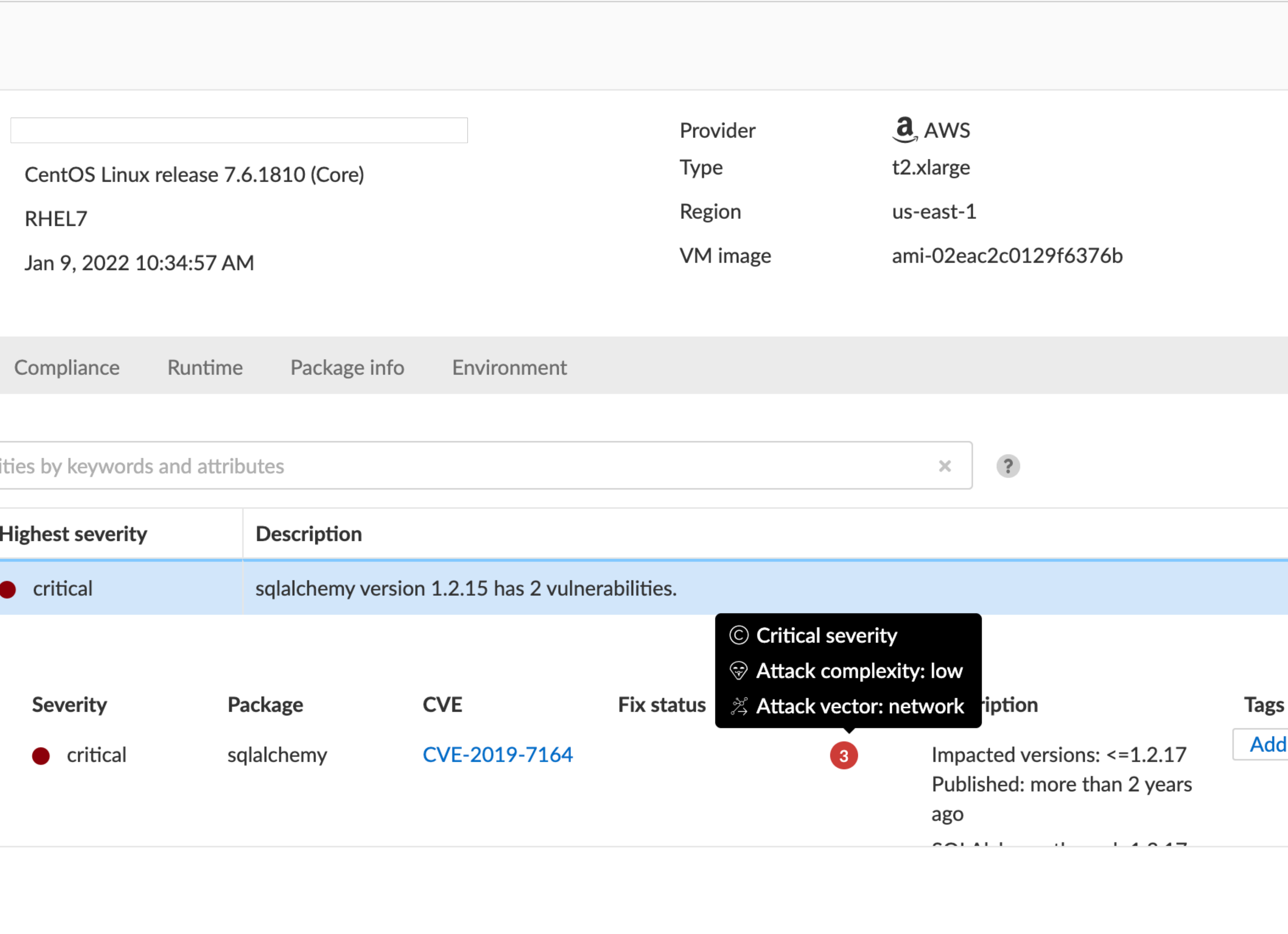
Agentless scan results are cohesively integrated with Defender results throughout the Console to provide seamless experience.

Vulnerability scan rules control the data surfaced in Prisma Cloud Console, including scan reports and Radar visualizations. To modify these rules, see vulnerability scan rules.

View Scan Results

Navigate to **Monitor > Vulnerabilities > Hosts** to view agentless vulnerability scan results. You can see a column named **Scanned by** in the results page. On the rows where entry is **Agentless**, scan results are provided by agentless scanning.

Agentless scans provide risk factors associated with each vulnerability such as package in use, exposed to internet, etc. (here). You can add tags and create policies in alert mode for exceptions. Agentless scanning is integrated with Vulnerability Explorer and Host Radar.

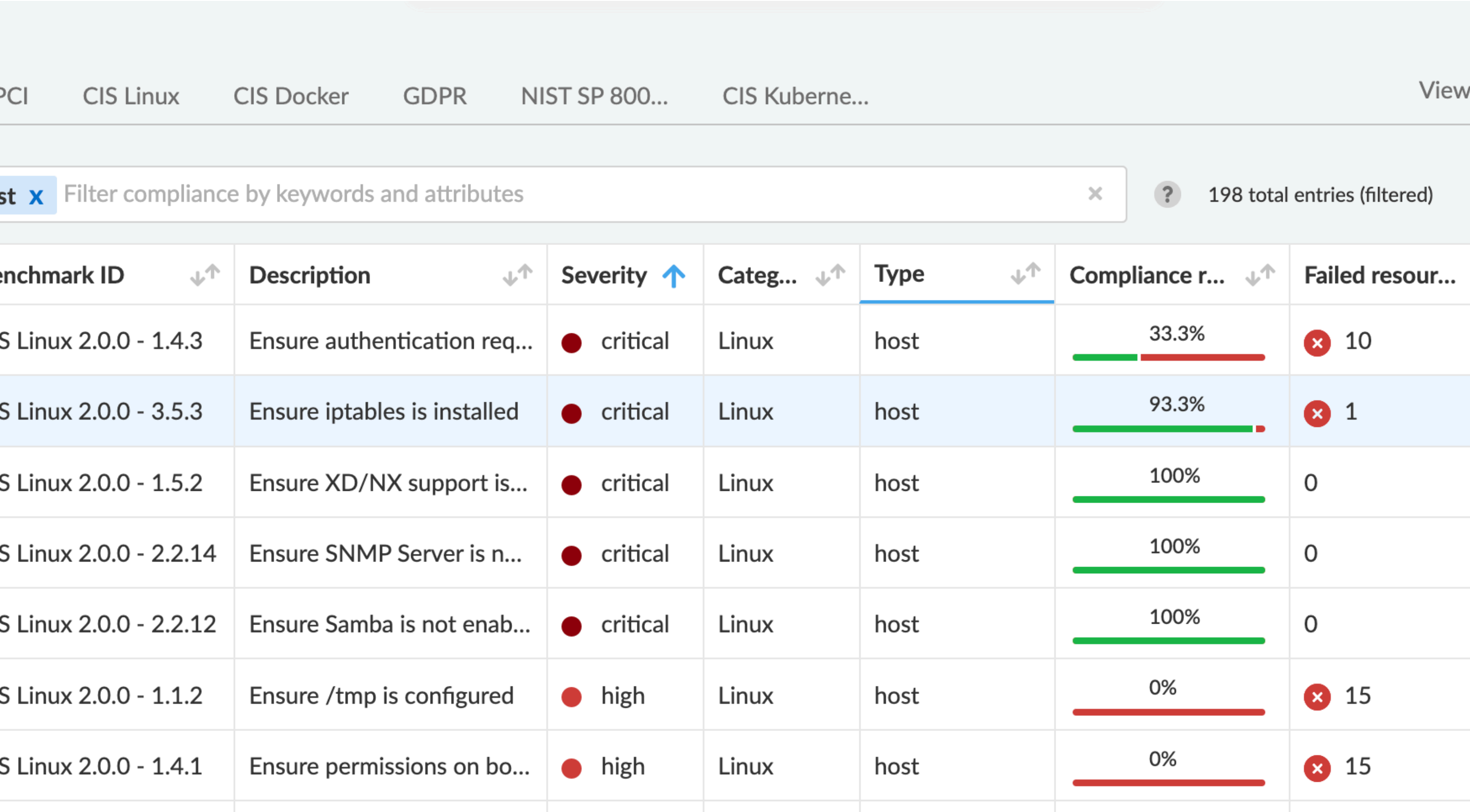


Agentless scans also provide:

- Host Compliance and custom compliance scans
- Unscanned region detection with Cloud Discovery integration
- Pending OS security updates

Compliance Scans

Navigate to **Monitor > Compliance > Hosts** to view agentless compliance scan results. You can see a column named **Scanned by** in the results page. On the rows where entry is **Agentless**, scan results are provided by agentless scanning.



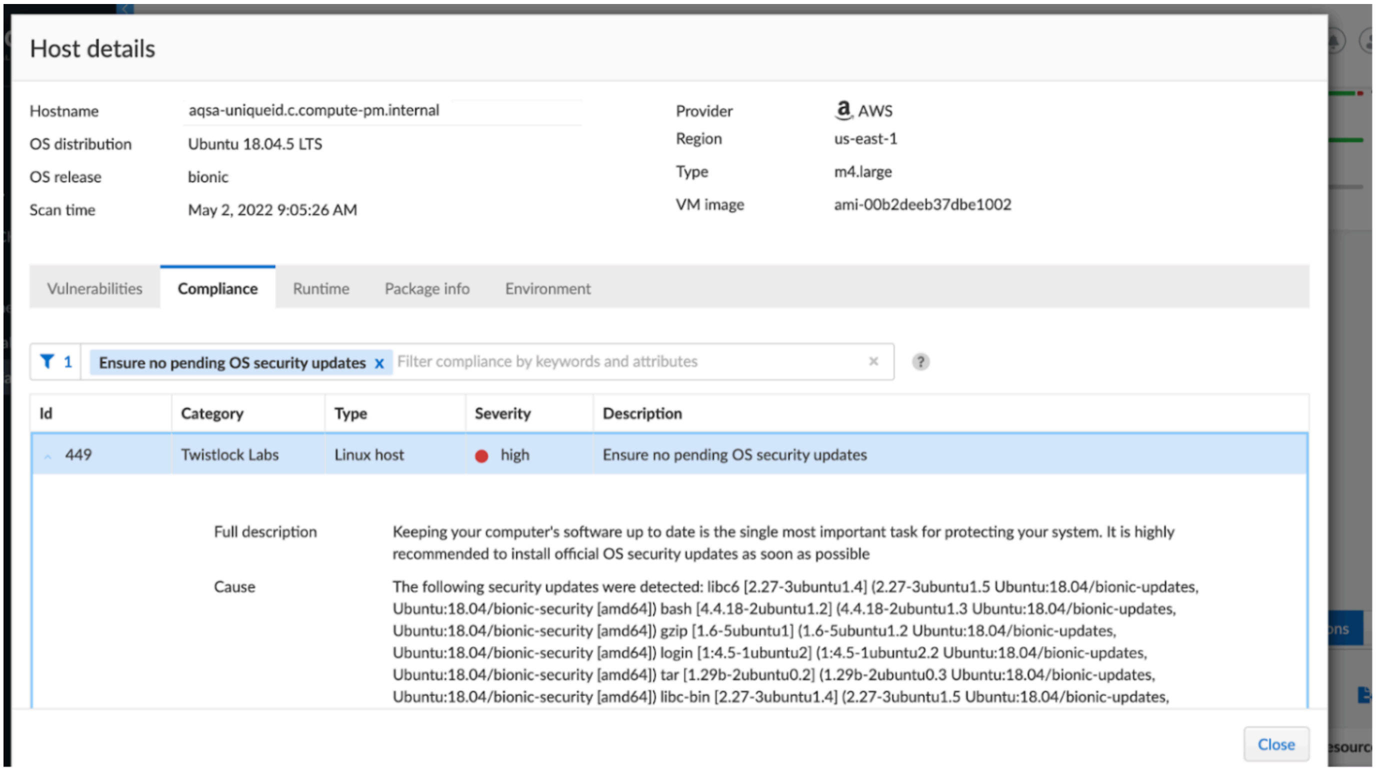
Agentless scans provide risk factors associated with each compliance issue and overall compliance rate for host benchmarks. (learn more here). You can add tags and create policies in alert mode for exceptions. Agentless scanning is integrated with Compliance Explorer and Host Radar.

Custom Compliance Scans

You can create custom compliance checks on file systems for your host and add them to your compliance policy for scanning. Follow the instructions to enable custom compliance checks in a single step for both Defenders and Agentless scans.

Pending OS Updates

Unpatched OSes lead to security risks and greater possibility of exploits. Through agentless scanning, find pending OS security updates as a compliance check.

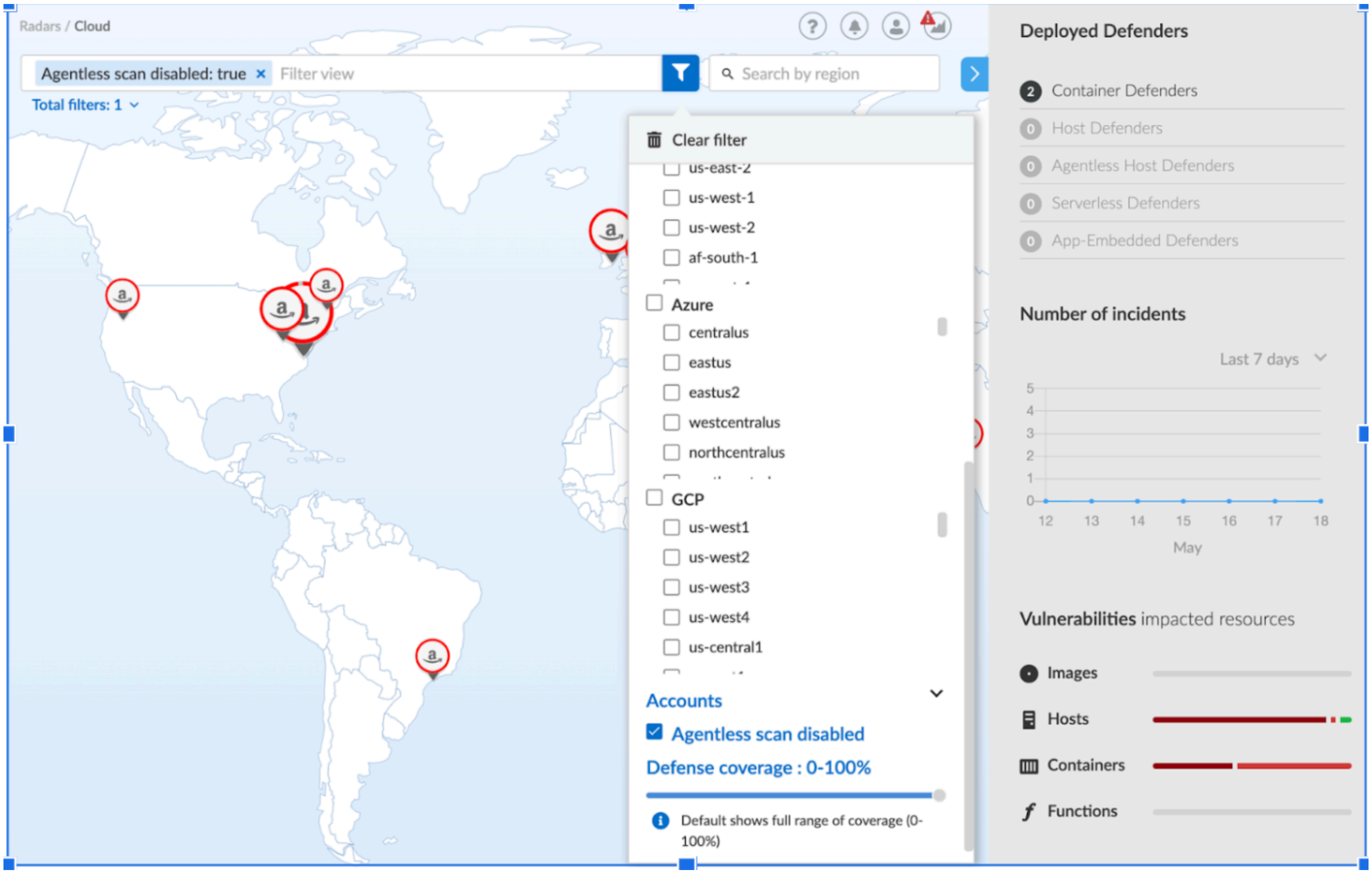


You can search for all hosts with pending OS updates by searching for "Ensure no pending OS updates" string in Compliance explorer page (Monitor > Compliance > Compliance eExplorer tab).

Syntax: <package name> [<current version>] (<new version available> ...)

Cloud Discovery

When cloud discovery is enabled, agentless scans are automatically integrated with the results to provide visibility into all regions and cloud accounts where agentless scanning is not enabled along with undefended hosts.


Radars / Cloud
Agentless scan disabled: true ×
Filter view
Search by region
Total filters: 1
Clear filter
us-east-2
us-west-1
us-west-2
af-south-1
Azure
centralus
eastus
eastus2
westcentralus
northcentralus
GCP
us-west1
us-west2
us-west3
us-west4
us-central1
Accounts
Agentless scan disabled
Defense coverage : 0-100%
Default shows full range of coverage (0-100%)
Deployed Defenders
2 Container Defenders
0 Host Defenders
0 Agentless Host Defenders
0 Serverless Defenders
0 App-Embedded Defenders
Number of incidents
Last 7 days
May
Vulnerabilities impacted resources
Images
Hosts
Containers
Functions

Malware scanning

Edit on GitHub

Besides detecting software vulnerabilities (CVEs) and compliance issues (such as images configured to run as root), Prisma Cloud also detects malware in your container images. No special configuration is required to enable this feature.

To perform malware analysis, Prisma Cloud uses the inputs from:

- Intelligence Stream—Searches the feed for a match against the hash for the executable.
- Wildfire service—Queries the WildFire service for a match against the hash for the executable. If there is no match and upload is enabled, the file is uploaded for analysis.

  WildFire is only supported for image scanning when used in CI.

- Custom feeds—Searches for a match for the executable hash against any custom malware data you import for image scanning.

Malware scanning and detection is supported for Linux container images only. Windows containers are not supported.

Detecting malware

The image scanner looks for malware in binaries in the image layers, including the base layer. When Prisma Cloud detects malware in an image, it includes the malware information as a compliance violation in the image scan report.

To review the results of an image scan:

STEP 1 | Open Console, then go to **Monitor > Vulnerabilities > Images**.

STEP 2 | Click on an image to get a detailed report from the last image scan.

STEP 3 | In the detailed report, click on the **Compliance** tab.

Issues with vulnerability ID 422 means that your image contains a file with an md5 signature of known malware.

Vulnerability risk tree

Edit on GitHub

Because Prisma Cloud knows the state of all the images in your environment, it can show you all the places you might be at risk to a given set of vulnerabilities. To generate a risk tree, provide a CVE, and Prisma Cloud returns:

- A list of images that contain packages affected by the specified CVE.
- A list of running containers (created from the images listed above) that are affected by the specified CVE.
- A list of namespaces where the containers affected by the specified CVE reside.
- A list of hosts where the images affected by the specified CVE reside.
- A list of serverless functions that are affected by the specified CVE.

The risk tree lets you create a detailed map of your exposure to a vulnerability, and can help you identify the best way to resolve it in your upstream images.

Generating a risk tree

Prisma Cloud's Vulnerability Explorer shows you risk trees for the top ten vulnerabilities in your container ecosystem. To see the risk tree for any arbitrary CVE, use the search tool at the top of the "Top Ten lists" table or Prisma Cloud API.

To generate a risk tree, submit a CVE to the API. The API returns an ordered tree of the images that contain those vulnerabilities, containers that are derived from those images, namespaces where these containers reside, and hosts where those images live. This allows you to automate, with a single API call, the creation of a detailed map of your exposure to the vulnerabilities.

To generate a risk tree, use the following endpoint:

```
GET /api/v1/stats/vulnerabilities/impacted-resources?cve=<CVE-ID>
```

For example, to generate a risk tree for CVE-2016-2109:

```
GET /api/v1/stats/vulnerabilities/impacted-resources?
cve=CVE-2016-2109
```

The following listing shows an example response. For complete details about the response object, see the API reference.

```
{
  "_id": "CVE-2017-6983",
  "riskTree": {

 "sha256:154de23b60b2a0651401012afff4c2da485f8076043e7241288bbeb88c9965fa":
 [
      {
        "image": "docker.io\/library\/ubuntu-exploit:latest",
        "container": "",
        "host": "",
        "namespace": "",
```



```
        "factors": {
          "network": false,
          "internet": false,
          "rootPrivilege": false,
          "noSecurityProfile": false,
          "privilegedContainer": false
        }
      }
    ],

 "sha256:45919d98e870eeb1b1d4ccb9458f992f372d9fadc5f3efd034bc569a575f0ff9":
 [
      {
        "image": "docker.io\/morello\/docker-whale:latest",
        "container": "confident_archimedes",
        "host": "ian-23.c.cto-sandbox.internal",
        "namespace": "",
        "factors": {
          "network": false,
          "internet": false,
          "rootPrivilege": true,
          "noSecurityProfile": false,
          "privilegedContainer": false
        }
      }
    ],

 "sha256:67759a80360cbaef77ec1eee8aa0590f07ba04c26ef496efbc90391f217fd9d6":
 [
      {
        "image": "docker.io\/library\/ubuntu:14.04",
        "container": "",
        "host": "",
        "namespace": "",
        "factors": {
          "network": false,
          "internet": false,
          "rootPrivilege": false,
          "noSecurityProfile": false,
          "privilegedContainer": false
        }
      }
    ]
  },
  "registryImages": {

  },
  "hosts": [
    "ian-23.c.cto-sandbox.internal"
  ],
  "functions": {
  }
}
```


Vulnerabilities Detection

Edit on GitHub

Supported packages and languages

Scan reports have a Package info tab, which lists all the packages installed in an image or host. The following list shows the package types that are currently supported and can be seen in scan results, by the name they are shown in the scan.

- **Package** - supported Operating Systems packages, such as an RPM (Red Hat and derived distributions), dpkg/deb (Debian and derived distributions), or apk (Alpine Linux).
- **Jar** - the Java Archive format, which is a zip file with a standard structure. The war file format, or web app archive, is also supported.
- **Python** - a Python library.
- **Nodejs** - a Node.js library.
- **Gem** - a Ruby gem library.
- **Go** - a GoLang library
- **App** - a binary associated with a well-known application, such as Nginx or PostgreSQL. A full list of supported applications is listed below.

Prisma Cloud uses a variety of approaches for package detection, these are purpose-built differently for images and hosts. For example, the host Defender only scans applications, as well as language-based packages, if the processes are running. The Windows Defender only scans packages that are installed with a package manager, missing Microsoft hotfixes, and .net framework applications.

Unpackaged software

Typically, the software is added to container images and hosts with a package manager, such as *apt*, *yum*, and *npm*. Prisma Cloud has a diverse set of upstream vulnerability data sources covering many different package managers across operating systems, including coverage for *Go*, *Java*, *Node.js*, *Python*, and *Ruby* components. Prisma Cloud typically uses the package manager's metadata to discover installed components and versions, comparing this data to the data in the Intelligence Stream's real-time CVE feed.

Sometimes, you might install software without a package manager. For example, the software might be built from a source and then added to an image with the Dockerfile *ADD* instruction, or your developers might unzip software from a tarball to a location on a host, and utilize the application. In these cases, there is no package manager data associated with the application.

Prisma Cloud uses a variety of analysis techniques to detect metadata about software not installed by package managers. These are purpose-built differently for images and hosts.

This analysis augments existing vulnerability detection and blocking mechanisms, giving you a single view of all vulnerabilities, regardless of how the software is installed (distro's package manager, language runtime package manager, or without a package manager).

Supported apps

The following list shows examples of the apps currently supported. Download IS data and read the *cve.json* file to get the most recent list of packages.

- .NET Core
- ASP.NET Core
- BusyBox
- Consul
- CRI-O
- Docker
- GO
- Istio
- OMI
- Vault
- Websphere Application Server
- Webshpere Open Liberty
- Kubernetes
- OpenShift
- Jenkins
- Envoy
- Hashicorp Vault
- Hashicorp Consul
- WordPress
- Redis
- Nginx
- Mongo
- MySQL
- Httpd
- Java- Oracle, openJDK
- Apache
- Postgres
- Node
- Ruby
- Python
- PHP

Vulnerabilities of type **Application** are carried in the Intelligence Stream's app feed. Go to the CVE statistics section on the **Manage > System > Intelligence** page for more information.

Distro / package	Number of CVEs	Last updated from source
alpine/3.10	3232	Feb 16, 2021 4:37:41 PM
alpine/3.11	3212	Feb 16, 2021 4:37:41 PM
alpine/3.12	3435	Feb 16, 2021 4:37:41 PM
alpine/3.13	3421	Feb 16, 2021 4:37:41 PM
alpine/3.7	3148	Feb 16, 2021 4:37:41 PM
alpine/3.8	3084	Feb 16, 2021 4:37:41 PM
alpine/3.9	3104	Feb 16, 2021 4:37:41 PM
amzn/AL2	4698	Feb 16, 2021 4:37:52 PM
amzn/AMI	11852	Feb 16, 2021 4:37:52 PM
app	1970	Feb 16, 2021 4:37:42 PM
debian/bullseye	17177	Feb 16, 2021 4:36:36 PM

Nothing is required to enable the functionality described in this article. It is enabled by default.

CVSS scoring

Edit on GitHub

Because severity terminology can vary between projects, Prisma Cloud normalizes severity ratings into a common schema.

Prisma Cloud leverages the CVSS 3.0 scoring system. The CVSS framework captures the principal characteristics of a vulnerability and produces a numerical score that reflects the severity of the vulnerability. CVSS scores range from 0.0 to 10.0. The higher the number, the higher the degree of severity.

Mappings

We only normalize vulnerability ratings for the purpose of creating rules. Console's Monitoring section shows vendor terminology, not Prisma Cloud's normalized scores (low, medium, high, critical).

The following table maps popular vendor terminology to Prisma Cloud normalized scores:

Vendor terminology	Prisma Cloud score
Unimportant	Low
Unassigned	Low
Negligible	Low
Not yet assigned	Low
Low	Low
Medium	Medium
Moderate	Medium
High	High
Important	High
Critical	Critical

In the absence of project-specific terminology, Prisma Cloud normalizes using the CVSS base scores defined by NIST. In addition to the numeric CVSS scores, NVD provides severity rankings of Low, Medium, High, and Critical. These qualitative grades are simply derived from the numeric CVSS scores:

CVSS base score	Prisma Cloud severity
0.0 - 3.9	Low
4.0 - 6.9	Medium
7.0 - 8.9	High
9.0 -10.0	Critical

In some cases, the OS vendor's CVSS scoring and severity rating can differ from NVD's rating. This is based on the vendor's analysis of the impact of the CVE specific to their OS and distro, which is the more accurate view of the vulnerability. Prisma Cloud shows the vendor's rating when reporting findings from workloads running the vendor's OS, and falls back to NVD's rating where applicable.

Example: CVE-2021-33574 *has a CVSS 3.0 score of 9.8 and it's graded as 'critical' by NVD. The same CVE is graded as 'low' by* Ubuntu *and 'medium' with different CVSS score by* Redhat*. For workloads running Ubuntu, Prisma Cloud shows Ubuntu's rating, rather than NVD's rating.*

Windows container image scanning

Edit on GitHub

To scan Windows images, the Windows Intelligence Stream must be enabled. You can find the setting under **Manage > System > Intelligence**. By default, the Windows Intelligence Stream is disabled.

There are a number of things to consider when scanning Windows container images:

- Prisma Cloud Console only runs on Linux hosts. Prisma Cloud Defender, which does the actual scanning work, comes in a number of flavors. On Windows, Prisma Cloud supports Container Defender and Host Defender.
- The container OS version must match the host OS version where Defender runs. For example, Defender on Windows Server 1803 can scan nanoserver:1803, but it can't scan nanoserver:1809. Conversely, Defender on Windows Server 1809 can scan nanoserver:1809, but it can't scan nanoserver:1803.
- Prisma Cloud requires a privileged user inside the container to scan it. In more recent versions of Windows (Windows Server, version 1803 or higher, build 17134 or higher), Prisma Cloud uses the ContainerAdministrator account. This account has complete access to the whole file system and all of the resources in the container. In older versions of Windows, specifically Windows Server 2016 (version 1607, build 14393), ContainerAdministrator does not exist, so Prisma Cloud uses the default user.

Serverless function scanning

Edit on GitHub

Prisma Cloud can scan serverless functions for vulnerabilities. Prisma Cloud supports AWS Lambda, Google Cloud Functions, and Azure Functions.

Serverless computing is an execution model in which a cloud provider dynamically manages the allocation of machine resources and schedules the execution of functions provided by users. Serverless architectures delegate the operational responsibilities, along with many security concerns, to the cloud provider. In particular, your app itself is still prone to attack. The vulnerabilities in your code and associated dependencies are the footholds attackers use to compromise an app. Prisma Cloud can show you a function's dependencies, and surface the vulnerabilities in those dependent components.

Capabilities

For serverless, Prisma Cloud can scan Node.js, Python, Java, C#, Ruby, and Go packages. For a list of supported runtimes see system requirements.

Prisma Cloud scans are triggered by the following events:

- When the settings change, including when new functions are added for scanning.
- When you explicitly click the **Scan** button in the **Monitor > Vulnerabilities > Functions > Scanned Functions** page.
- Periodically. By default, Prisma Cloud rescans serverless functions every 24 hours, but you can configure a custom interval in **Manage > System > Scan**.

Scanning a serverless function

Configure Prisma Cloud to periodically scan your serverless functions. Unlike image scanning, all function scanning is handled by Console.

STEP 1 | Open Console.

STEP 2 | Go to **Defend > Vulnerabilities > Functions > Functions**.

STEP 3 | Click on **Add scope**. In the dialog, enter the following settings:

1. (AWS only) Select **Scan only latest versions** to only scan the latest version of each function. Otherwise, the scanning will cover all versions of each function up to the specified **Limit** value.
2. (AWS only) Select **Scan Lambda Layers** to enable scanning function layers as well.
3. (AWS only) Specify which regions to scan in **AWS Scanning scope**. By default, the scope is applied to **Regular regions**. Other options include **China regions** or **Government regions**.
4. Specify a **Limit** for the number of functions to scan.

   Prisma Cloud scans the X most recent functions, where X is the limit value. Set this value to '0' to scan all functions.

   For scanning Google Cloud Functions with GCP organization level credentials, the limit value is for the entire organization. Increase the limit as needed to cover all the projects within your GCP organization.
5. Select the accounts to scan by credential. If you wish to add an account, click on **Add credential**.

   *For Azure: if you create a credential in the credentials store (**Manage > Authentication > Credentials store**), your service principal authenticates with a password. To authenticate with a certificate, create a cloud account.*
6. Click **Add**.

STEP 4 | Click the green save button.

STEP 5 | View the scan report.

Go to **Monitor > Vulnerabilities > Functions > Scanned functions**.

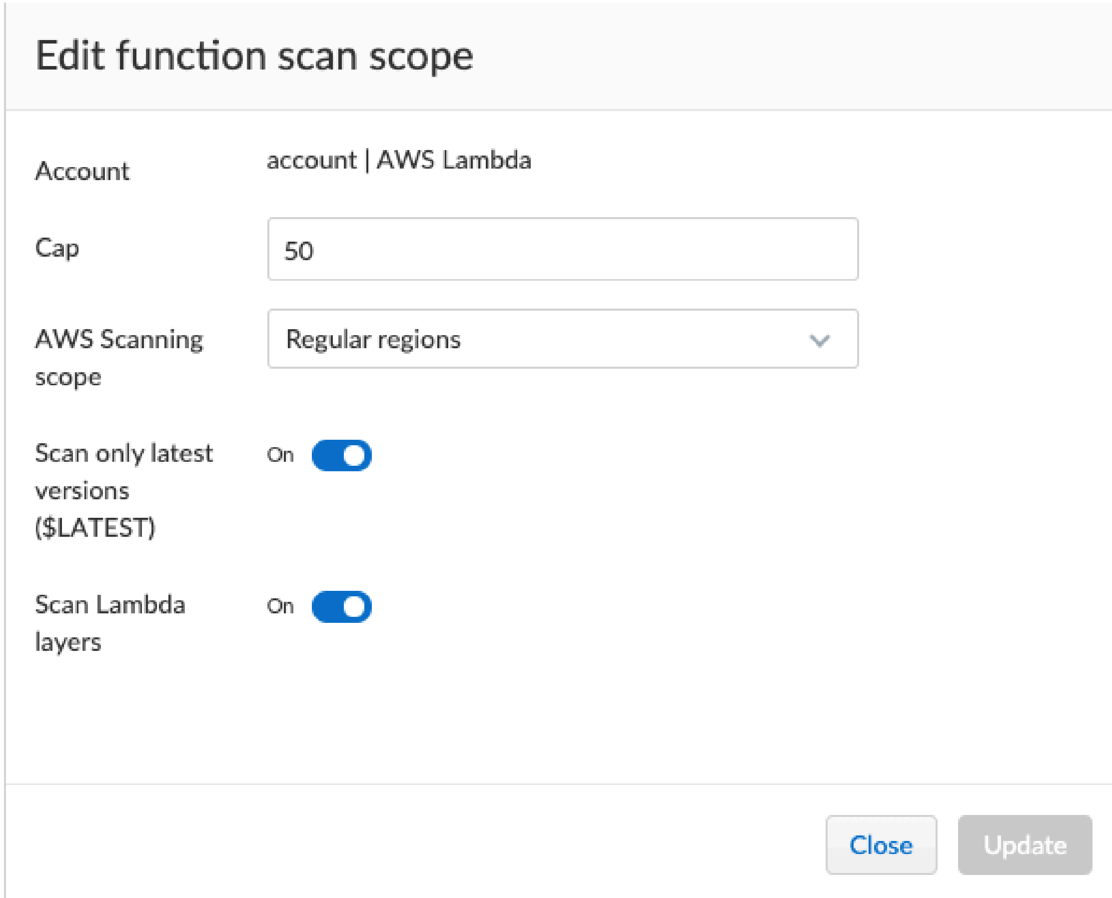
All vulnerabilities identified in the latest serverless scan report can be exported to a CSV file by clicking on the CSV button in the top right of the table.

View AWS Lambda Layers scan report

Prisma Cloud can scan the AWS Lambda Layers code as part of the Lambda function's code scanning. This capability can help you determine whether the vulnerability issues are associated with the function or function Layers. Follow the steps below to view the Lambda Layers scan results:

STEP 1 | Open Console.

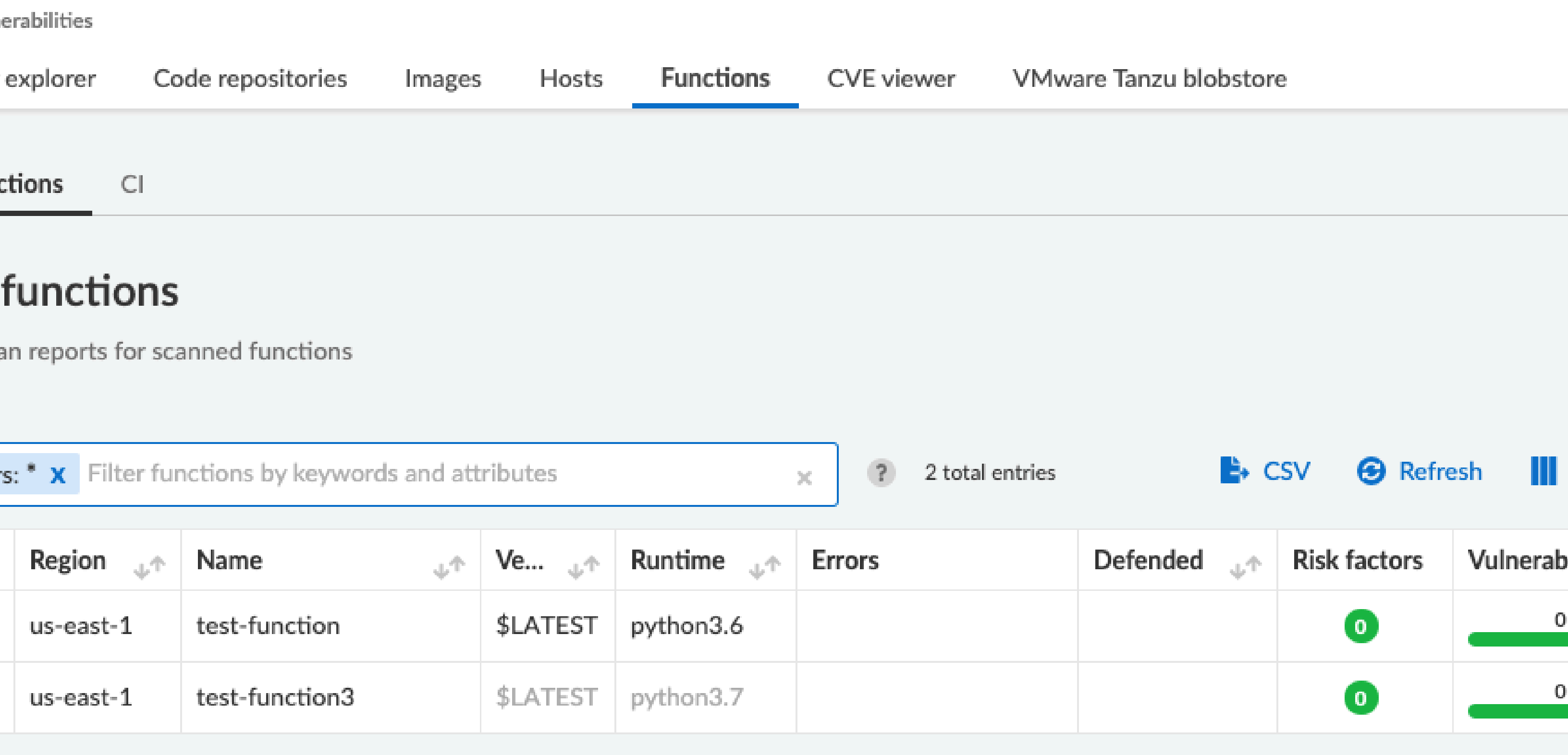
STEP 2 | Make sure you selected the **Scan Lambda layers** in the Defend > Vulnerabilities > Functions > Functions > Serverless Accounts > **Function scan scope**



STEP 3 | Go to **Monitor > Vulnerabilities > Functions > Scanned functions**.

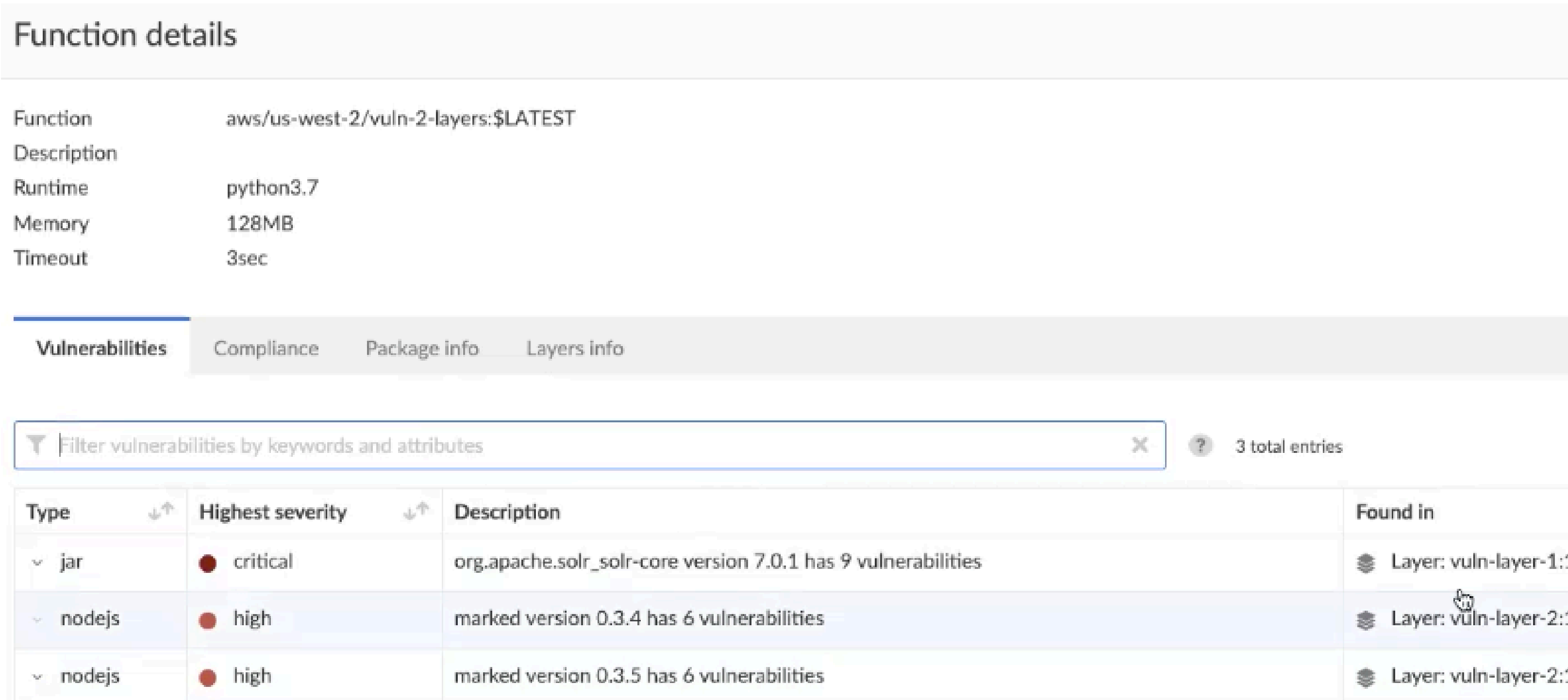
STEP 4 | Filter the table to include functions with the desired Layer by adding the **Layers** filter.

You can also filter the results by a specific layer name or postfix wildcards. Example: *Layers:* OR Layers:arn:aws:lambda:**



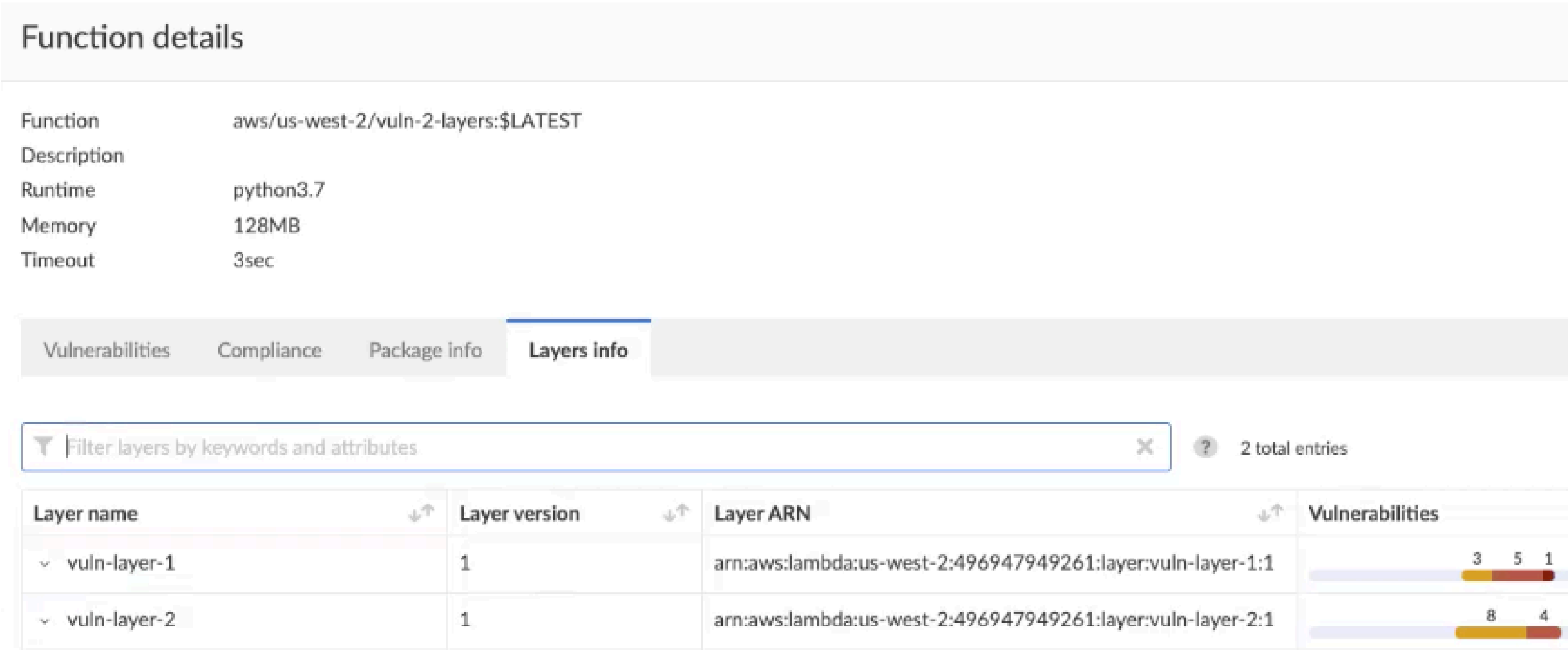
STEP 5 | Open the **Function details** dialog to view the details about the Layers and the vulnerabilities associated with them:

1. Click on a specific function
2. See the Function’s vulnerabilities, compliance issues and package info in the related tabs. Use the **Found in** column to determine if the component is associated with the Function or with the Function’s Layers.



3. Use the **Layers info** tab to see the full list of the Function’s Layers, and aggregated information about the Layers vulnerabilities. In case that there are vulnerabilities

associated with the layer you will be able to expand the layer raw to list all the vulnerabilities.



Authenticating with AWS

The serverless scanner is implemented as part of Console. The scanner requires the following permissions policy:

+

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "PrismaCloudComputeServerlessScan",
            "Effect": "Allow",
            "Action": [
                "lambda:ListFunctions",
                "lambda:GetFunction",
                "iam:GetPolicy",
                "iam:GetPolicyVersion",
                "iam:GetRole",
                "iam:GetRolePolicy",
                "iam:ListAttachedRolePolicies",
                "iam:ListRolePolicies",
                "lambda:GetLayerVersion",
                "kms:Decrypt"
            ],
            "Resource": "*"
        }
    ]
}
```


IAM User

If authenticating with an IAM user, use the Security Token Service (STS) to temporarily issue security credentials to Prisma Cloud to scan your Lambda functions. AWS STS is considered a best practice for IAM users per the AWS Well-Architected Framework. For more on how to use AWS STS, see here.

When authenticating with an IAM user, Console can access and scan functions across multiple regions.

IAM Role

The Prisma Cloud serverless scanner can also authenticate with AWS using an IAM role. If Console authenticates with AWS using an IAM role, it can assume roles using STS to assume roles in other regions.

Scanning Azure Functions

Azure Functions are architected differently than AWS Lambda and Google Cloud Functions. Azure function apps can hold multiple functions. The functions are not segregated from each other. They share the same file system. Rather than separately scanning each function in a function app, download the root directory of the function app, which contains all its functions, and scan them as a bundle.

*Prisma Cloud supports scanning both Windows and Linux functions. For Linux functions, the support is only for functions that use **External package URL** as the deployment technology. For more information, see* Deployment technologies in Azure Functions.

To do this, you must know the Region, Name (of the function), and Service Key. To get the Service Key, download and install the Azure CLI, then:

STEP 1 | Within your Azure portal, create a custom role with the following permissions:

```
{
    "permissions": [
        {
            "actions": [
                "Microsoft.Web/sites/Read",
                "Microsoft.Web/sites/config/list/Action",
                "Microsoft.web/sites/functions/action",
                "Microsoft.web/sites/functions/read",
                "Microsoft.Web/sites/publishxml/Action"
            ],
            "notActions": [],
            "dataActions": [],
            "notDataActions": []
        }
    ]
}
```

STEP 2 | Using the CLI, log into your account with a user that has the User Administrator role.

```
$ az login
```

STEP 3 | Get the service key.

```
$ az ad sp create-for-rbac --sdk-auth --name twistlock-azure-
serverless-scanning --role CUSTOM_ROLE_NAME
```

Sample output from the previous command:

```
{
  "clientId": "f8e9de2o-45bd-af94-ae11-b9r8c5tfy3b6",
  "clientSecret": "4dfds482-6sdd-4dsb-b5ff-56123043c4dc",
  "subscriptionId": "ea19322m-z2bd-501c-dd11-234m547a944e",
  "tenantId": "c189c61a-6c27-41c3-9949-ca5c8cc4a624",
  "activeDirectoryEndpointUrl": "https://
login.microsoftonline.com",
  "resourceManagerEndpointUrl": "https://management.azure.com/",
  "activeDirectoryGraphResourceId": "https://graph.windows.net/",
  "sqlManagementEndpointUrl": "https://
management.core.windows.net:8443/",
  "galleryEndpointUrl": "https://gallery.azure.com/",
  "managementEndpointUrl": "https://management.core.windows.net/"
}
```

STEP 4 | Copy the JSON output, which is your secret key, and paste it into the **Service Key** field for your Azure credentials in Prisma Cloud Console.

Scanning Google Cloud Functions

To scan Google Cloud Functions, you must create an appropriate credential to authenticate with GCP. The service account should include the following custom permissions:

```
cloudfunctions.functions.sourceCodeGet
cloudfunctions.functions.get
cloudfunctions.functions.list
cloudfunctions.locations.get
cloudfunctions.locations.list
cloudfunctions.operations.get
cloudfunctions.operations.list
cloudfunctions.runtimes.list
```

Prisma Cloud currently supports scanning functions that are packaged with local depenendencies.

Scanning functions at build time with twistcli

You can also use the *twistcli* command line utility to scan your serverless functions. First download your serverless function as a ZIP file, then run:

```
$ twistcli serverless scan <SERVERLESS_FUNCTION.ZIP>
```

To view scan reports in Console, go to **Monitor > Vulnerabilities > Functions > CI** or **Monitor > Compliance > Functions > CI**.

Twistcli Options

- ***--address URI*** --

  Required. Complete URI for Console, including the protocol and port. Only the HTTPS protocol is supported. By default, Console listens to HTTPS on port 8083, although your administrator can configure Console to listen on a different port.

  Example: --address https://console.example.com:8083

- ***-u, --user USERNAME*** --

  Username to access Console. If not provided, the *TWISTLOCK_USER* environment variable will be used if defined, or "admin" is used as the default.

- ***-p, --password PASSWORD*** --

  Password for the user specified with *-u*, *--user*. If not specified on the command-line, the *TWISTLOCK_PASSWORD* environment variable will be used if defined, or otherwise will prompt for the user's password before the scan runs.

- ***--project PROJECT NAME*** --

  Interface with a specific supervisor Console to retrieve policy and publish results.

  Example: --project "Tenant Console"

- ***--details*** --

  Show all vulnerability details.

- ***--tlscacert PATH*** --

  Path to Prisma Cloud CA certificate file. If no CA certificate is specified, the connection to Console is insecure.

- ***--include-js-dependencies*** --

  Include javascript package dependencies.

- ***--token TOKEN*** --

  Token to use for Prisma Cloud Console authentication. Tokens can be retrieved from the API endpoint *api/v1/authenticate* or from the **Manage > Authenticate > User Certificates** page in Console.

- ***--cloudformation-template PATH*** --

  Path to the CloudFormation template file in JSON or YAML format. Prisma Cloud scans the function source code for AWS service APIs being used, compares the APIs being used to the function permissions, and reports when functions have permissions for APIs they don't need.

- ***--function NAME*** --

  Function name to be used in policy detection and Console results. When creating policy rules in Console, you can target specific rules to specific functions by function name. If this field is left unspecified, the function zip file name is used.

- ***--output-used-apis*** --

  Report APIs used by the function

- ***--publish*** --

  Publish the scan result to the Console. True by default.

VMware Tanzu blobstore scanning

Edit on GitHub

Prisma Cloud for TAS can scan the droplets in your blobstores for vulnerabilities. Prisma Cloud can be configured to scan your blobstores periodically. Defenders are the entities that perform the scanning.

When you install Tanzu Application Service (TAS) Defender in your environment, it automatically scans the running apps and hosts in your environment without any special configuration required.

Tanzu stores large binary files in blobstores. Blobstores are roughly equivalent to registries. One type of file stored in the blobstore is the droplet.

Droplets are archives that contain ready to run applications. They are roughly equivalent to container images. Droplets contain the OS stack, a buildpack (which contains the languages, libraries, and services used by the app), and custom app code. Before running an app on your infrastructure, the Cloud Controller stages it for delivery by combining the OS stack, buildpack, and source code into a droplet, then storing the droplet in a blobstore.

The *twistcli* command line tool also lets you scan droplet files directly. You can integrate *twistcli* into your CLI to pass or fail builds based on vulnerability thresholds.

Configure Prisma Cloud to scan a blobstore

Prisma Cloud can scan both internal and external blobstores, and blobstores configured to use the Fog Ruby gem or WebDAV protocol.

Prequisite: You've already installed TAS Defender in your environment.

STEP 1 | Log into Prisma Cloud Console.

STEP 2 | Go to **Defend > Vulnerabilities > VMware Tanzu blobstore**.

STEP 3 | Click **Add blobstore**.

STEP 4 | In **Blobstore location**, select if scanning is Local or Remote.

Prisma Cloud allows you to scan a blobstore by a Defender within the same TAS environment, or to scan it by a Defender in a remote TAS environment. If the Defender (the Scanner) runs in the same TAS environment as the blobstore, select **Local**. If you want a Defender to scan a blobstore in a different TAS environment, select **Remote**.

STEP 5 | In **Blobstore's cloud controller**, specify the cloud controller address of the blobstore you want to scan.

STEP 6 | For **Remote** scanning:

1. (Optional) In **Foundation**, specify the foundation of the blobstore to scan. The foundation name will then be added as a label to the droplets scanned on this blobstore, which allows you to use it as a criteria for Collections.
2. In **Credentials**, enter the credentials required to access the remote blobstore. If the credentials have already been created in the Prisma Cloud credential store, select it. If not, click **Add** to create new credentials.

   The user role of the credentials you use should be one of the following: Admin, Admin Read-Only, Global Auditor, Org Manager, Space Auditor, Space Developer, or Space Manager. For non-admin users, the *cloud_controller.read scope* is also required.
3. (Optional) In **CA certificate**, enter a CA certificate in PEM format.
4. In **Scanner's cloud controlles**, specify the cloud controller address of the TAS environment where the scanning Defender is located.

STEP 7 | In **Scanner**, specify a Defender to execute the scanning.

Prisma Cloud lists all the agentIDs where Defender is installed. To correlate the agentID to the Diego cell's IP address, and determine which host runs a Defender, log into any Diego cell, and inspect */var/vcap/instance/dns/records.json*. This file shows the correlation between agentID and host IP address.

STEP 8 | In **Application name**, specify the droplets to scan. Wildcards are supported. To scan all droplets, enter a single wildcard (*).

STEP 9 | In **Cap**, specify the maximum number of droplets to scan. To scan all droplets, enter 0.

STEP 10 | Click **Add**.

STEP 11 | Click **Save**.

Review scan reports

Scan reports show all vulnerabilities found in the droplets in your blobstores. By default, droplets are rescanned every 24 hours.

A droplet, which is an artifact of the app staging process, contains the minimum required data to specify an app (binaries/libraries). Droplets are stored in blobstores. Review scan reports for droplets in **Monitor > Vulnerabilities > VMware Tanzu blobstore**.

When an application is run in a Diego cell, it's run on top of a stack, currently cflinuxfs3, which is derived from Ubuntu Bionic 18.04. Defender automatically scans all running applications (buildpack and docker). Review the scan reports for running apps in **Monitor > Vulnerabilities > Images**.

If you compare the findings for a buildpack app in **Monitor > Vulnerabilities > VMware Tanzu blobstore** and **Monitor > Vulnerabilities > Images**, you'll notice a difference in the number of findings. Remember that **Monitor > Vulnerabilities > Images** reports any additional findings in the app's underlying stack that would not be found in the droplet alone.

When TAS stages Docker-based apps, it doesn't stage an associated droplet in the blobstore. Therefore, blobstore scanning alone won't cover Docker-based apps. If you're running Docker containers in TAS, and you want to scan the images before they run, then configure Prisma Cloud to scan the container registry.

STEP 1 | Log into Prisma Cloud Console.

STEP 2 | Go to **Monitor > Vulnerabilities > VMware Tanzu blobstore** to see a list of summary reports for each droplet.

STEP 3 | To drill into a specific scan report, click on a row in the table.

Scan App-Embedded workloads

Edit on GitHub

App-Embedded Defenders can scan their workloads for vulnerabilities.

To see the scan reports, go to **Monitor > Vulnerabilities > Images > Deployed** You can filter the table by:

- **App-Embedded: Select** — Narrows the results to just images protected by App-Embedded Defender.
- **App ID** — Narrows the list to specific images. App IDs are listed under the table's **Apps** column.

  For ECS Fargate tasks, the App ID is partially constructed from the task name. AWS Fargate tasks can run multiple containers. All containers in a Fargate task have the same App ID.

  For all other workloads protected by App-Embedded Defender, the App ID is partially constructed from app name, which is a deploy-time configuration set in the App ID field of the embed workflow.

You can use wildcards to filter the table by app/image name. For example, if the app name is *dvwa*, then you could find all deployments with *Repository: dvwa**. This filter would show *dvwa:0438dc81a9144fab8cf09320b0e1922b* and *dvwa:538359b5f7f54559ab227375fe68cd7a*.

Create vulnerability rules

Create a vulnerability rule for a segment of App-Embedded workloads.

STEP 1 | Login to the Console.

STEP 2 | Go to **Defend > Vulnerabilities > Images > Deployed**.

STEP 3 | Click **Add rule**.

STEP 4 | Entar a rule name.

STEP 5 | Click on **Scope** to select a relevant collection, or create a new collection.

Workloads are scoped by App ID. App ID is specified when you embed the App-Embedded Defender into a workload, and it's a unique identifier for the Defender/task pair.

1. If creating a collection, click **Add collection**.
2. Enter collection name.
3. In the **App ID** field, enter one or more App IDs.

   Postfix wildcards are supported.
4. Click **Save**.
5. Select the new collection.
6. Click **Select collection**.

STEP 6 | Select an alert threshold.

Thresholds select, by severity, which vulnerabilities Prisma Cloud should report.

App-Embedded Defenders don't support the block action.

STEP 7 | Click **Save**.

Deploy an example Fargate task

Deploy the *fargate-vulnerability-compliance-task* Fargate task. Follow the steps in embed App-Embedded Defender into Fargate tasks.

You can use the following task definition to test Prisma Cloud's App-Embedded Defender. It's based on an Ubuntu 18.04 image.

```
{
  "containerDefinitions": [
     {
        "command": [
           "/bin/sh -c 'cp /bin/sleep /tmp/xmrig && echo \"[+]
 Sleeping...\" && while true; do sleep 1000 ; done'"
        ],
        "entryPoint": [
           "sh",
           "-c"
        ],
        "essential": true,
        "image": "ubuntu:18.04",
        "logConfiguration": {
           "logDriver": "awslogs",
           "options": {
              "awslogs-group" : "/ecs/fargate-task-definition",
              "awslogs-region": "us-east-1",
              "awslogs-stream-prefix": "ecs"
           }
        },
        "name": "Fargate-vul-comp-test",
        "portMappings": [
           {
              "containerPort": 80,
              "hostPort": 80,
              "protocol": "tcp"
           }
        ]
     }
  ],
  "cpu": "256",
  "executionRoleArn": "arn:aws:iam::012345678910:role/
ecsTaskExecutionRole",
  "family": "fargate-vulnerability-compliance-task",
  "memory": "512",
  "networkMode": "awsvpc",
  "requiresCompatibilities": [
      "FARGATE"
```

```
    ]
}
```

Review vulnerability scan reports

Review the scan results in Console.

*If an App-Embedded Defender protects a container where the user isn't root, the vulnerability and compliance scanning procedure will encounter permission denied errors that you can see in the Defender logs (**Manage > Defenders > Manage > Defenders > Actions > Logs**).*

If an App-Embedded Defender protects a Fargate task with a container where the user isn't root, the vulnerability and compliance scanning procedure will also encounter permission denied errors. However, the errors won't be visible unless you download and inspect the Defender logs.

In both cases, the scan flow continues even though errors are encountered.

For Fargate version 1.3.0 and older, Prisma Cloud shows only a single scan report if the same image is run simultaneously as:

- *A task on ECS Fargate, protected by App-Embedded Defender.*
- *A container on a host, protected by Container Defender.*

*In this case, the image is categorized as "App-Embedded". As a result, when the scan report table is filtered by **App-Embedded: Select**, a scan report will be shown. When the table is filtered by **App-Embedded: Exclude**, it will be hidden. And when filtering by **Hosts**, it will be hidden, even if the host matches, because the image is considered as App-Embedded.*

For Fargate version 1.4.0, two separate scan reports are shown, one for App-Embedded and one for Container Defender.

STEP 1 | Navigate to **Monitor > Vulnerabilities > Images > Deployed** and validate that the deployed image appears and contains vulnerabilities.

STEP 2 | To see all images that are related to Fargate tasks, filter the image table by **App-Embedded: Select**.

You can also filter the results by a specific task name or postfix wildcards. For example, *fargate-task* or *fargate-task**.

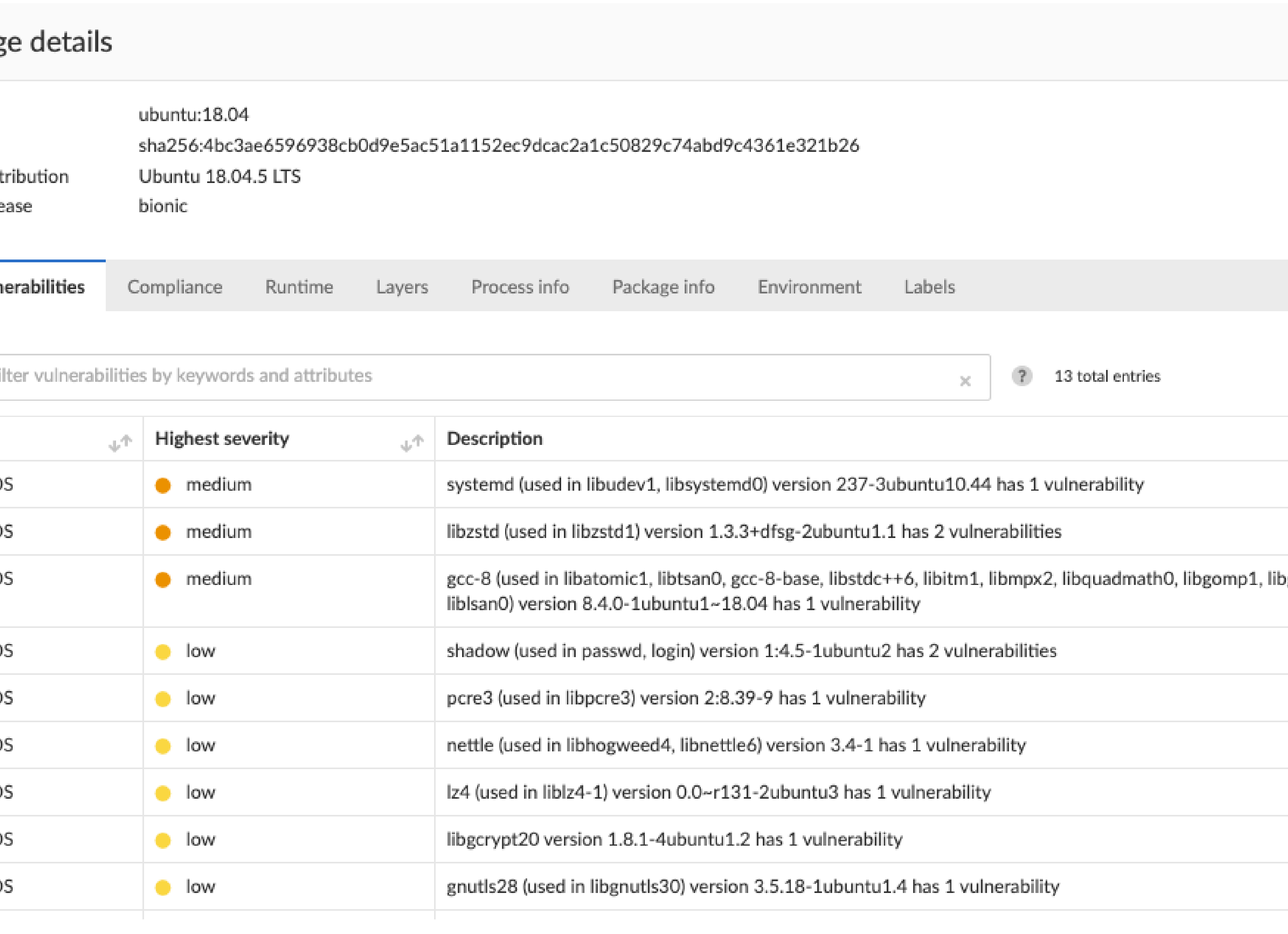
STEP 3 | Search for the *fargate-vulnerability-compliance-task* Fargate task.

STEP 4 | Click on the image to see more details.

*The **Apps** column shows a count of the number of running containers protected by App-Embedded Defender.*

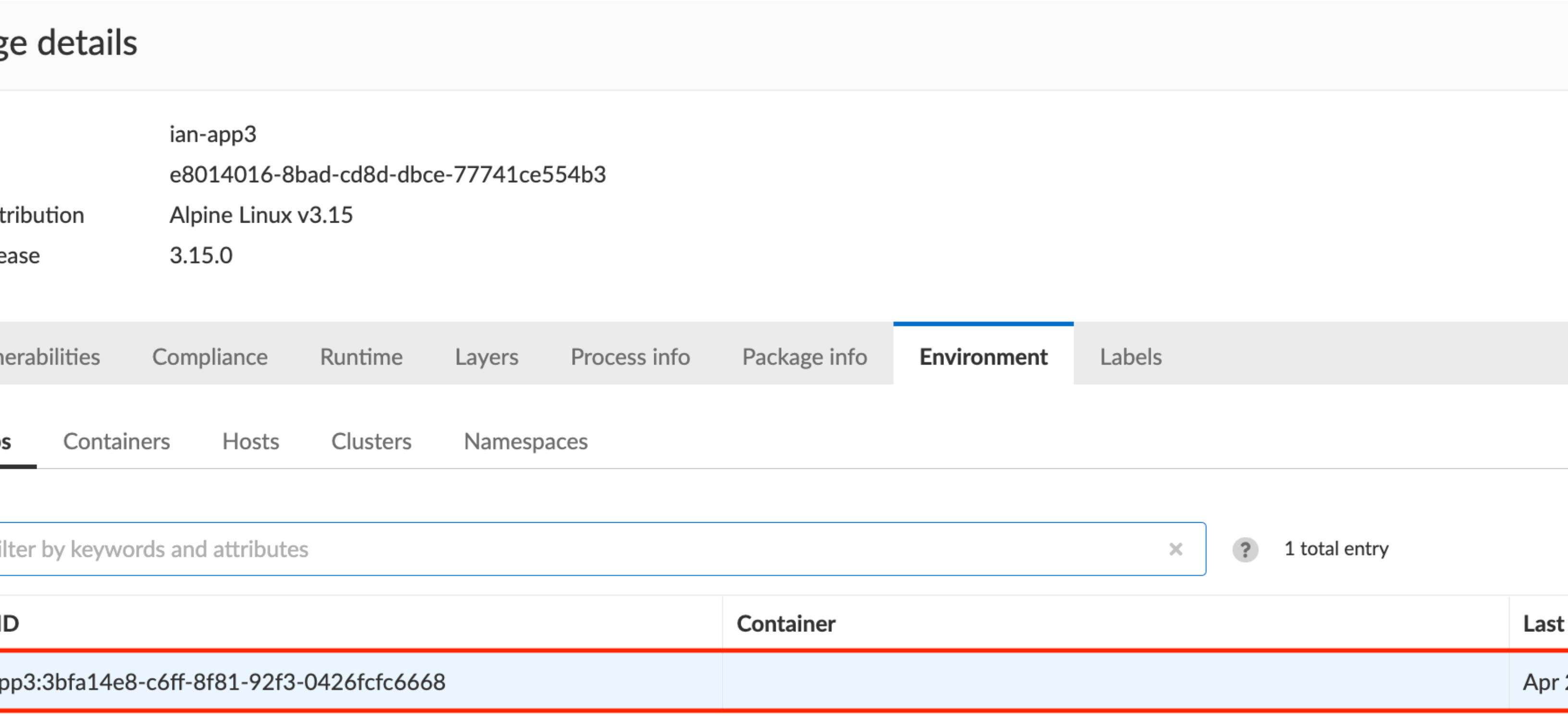
*The **Layers**, **Processes info**, **Labels**, **Runtime**, and **Trust groups** tabs aren't supported for images scanned by App-Embedded Defenders.*

1. Click the **Vulnerabilities** tab to review all findings.

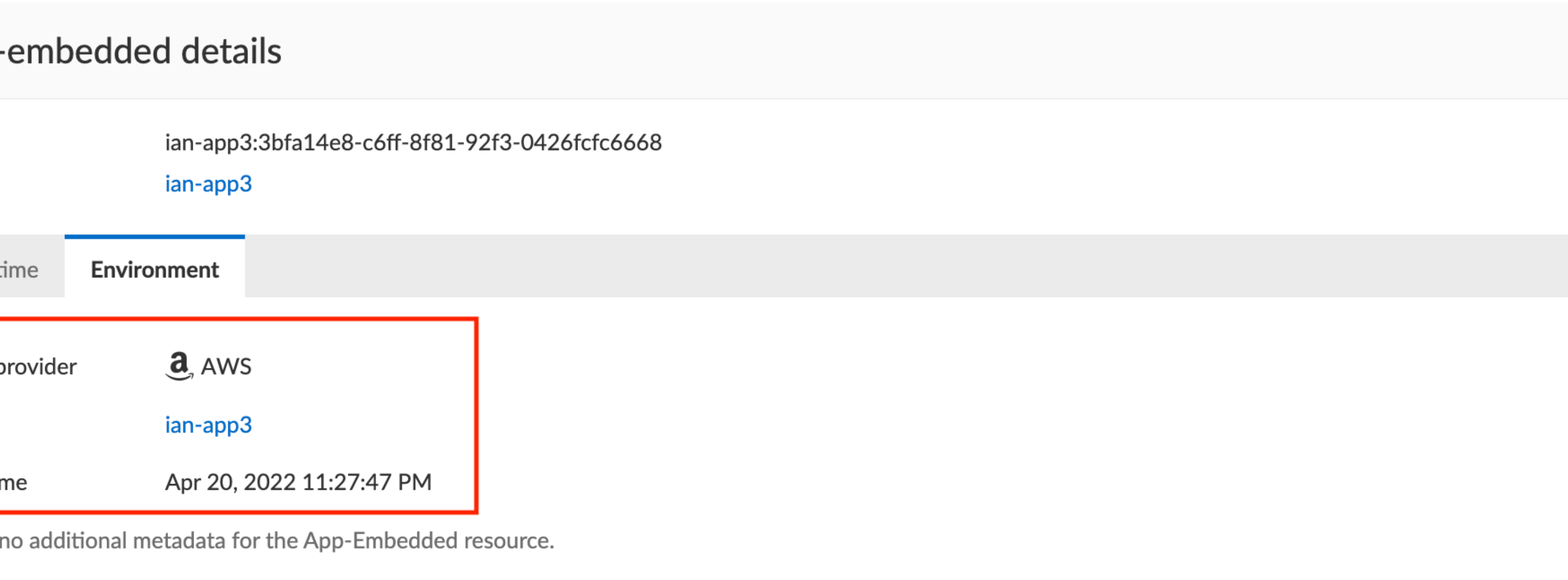


2. Review runtime information for the container.

Go to the **Environment > Apps** tab, and then click on the app in the table to open the App-Embedded observations. You can bring up the same view by going directly to **Monitor > Runtime > App-Embedded observations**, and clicking on the same app.



The **Environment** tab shows cloud-provider metadata that App-Embedded Defender collected about the running container. For more information about the type of cloudprovider metadata App-Embedded Defender can collect, see Monitoring workloads at runtime.



Troubleshoot vulnerability detection

Edit on GitHub

Prisma Cloud offers a comprehensive Intelligence Stream for vulnerability management that draws on threat intelligence from commercial providers, the open source community, as well as distinctive vulnerability intelligence curated by Prisma Cloud vulnerability researchers.

Use this troubleshooting section to verify the accuracy of Prisma Cloud scan results, understand the logic behind scan reports, and provide the details requested in the template when you submit a support request for further analysis.

The information in this section will help answer the most common questions related to CVE scan reports –

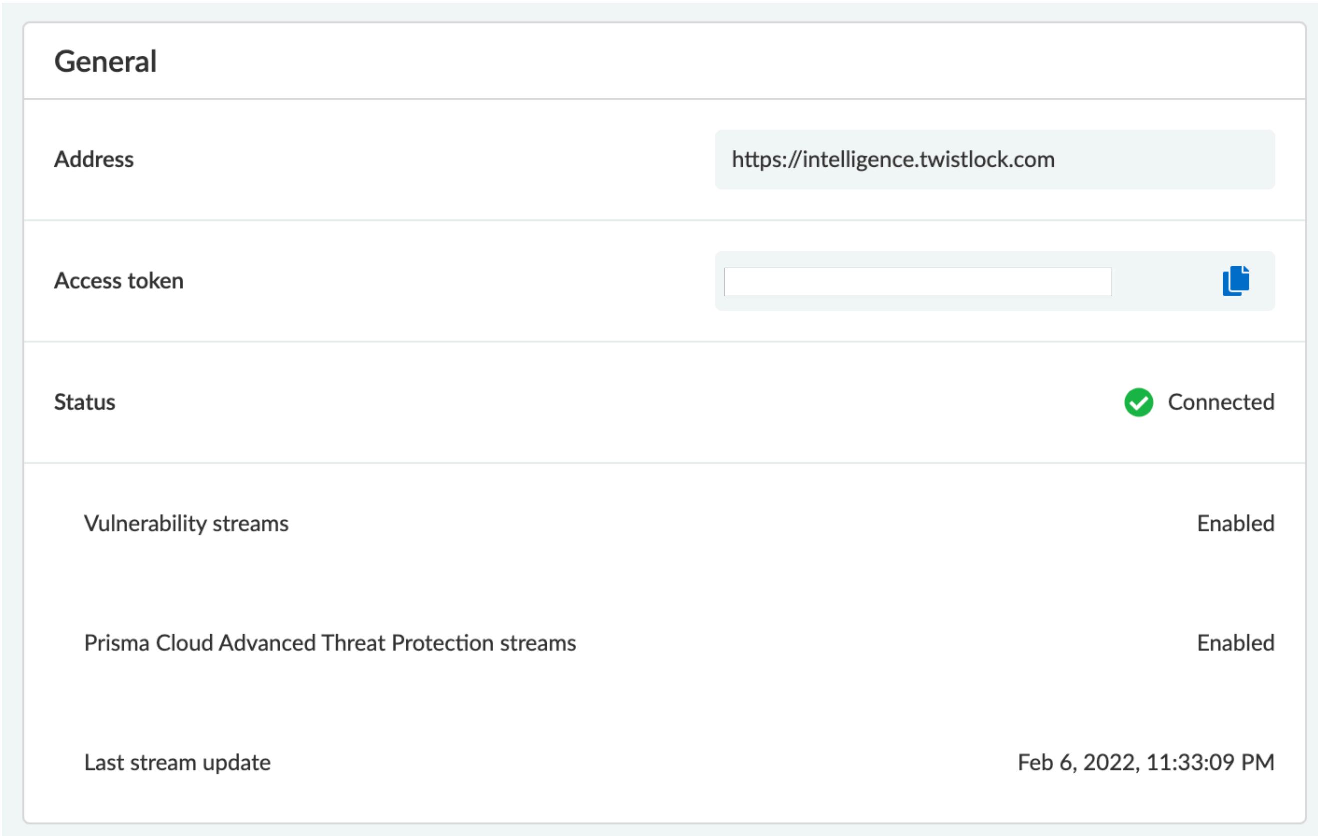
1. Whether a CVE reported by Prisma Cloud is suspected to be a false positive (meaning there is an assumption that the CVE doesn't exist on a package / image, but it displays on the Prisma Cloud Console)
2. Whether a CVE in Prisma Cloud is suspected to be a false negative (meaning there is an assumption that a CVE does exist on package / image, but it does not display on Prisma Cloud Console)

Prerequisites

Before you start with the troubleshooting workflow, check the following prerequisites for accurate scan results.

1. Ensure you are running the latest version of Prisma Cloud Compute Console.
2. Ensure you are running a supported version of Prisma Cloud Compute Defenders. Prisma Cloud Defender version is backward compatible for up to two major releases of Console.
3. Ensure that the image or OS is supported.
    1. If the problem is in a container, ensure that the image is based on a supported OS.
    2. If the problem is in a host, ensure it is running a supported OS.

4. Connection to Intelligence Stream is up to date.
    1. Navigate to **Manage > System > Intelligence**.
    2. Verify that the **status** is **Connected**.



Troubleshooting Steps

After you complete the prerequisite checks, continue to troubleshoot further. The commands below are for Linux distributions but you can use the same process for Windows distributions.

Step 1: Running the image in a container

Whether troubleshooting for a false positive or a false negative scenario, the image should be searched for signs of the given package or file that has been associated with the CVE. Running the image in a container is a good way to proceed. As a best security practice, always run these experiments in a sandbox environment instead of production.

Download the image or load it from a tar on a host protected by a container Defender environment:

```
docker pull <imagename> OR docker load -i <image.tar>
```

Instantiate a container from the image:

```
docker run —rm —detach —name vuln_testing <imagename>
```


If the image exits immediately, the entrypoint or CMD associated with it most likely doesn't spawn a long running process. In that case, the docker run command given above can have a command and arguments appended to it overriding the built in directives and ensuring that the container remains up while it is being investigated. For example:

```
$ docker run –rm –detach –name vuln_testing <imagename>
 sleep infinity
```

Step 2: Investigate the Container

Get the ID of the running container then exec into it:

```
$ docker ps | grep vuln_testing
$ docker exec -ti <containerID> /bin/bash
```

If the bash shell isn't installed in the image, try alternate shells such as /bin/sh or /bin/ash.

Step 3: Find the Linux distribution of the image

Match the detected OS type in the Console against the listed OS inside running container to ensure that it was correctly identified.

```
$ cat /etc/os-release
```

Tip: If the os-release file is not found, look for /etc/redhat-release, /etc/lsb-release, or other files matching /etc/-release.*

Step 4: Locate the package associated with the CVE

Locate the package or file that is associated with the CVE that was listed, or that was not detected despite the expectations. Additionally, confirm the version of the package detected inside the container with the one shown in Prisma Cloud Console or, in case of false negative, which is shown in the other source confirming the CVE.

An easy command to do this is to use find the package in the whole filesystem is:

```
$ find / -name <package_name>
```

Example:

```
abc@3f61f8497e23:/# find / -name console
/dev/console
/sys/devices/virtual/tty/console
/sys/class/tty/console
```

Run the package binary with --version tag if available. You can also search for the version in Console. Go to **Monitor > Vulnerabilities**, then click on an image, and select the **Package Info** tab.

Example:

```
abc@3f61f8497e23:/# /usr/bin/wget --version
```



```
GNU Wget 1.20.3 built on linux-gnu.
```

Some other ways to find the package, depending on the type of package are -

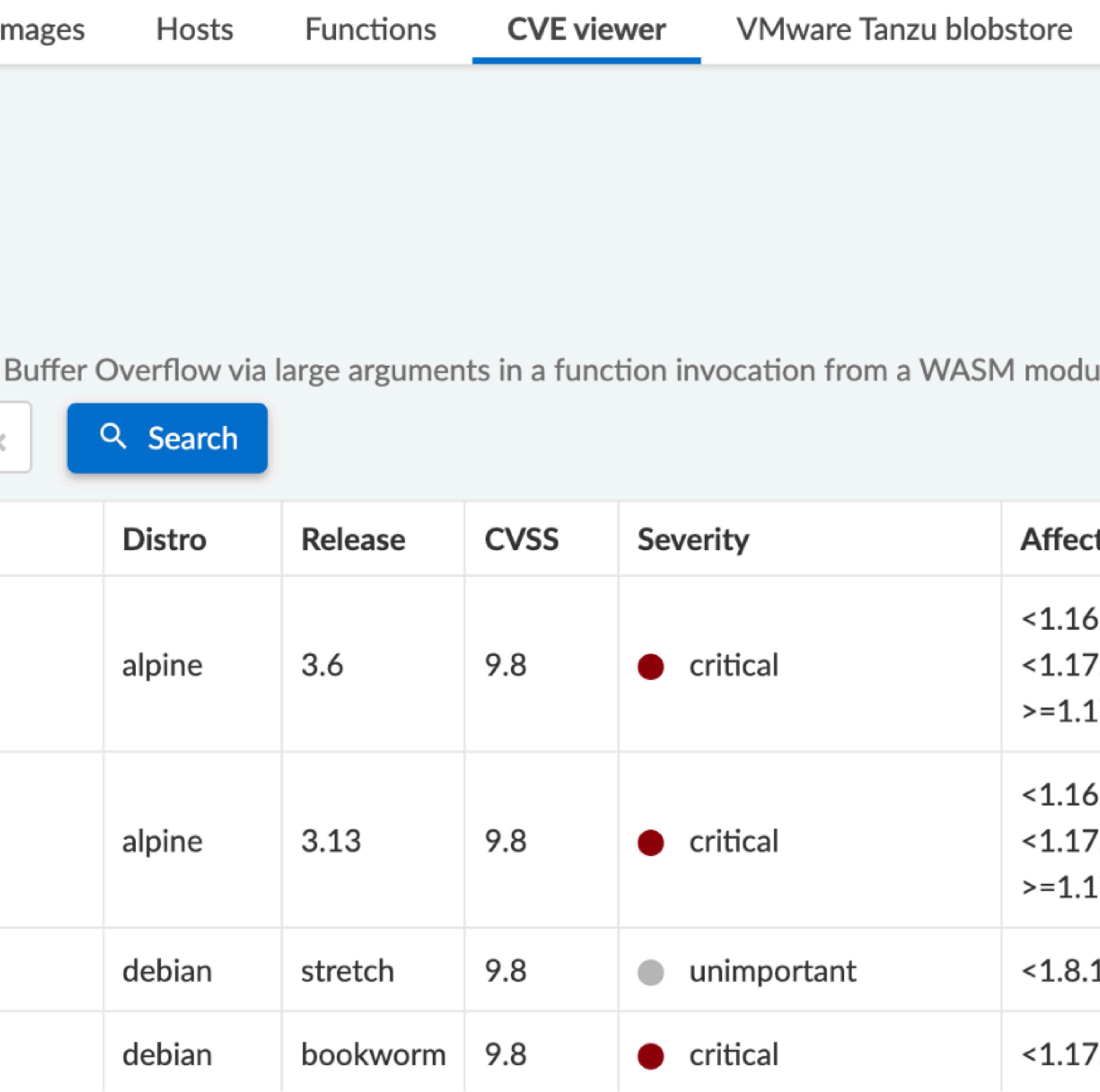
<table>
<tr><th>Package Type</th><th>Command</th></tr>
<tr><td>jar</td><td><pre>find / -iname '*.jar' | grep <jar_name></pre>Get the version from the jar name.
Example output:<pre>/opt/amq/webapps/hawtio/WEB-INF/lib/
httpcore-4.4.4.jar</pre></td></tr>
<tr><td>Npm/node packages</td><td><pre>npm list | grep -i <package_name></pre><pre>sh-4.2$ cd <path>
sh-4.2$ cat package.json | grep -i version</pre>If investigating false positives, find the package path from image details in Console. Select Monitor > Vulnerabilities > Images, click on the image, and select the Package Info tab.</td></tr>
<tr><td>OS</td><td>For OS packages, use the OS package manager to find the installed package and version.
For example, you can use the following for RHEL/CentOS/SUSE packages (here searching for the curl package): rpm -qa | grep curl yum list installed | grep -i curl dnf list installed | grep -i curl Another example for Debian/Ubuntu: apt list --installed | grep -i curl dpkg --list | grep -i curl</td></tr>
<tr><td>python</td><td>For python packages, you can run the following command in the package path (if already known)<pre>$ cat __init__.py | grep -i __version__</pre>(OR) in the .dist-info directory.<pre>$ cat METADATA | grep -i version</pre></td></tr>
</table>

Analyzing Results

The above steps should help answer whether the vulnerable package exists in the image or not and answer if a CVE is truly false positive. If you found the package and the vulnerable version in image but have questions on the report's accuracy, you can search the vendor's official feeds to confirm the source of the CVE report.

1. "I found the package but I'm not sure if it's truly vulnerable."

Navigate to **Monitor > Vulnerabilities > CVE Viewer**, type the CVE ID and verify the source matching OS of your image, or look for the reference with empty Distro and Release if it's a specific language library.


mages
Hosts
Functions
CVE viewer
VMware Tanzu blobstore
Buffer Overflow via large arguments in a function invocation from a WASM modu
Search
Distro
Release
CVSS
Severity
Affect
alpine
3.6
9.8
critical
<1.16
<1.17
>=1.1
alpine
3.13
9.8
critical
<1.16
<1.17
>=1.1
debian
stretch
9.8
unimportant
<1.8.1
debian
bookworm
9.8
critical
<1.17

You can then directly search vendor feeds to confirm CVE's authenticity. For OS packages, the relevant vendor site should be consulted. For specific language libraries, the site of that project should be visited. The National Vulnerability Database (NVD) should be used for locating CVE information that is not available on official vendor feeds.

Vendor vulnerability data may differ between feeds and NVD analysis. For example, in severity, description or affected versions. Prisma Cloud gives more weight to specific vendor analysis to provide accurate vulnerability data.

Example 1: A vulnerability was determined to be high severity per NVD analysis, but Red Hat Linux analysis determined the vulnerability to be of high severity on RHEL releases. Prisma Cloud should display high severity in this case.

Example 2: A vulnerability was discovered in an open-source package and was fixed in the latest release. NVD analysis mentioned the vulnerability affects all releases earlier than the latest release. At the same time, the vulnerability could be fixed on earlier releases on RHEL, with maintainers having backported the patch to earlier releases of the package for RHEL.

2. "I found the vulnerable package but Prisma Cloud doesn't show it's CVE."

When looking into a false negative, it is important to confirm the type of the vulnerability (that is anticipated to be 'missing' from scan results), where type equals one of the supported formats that Compute currently detects when interrogating an image.

Supported types:

- package - an OS package, such as an RPM (Red Hat and derived distributions), dpkg/deb (Debian and derived distributions), or apk (Alpine Linux).
- jar - the Java Archive format, which is a zip file with a standard structure. The war file format, or web app archive, is also supported.
- python - a Python library, sometimes consisting of zip archives with varying structures and names (eggs, wheels) or plain text files on disk with supporting metadata text files.
- nodejs - a NodeJS library, primarily consisting of text files on disk with supporting metadata text files.
- gem - a Ruby library, consisting of text files on disk with supporting metadata text files.
- go - a Golang binary, which typically contains dependencies that are statically compiled into it. Where most C programs make use of dynamically linked libraries/shared-objects that are present on the host and pulled in at run time, Golang binaries usually have their dependencies embedded within them at compile time.
- app - a binary associated with a well known application, such as Nginx or PostgreSQL.

If it is one of the above supported types yet missing in Prisma Cloud Compute's scan report, then open a support case and provide the following information so our teams can investigate further.

Submit a Support Request

When submitting a technical support request with Palo Alto Networks, provide the following information to help our teams identify the root cause more quickly. This information is required to review escalations.

1. Debug logs: Provide full debug logs through **Manage > System > View Logs > Upload / Download Debug logs**. You can also use twistcli to upload logs:

```
$ ./linux/twistcli support upload --help
```

   1. The debug log option is only available on self-hosted Consoles. In the event that you have a SaaS Console, gather the console.log (from **Manage > System > View Logs**) and the defender.log (under /var/lib/twistlock/log directory on host) from the host where the image was first scanned.
2. **Image details**: If the issue is in a container image, provide the affected container image (image.tar). You may also check if the image can be downloaded from Docker Hub and share a link to pull the image. Always validate the Image ID SHA to ensure it's the same image. If you are unable to share the image, please provide an image where the issue reproduces that we can analyse.
3. **Scan discrepancy report sheet**: Ensure you have a spreadsheet with following columns info filled out from your prior analysis.

CVE ID	Package Type	Package Name	Package Version	Path where package is found in image	CVE Reported in Console? Yes/No	CVE Reported by any other vendor/ source?	Your explanation comments
Example: CVE-2021-38297	OS	gnutls28	3.6.7-4+deb10u5	/usr/bin/ gnutls	Yes. Suspect it to be a false positive	Yes, NVD: https://nvd.nist.gov/vuln/detail/CVE-2021-38297	I don't believe this CVE should be reported for this version of package because I don't see version in NVD.

Frequently Asked Questions

I see a CVE in the scan, but it does not appear on NVD or is still under analysis. What is the information I'm seeing?

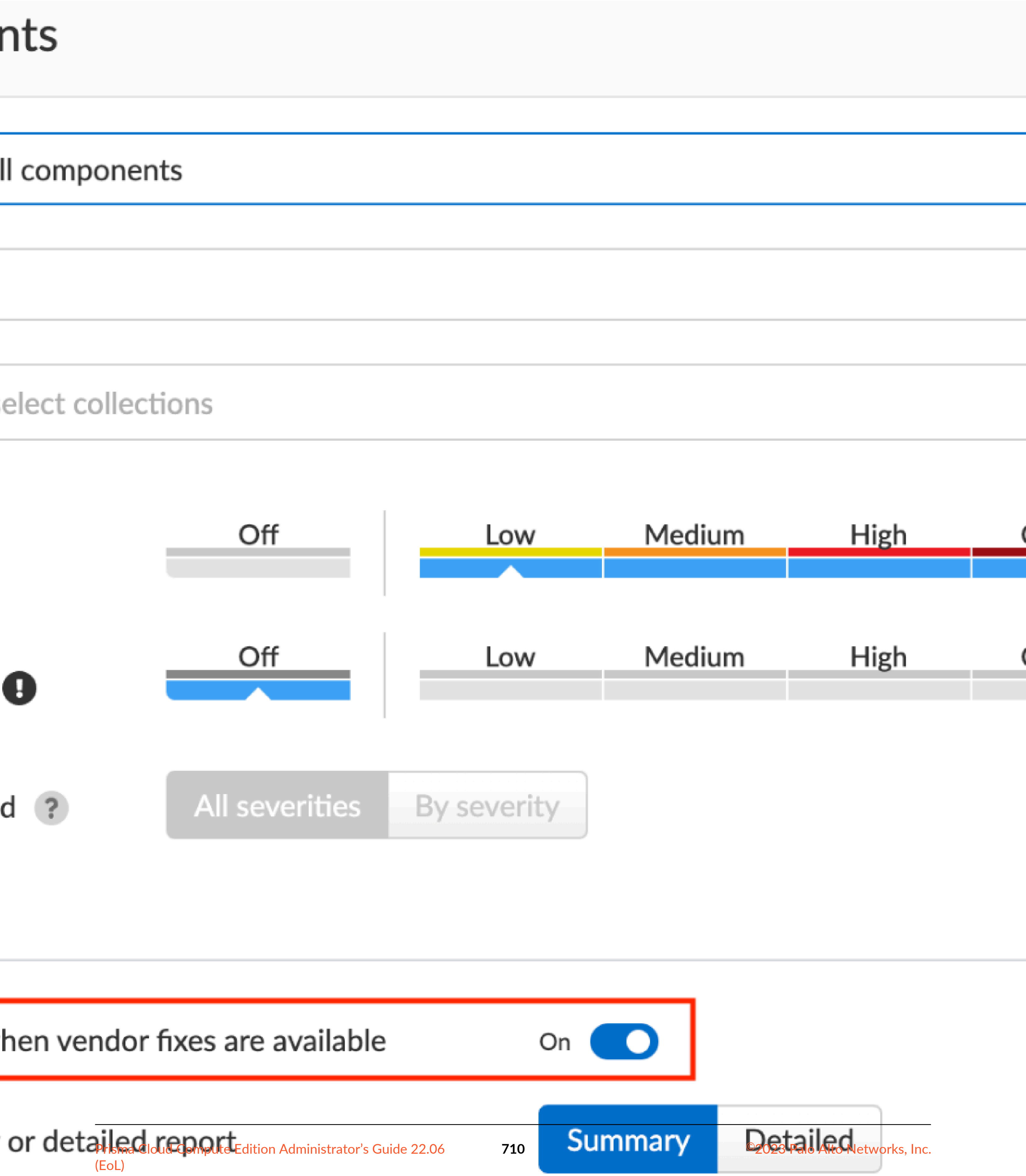
When a CVE is assigned to a vulnerability, usually NVD analysis takes place, and it may take multiple days for the NVD site to update with description and the affected releases range. Instead of waiting for the official analysis to complete, our researchers manually review the details of the CVE and add it as a pre-filled CVE to our Intelligence Stream, so you can know you are vulnerable and mitigate the vulnerability before the official analysis is done. See the Prisma Cloud vulnerability feed doc for more information.

What are PRISMA-* vulnerabilities?

Our researchers assign a PRISMA-* identifier for vulnerabilities that lack a CVE ID. Many vulnerabilities are publicly discussed or patched without a CVE ever being assigned to them. Our researchers find those vulnerabilities, analyse them and assign a PRISMA ID whenever applicable, so you can know what you need to be aware of. See the Prisma Cloud vulnerability feed doc for more information.

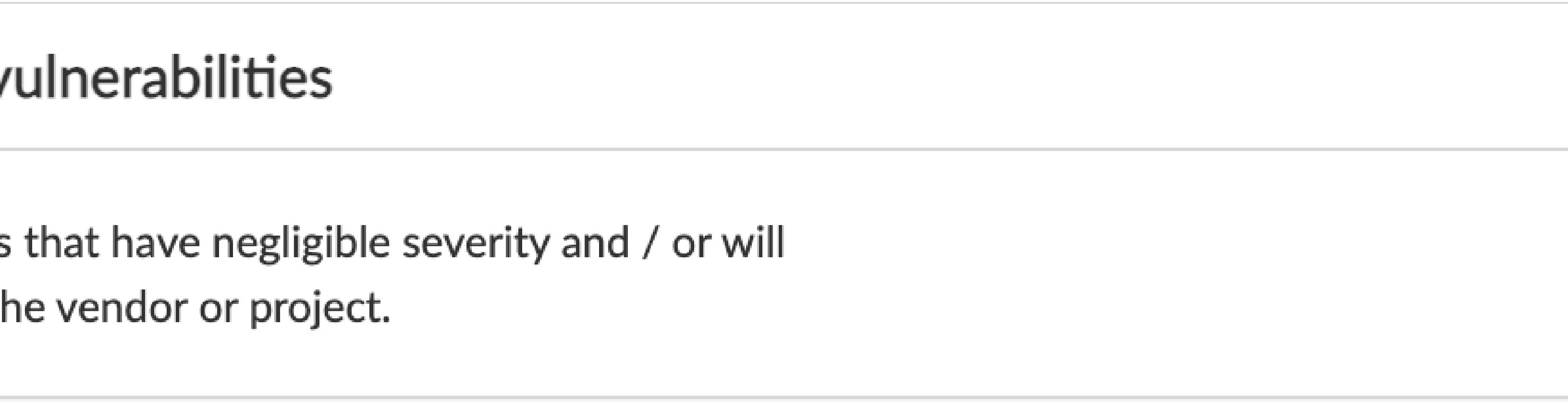
I see CVEs with Fix status “affected”. What are these? Are they false positives?

CVEs with the status “affected” are CVEs that don’t have a fix yet, and the vendor marked them as affecting the current OS release. Some other vulnerability scanners don’t show them, but these are not false positives. You can also decide not to show vulnerabilities with no fix - go to Defend > Vulnerabilities > edit your desired rule > Advanced settings > turn on the toggle “Apply rule when vendor fixes are available”.


nts
ll components
elect collections
Off
Low
Medium
High
Off
Low
Medium
High
d
All severities
By severity
hen vendor fixes are available
On
or detailed report
Summary
Detailed

I see a lot of low severity CVEs. What are these? Are they false positives?

You can decide if you want to see vulnerabilities that have negligible severity or “will not fix” status. These CVEs have already been reviewed by the vendor and are not going to be fixed. Although they are not truly false positives, Prisma Cloud Compute doesn’t show these CVEs by default, since the vendor decided a fix is not necessary. You can change this configuration - go to **Manage > System > Scan > Unactionable vulnerabilities**.



Where do you take CVE information such as severity and fixed version from?

For known vulnerabilities with a CVE, we rely on the most authoritative source - for OS packages (packages that are maintained by the OS vendor, marked as type “package” in Compute), the CVE details are taken from the specific vendor feed. For other CVEs, the information is taken from official sources like NVD and vendor specific Security Advisories. For new vulnerabilities missing analysis or undocumented vulnerabilities (such as PRISMA-IDs), we rely on severity determined by our researchers.

Do all CVEs reported by Prisma Cloud rely on information from NVD?

The National Vulnerability Database (NVD) is one of the major sources on which the Intelligence Stream relies or accurate CVE information. In addition to using NVD and other vendor sources, Prisma Cloud security researchers analyze vulnerabilities on a daily basis. In case we find any discrepancies between our analysis to that of NVD or any other vendor, we partner with them to correct any missing or inaccurate information. We strive to contribute to the security of the open-source community.

I see on the Red Hat security page that a CVE affects my OS release, but it doesn’t show up in Prisma’s scan. What happened?

Our Intelligence stream is drawing CVE information from Red Hat API - using OVAL v2 streams. While the HTML CVE page is already updated, there could be a delay in the API update.

Why does Prisma Cloud show more vulnerabilities than what I see in the Red Hat catalog?

The Red Hat Container Health Index analysis is based on RPM packages signed and created by Red Hat, and does not grade other software that may be included in a container image. Thus, non-OS vulnerabilities like jar, python, etc., will not be listed on Red Hat Catalog. Furthermore, the

Red Hat catalog only shows CVEs that have a fix, meaning there is a security advisory with the fix. "Affected" CVEs (see above) don't have a fix, and they won't appear in the Red Hat catalog.

What is the "Published Date" in Console?

Published date is the date that the CVE was published by the vendor / project or by NVD. This information is taken from the relevant feed - either the vendor feed or NVD. Please note that the date a CVE is published in NVD is not the date it was analyzed. The CVE can be published in NVD and only later updated with the analysis.

What is the "Fix Date" in Console?

Fix date is the date the vulnerability data was fixed by the vendor. When we can't find the relevant fix date in the official feeds, the published date in NVD is considered as the fix date.

A new vulnerability is affecting Compute - what should I do?

If the vulnerability affects Compute that has not yet been addressed, please report it through support channels or to PSIRT.

A CVE exists in NVD, but I don't see it in the CVE viewer, what should I do?

If you believe a CVE that was fully analysed by NVD is missing from our feeds, please report it through the support channels.

Compliance

Edit on GitHub

Prisma Cloud helps enterprises monitor and enforce compliance for hosts, containers and serverless environments. Use the compliance management system to enforce standard configurations and security best practices.

- Compliance Explorer
- Enforce compliance checks
- CIS Benchmarks
- Prisma Cloud Labs compliance checks
- Serverless functions compliance checks
- Windows compliance checks
- DISA STIG compliance checks
- Custom compliance checks
- Trusted images
- Host scanning
- VM image scanning
- App-Embedded scanning
- Detect secrets
- Cloud discovery
- OSS license management

Compliance Explorer

Edit on GitHub

Compliance Explorer is a reporting tool for compliance rate. Metrics present the compliance rate for resources in your environment on a per-check, per-rule, and per-regulation basis. Report data can be exported to CSV files for further investigation.

The key pivot for Compliance Explorer is failed compliance checks. Compliance Explorer tracks each failed check, and the resources impacted by each failed check. From there, you can further slice and dice the data by secondary facets, such as collection, benchmark, and issue severity.

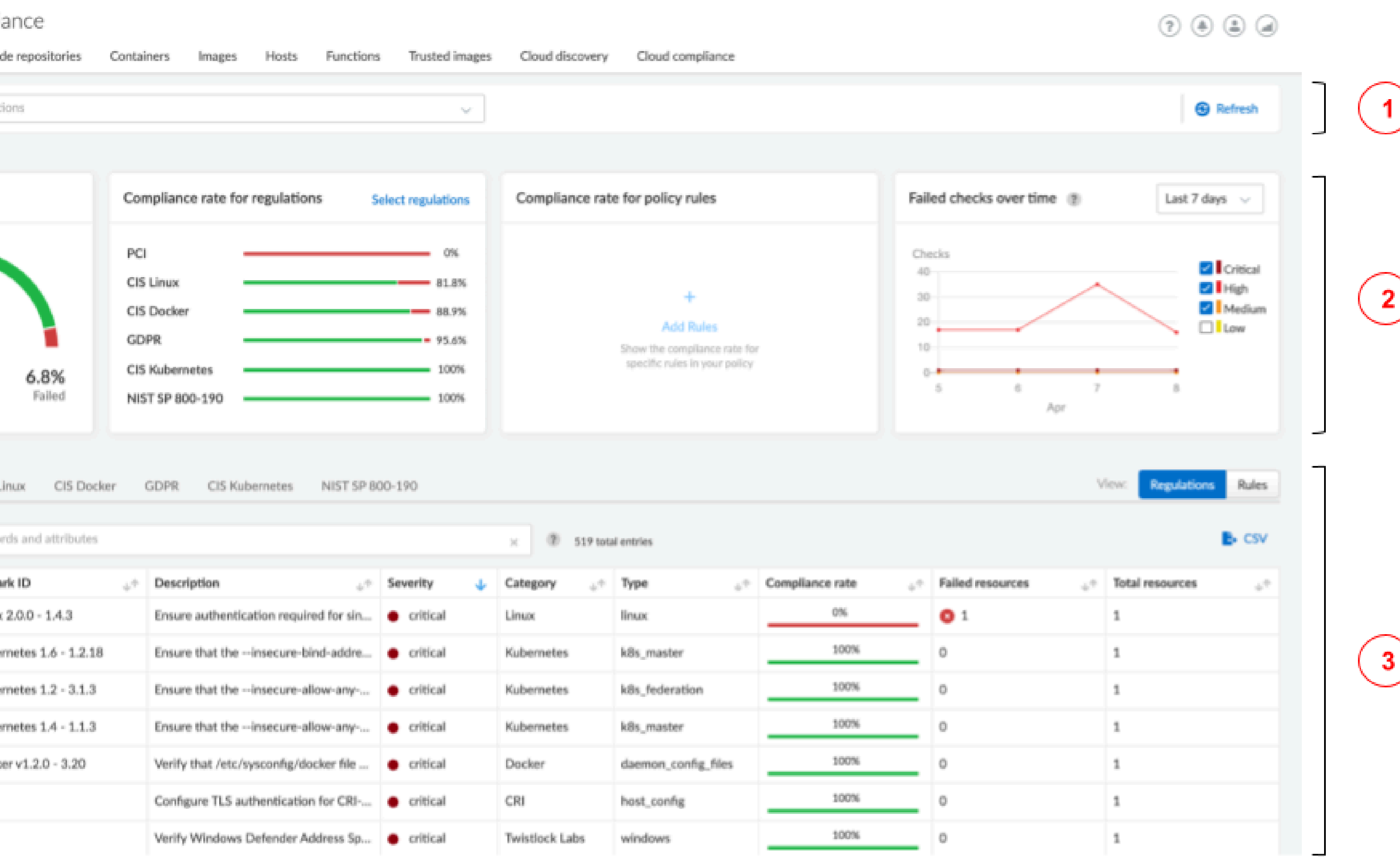
Compliance Explorer helps you answer these types of questions:

- What is the compliance rate for the entire estate?
- What is the compliance rate for some segment of the estate?
- What is the compliance rate relative to the checks that you consider important?
  - Segment by benchmark.
  - Segment by specific compliance policy rules. Prisma Cloud supports compliance policies for containers, images, hosts, and serverless functions.
- Which resources (containers, images, hosts, serverless functions) are failing the compliance checks you care about?

To view Compliance Explorer, go to **Monitor > Compliance > Compliance Explorer**.

Page organization

The Compliance Explorer view is organized into three main areas:



1. Collection filter. Collections group related resources together. Use them to filter the data in Compliance Explorer. For example, you might want to see how all the entities in the sock-shop namespace in your production cluster comply to the checks in the PCI DSS template.

2. Roll-up charts. Configurable charts that summarize compliance data for the perspectives you care about. They report the following data:

- Total compliance rate for your entire estate.
- Compliance rate by regulation. Click **Select regulations** to configure which benchmarks and templates are shown on the page. Benchmarks are industry-standard checklists, such as the CIS Docker Benchmark. Templates are Prisma Cloud-curated checklists Checks are selected from the universe of checks provided in the product that pertain to directives in a regulatory regime, such as the Payment Card Industry Data Security Standard (PCI DSS).

- Compliance rate by rule. Provides another mechanism to surface specific segments of your environment when scrutinizing compliance. Click **Add rule** to configure the card.
- Historical trend chart for compliance rate. Shows how the compliance rate has changed over time.

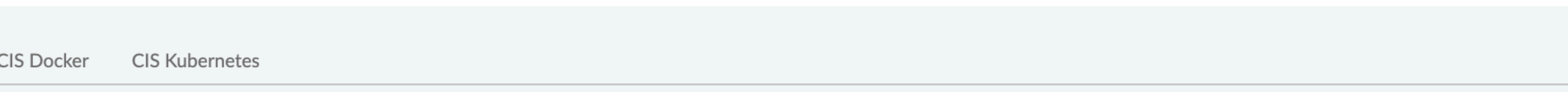
The lists in the regulation and rule cards are sorted by compliance rate (the lowest compliance rate first).

3. Results table. Shows the universe of compliance checks, and the compliance rate for each check, relative to:

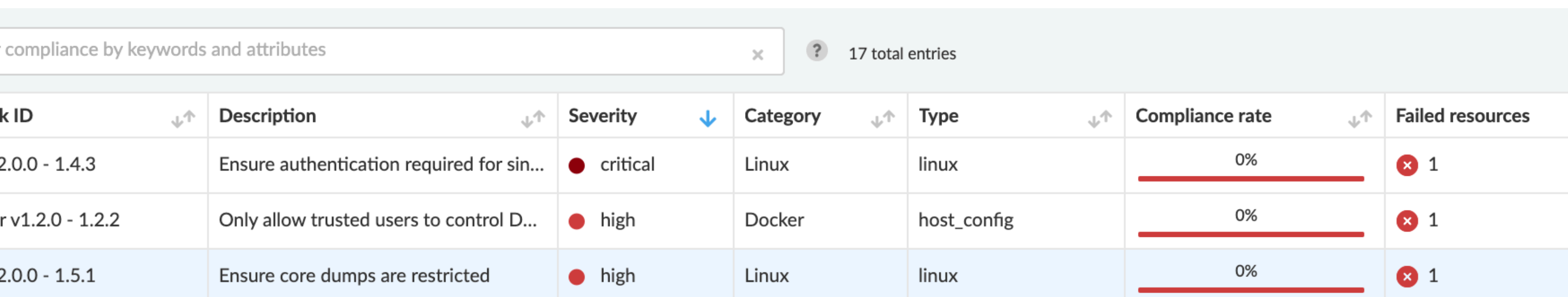
- Your policies and the checks that are enabled. Every compliance check has an ID, and it's either enabled or disabled.
- The collection selected at the top of the page.
- The filters applied (e.g. show critical severity issues only.)

By default, columns are sorted by severity (primary sort key), and then by compliance rate (secondary sort key). If no parameters are specified, Compliance Explorer shows all IDs by default.

The tabs organize results by benchmark or template. The tabs are shown or hidden based on how you configure the corresponding **Compliance rate for regulations** roll-up card. If you select the **Rules** view, the tabs show the rules selected in the **Compliance rate for policy rules** roll-up card.



Filters let you show failed checks only by setting the **Status** key to **Failed**:



k ID	Description	Severity	Category	Type	Compliance rate	Failed resources
2.0.0 - 1.4.3	Ensure authentication required for sin...	critical	Linux	linux	0%	1
r v1.2.0 - 1.2.2	Only allow trusted users to control D...	high	Docker	host_config	0%	1
2.0.0 - 1.5.1	Ensure core dumps are restricted	high	Linux	linux	0%	1

After narrowing your view of the data with collections and filters, you can export the data in the table to a CSV file.

Statistics

The data in Compliance Explorer is calculated every time the page is loaded, and it's based on data from the latest scan. Data in the trend graph is based on snapshots taken every 24 hours.

You can force Console to recalculate statistics from the latest scan data by clicking the **Refresh** button. The **Refresh** button displays a red indicator when there's a change in the following resources in your environment:

- Containers.
- Images.

- Hosts.
- Serverless functions.

For example, the refresh indicator is shown when new containers are detected. It's also shown when containers are deleted.

No red refresh indicator is shown if you simply change the compliance policy. If you change the compliance policy, manually force Prisma Cloud to rescan your environment (or wait for the next periodic scan), and then refresh the Compliance Explorer.

Enforce compliance checks

Edit on GitHub

Prisma Cloud can monitor and enforce compliance settings across your environment. Out of the box, Prisma Cloud supports hundreds of discrete checks that cover images, containers, hosts, clusters, and clouds.

Prisma Cloud provides predefined checks that are based on industry standards, such as the CIS benchmarks, as well as research and recommendations from Prisma Cloud Labs. Your security teams can review these best practices and enable the ones that align with your organization's security mandate and consistently enforce them across your environment. Additionally, you can implement your own compliance checks with scripts.

Enforcement

Compliance rules are defined and applied in the same way as vulnerability rules. When there is no matching rule for compliance checks on specific resources, Prisma Cloud generates alerts on all violations that are found. For checks that can be performed on static images, those checks are performed as images are scanned (either in the registry or on local hosts). Results are then displayed in the compliance reports under **Monitor > Compliance** on the Console.

When compliance rules are configured with block actions, they are enforced when a container is created. If the instantiated container violates your policy, Prisma Cloud prevents the container from being created.

Note that compliance enforcement is only one part of a defense in depth approach. Because compliance enforcement is applied at creation time, it is possible that a user with appropriate access could later change the configuration of a container, making it non-compliant after deployment. In these cases, the runtime layers of the defense-in-depth model provide protection by detecting anomalous activity, such as unauthorized processes.

Assume that you want to block any container that runs as root. The flow for blocking such a container is:

1. Prisma Cloud admin creates a new compliance rule that blocks containers from running as root.
2. The admin optionally targets the rule to a specific resources, such as a set of hosts, images, or containers.
3. Someone with rights to create containers attempts to deploy a container to the environment.
4. Prisma Cloud compares the image being deployed to the compliance state that it detected when it scanned the image. For deploy-time parameters, the specific Docker client commands sent are also analyzed.
    1. If the comparison determines that the image is compliant with the policy, the 'docker run' command is allowed to proceed as normal, and the return message from Docker Engine is sent back to the user.
    2. If the comparison determines that the image is not compliant, the container_create command is blocked and Prisma Cloud returns an error message back to the user describing the violation.
5. In both success and failure cases, all activities are centrally logged in Console and (optionally) syslog.

Supported runtimes

The supported runtimes for compliance are:

- Docker
- CRIO
- Containerd

Surveying Prisma Cloud compliance checks

As you configure your compliance policy, you might want more details for the built-in checks. Teams that address compliance issues might also need more information about why checks fail, so that they can resolve the underlying issues.

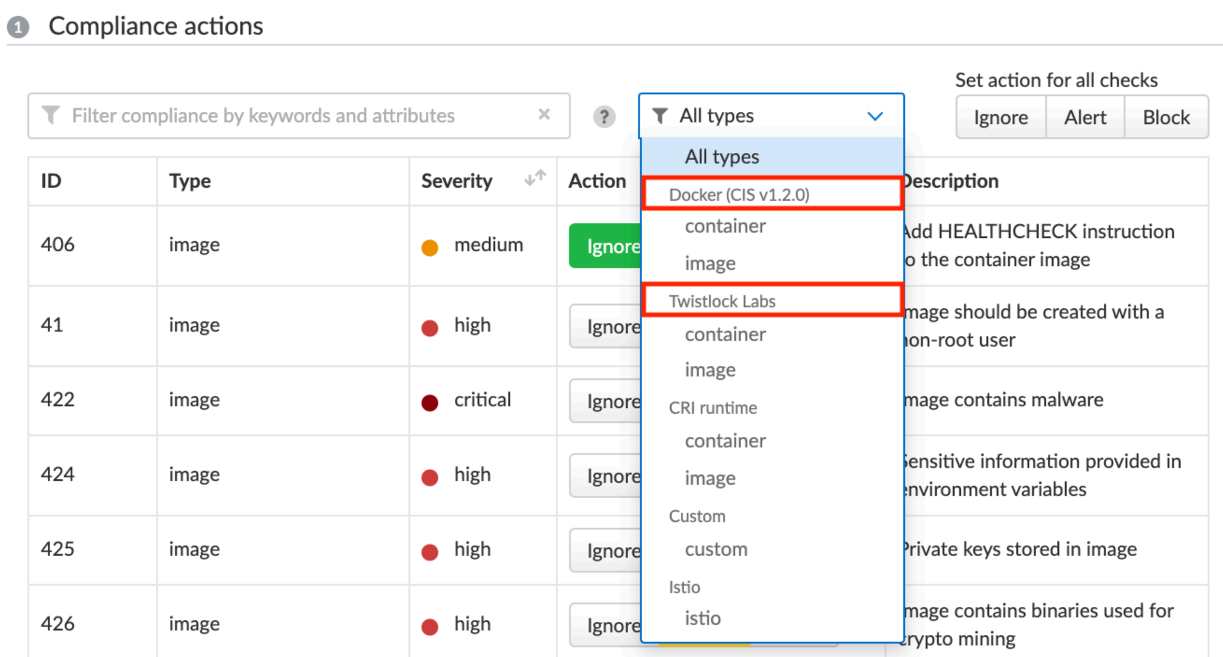
As you explore the built-in checks, consider the following points.

CIS

Most built-in checks are based on the CIS benchmarks. For full details about what a check does, and why, refer to the CIS benchmark documentation. Prisma Cloud check IDs map to CIS benchmark IDs. For example, Prisma Cloud check ID 51 maps to CIS Docker Benchmark 5.1

Check IDs

When creating compliance rules, there's a drop-down menu that lets you filter checks by type. Each type has a heading, which indicates the origin of the checks. Checks from the CIS benchmarks are clearly labeled.



CRI checks

All CRI checks are direct analogs of the Docker CIS checks, but repurposed for CRI environments.

Twistlock Labs checks

For all other checks, including those from Twistlock Labs, we provide documentation.

- Twistlock Labs compliance checks for containers, images, Istio, and Linux hosts.
- Twistlock Labs compliance checks for serverless functions.
- Twistlock Labs compliance checks for Windows.

Creating compliance rules

This procedure shows you how to set up a container compliance rule to block any containers running as root.

STEP 1 | Open Console, then go to **Defend > Compliance > Containers and Images**.

STEP 2 | Click **Add rule**.

1. Enter a rule name, such as **my-rule**.
2. In the search field under **Compliance actions**, enter **Container is running as root**.

   As you type, the available checks are filtered to match your search query.
3. For check 599 (Container is running as root), set the action to **Block**.
4. In **Scope**, accept the default collection, **All**. The default collection applies the rule to all containers in your environment.
5. Click **Save**.

   Your rule is now activated.

STEP 3 | Verify that your rule is being enforced.

1. Connect to a host running Defender, then run the following command, which starts an Ubuntu container with a root user (uid 0).

```
$ docker run -u 0 -ti library/ubuntu /bin/sh
```

   Defender should block the command with the following message:

```
docker: Error response from daemon: oci runtime error: [Prisma
 Cloud] Container operation blocked by policy: my-rule, has 1
 compliance issues.
```

Reporting full results

By default, Prisma Cloud reports only the compliance checks that fail. Sometimes you need both negative and affirmative results to prove compliance. You can configure Prisma Cloud to report checks that both pass and fail.

The contents of a full compliance report (both passed and failed checks) is the sum of all applied rules. If your compliance policy raises an alert for only two checks, your compliance report will show the results of two checks. To report on *all* compliance checks, set all compliance checks to either alert or block.

STEP 1 | Open Console, then go to **Defend > Compliance > {Containers and Images | Hosts}**.

STEP 2 | Click **Add rule**.

1. Enter a rule name.
2. Under **Reported results**, click **Passed and Failed Checks**.
3. Click **Save**.

   Your rule is now activated.

STEP 3 | Verify that the compliance reports show both passed and failed checks.

1. Go to **Defend > Compliance**, select any tab, then click on a resource in the table to open its scan report. You will see a list of checks that have both passed and failed.

Vulnerabilities | Compliance | Layers | Process Info | Package Info | Hosts | Labels

Id	Type	Severity	Result	Description
41	CIS	high	Fail	(CIS_Docker_CE_v1.1.0 - 4.1) Image should be created with a user Show details
406	CIS	medium	Fail	(CIS_Docker_CE_v1.1.0 - 4.6) Add HEALTHCHECK instruction to the container image Show details
422	twistlock	critical	Pass	Image contains malware
426	twistlock	high	Pass	Image contains binaries used for crypto mining
425	twistlock	high	Pass	Private keys stored in image
424	twistlock	high	Pass	Sensitive information provided in environment variables
423	twistlock	high	Pass	Image is not trusted
420	twistlock	medium	Pass	Image is not updated to latest

CIS Benchmarks

Edit on GitHub

The CIS Benchmarks provide consensus-oriented best practices for securely configuring systems. Prisma Cloud provides checks that validate the recommendations in the following CIS Benchmarks:

- Docker Benchmark
- Kubernetes Benchmark
- Openshift Benchmark
- Distribution Independent Linux
- Amazon Web Services Foundations

We have graded each check using a system of four possible scores: critical, high, medium, and low. This scoring system lets you create compliance rules that take action depending on the severity of the violation. If you want to be reasonably certain that your environment is secure, you should address all critical and high checks. By default, all critical and high checks are set to alert, and all medium and low checks are set to ignore. We expect customers to review, but probably never fix, medium and low checks.

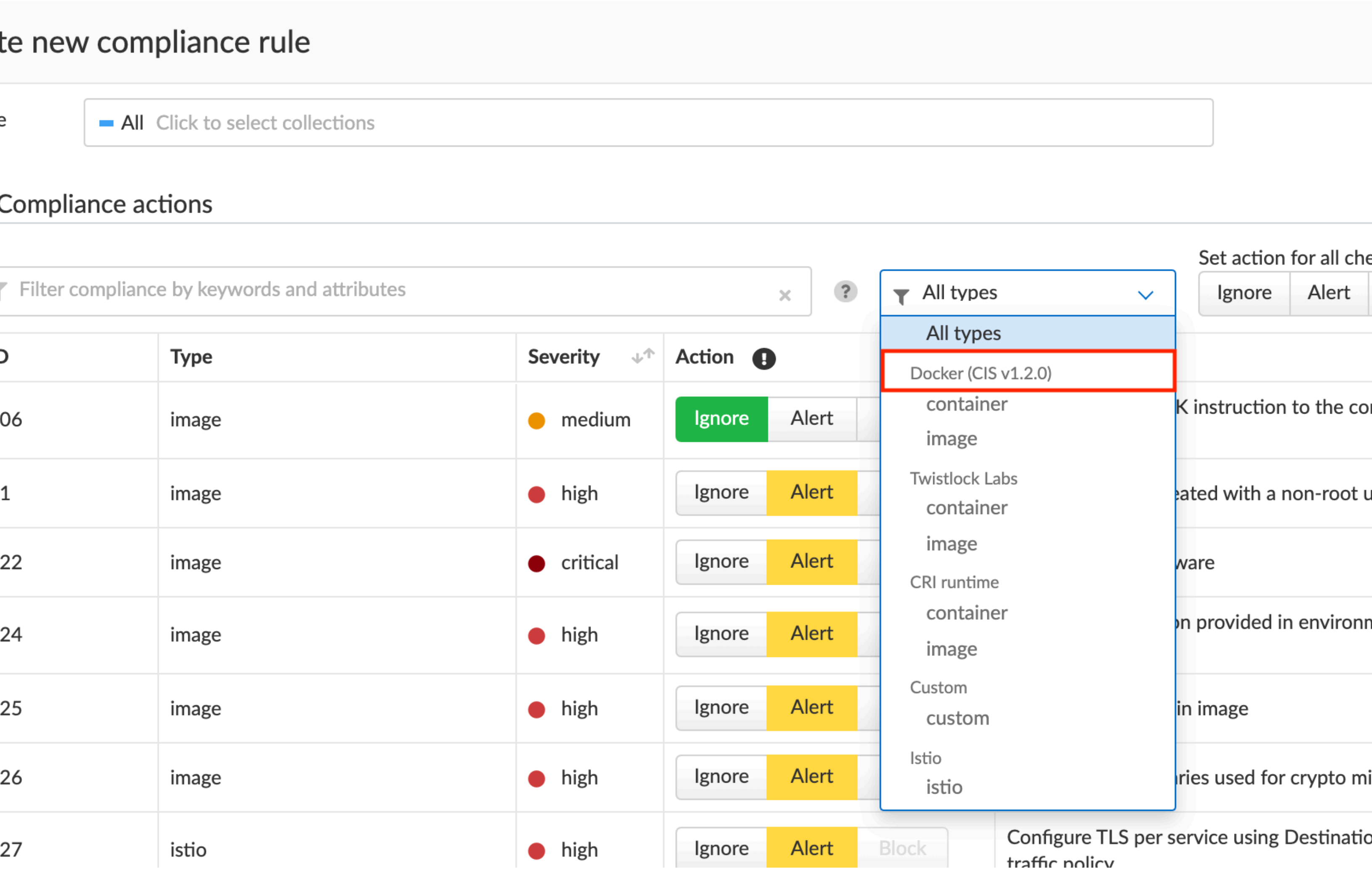
There are just a handful of checks graded as critical. Critical is reserved for things where your container environment is exposed to the Internet, and can result in a direct attack by somebody on the outside. They should be addressed immediately.

Prisma Cloud has not implemented CIS checks marked as *Not Scored*. These checks are are hard to define in a strict way. Other checks are might not implemented because the logic is resource-heavy, results depend on user input, or files cannot be parsed reliably.

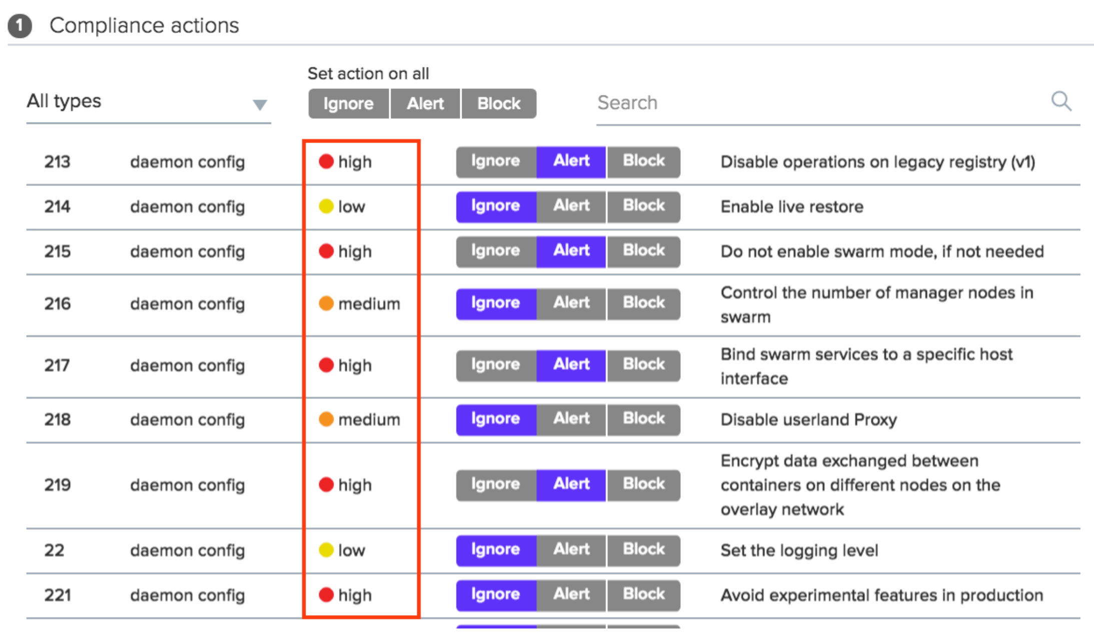
Additional details about Prisma Cloud's implementation of the CIS benchmarks

The compliance rule dialog provides some useful information. Compliance rules for containers can be created under **Defend > Compliance > Containers and Images**, while compliance rules for hosts can be created under **Defend > Compliance > Hosts**.

Benchmark versions—To see which version of the CIS benchmark is supported in the product, click on the **All types** drop-down list.



Grades — To see Prisma Cloud's grade for a check, see the corresponding **Severity** column.



Built-in policy library—To enable the checks for the PCI DSS, HIPAA, NIST SP 800-190, and GDPR standards, select the appropriate template.

Notes on the CIS OpenShift benchmark

When Prisma Cloud detects OpenShift Container Platform (OCP) 4, we assess the cluster against the CIS OpenShift benchmark. Prisma Cloud supports the CIS OpenShift benchmark on OCP 4.6 and later.

The following checks from the CIS OpenShift benchmark haven't been implemented:

- 1.2.7 - Ensure that the --authorization-mode argument is not set to AlwaysAllow.
- 1.2.10 - Ensure that the APIPriorityAndFairness feature gate is enabled.
- 1.2.11 - Ensure that the admission control plugin AlwaysAdmit is not set.
- 1.2.16 - Ensure that the admission control plugin SecurityContextConstraint is set.
- 1.2.21 - Ensure that the healthz endpoint is protected by RBAC.
- 1.2.23 - Ensure that the audit logs are forwarded off the cluster for retention.
- 1.2.33 - Ensure that the --encryption-provider-config argument is set as appropriate.
- 1.2.34 - Ensure that encryption providers are appropriately configured.
- 1.2.35 - Ensure that the API Server only makes use of Strong Cryptographic Ciphers.
- 1.3.1 - Ensure that garbage collection is configured as appropriate.
- 1.3.2 - Ensure that controller manager healthz endpoints are protected by RBAC.
- 1.4.1 - Ensure that the healthz endpoints for the scheduler are protected by RBAC.
- 1.4.2 - Verify that the scheduler API service is protected by authentication and authorization.
- 3.1.1 - Client certificate authentication should not be used for users.
- 3.2.2 - Ensure that the audit policy covers key security concerns.
- 4.2.2 - Ensure that the --authorization-mode argument is not set to AlwaysAllow.
- 4.2.7 - Ensure that the --make-iptables-util-chains argument is set to true.
- 4.2.8 - Ensure that the --hostname-override argument is not set.
- 4.2.9 - Ensure that the kubeAPIQPS [--event-qps] argument is set to 0 or a level which ensures appropriate event capture.
- 4.2.13 - Ensure that the Kubelet only makes use of Strong Cryptographic Ciphers.
- Section 5 - Policies.

Notes on the CIS Distribution Independent Linux benchmark

Prisma Cloud hasn't implemented the following checks from the CIS Distribution Independent Linux benchmark:

- *1.7.2 - Ensure GDM login banner is configured*—By default, most server distributions ship without a windows manager. A manual assessment is required.

- *2.2.1.2 - Ensure ntp (Network Time Protocol) is configured*—CIS did not score this recommendation. A manual assessment is required.
- *2.2.1.3 - Ensure chrony is configured*—CIS did not score this recommendation. A manual assessment is required.
- *5.3.1 - Ensure password creation requirements are configured*—This recommendation cannot be implemented generically because password requirements vary from organization to organization. A manual assessment is required.

Prisma Cloud Labs compliance checks

Edit on GitHub

Prisma Cloud Labs compliance checks are designed by our research team and fill gaps not offered by other benchmarks. Like all compliance checks, Prisma Cloud's supplementary checks monitor and enforce a baseline configuration across your environment.

Prisma Cloud Labs compliance checks can be enabled or disabled in custom rules. New rules can be created under **Defend > Compliance > Policy**.

Container checks

- **596 — Potentially dangerous NET_RAW capability enabled** --

  Checks if a running container has the NET_RAW capability enabled. This capability grants an application the ability to craft raw packets. In the hands of an attacker, NET_RAW can enable a wide variety of networking exploits, such as ARP-spoofing and hijacking a cluster's DNS traffic.

- **597 — Secrets in clear text environment variables (container and serverless function check)** --

  Checks if a running container (instantiated from an image) or serverless function contains sensitive information in its environment variables. These env vars can be easily exposed with docker inspect, and thus compromise privacy.

- **598 — Container app is running with weak settings** --

  Weak settings incidents indicate that a well-known service is running with a non-optimal configuration. This covers settings for common applications, specifically: Mongo, Postgres, Wordpress, Redis, Kibana, Elasitc Search, RabbitMQ, Tomcat, Haproxy, KubeProxy, Httpd, Nginx, MySql, and registries. These check for things such as the use of default passwords, requiring SSL, etc. The output for a failed compliance check will contain a "Cause" field that gives specifics on the exact settings detected that caused a failure.

- **599 — Container is running as root (container check)** --

  Checks if the user value in the container configuration is root. If the user value is 0, root, or "" (empty string), the container is running as a root user, and the policy's configured effect (ignore, alert, or block) is actuated.

Container image checks

- **422 — Image contains malware (image check)** --

  Checks if any binary in the image matches the md5 checksum for known malicous software.

- **423 — Image is not trusted (image check)** --

  Checks if unauthorized (untrusted) images are pulled or loaded into your environment.

  Prisma Cloud provides a mechanism to specify specific registries, repositories, and images that are considered trusted. Enable this check to prevent unauthorized containers from running in your critical environment. For more information, see Trusted images.

- **424 — Sensitive information provided in environment variables (image and serverless function check)** --

  Checks if images or serverless functions contain sensitive information in their environment variables. Container images define environment variables with the Dockerfile ENV instruction. These environment variables can be easily exposed with *docker inspect*.

- **425 — Private keys stored in image (image and serverless function check)** --

  Searches for private keys stored in an image or serverless function. If found, the policy effect (ignore, alert, block) is applied on deployment.

- **426 — Image contains binaries used for crypto mining (image check)** --

  Detects when there are crypto miners in an image. Attackers have been quietly poisoning registries and injecting crypto mining tools into otherwise legitimate images. When you run these images, they perform their intended function, but also mine Bitcoin for the attacker. This check is based on research from Prisma Cloud Labs. For more information, see Real World Security: Software Supply Chain.

- **448 — Package binaries should not be altered** --

  Checks the integrity of package binaries in an image. During an image scan, every binary's checksum is compared with its package info. If there's a mismatch, a compliance issue is raised.

  Besides scan-time, this compliance issue an also be raised at run-time if a modified binary is spawned.

Prisma Cloud Labs Istio compliance checks

Istio compliance checks help enforce a secure Istio configuration. They address risks such as misconfigured TLS settings and universally scoped service roles.

The goals of the compliance rules are to:

- Ensure mutual TLS is configured correctly (enabled and over HTTPs).
- Ensure RBAC policy is configured with service level access control (service x can only talk with service y).
- Ensure RBAC policy is not too permissive.

Istio checks

427 — Configure TLS per service using Destination Rule traffic policy

450 — Enable mesh-wide mutual TLS authentication using Peer Authentication Policy

451 — Avoid using permissive authorization policies without rules as it can compromise the target services

452 — Enable Istio access control on all workloads in the mesh using Authorization Policies

In Istio versions 1.6 and later, Mesh Policy is deprecated and replaced with Peer Authentication Policy.

Linux host checks

449 — Ensure no pending OS security updates

On each host scan, Prisma Cloud checks for available package updates marked as security updates. If such updates are found, they're listed under the security updates tab in **Monitor > Runtime > Host observations > <HOST>** Supported for Ubuntu and Debian hosts only.

Serverless functions compliance checks

Edit on GitHub

Prisma Cloud Labs has developed compliance checks for serverless functions. Currently, only AWS Lambda is supported.

In AWS Lambda, every function has an execution role. Execution roles are identities with permission policies that control what functions can and cannot do in AWS. When you create a function, you specify an execution role. When the function is invoked, it assumes this role.

When Prisma Cloud scans the functions in your environment, it inspects the execution role for overly permissive access to AWS services and resources. Two fields are inspected: *resource* and *action*.

Resource

Specifies the objects to which the permission policy applies. Resources are specified with ARNs. ARNs let you unambiguously specify a resource across all of AWS. ARNs have the following format:

```
arn:partition:service:region:account-id:resource
```

Where:

- *service* — Identifies the AWS product, such as Amazon S3, IAM, or CloudWatch Logs.
- *resource* — Identies the objects in the service. It often includes the resource type, followed by the resource name itself. For example, the following ARN uniquely identifies the user Francis in the IAM service:

  ```
  arn:aws:iam::586975633310:user/Francis
  ```

Action

Describes the tasks that can be performed on the service. For example, ec2:StartInstances, iam:ChangePassword, and s3:GetObject. Wildcards can be used to grant access to all the actions of a given AWS service. For example, s3:* applies to all S3 actions.

Types of issues

The following permission policy is tightly scoped. It grants read-write only access to the Books table. Prisma Cloud would not flag an execution role with this type of permissions policy.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": [
      "dynamodb:GetItem",
      "dynamodb:BatchGetItem"
    ],
    "Resource": "arn:aws:dynamodb:us-east-1:125643784111:table/Books"
```



```
  }
}
```

The following permissions policy has been implemented carelessly. It allows all DyanmoDB operations on all tables owned by the AWS account in the current region, including dynamodb:DeleteTable, which has serious implications for the integrity and availability of your data. This type of configuration would raise compliance check 437 because the execution role permits all DyanmoDB operations, and it's unlikely a function actually needs this range of capabilities.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Sid": "AllAPIActionsOnBooks",
    "Effect": "Allow",
    "Action": "dynamodb:*",
    "Resource": "*"
  }
}
```

Compliance check details

The following checks are supported:

- **434: Sensitive information provided in environment variables** --

  Detects when functions contain environment variables (such as MYSQL_PASSWORD) that expose sensitive information.
- **435: Private keys stored in function** --

  Detects private keys in functions.
- **436: Unbounded service access** --

  Detects functions with permission to run all actions on all services and their resources.
- **437: Overly permissive service access** --

  Detects functions with permission to run all actions on one or more services.
- **438: Broad resource access** --

  Detects functions that granted access to all resources in one or more services.
- **439: Suspicious function actions** --

  Detects functions with permission to run actions that are used in exploits and attacks. Includes things like cloudtrail:StopLogging, cloudtrail:UpdateTrail that allow disabling and changing the output of CloudTrail logging.
- **440: Unused service API with information disclosure risk** --

  Detects functions with permissions to unused APIs that could allow information disclosure.
- **441: Unused service API with data leakage risk** --

  Detects functions with permissions to unused APIs that could leak data.

- **442: Unused service API with data tampering risk** --

  Detects functions with permissions to unused APIs that could allow data tampering.
- **443: Unused service API with lateral movement risk** --

  Detects functions with permissions to unused APIs that could allow an attacker to move laterally.
- **444: Unused service API with denial of service risk** --

  Detects functions with permissions to unused APIs that could facilitate a denial of service attack.
- **445: Unused service API with information exfiltration risk** --

  Detects functions with permissions to unused APIs that could allow data exfiltration.
- **446: Unused service API with persistent access risk** --

  Detects functions with permissions to unused APIs that allow persistent access.
- **447: Unused service API with privilege elevation risk** --

  Detects functions with permissions to unused APIs that allow privilege elevation.

Scanning serverless functions

Configure Prisma Cloud to periodically scan your serverless functions. Function scanning is handled by Console.

STEP 1 | Open Console.

STEP 2 | Go to **Defend > Compliance > Functions**.

STEP 3 | Click on **Add scope**. In the dialog, enter the following settings:

1. Specify a cap for the number of functions to scan.

   Prisma Cloud scans the X most recent functions, where X is the cap value. Set this value to '0' to scan all functions.
2. (AWS only) Specify which regions to scan. By default, the scope is applied to **Regular regions**. Other options include **China regions** or **Goverment regins**.
3. (AWS only) Select **Scan only latest versions** to only scan the latest version of each function. Otherwise, the scanning will cover all versions of each function up to the specified **cap** value.
4. (AWS only) Select **Scan Lambda Layers** to enable scanning the function's Layers as well.
5. Select the accounts to scan by credential. If you wish to add an account, click on **Add credential**.
6. Click **Add**.

STEP 4 | Verify that you have assigned the correct permissions required to scan.

STEP 5 | To view the scan report, go to **Monitor > Compliance > Functions**.

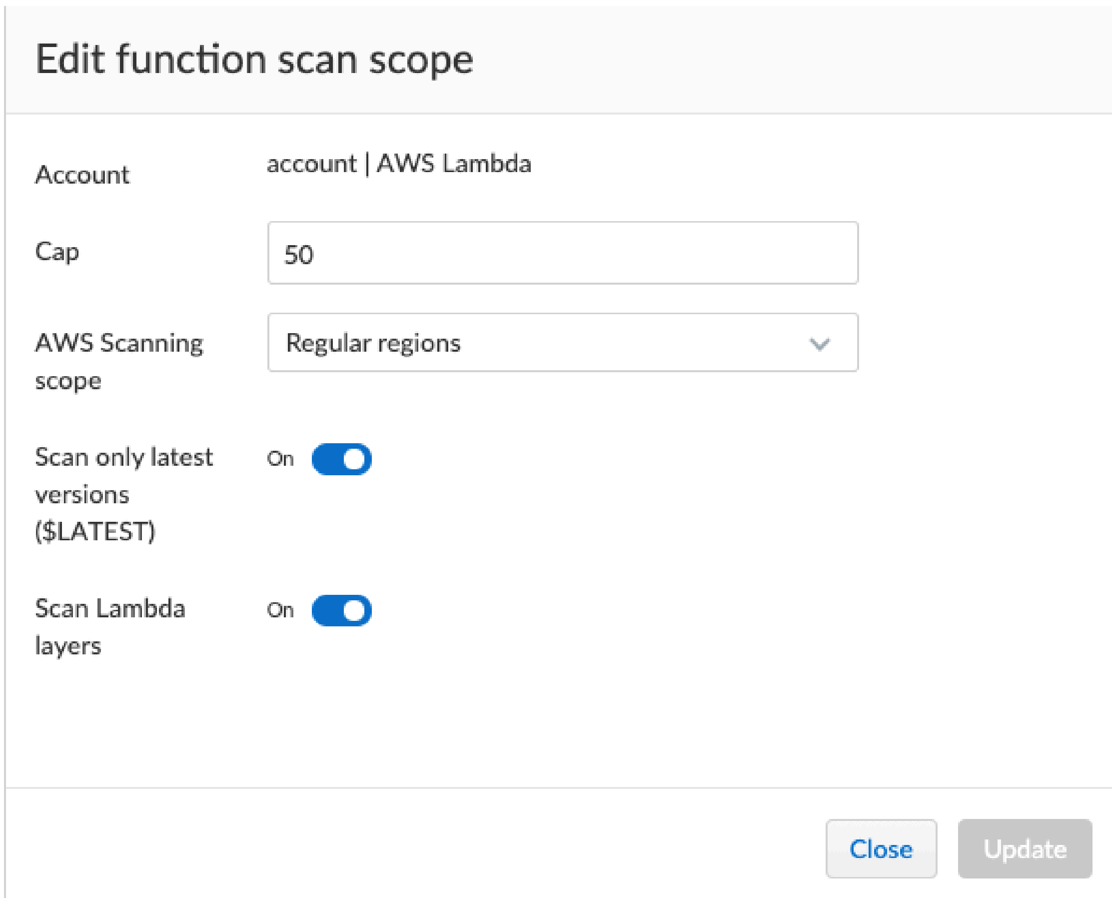
All compliance issues identified in the latest serverless scan report can be exported to a CSV file by clicking on the CSV button in the top right of the table.

View AWS Lambda Layers scan report

Prisma Cloud can scan the AWS Lambda Layers code as part of the Lambda function's code scanning. This capability can help you determine whether the Compliance checks are associated with the function or function Layers. Follow the steps below to view the Lambda Layers compliance scan results:

STEP 1 | Open Console.

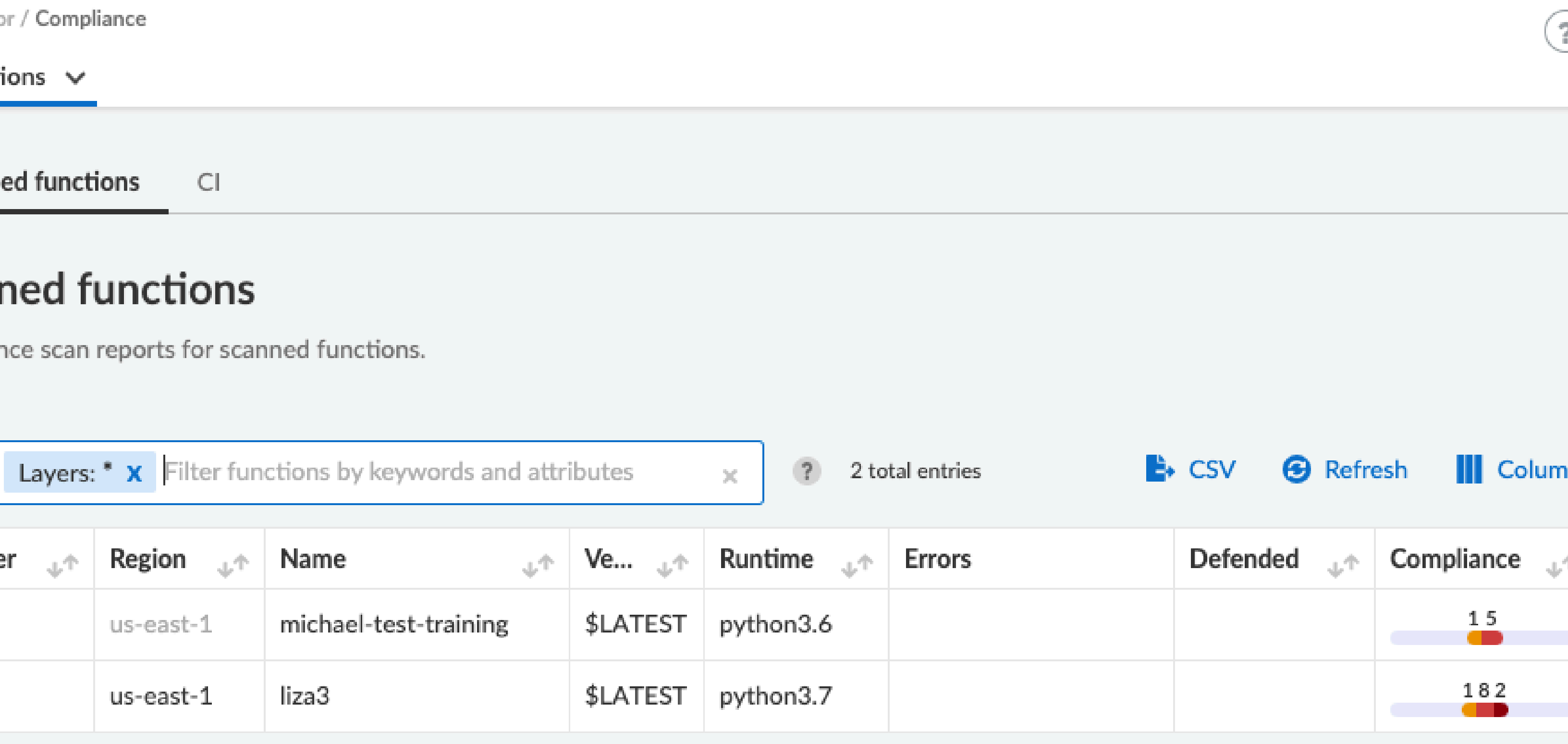
STEP 2 | Make sure you selected the **Scan Lambda layers** in the Defend > Compliance > Functions > Functions > Serverless Accounts > **Function scan scope**



STEP 3 | Go to **Monitor > Compliance > Functions > Scanned functions**.

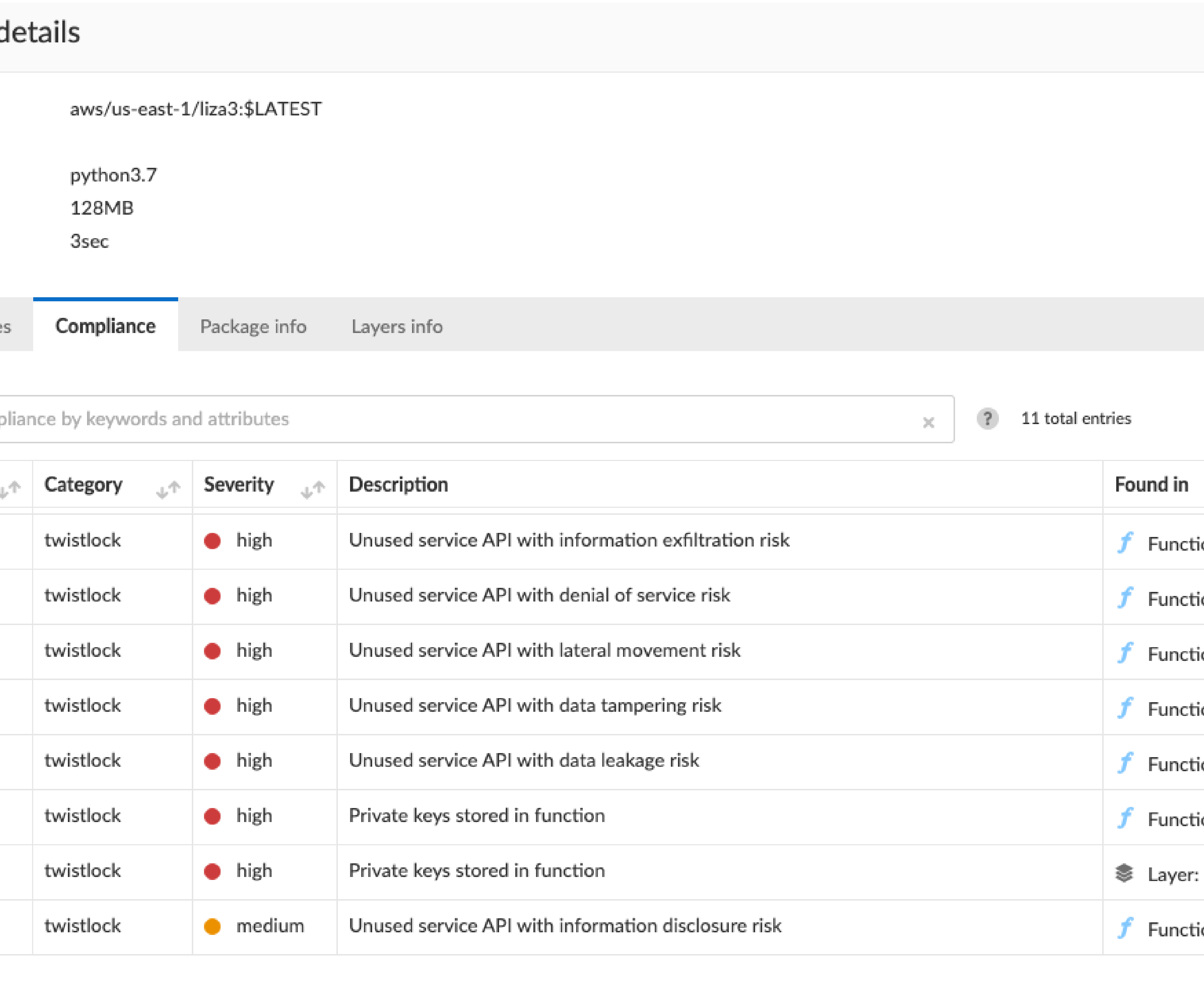
STEP 4 | Filter the table to include functions with the desired Layer by adding the **Layers** filter.

You can also filter the results by a specific layer name or postfix wildcards. Example: *Layers:* OR Layers:arn:aws:lambda:**

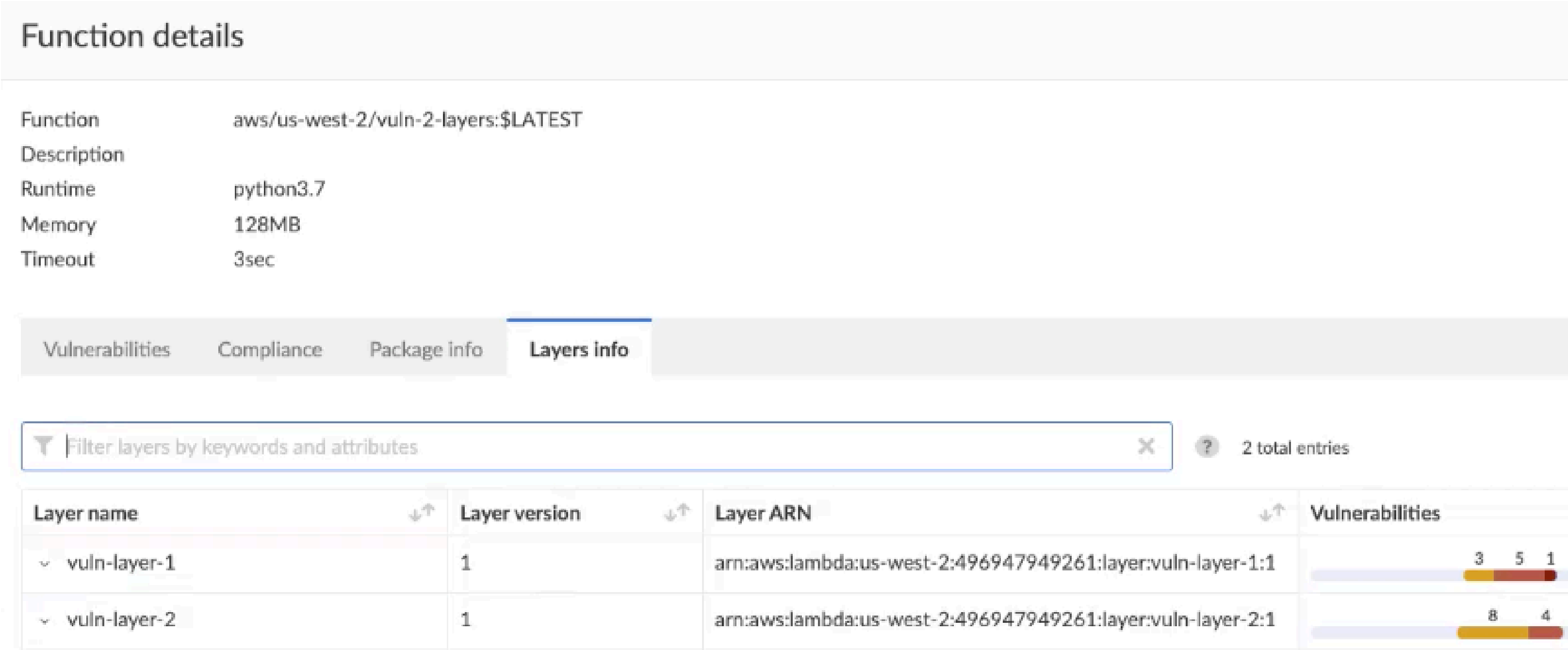


STEP 5 | Open the **Function details** dialog to view the details about the Layers and the Compliance issues associated with them:

1. Click on a specific function
2. See the Function's vulnerabilities, compliance issues and package info in the related tabs. Use the **Found in** column to determine if the component is associated with the Function or with the Function's Layers.



3. Use the **Layers info** tab to see the full list of the function's the Layers.

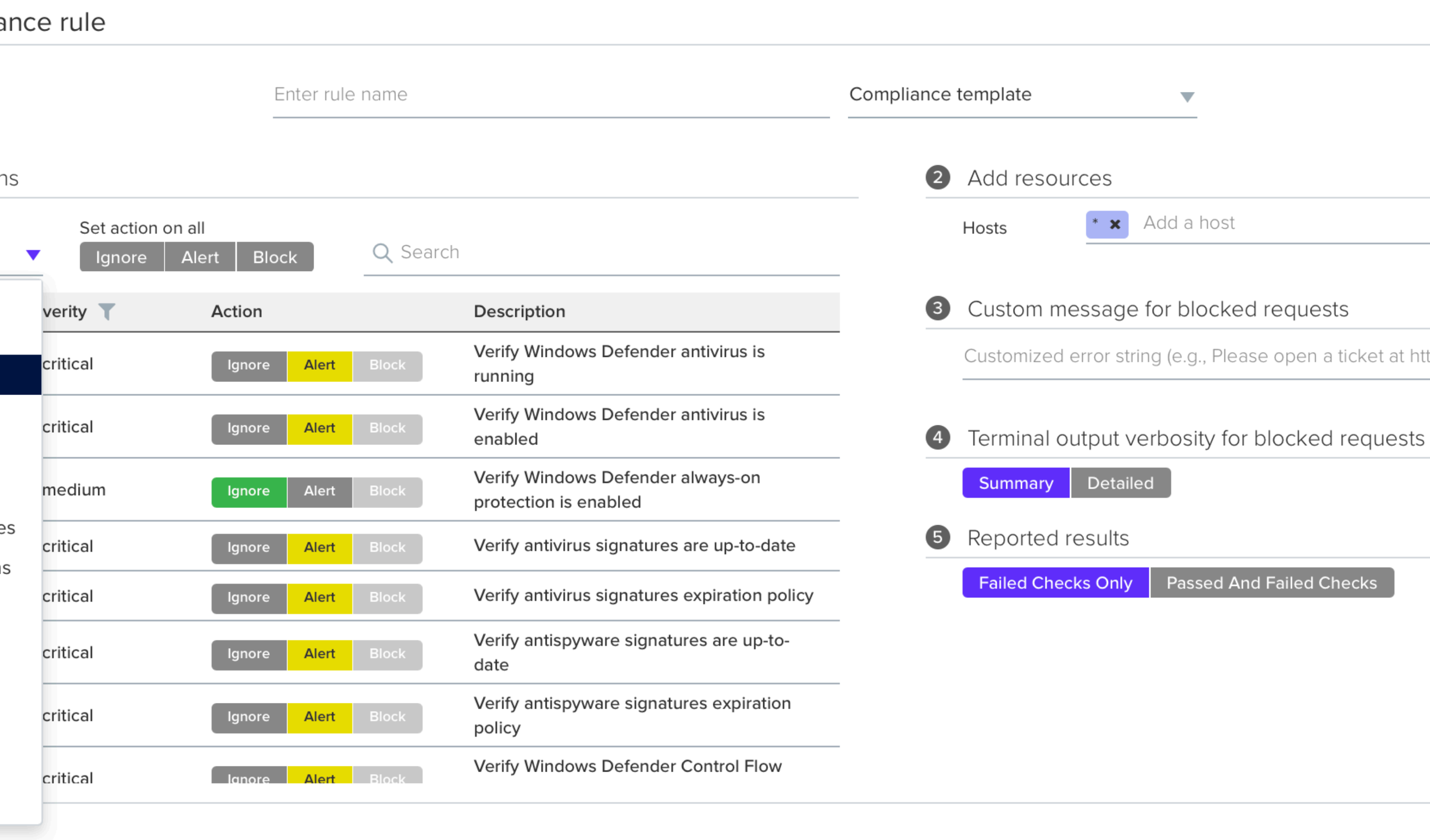

Function details
Function
aws/us-west-2/vuln-2-layers:$LATEST
Description
Runtime
python3.7
Memory
128MB
Timeout
3sec
Vulnerabilities
Compliance
Package info
Layers info
Filter layers by keywords and attributes
2 total entries
Layer name
Layer version
Layer ARN
Vulnerabilities
vuln-layer-1
1
arn:aws:lambda:us-west-2:496947949261:layer:vuln-layer-1:1
3 5 1
vuln-layer-2
1
arn:aws:lambda:us-west-2:496947949261:layer:vuln-layer-2:1
8 4

Windows compliance checks

Edit on GitHub

Windows compliance checks were developed by Prisma Cloud Labs. They can be enabled in your host compliance policy.

Create Windows host compliance rules in **Defend > Compliance > Hosts**. In the new rule dialog, select **Windows host** from the **Types** drop-down list.



The following checks are supported:

- **200001: Verify Windows Defender antivirus is running** --

  Microsoft's built in Windows Defender antivirus service is running.
- **200002: Verify Windows Defender antivirus is enabled** --

  Microsoft's built in Windows Defender service antivirus, anti-malware, and anti-spyware features are enabled

- **200003: Verify Windows Defender always-on protection is enabled** --

  Always-on protection consists of real-time protection, behavior monitoring, and heuristics to identify malware based on known suspicious and malicious activities.
- **200004: Verify antivirus signatures match defined frequency** --

  Windows antivirus signatures are overdue based on your frequency policy.
- **200005: Verify antivirus signatures are up-to-date** --

  Windows antivirus signatures list must be updated within the last 14 days. If 14 days elapse without an update, signatures are stale. This interval is required to be effective against current threats.
- **200006: Verify anti-spyware signatures match defined frequency** --

  Windows anti-spyware signatures are overdue based on your frequency policy.
- **200007: Verify anti-spyware signatures are up-to-date** --

  Windows anti-spyware signatures list must be updated within the last 14 days. If 14 days elapse without an update, signatures are stale. This interval is required to be effective against current threats.
- **200201: Verify Windows Defender Control Flow Guard (CFG) is enabled** --

  Control Flow Guard (CFG) is a platform security feature that combats memory corruption vulnerabilities. By placing tight restrictions on where an application can execute code, CFS makes it harder for exploits to execute arbitrary code through vulnerabilities, such as buffer overflows. This check is applicable to Windows Server 2019 only.
- **200202: Verify Windows Defender Data Execution Prevention (DEP) is enabled** --

  Data Execution Prevention (DEP) monitors memory to stop malicious code from running. It monitors all processes and services and stops a program if it isn't running correctly in memory. This check is applicable to Windows Server 2019 only.
- **200203: Verify Windows Defender Address Space Layout Randomization (ASLR) is enabled** --

  Address space layout randomization (ASLR) prevents exploitation of memory corruption vulnerabilities. It prevents an attacker from reliably jumping to an exploited function in memory by randomly arranging the position (address) of the stack, heap, and loaded libararies. This check is applicable to Windows Server 2019 only.
- **200300: Verify Windows Firewall public profile is enabled** --

  This setting is applied when a connection to a domain is made through a public network, such as at an airport, hotel, or coffee shop. Since the security of these networks is unknown and not really controlled by the user running the computer, it is suggested that the Public network profile of settings be more restrictive than either the Domain network or Private network.
- **200400: Verify Windows Update is enabled** --

  Windows Update is a service which automates downloading and installing Microsoft Windows software updates.
- **200401: Verify Windows Update is set to automatically install** --

  Verify that Windows is configured to automatically download and install updates at a regular interval.

- *If Windows Defender antivirus is not installed or running, all Windows Defender related checks (200001, 200002, 200003, 200201, 200202, 200203) fail with the following cause: "Windows Defender antivirus service is not installed/running".*
- *Although checks 200004/5 and 200006/7 look similar, they clarify the root cause of the issue when assessed separately. Checks 200004/6 verify the update frequency policy, while 200005/7 verify that signatures are actually up-to-date. Checks 200004/6 show whether the defined frequency is suboptimal (greater than 14 days), while checks 200005/7 show if there was a failure to update the signatures according to the defined policy (whether it's 14 days or some other interval).*
- *If no definition files (signatures) are available, checks 200004 and 200006 fail with the following cause: "Windows Defender definition files are not available". Definitions can be removed with the following command:*

```
"%ProgramFiles%\Windows Defender\MpCmdRun.exe" -
removedefinitions
```


DISA STIG compliance checks

Edit on GitHub

Prisma Cloud supports the Docker Enterprise 2.x Linux/Unix STIG - Ver 2, Rel 1 and the Kubernetes STIG - Ver 1, Rel 2 compliance checks. Defense Information Systems Agency Security Technical Implementation Guides (DISA STIGs) contain technical guidance to lock down systems that might otherwise be vulnerable to attack. These STIGs help ensure your environments are properly secured, based on Department of Defense guidance. Prisma Cloud will continue to incorportate DISA STIG guidance as existing STIGs are updated and new STIGs are published.

For an overview of the STIG, see here.

To download the STIGs, see here.

Checks

Prisma Cloud Compute has a compliance template "DISA STIG" for images, containers and hosts. This compliance template maps individual STIG rules to existing compliance checks within Compute. In some cases, we've implemented checks specficially to support the STIGs. When configuring your compliance policy, simply select the DISA STIG template to enable ("Alert") all relevant checks.

CAT I

CAT I is a category code for any vulnerability, which when exploited, will *directly and immediately* result in loss of Confidentiality, Availability, or Integrity. These risks are the most severe.

The following table lists the CAT I checks implemented in Prisma Cloud, and how they map to existing Prisma Cloud checks. All CAT I checks, except DKER-EE-001070, map to CIS Docker Benchmark checks. A separate check has been implemented for DKER-EE-001070 to support the Docker Enterprise STIG.

STIG ID	Prisma Cloud ID	Description
DKER-EE-001070	N/A	FIPS mode must be enabled on all Docker Engine - Enterprise nodes.
DKER-EE-002000	59	Docker Enterprise hosts network namespace must not be shared.
DKER-EE-002030	512	All Docker Enterprise containers root filesystem must be mounted as read only.
DKER-EE-002040	517	Docker Enterprise host devices must not be directly exposed to containers.
DKER-EE-002070	521	The Docker Enterprise default seccomp profile must not be disabled.

STIG ID	Prisma Cloud ID	Description
DKER-EE-002080	224	Docker Enterprise exec commands must not be used with privileged option.
DKER-EE-002110	525	All Docker Enterprise containers must be restricted from acquiring additional privileges.
DKER-EE-002120	530	The Docker Enterprise hosts user namespace must not be shared.
DKER-EE-002130	531	The Docker Enterprise socket must not be mounted inside any containers.
DKER-EE-002150	57	Docker Enterprise privileged ports must not be mapped within containers.
DKER-EE-005170	31	Docker Enterprise docker.service file ownership must be set to root:root.
DKER-EE-005190	33	Docker Enterprise docker.socket file ownership must be set to root:root.
DKER-EE-005210	35	Docker Enterprise /etc/docker directory ownership must be set to root:root.
DKER-EE-005230	37	Docker Enterprise registry certificate file ownership must be set to root:root.
DKER-EE-005250	39	Docker TLS certificate authority (CA) certificate file ownership must be set to root:root
DKER-EE-005270	311	Docker server certificate file ownership must be set to root:root
DKER-EE-005300	314	Docker server certificate key file permissions must be set to 400
DKER-EE-005310	315	Docker Enterprise socket file ownership must be set to root:docker.
DKER-EE-005320	316	Docker Enterprise socket file permissions must be set to 660 or more restrictive.
DKER-EE-005330	317	Docker Enterprise daemon.json file ownership must be set to root:root.

STIG ID	Prisma Cloud ID	Description
DKER-EE-005340	318	Docker Enterprise daemon.json file permissions must be set to 644 or more restrictive.
DKER-EE-005350	319	Docker Enterprise /etc/default/docker file ownership must be set to root:root.
DKER-EE-005360	320	Docker Enterprise /etc/default/docker file permissions must be set to 644 or more restrictive.
CNTR-K8-000220	8134	The Kubernetes Controller Manager must create unique service accounts for each work payload.
CNTR-K8-000320	8117	The Kubernetes API server must have the insecure port flag disabled.
CNTR-K8-000330	8215	The Kubernetes Kubelet must have the read-only port flag disabled.
CNTR-K8-000340	8116	The Kubernetes API server must have the insecure bind address not set.
CNTR-K8-000360	8112	The Kubernetes API server must have anonymous authentication disabled.
CNTR-K8-000370	8212	The Kubernetes Kubelet must have anonymous authentication disabled.
CNTR-K8-000380	8213	The Kubernetes kubelet must enable explicit authorization.
CNTR-K8-001160	597	Secrets in Kubernetes must not be stored as environment variables.
CNTR-K8-001620	8217	Kubernetes Kubelet must enable kernel protection.
CNTR-K8-001990	81120	Kubernetes must prevent non-privileged users from executing privileged functions to include disabling, circumventing, or altering implemented security safeguards/countermeasures or the installation of patches and updates.
CNTR-K8-002010	81125	Kubernetes must have a pod security policy set.
CNTR-K8-002620	8113	Kubernetes API Server must disable basic authentication to protect information in transit.

CAT II

CAT II is a category code for any vulnerability, which when exploited, *has a potential* to result in loss of Confidentiality, Availability, or Integrity.

The following table lists the CAT 1 checks implemented in Prisma Cloud, and how they map to existing checks. Some CAT 1 checks don't map to any existing checks, and have been implemented specifically for this DISA STIG.

STIG ID	Prisma Cloud ID	Description
DKER-EE-001050	26	TCP socket binding for all Docker Engine - Enterprise nodes in a Universal Control Plane (UCP) cluster must be disabled.
DKER-EE-001240	515	The Docker Enterprise hosts process namespace must not be shared.
DKER-EE-001250	516	The Docker Enterprise hosts IPC namespace must not be shared.
DKER-EE-001800	24	The insecure registry capability in the Docker Engine - Enterprise component of Docker Enterprise must be disabled.
DKER-EE-001810	25	On Linux, a non-AUFS storage driver in the Docker Engine - Enterprise component of Docker Enterprise must be used.
DKER-EE-001830	218	The userland proxy capability in the Docker Engine - Enterprise component of Docker Enterprise must be disabled.
DKER-EE-001840	221	Experimental features in the Docker Engine - Enterprise component of Docker Enterprise must be disabled.
DKER-EE-001930	51	An appropriate AppArmor profile must be enabled on Ubuntu systems for Docker Enterprise.
DKER-EE-001940	52	SELinux security options must be set on Red Hat or CentOS systems for Docker Enterprise.
DKER-EE-001950	53	Linux Kernel capabilities must be restricted within containers as defined in the System Security Plan (SSP) for Docker Enterprise.
DKER-EE-001960	54	Privileged Linux containers must not be used for Docker Enterprise.
DKER-EE-001970	56	SSH must not run within Linux containers for Docker Enterprise.

STIG ID	Prisma Cloud ID	Description
DKER-EE-001990	58	Only required ports must be open on the containers in Docker Enterprise.
DKER-EE-002010	510	Memory usage for all containers must be limited in Docker Enterprise.
DKER-EE-002050	519	Mount propagation mode must not set to shared in Docker Enterprise.
DKER-EE-002060	520	The Docker Enterprise hosts UTS namespace must not be shared.
DKER-EE-002100	524	cgroup usage must be confirmed in Docker Enterprise.
DKER-EE-002160	513	Docker Enterprise incoming container traffic must be bound to a specific host interface.
DKER-EE-002400	223	Docker Enterprise Swarm manager must be run in auto-lock mode.
DKER-EE-002770	406	Docker Enterprise container health must be checked at runtime.
DKER-EE-002780	528	PIDs cgroup limits must be used in Docker Enterprise.
DKER-EE-003200	41	Docker Enterprise images must be built with the USER instruction to prevent containers from running as root.
DKER-EE-004030	514	The on-failure container restart policy must be is set to 5 in Docker Enterprise.
DKER-EE-004040	518	The Docker Enterprise default ulimit must not be overwritten at runtime unless approved in the System Security Plan (SSP).
DKER-EE-005180	32	Docker Enterprise docker.service file permissions must be set to 644 or more restrictive.
DKER-EE-005200	34	Docker Enterprise docker.socket file permissions must be set to 644 or more restrictive.
DKER-EE-005220	36	Docker Enterprise /etc/docker directory permissions must be set to 755 or more restrictive.

STIG ID	Prisma Cloud ID	Description
DKER-EE-005240	38	Docker Enterprise registry certificate file permissions must be set to 444 or more restrictive.
DKER-EE-005260	310	Docker TLS certificate authority (CA) certificate file permissions must be set to 444 or more restrictive
DKER-EE-005280	312	Docker server certificate file permissions must be set to 444 or more restrictive
DKER-EE-005290	313	Docker server certificate key file ownership must be set to root:root
DKER-EE-006270	217	Docker Enterprise Swarm services must be bound to a specific host interface.
CNTR-K8-000180	8153	The Kubernetes etcd must use TLS to protect the confidentiality of sensitive data during electronic dissemination (--auto-tls argument is not set to true).
CNTR-K8-000190	8156	The Kubernetes etcd must use TLS to protect the confidentiality of sensitive data during electronic dissemination. (--peer-auto-tls argument is not set to true).
CNTR-K8-000270	81141 & 81132	The Kubernetes API Server must enable Node,RBAC as the authorization mode.
CNTR-K8-000300	8122	The Kubernetes Scheduler must have secure binding.
CNTR-K8-000350	8118	The Kubernetes API server must have the secure port set.
CNTR-K8-000850	82110	Kubernetes Kubelet must deny hostname override.
CNTR-K8-000860	81418 & 8142 & 81424 & 81422	The manifest files contain the runtime configuration of the API server, proxy, scheduler, controller, and etcd. If an attacker can gain access to these files, changes can be made to open vulnerabilities and bypass user authorizations inherit within Kubernetes with RBAC implemented.
CNTR-K8-000910	8132	Kubernetes Controller Manager must disable profiling.
CNTR-K8-001400	605213	The Kubernetes API server must use approved cipher suites.

STIG ID	Prisma Cloud ID	Description
CNTR-K8-001410	81122	Kubernetes API Server must have the SSL Certificate Authority set.
CNTR-K8-001420	81130 & 8214	Kubernetes Kubelet must have the SSL Certificate Authority set.
CNTR-K8-001430	8136	Kubernetes Controller Manager must have the SSL Certificate Authority set.
CNTR-K8-001450	8152	Kubernetes etcd must enable client authentication to secure service.
CNTR-K8-001460	82112	Kubernetes Kubelet must enable tls-private-key-file for client authentication to secure service.
CNTR-K8-001480	8155	Kubernetes etcd must enable client authentication to secure service.
CNTR-K8-001490	81127	Kubernetes etcd must have a key file for secure communication.
CNTR-K8-001510	81131	Kubernetes etcd must have the SSL Certificate Authority set.
CNTR-K8-001550	8154	Kubernetes etcd must have a peer-key-file set for secure communication.
CNTR-K8-002600	81138	Kubernetes API Server must configure timeouts to limit attack surface.
CNTR-K8-003120	81412	The Kubernetes component etcd must be owned by etcd.
CNTR-K8-003130	81414 & 8145	The Kubernetes conf files must be owned by root.
CNTR-K8-003140	8231	The Kubernetes Kube Proxy must have file permissions set to 644 or more restrictive.
CNTR-K8-003150	8232	The Kubernetes Kube Proxy must be owned by root.
CNTR-K8-003160	8227	The Kubernetes Kubelet certificate authority file must have file permissions set to 644 or more restrictive.

STIG ID	Prisma Cloud ID	Description
CNTR-K8-003170	8228	The Kubernetes Kubelet certificate authority must be owned by root.
CNTR-K8-003180	81427	The Kubernetes component PKI must be owned by root.
CNTR-K8-003210	8230	The Kubernetes kubeadm.conf must be owned by root.
CNTR-K8-003220	8229	The Kubernetes kubeadm.conf must have file permissions set to 644 or more restrictive.
CNTR-K8-003230	8234	The Kubernetes kubelet config must have file permissions set to 644 or more restrictive.
CNTR-K8-003240	8233	The Kubernetes kubelet config must be owned by root.
CNTR-K8-003250	81419 & 81421 & 81423 & 81425	The Kubernetes API Server must have file permissions set to 644 or more restrictive.
CNTR-K8-003260	81411	The Kubernetes etcd must have file permissions set to 644 or more restrictive.
CNTR-K8-003270	81413	The Kubernetes admin.conf must have file permissions set to 644 or more restrictive.
CNTR-K8-003290	81119	The Kubernetes API Server must be set to audit log max size.
CNTR-K8-003290	81118	The Kubernetes API Server must be set to audit log maximum backup.
CNTR-K8-003310	81117	The Kubernetes API Server audit log retention must be set.
CNTR-K8-003320	81116	The Kubernetes API Server audit log path must be set.
CNTR-K8-003330	81428	The Kubernetes PKI CRT must have file permissions set to 644 or more restrictive.
CNTR-K8-003340	81429	The Kubernetes PKI keys must have file permissions set to 600 or more restrictive.

STIG ID	Prisma Cloud ID	Description
CNTR-K8-002630	81121	Kubernetes API Server must disable token authentication to protect information in transit.
CNTR-K8-002640	81123	Kubernetes endpoints must use approved organizational certificate and key pair to protect information in transit.

CAT III

CAT III is a category code for any vulnerability, which when it exists, *degrades measures* to protect against loss of Confidentiality, Availability, or Integrity.

The following table lists the CAT III checks implemented in Prisma Cloud, and how they map to existing Prisma Cloud checks. All checks map to CIS Docker Benchmark checks.

STIG ID	Prisma Cloud ID	Description
DKER-EE-002020	511	Docker Enterprise CPU priority must be set appropriately on all containers.

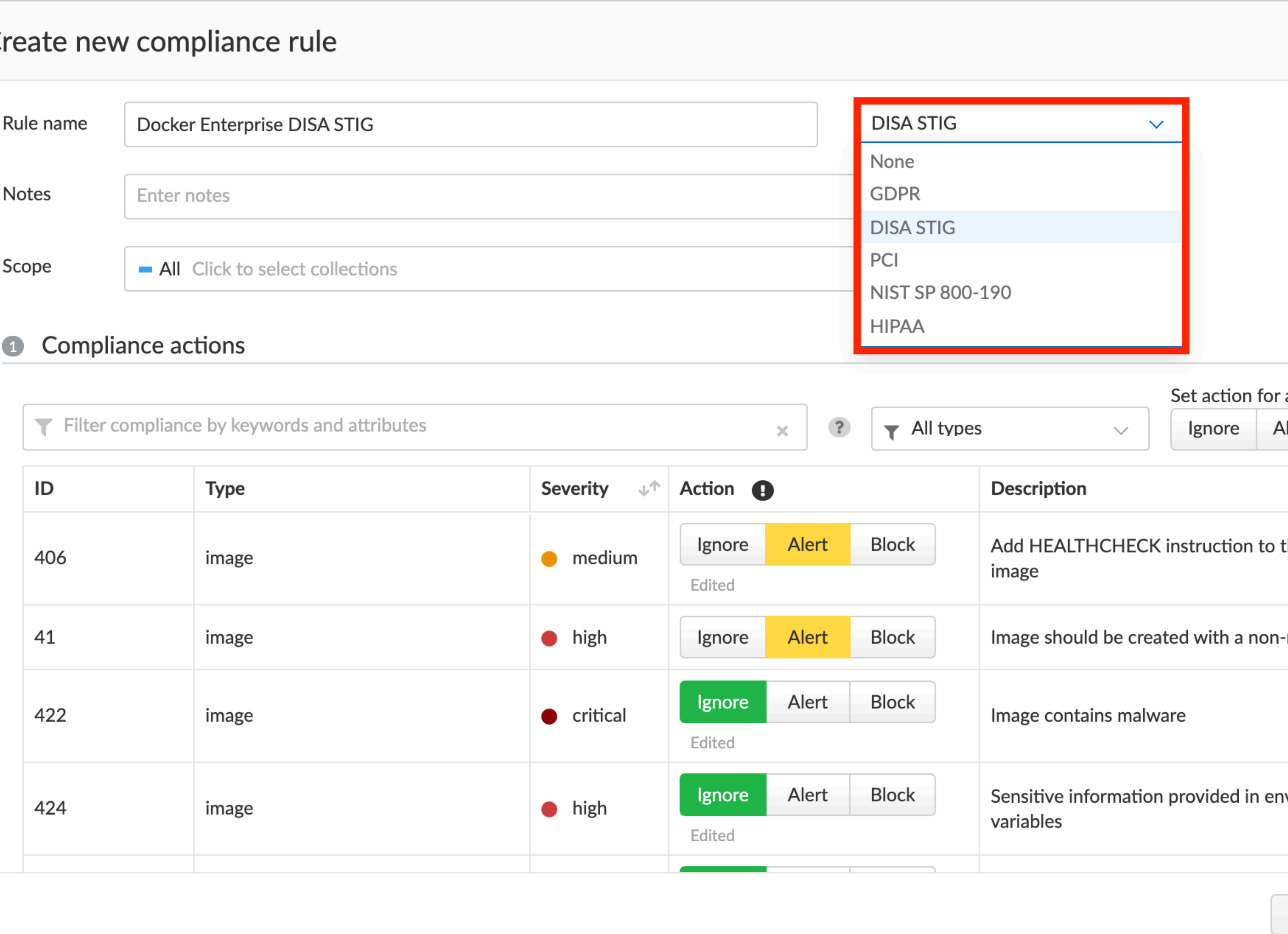
Enable DISA STIG for Docker Enterprise checks

DISA STIG for Docker Enterprise checks have been grouped into a template. Checks are relevant to containers, images, and hosts.

STEP 1 | Log into Console.

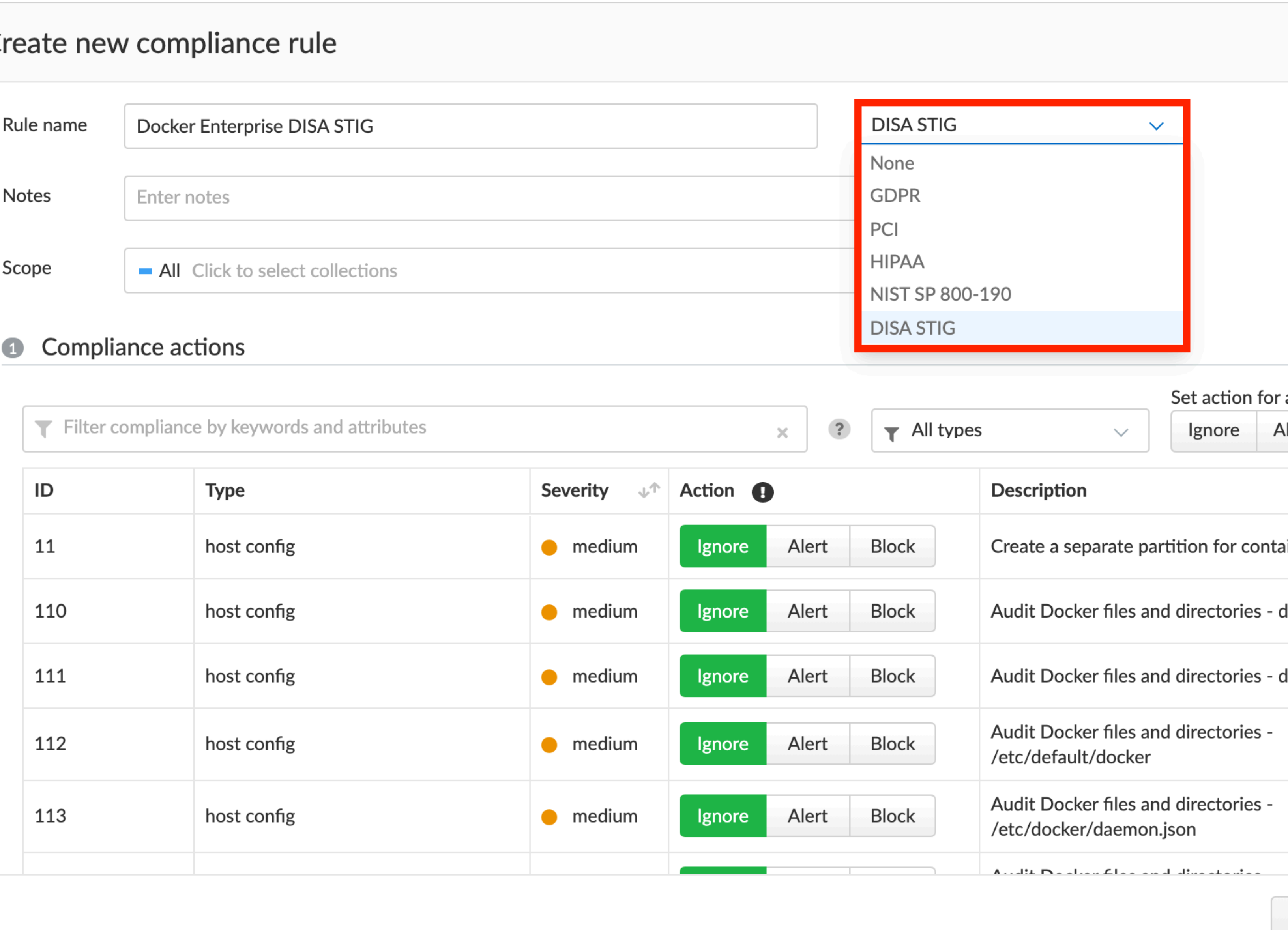
STEP 2 | Enable the container checks.

1. Go to **Defend > Compliance > Containers and images > {Deployed | CI}**.
2. Click **Add rule**.
3. Enter a rule name.
4. In the **Compliance template** drop-down, select **DISA STIG**.
5. Click **Save**.



STEP 3 | Enable host checks.

1. Go to **Defend** > **Compliance** > **Hosts** > **{Running hosts | VM images}**.
2. Click **Add rule**.
3. Enter a rule name.
4. In the **Compliance template** drop-down, select **DISA STIG**.
5. Click **Save**.



Custom compliance checks

Edit on GitHub

Custom image checks give you a way to write and run your own compliance checks to assess, measure, and enforce security baselines in your environment.

Prisma Cloud lets you implement your own custom image checks with simple scripts. Custom compliance checks are supported for Linux containers (docker or CRI-O), Windows containers (docker) and Linux Hosts

A custom image check consists of a single script. The script's exit code determines the result of the check, where 0 is pass and 1 is fail.

Scripts are executed in the default shell. The most common default shell for Linux is bash, but that's not always the case. For Windows container images, the default shell is cmd.exe.

If you want to use a specific shell, or if your default shell is in a non-standard location, use the shebang interpreter directive at the top of your compliance check to specify the path to the executable.

For example, #!/bin/bash specifies that the Linux Bourne-again (bash) shell should parse and interpret the compliance check.

For containers, Defender runs the compliance checks inside a restricted sandboxed container instantiated from the image being scanned, thus avoiding the unnecessary risk associated with running arbitrary code.

For hosts, Defender runs the compliance checks on the host itself with unrestricted privileges to allow execution of any script. In order to limit exposure, this feature is disabled by default.

Every compliance check in the system has a unique ID. Custom checks are automatically assigned an ID, starting with the number 9000. As new custom checks are added, they are automatically assigned the next available ID (9001, 9002, and so on).

If a new rule with custom compliance checks is added, or an existing rule is updated with a new custom compliance check, Prisma Cloud drops the cached compliance and vulnerability scan results for registries, and rescans registry images. In a scaled-out environment with large registries, repeated changes to custom compliance checks could have a negative impact on Prisma Cloud's performance.

Enabling custom compliance checks for hosts

By default, custom compliance checks for hosts is disabled.

If you enable the feature, and then later disable it, the disabled state is effective immediately. You don't need to redeploy Defenders when you switch to the disabled state. You only need to redeploy Defenders when switching to the enabled state.

STEP 1 | Go to **Manage > Defenders > Advanced Settings**.

STEP 2 | Set **Custom Compliance Checks for hosts** to enabled.

STEP 3 | Deploy Defenders to your environment. Or if already deployed, redeploy your Defenders.

Creating a new custom check

The flow for writing and operationalizing a custom check is:

- Write a custom check.
- Create a new compliance rule that includes your custom check, and specifies the action to take when the check fails (ignore, alert, block).

STEP 1 | Open Console

STEP 2 | Write a new custom check.

1. Go to **Defend > Compliance > Custom**.
2. Click **Add check**.
3. Enter a name and description.
4. Specify the severity of the compliance issue.
5. Enter a script.
6. Click **Save**.

STEP 3 | Update the compliance policy to run your check.

1. Go to **Defend > Compliance > Containers and Images** for containers or **Defend > Compliance > Hosts** for hosts.
2. Click **Add rule**.
3. Enter a rule name.
4. Under **Compliance actions**, narrow the compliance checks displayed.

   For containers, on the **All types** drop-down list, select **Custom > Image**.

   For hosts, on the **All types** drop-down list, select **Custom > Custom**.

   You should see a list of custom checks you've implemented, starting with ID 9000.
5. Select an action for your custom check (**Ignore**, **Alert**, or **Block**).
6. Click **Save**.

STEP 4 | Validate your setup by reviewing the compliance reports under **Monitor > Compliance**.

Example scripts

The following example scripts show how to run some basic checks, such as checking file permissions. Use them as starting point for your own scripts. Any special utilities or programs required by your script must be installed in the image being evaluated.

File permissions (Linux)

The following script checks the permissions for the */bin/busybox* file. Assuming busybox is installed in your image, this check should pass.

```
if [ $(stat -c %a /bin/busybox) -eq 755 ]; then
    echo 'test permission failure' && exit 1;
```



```
fi
```

File exists (Linux)

The following script checks if */tmp/foo.txt* exists in the container file system. If it doesn't exist, the check fails.

```
if [ ! -f /tmp/foo.txt ]; then
    echo "File not found!"
    exit 1
fi
```

User exists (Linux)

The following script checks if the user John exists. If the user exists, the check passes. Otherwise, it fails.

```
if grep -Fxq "John" /etc/passwd
then
    echo yes
else
    echo "user not found!"
    exit 1
fi
```

File exists (Windows)

The following script checks if *C:\Users* exists. If it does, the check passes.

```
IF EXIST C:\Users Echo test permission failure && exit 1
```

File does not exist (Windows)

This check is the inverse of the previous check. The script checks if *C:\Users* doesn't exist. If it doesn't exist, the check passes.

```
IF NOT EXIST C:\Users Echo test permission failure && exit 1
```


Trusted images

Edit on GitHub

Trusted images is a security control that lets you declare, by policy, which registries, repositories, and images you trust, and how to respond when untrusted images are started in your environment.

Image provenance is a core security concern. In NIST SP 800-190 (Application Container Security Guide), the section on countermeasures for major risks (Section 4) says:

"Organizations should maintain a set of trusted images and registries and ensure that only images from this set are allowed to run in their environment, thus mitigating the risk of untrusted or malicious components being deployed."

Container runtimes, such as Docker Engine, will run, by default, any container you ask it to run. Trusted images lets you explicitly define which images are permitted to run in your environment. If an untrusted image runs, Prisma Cloud emits an audit, raises an alert, and optionally blocks the container from running.

Modern development has made it easy to reuse open source software. Pulling images from public registries, such as Docker Hub, is simple and fast, and it removes a lot of friction in operations. Retrieving and executing software with such ease, however, runs contrary to many organizations' security policies, which mandate that software originates from approved providers and distribution points. The Trusted Images rule engine lets you specify registries, repositories, and images that are considered trustworthy.

Feature overview

Trusted images is disabled by default. To enable it, go to **Defend > Compliance > Trusted Images > Policy**.

After enabling the feature, you must specify the images you trust. Declare trust using objects called *trust groups*. Trust groups collect related registries, repositories, and images in a single entity. Then use those entities for writing policy rules.

The default policy consists of a single rule that alerts on all images started in your environment. Build out your policy by writing new rules. Rules let you define:

- Explicitly allowed trust groups.
- Explicitly denied trust groups
- An action to take when an image isn’t trusted.

When a container starts in your environment, Defender assesses the event against your trust policy, and then acts accordingly. Rules in a policy are evaluated top-down. The criteria for matching an event to a rule is the cluster or the hostname. When a matching rule is found, the rule is processed. No subsequent rules are processed. The first rule that matches the cluster or hostname holds the verdict for all images that can run on that cluster/host. If the image being started matches an explicitly denied trust group, the rule effect is applied. If an image doesn’t match either the list of explicitly allowed trust groups or explicitly denied trust groups, the rule effect is also applied.

Audits are created when the effect of a rule is alert or block. You can review audits in **Monitor > Events**. When reviewing audits, you can optionally add the image to a trust group to quickly adjust your policy and clean up false positives.

The Console UI provides a number of features to surface trust in your environment.

- Image scan reports have indicators in the report header to show whether an image is trusted or not. See:
  - **Monitor > Compliance > Containers and Images**
  - **Monitor > Vulnerabilities > Images**
- A dedicated page in **Monitor > Compliance > Trusted Images**, shows a snapshot of all running images in your environment and their trust status. The table is updated at scan-time, which is once per 24 hours by default. However, the page lets you force a re-scan and refresh the results.

  Also note that updated policies aren't automatically reflected in the view. If you change a rule in your Trusted Images policy, re-scan the images in your environment to update the view.

Trusted images aren't supported for workloads protected by App-Embedded Defender, including Fargate tasks.

Trust indicators in the Console UI

Badges are shown throughout Console to clearly delineate between trusted and unstrusted images. The following badges are used:

- Trusted — Explicitly trusted by a user-defined rule (**Defend > Compliance > Trusted Images > Policy**).

- Untrusted. An image is considered untrusted if it's untrusted on at least one host.

Badges are shown in the following pages:

- Scan reports (click on a row in the table to open an image scan report):
  - **Monitor > Compliance > Containers and Images**
  - **Monitor > Vulnerabilities > Images**
- Snapshot of all running containers and their trust status. The table is updated at scan-time.
  - **Monitor > Compliance > Trusted Images**

Events Viewer

Prisma Cloud generates an audit for every image that is started in your environment, but fails to comply with your trust policy. Audits can be reviewed under **Monitor > Events > Trust Audits**. When reviewing audits, you can optionally add the image to a trust group.

Establishing trust with rules

Prisma Cloud monitors the origin of all containers on the hosts it protects.

Policies are built on rules, and rules reference trust groups. Trust groups are collections of related registries and repositories.

Trust is established by one of the following factors:

- Point of origin (registry and/or repository),
- Base layer(s).

Trusting images by image tag isn't supported.

Establishing trust by registry and repository

Prisma Cloud lets you specify trust groups by registry and repository. If you specify just a registry, all images in the registry are trusted.

Establishing trust by base layer

Images can have layers in common. If your organization builds and approves specific base images for use in containerized apps, then you can use this mechanism to enforce compliance.

For example, consider the ubuntu:16.04 image. If you run *docker inspect*, the layers are:

```
"Layers": [

 "sha256:a94e0d5a7c404d0e6fa15d8cd4010e69663bd8813b5117fbad71365a73656df9",

 "sha256:88888b9b1b5b7bce5db41267e669e6da63ee95736cb904485f96f29be648bfda",

 "sha256:52f389ea437ebf419d1c9754d0184b57edb45c951666ee86951d9f6afd26035e",

 "sha256:52a7ea2bb533dc2a91614795760a67fb807561e8a588204c4858a300074c082b",

 "sha256:db584c622b50c3b8f9b8b94c270cc5fe235e5f23ec4aacea8ce67a8c16e0fbad"
]
```

Now consider a new image, where ubuntu:16.04 is the base OS. The following Dockerfile shows how such an image is constructed:

```
FROM ubuntu:16.04
RUN apt-get update
ADD hello.txt /home/hello.txt
WORKDIR /home
```

After building the image, and inspecting the layers, you can see that both images share the same first five layers.

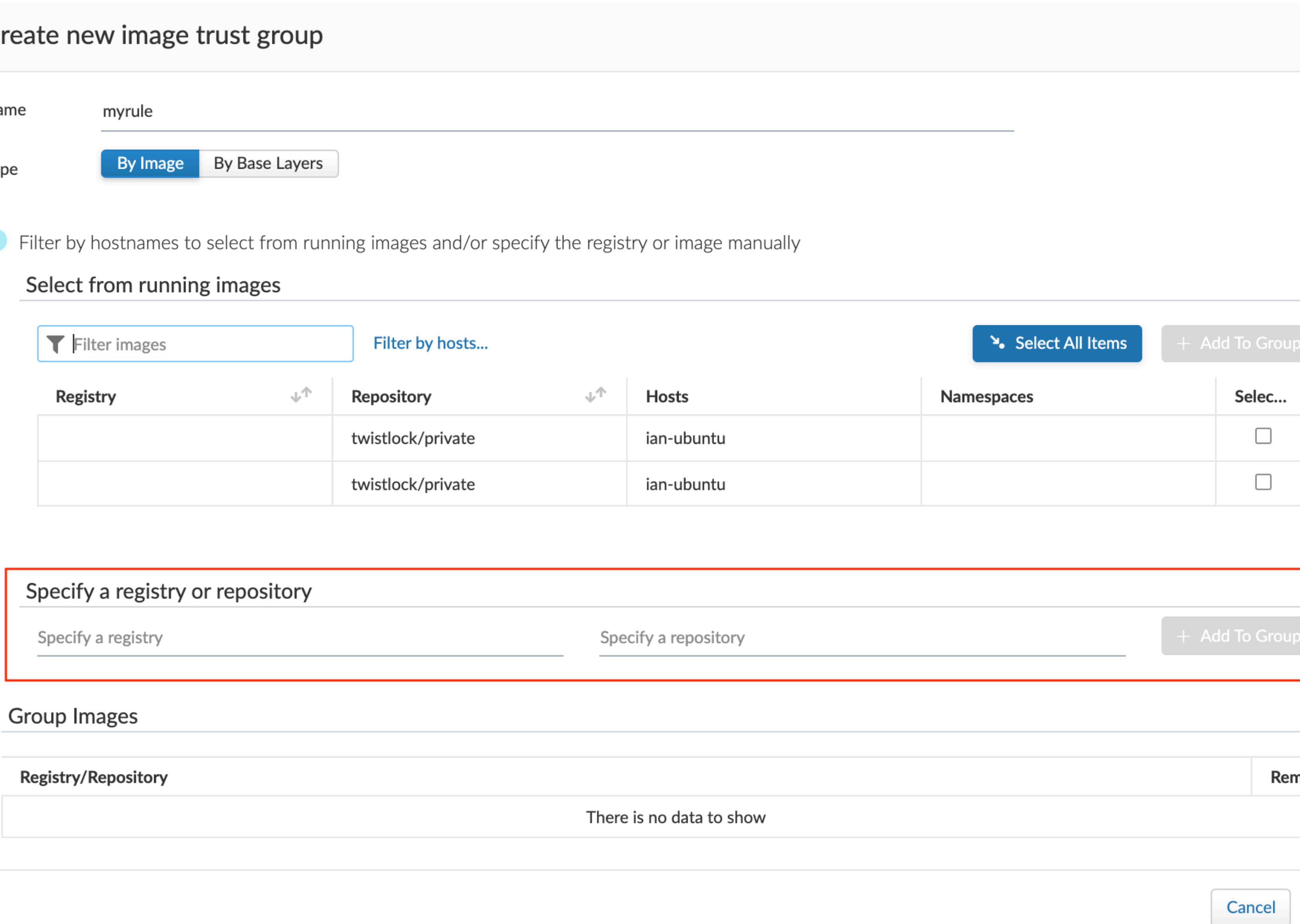
```
"Layers": [

 "sha256:a94e0d5a7c404d0e6fa15d8cd4010e69663bd8813b5117fbad71365a73656df9",
```

```
 "sha256:88888b9b1b5b7bce5db41267e669e6da63ee95736cb904485f96f29be648bfda",

 "sha256:52f389ea437ebf419d1c9754d0184b57edb45c951666ee86951d9f6afd26035e",

 "sha256:52a7ea2bb533dc2a91614795760a67fb807561e8a588204c4858a300074c082b",

 "sha256:db584c622b50c3b8f9b8b94c270cc5fe235e5f23ec4aacea8ce67a8c16e0fbad",

 "sha256:29d16833b7ef90fcf63466967c58330bd513d4dfe1faf21bb8c729e69084058f",

 "sha256:1d622b0ae83a00049754079a2bbbf7841321a24cfd2937aea2d57e6e3b562ab9"
]
```

Creating trust groups manually

Trust groups are collections of related registries and repositories. Policies are built on rules, and rules reference trust groups.

When setting up a trust group, you can explicitly specify registries and repositories to trust.



Prisma Cloud supports leading and trailing wildcard matches as described in the following table:

Match type	Registry only	Repository only	Both
Exact match	reg	repo	reg/repo
Suffix match	reg*	repo* repo/*	reg/repo* reg/repo/*
Prefix match	*reg	*repo	*reg/repo
Both suffix & prefix	*reg/*	*repo/*	*reg/repo/*

Examples:

- All repos under a parent repo:

  reg: reg

  repo: parent-repo/*
- A nested repo:

  reg: reg

  repo: parent-repo/some-repo
- All registries ending with "gcr.io":

  reg: *gcr.io

  repo: <unspecified>

Prerequisites:

- You've enabled the trusted images feature in **Defend > Compliance > Trusted Images > Policy**.

STEP 1 | Open Console.

STEP 2 | Go to **Defend > Compliance > Trusted Images > Trust Groups**.

STEP 3 | Click **Add New Group**.

STEP 4 | In **Name**, enter a group name.

STEP 5 | In **Type**, select how you want to specify an image.

By Image:

There are two ways to specify images:

Method 1 - Choose from a list of containers already running in your environment. In the table, select the images you trust, and click **Add To Group**.

Method 2 - Specify a registry address and/or repository, and click **Add To Group**. If you specify just a registry, then all images in the registry are trusted. If you specify just a repository, the registry is assumed to be Docker Hub.

As you add entries to the trust group, the entries are enumerated in the **Group Images** table at the bottom of the dialog.

By Base Layer:

Prisma Cloud lets you import the base layers from any image in your environment. If Prisma Cloud has seen and scanned an image, it is available in the **Image** drop-down list.

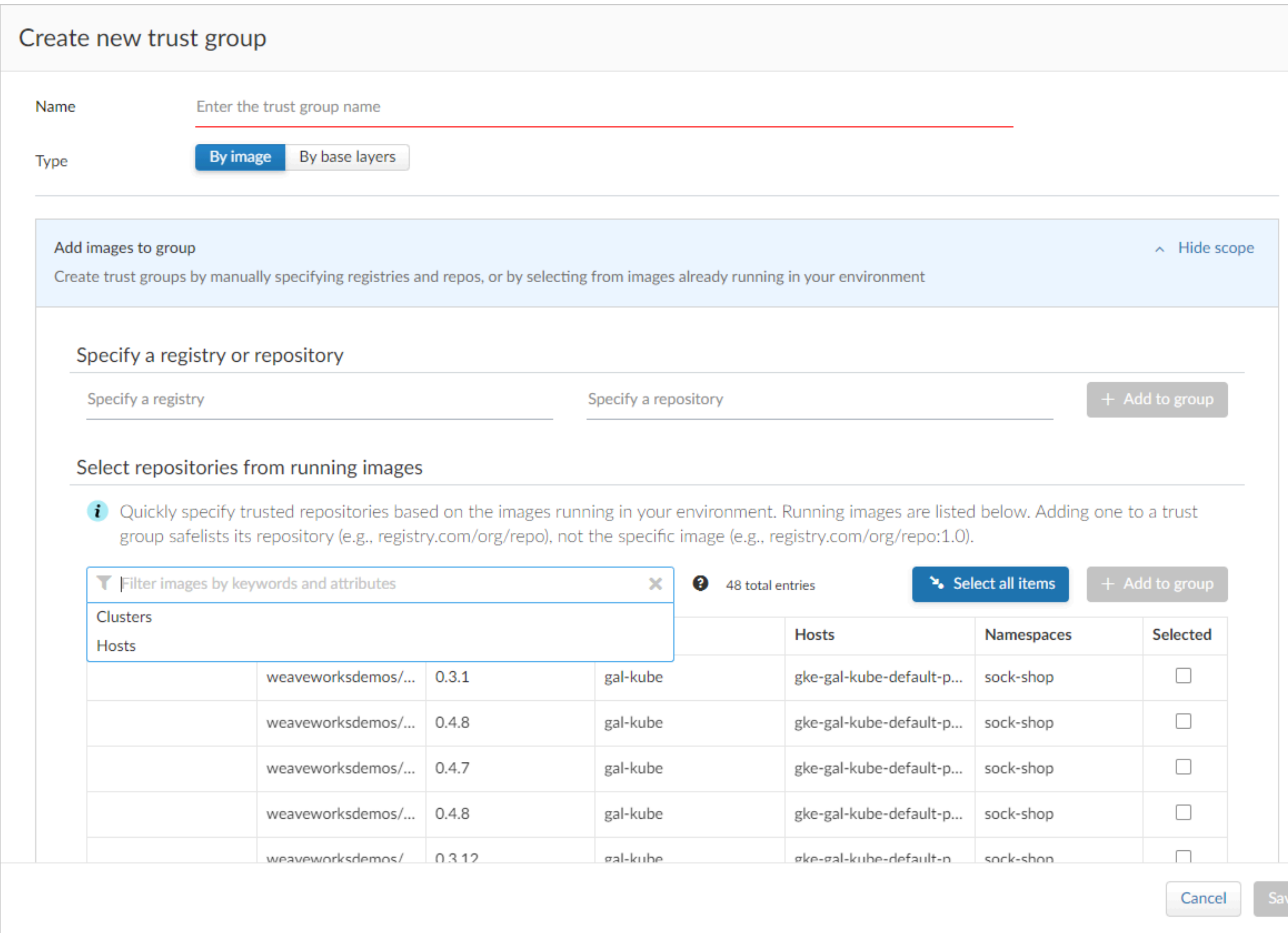
Select an image, import it, and then review the SHA256 hashes for the base layers. For example, if the secteam/ubuntu:16.04 is your trusted base OS, select it from the **Image** drop-down list, and click **Import**.

STEP 6 | Click **Save**.

Creating trust groups based on what's running in your environment

When setting up a trust group, Prisma Cloud shows you all running images in your environment You can use the filters to narrow the set, and them all to a trust group.

Filtering images by cluster is the most convenient option. For example, consider an environment with two clusters called "prod" and "dev". To create a trust group called "production images", select all the images running on the "prod" cluster. You would type "prod" in the filter line, and click Enter to filter. Then you could select all images on cluster and add them to the trust group. Later, you could create a rule for this prod cluster by specifying the cluster resource as "prod", and add the new trust group to the allowed groups. For more specific needs, you can also filter the running images by hosts.



Writing policy

After declaring the images you trust with trust groups, write the rules that make up your policy.

Prisma Cloud evaluates the rules in your trusted images policy from top to bottom until a match is found based on cluster and hostname. If the image being started in your environment matches a cluster/hostname in a rule, Prisma Cloud applies the actions in the rule and stops processing any further rules. If no match is found, no action is taken.

You should never delete the default rule, *Default - alert all*, and it should always be the last rule in your policy. The default rule matches all clusters and hosts (*). It serves as a catchall, alerting you to images that aren't captured by any other rule in your policy.

If you delete all rules in your policy, including the default rule, all images in your environment will be considered trusted.

Assuming the default rule is in place, policy is evaluated as follows:

- **A rule is matched** — The rule is evaluated.
- **A rule is matched, but no trust group is matched** — The image is considered untrusted. Prisma Cloud takes the same action is if it were explicitly denied.
- **No rule match is found** — The default rule is evaluated, and an alert is raised for the image that was started. The default rule is always matched because the cluster and hostname are set to a wildcard

STEP 1 | Open Console.

STEP 2 | Go to **Defend** > **Compliance** > **Trusted Images** > **Policy**.

STEP 3 | Click **Add Rule**.

STEP 4 | Enter a rule name.

STEP 5 | In **Effect**, specify how Prisma Cloud responds when it detects an explicitly denied image starting in your environment. This action is also used when a rule is matched (by cluster/hostname), but no trust group in the rule is matched.

Ignore — Do nothing if an untrusted image is detected.

Alert — Generate an audit and raise an alert.

Block — Prevent the container from running on the affected host. Blocking isn't supported for Windows containers.

STEP 6 | Specify the rule's scope.

By default, the rule applies to all clusters and hosts in your environment. Pattern matching is supported.

STEP 7 | Explicitly allow or deny images by trust group.

STEP 8 | (Optional) Append a custom message to the block action message.

Custom messages help the operator better understand how to handle a blocked action. You can enhance Prisma Cloud's default response by appending a custom message to the default message. For example, you could tell operators where to go to open a ticket.

STEP 9 | Click **Save**.

Your rule is added to the top of the rule list. Rules are evaluated from top to bottom. The rule at the top of the table has the highest priority. The rule at the bottom of the table should be your catch-all rule.

Host scanning

Edit on GitHub

Prisma Cloud scans all hosts where Defender is installed.

Defender scans hosts for the following types of vulnerabilities:

- **Host configuration**: Vulnerabilities in the host setup.
- **Docker daemon configuration**: Vulnerabilities that stem from misconfiguring your Docker daemons. Docker daemon derives its configuration from various files, including /etc/sysconfig/docker or /etc/default/docker. Misconfigured daemons affect all container instances on a host.
- **Docker daemon configuration files**: Vulnerabilities that arise from improperly securing critical configuration files with the correct permissions.
- **Docker security operations**: Recommendations and reminders for extending your current security best practices to include containers.

Prisma Cloud implements the checks from:

- CIS Distribution Independent Linux v2.0.0.
- CIS Amazon Linux 2 Benchmark v1.0.0 (for AL 2)
- CIS Amazon Linux Benchmark v2.1.0 (for AL 1)

Reviewing host scan reports

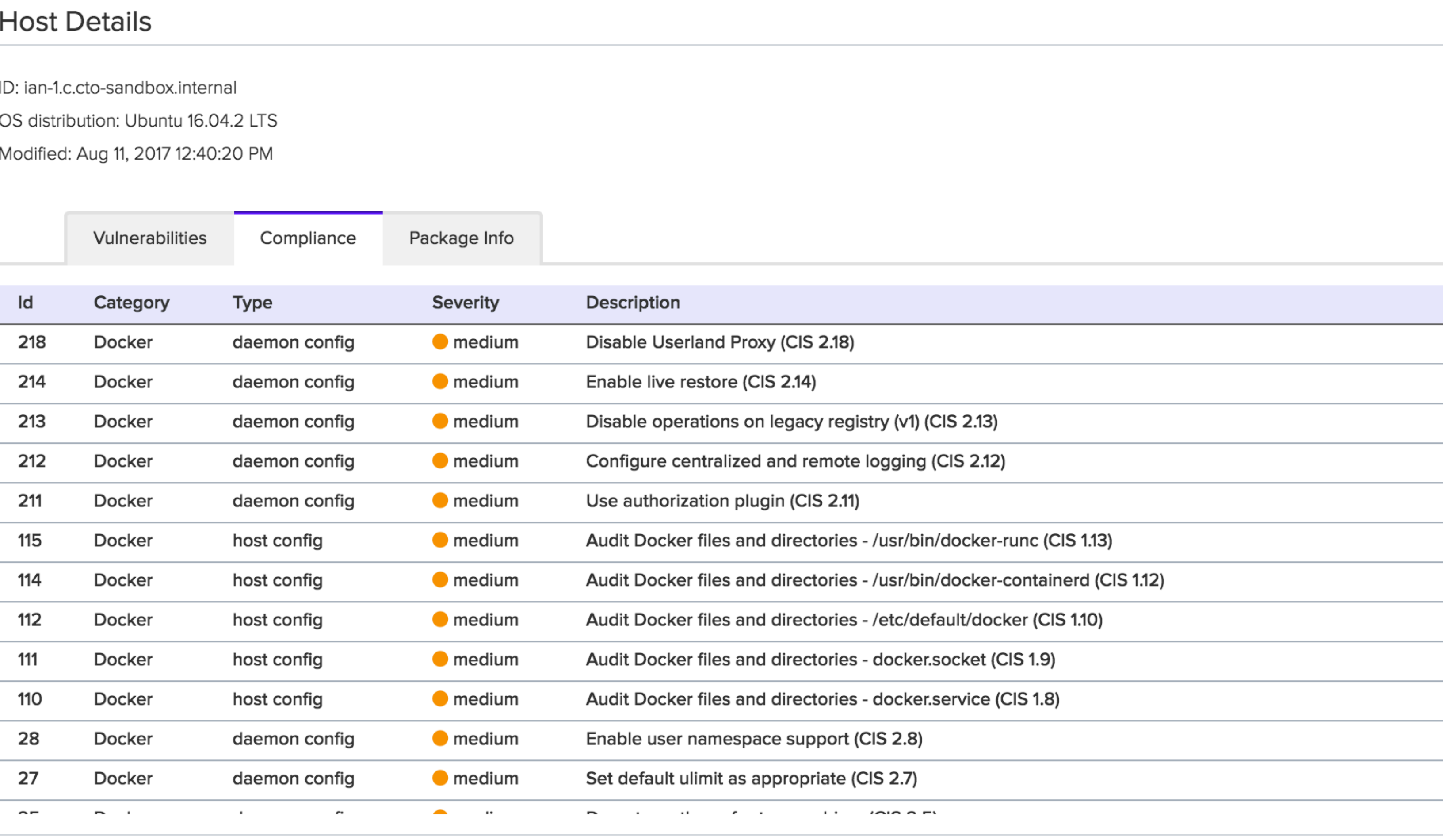
Prisma Cloud lets you filter the displayed hosts by searching for specific hosts or by collection. Collections support AWS tags. When creating new collections, specify the tags you want to use for filtering in the **Labels** field.

You can filter the displayed hosts by searching for specific hosts or by choosing a collection. Collections support AWS tags. When creating a new collection, add the tags you want to use for filtering to the **Labels** field.

STEP 1 | Open Console, then go to **Monitor > Compliance > Hosts > Running Hosts**.

STEP 2 | Click on a host in the list.

A report for the compliance issues on the host is shown.



All vulnerabilities identified in the latest host scan can be exported to a CSV file by clicking on the **CSV** button in the top right of the table.

VM image scanning

Edit on GitHub

Prisma Cloud can scan the virtual machine (VM) images in your AWS environment for the following types of vulnerabilities:

- **Host configuration**: Vulnerabilities in the VM image setup.
- **Docker daemon configuration**: Vulnerabilities that stem from misconfiguring your Docker daemon. The Docker daemon derives its configuration from various files, including /etc/sysconfig/docker or /etc/default/docker.
- **Docker daemon configuration files**: Vulnerabilities that arise from setting incorrect permissions on critical configuration files.
- **Docker security operations**: Recommendations and reminders for extending your current security best practices to include containers.
- **Linux configuration**: Compliance of Linux hosts. For example, ensure mounting of the *hfs* filesystem is disabled.

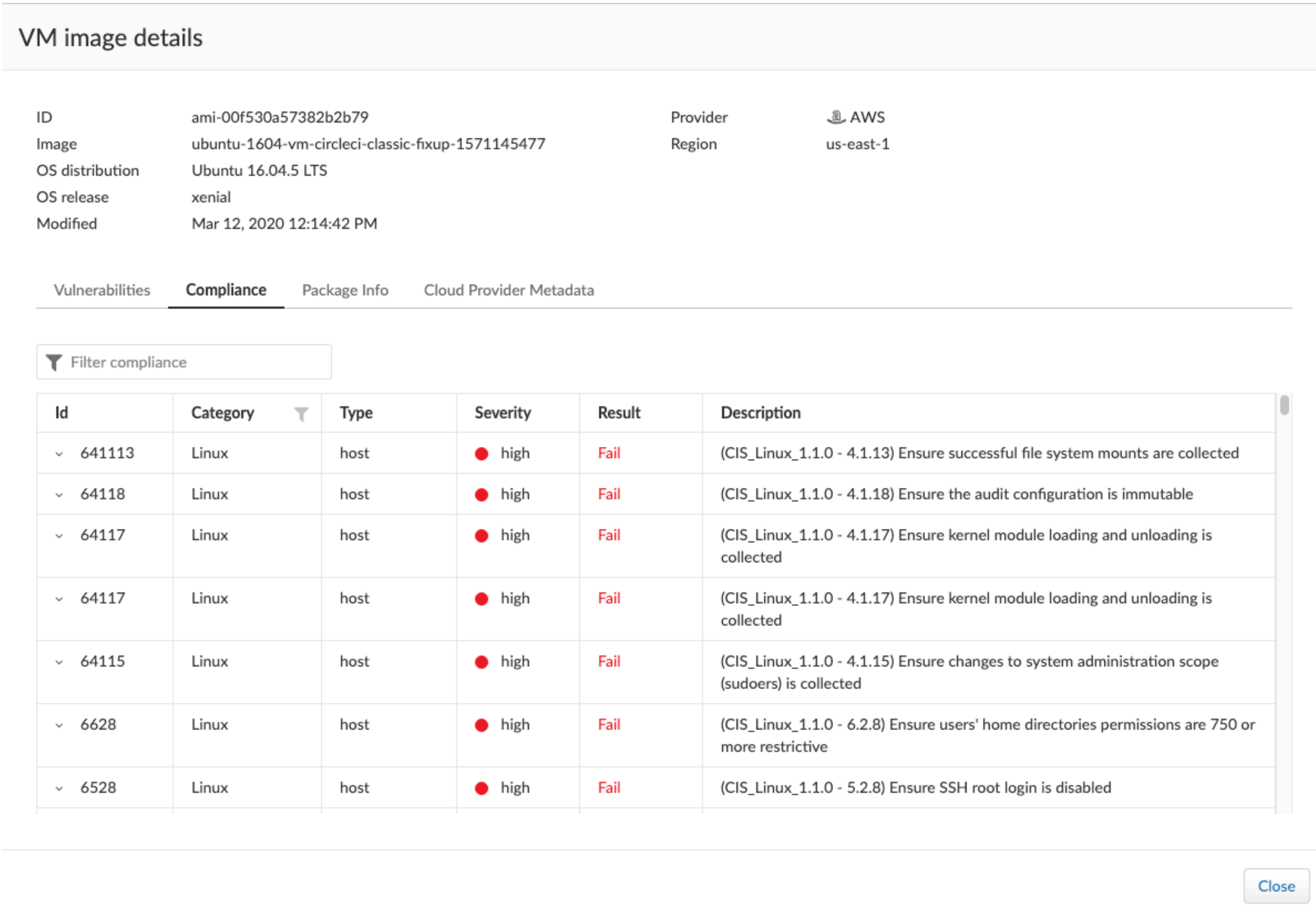
Reviewing VM image scan reports

To view the health of the VM images in your environment:

STEP 1 | Open Console, then go to **Monitor > Compliance > Hosts > VM images**.

STEP 2 | Click on a VM image on the list.

A report for the compliance issues on the VM image is shown.



All compliance issues identified in the latest VM image scan can be exported to a CSV file by clicking on the **CSV** button in the top right of the table.

App-Embedded scanning

Edit on GitHub

App-Embedded Defenders can scan their workloads for compliance issues.

App-Embedded Defender support the following types of compliance checks:

- Image compliance checks.
- Custom compliance checks.

To see compliance scan reports, go to **Monitor > Compliance > Images > Deployed**. You can filter the table by:

- **App-Embedded: Select** — Narrows the results to just images protected by App-Embedded Defenders.
- **App ID** — Narrows the list to specific images. App IDs are listed under the table's **Apps** column.

  For ECS Fargate tasks, the App ID is partially constructed from the task name. AWS Fargate tasks can run multiple containers. All containers in a Fargate task have the same App ID.

  For all other workloads protected by App-Embedded Defender, the App ID is partially constructed from app name, which is a deploy-time configuration set in the App ID field of the embed workflow.

You can use wildcards to filter the table by app/image name. For example, if the app name is *dvwa*, then you could find all deployments with *Repository: dvwa**. This filter would show *dvwa:0438dc81a9144fab8cf09320b0e1922b* and *dvwa:538359b5f7f54559ab227375fe68cd7a*.

Create compliance rules

Create a compliance rules for workloads protected by App-Embedded Defender.

STEP 1 | Login to the Console.

STEP 2 | Go to **Defend > Compliance > Containers and images > Deployed**.

STEP 3 | Click **Add rule**.

STEP 4 | Enter a rule name.

STEP 5 | Click on **Scope** to select a relevant collection, or create a new collection.

Workloads are scoped by App ID. App ID is specified when you embed the App-Embedded Defender into a workload, and represents a unique identifier for the Defender/workload pair.

1. If creating a collection, click **Add collection**.
2. Enter collection name.
3. In the **App ID** field, enter one or more App IDs.

   Postfix wildcards are supported.
4. Click **Save**.
5. Select the new collection.
6. Click **Select collection**.

STEP 6 | Click **Save**.

The block action doesn't apply to App-Embedded workloads.

Supported compliance checks

App-Embedded Defenders support the following built-in image compliance checks:

- **448: Package binaries should not be altered** — Checks the integrity of package binaries in an image. During an image scan, every binary's checksum is compared with its package info.
- **424: Sensitive information provided in environment variables** — Checks if images contain sensitive information in their environment variables.
- **425: Private keys stored in image** — Searches for private keys stored in an image or serverless function.
- **426: Image contains binaries used for crypto mining** — Detects when there are crypto miners in an image. Attackers have been quietly poisoning registries and injecting crypto mining tools into otherwise legitimate images.

App-Embedded Defenders also support custom compliance checks. Custom compliance checks let you write and run your own compliance checks to assess, measure, and enforce your own security baselines. Custom checks only work for workloads that allow users with root privileges.

Deploy an example Fargate task

Deploy the *fargate-vulnerability-compliance-task* Fargate task. Follow the steps in Embed App-Embedded Defender into Fargate tasks.

You can use the following task definition to test Prisma Cloud's App-Embedded Defender. It's based on an Ubuntu 18.04 image. On start up, it runs the */bin/sh -c 'cp /bin/sleep /tmp/xmrig* command to trigger the compliance check that detects crypto miners in images.

```
{
  "containerDefinitions": [
     {
        "command": [
```

```
            "/bin/sh -c 'cp /bin/sleep /tmp/xmrig && echo \"[+]
 Sleeping...\" && while true; do sleep 1000 ; done'"
         ],
         "entryPoint": [
            "sh",
            "-c"
         ],
         "essential": true,
         "image": "ubuntu:18.04",
         "logConfiguration": {
            "logDriver": "awslogs",
            "options": {
               "awslogs-group" : "/ecs/fargate-task-definition",
               "awslogs-region": "us-east-1",
               "awslogs-stream-prefix": "ecs"
            }
         },
         "name": "Fargate-vul-comp-test",
         "portMappings": [
            {
               "containerPort": 80,
               "hostPort": 80,
               "protocol": "tcp"
            }
         ]
      }
   ],
   "cpu": "256",
   "executionRoleArn": "arn:aws:iam::012345678910:role/
ecsTaskExecutionRole",
   "family": "fargate-vulnerability-compliance-task",
   "memory": "512",
   "networkMode": "awsvpc",
   "requiresCompatibilities": [
       "FARGATE"
    ]
}
```

Review compliance scan reports

Review the scan results in Console.

For Fargate version 1.3.0 and older, Prisma Cloud shows only a single scan report if the same image is run simultaneously as:

- *A task on ECS Fargate, protected by App-Embedded Defender.*
- *A container on a host, protected by Container Defender.*

In this case, the image is categorized as "App-Embedded". As a result, when the scan report table is filtered by ***App-Embedded: Select****, a scan report will be shown. When the table is filtered by* ***App-Embedded: Exclude****, it will be hidden. And when filtering by* ***Hosts****, it will be hidden, even if the host matches, because the image is considered as App-Embedded.*

For Fargate version 1.4.0, two separate scan reports are shown, one for App-Embedded and one for Container Defender.

STEP 1 | Navigate to **Monitor > Compliance > Images > Deployed** and validate that the deployed image appears with an alerted compliance check.

STEP 2 | To see all images protected by App-Embedded Defender, filter the table by **App-Embedded: Select**.

STEP 3 | If you deployed the example Fargate task, search for *fargate-vulnerability-compliance-task*.

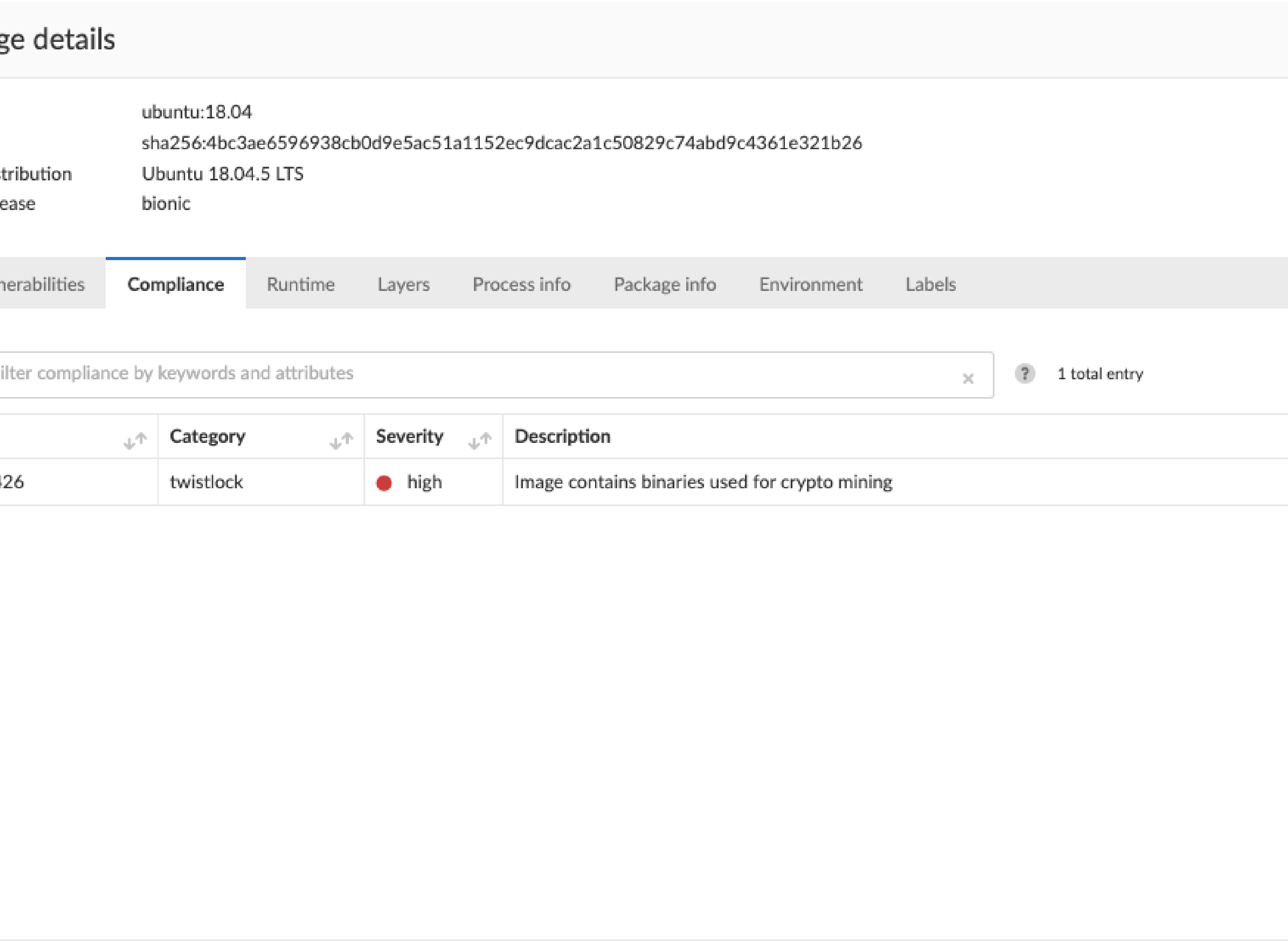
STEP 4 | Click on the image to view image details:

*The **Apps** column shows a count of the number of running containers protected by App-Embedded Defender.*

*The **Layers**, **Process info**, **Labels**, **Runtime**, and **Trust groups** tabs aren't supported for images scannned by App-Embedded Defenders.*

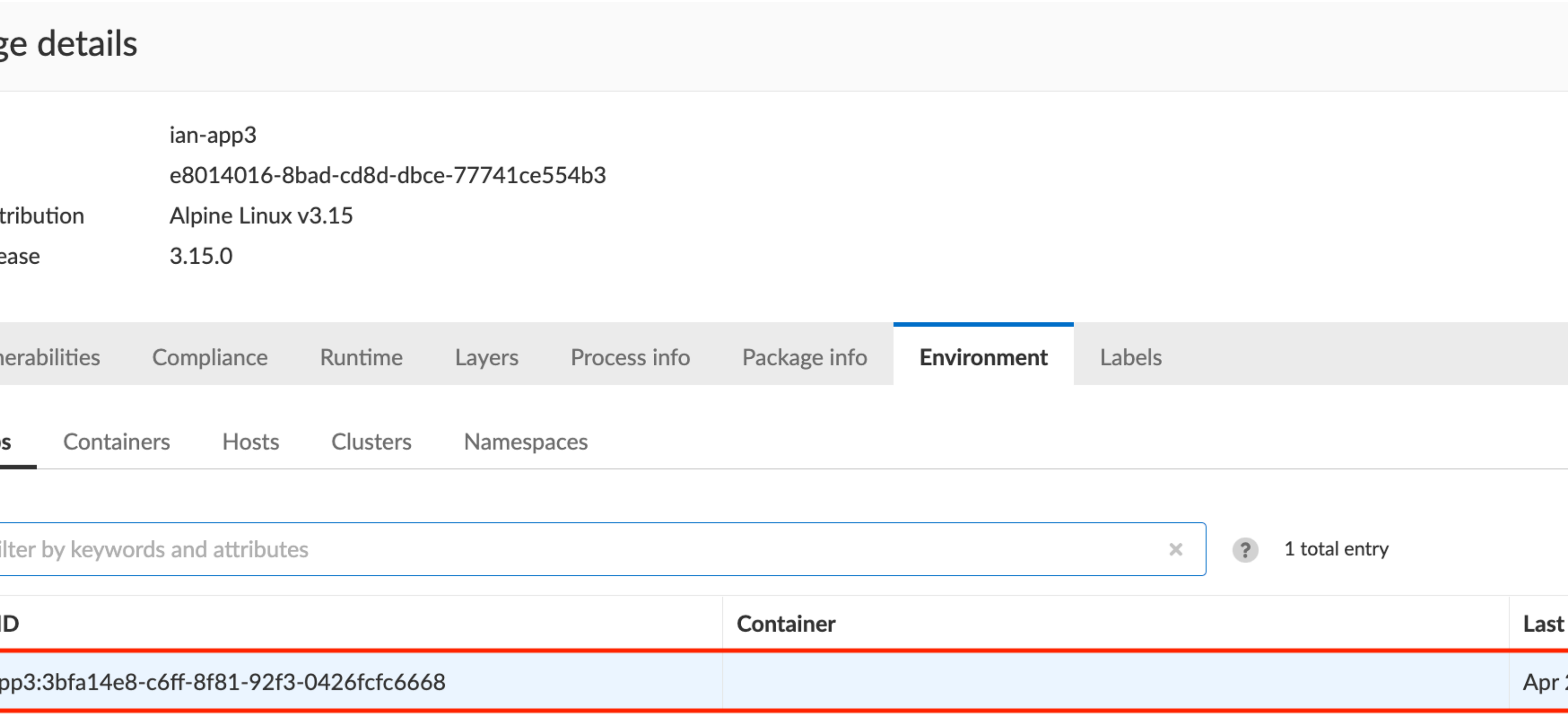
1. Click the **Compliance** tab to review compliance issues.

   You should seen an issue for **Image contains binaries used for crypto mining**.

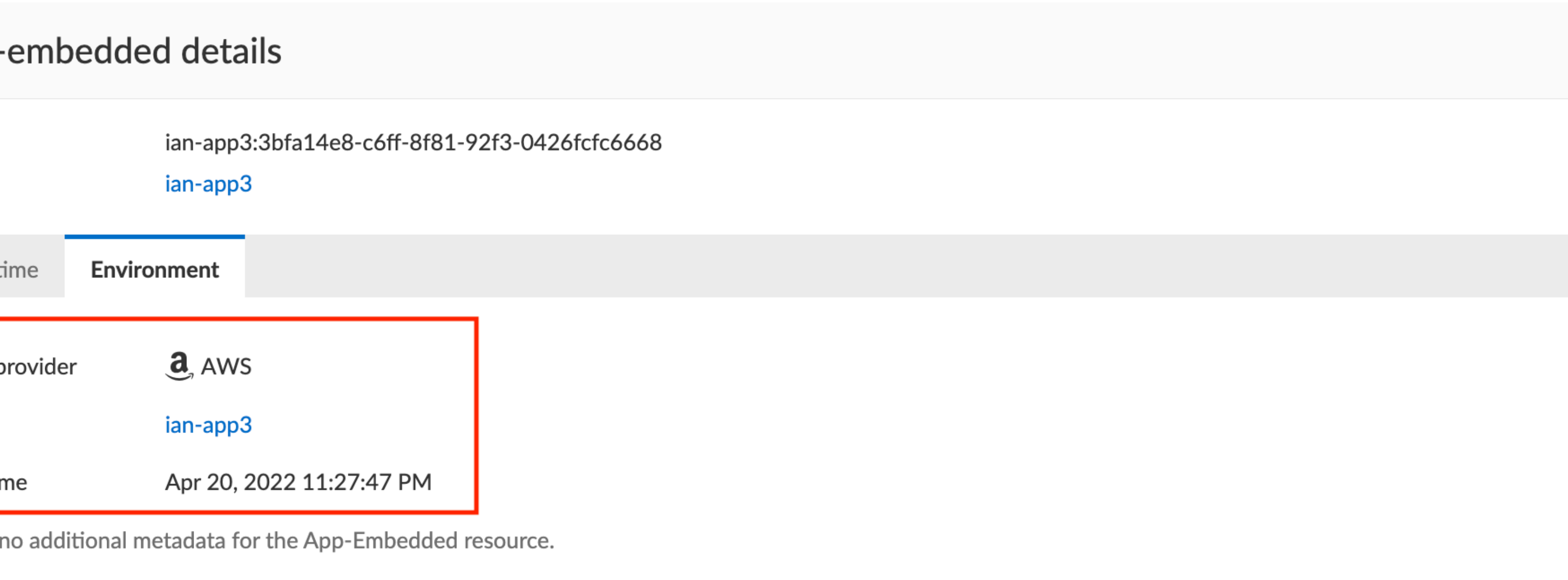


2. Review runtime information for the container.

Go to the **Environment > Apps** tab, and then click on the app in the table to open the App-Embedded observations. You can bring up the same view by going directly to **Monitor > Runtime > App-Embedded observations**, and clicking on the same app.



The **Environment** tab shows cloud-provider metadata that App-Embedded Defender collected about the running container. For more information about the type of cloudprovider metadata App-Embedded Defender can collect, see Monitoring workloads at runtime.



Detect secrets

Edit on GitHub

Prisma Cloud can detect sensitive information that is improperly secured inside images and containers. Scans can detect embedded passwords, login tokens, and other types of secrets. To detect improperly secured secrets, add the following checks to your compliance policy.

Compliance check ID 424

This check detects sensitive information provided in environment variables of image. The data so provided can be easily exposed by running *docker inspect* on the image and thus compromising privacy.

Example

```
$ docker --tlsverify -H :9998 build -t secret:v1 .
```

Response

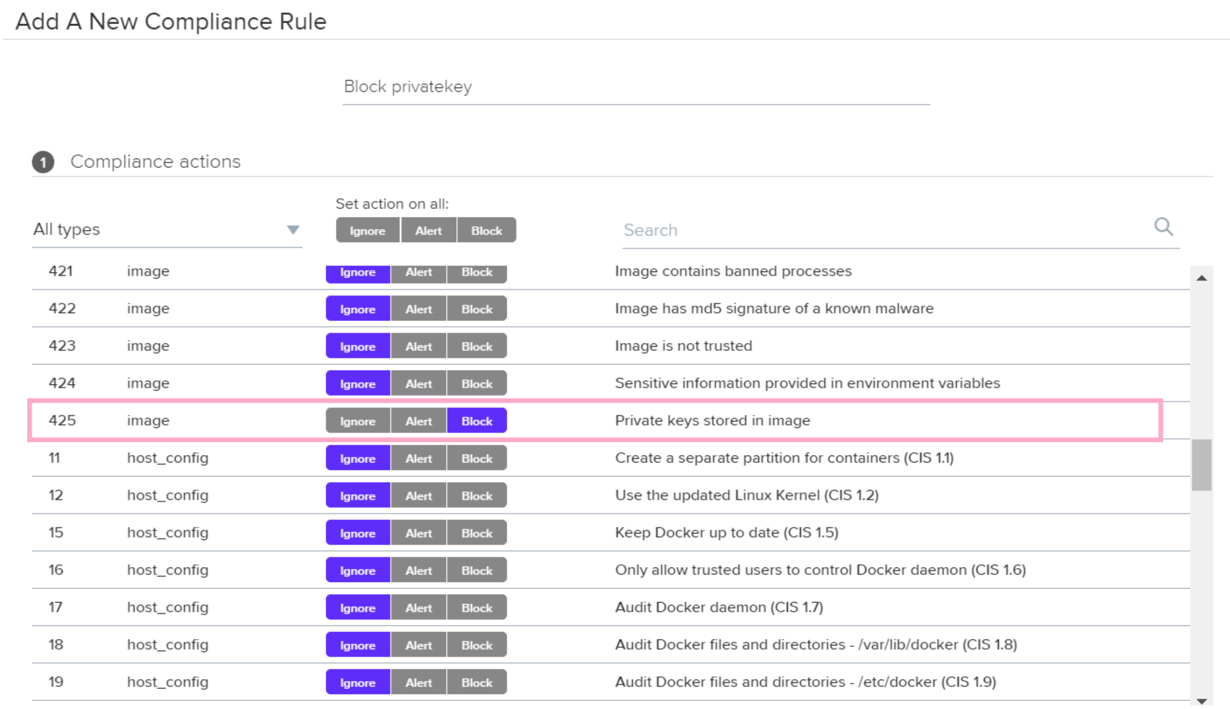
```
Sending build context to Docker daemon 2.048 kB
Step 1/2 : FROM alpine:latest
 ---> 88e169ea8f46
Step 2/2 : ENV PASSWORD = secret
 ---> Using cache
 ---> 8f3627bc339b
Error: [Prisma Cloud] Image operation blocked by policy: (No secrets
 attached), violates: The environment variable PASSWORD contains
 sensitive data
```

Compliance check ID 425

This check detects private keys stored in an image.

Example

Navigate to **Defend > Compliance**. Add a new compliance rule to block running an image with private key in it.



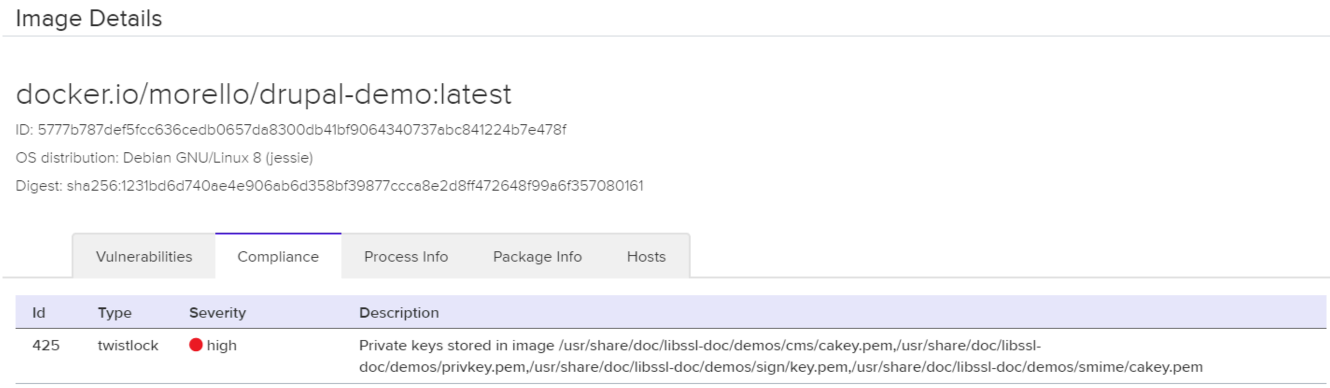
Test

```
$ docker --tlsverify -H  aqsa.c.cto-sandbox.internal:9998 build -t
 aqsa:secretv1
```

Response

```
Sending build context to Docker daemon 5.632 kB
Step 1/2 : FROM alpine:latest
 ---> 88e169ea8f46
Step 2/2 : ADD private_key /root/.ssh/id_rsa
 ---> Using cache
 ---> c6e8e2496663
Error: [Prisma Cloud] Image operation blocked by policy: (No secrets
 attached), violates: Private keys stored in image /root/.ssh/id_rsa
```

Set the action to **ALERT** instead of **BLOCK**, then go to **Monitor > Compliance** after running the image. Click on the image under **Images** tab.



Compliance check ID 597

This check detects sensitive information provided in environment variables of container.

Cloud discovery

Edit on GitHub

It's difficult to ensure that all your apps running on all the different types of cloud services are being properly secured. If you're using multiple cloud platforms, you might have many separate accounts per platform. You could easily have hundreds of combinations of providers, accounts, and regions where cloud native services are being deployed.

Cloud discovery helps you find all cloud-native services being used on AWS, Azure, and Google Cloud, across all regions, and across all accounts. It enables you to continuously monitor these accounts, detect when new services are added, and report on the services that are unprotected, so that you can mitigate your exposure to rogue deployments, abandoned environments, and sprawl.

Cloud discovery offers coverage for the following services.

Registries:

- AWS
- Azure
- Google Artifact Registry
- Google Container Registry[1,2]

Serverless functions:

- AWS
- Azure
- Google Cloud

[3] **Managed platforms:**

- AWS ECS
- AWS EKS
- Azure Kubernetes Service (AKS)
- Azure Container Instances (ACI)
- Google Kubernetes Engine (GKE)

Virtual machines:

- AWS EC2 instances
- Azure VMs[3]
- Google Cloud Platform (GCP) Compute Engine VM instances[3]

[1]Although Artifact Registry supports a number of content types (for example, Java, Node.js, and Python language packs), Prisma Cloud only supports discovering and scanning Docker images.

[2]Prisma Cloud doesn't support scanning Helm charts saved as OCI images and stored in Artifact Registry. The OCI image that represents a Helm chart has a single layer that contains the Helm package. It is only a way to store a Helm chart, but it has no meaning in terms of a container. Prisma Cloud has no way to run the image to scan it. Note that Helm charts stored as OCI images

will be shown in the list of resources discovered in the registry because we can't indicate their type until we actually pull and scan them.

[3]Auto-defend is currently not yet available for these services. Auto-defend utilizes rule-based policies to automatically deploy Prisma Cloud to protect resources in your environment.

Minimum permissions

Prisma Cloud needs one set of minimum permissions to discover and itemize all the resources in your account. After finding those resources, Prisma Cloud typically needs an additional set of permissions to protect them (e.g. retrieve those resources and inspect them for vulnerabilities and compliance issues.

For example, the service account for cloud discovery uses the *ecr:DescribeRepositories* permission to list all ECR repositories in your AWS accounts. If you find a repository that's not being scanned, and you want to configure Prisma Cloud to scan it, Prisma Cloud needs another service account with deeper permissions that lets it auth with the ECR service and download images from the repository (e.g., *ecr:GetAuthorizationToken*, *ecr:BatchGetImage*, etc). The permissions required for cloud discovery to scan your accounts are documented here. Permissions required to enable protection (e.g. scanning a repo) are documented in each protection feature's respective article.

AWS

For AWS, Prisma Cloud requires a service account with following minimum permissions policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "PrismaCloudComputeCloudDiscovery",
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeImages",
                "ec2:DescribeInstances",
                "ec2:DescribeRegions",
                "ec2:DescribeTags",
                "ecr:DescribeRepositories",
                "ecs:DescribeClusters",
                "ecs:ListClusters",
                "ecs:ListContainerInstances",
                "eks:DescribeCluster",
                "eks:ListClusters",
                "lambda:GetFunction",
                "lambda:ListFunctions"
            ],
            "Resource": "*"
        }
    ]
}
```

Azure

For Azure, Prisma Cloud requires a role with the following minimum permissions:

```
{
```



```
    "permissions": [
        {
            "actions": [
                "Microsoft.ContainerRegistry/registries/read",
                "Microsoft.ContainerRegistry/registries/metadata/
read",
                "Microsoft.ContainerService/managedClusters/read",
                "Microsoft.Web/sites/Read",
                "Microsoft.ContainerInstance/containerGroups/read",
                "Microsoft.ContainerInstance/containerGroups/
containers/exec",
                "Microsoft.Compute/virtualMachines/read",
                "Microsoft.Compute/virtualMachineScaleSets/read",
                "Microsoft.Compute/virtualMachineScaleSets/
virtualMachines/read",
                "Microsoft.Compute/virtualMachineScaleSets/
virtualMachines/instanceView/read"
            ],
            "notActions": [],
            "dataActions": [],
            "notDataActions": []
        }
    ]
}
```

The *Microsoft.ContainerInstance/containerGroups/containers/exec* checks for whether ACI is defended.

Google Cloud

For GCP, Prisma Cloud requires a service account with the viewer role. The basic role *roles/viewer*, however, is very broad with thousands of permissions across all Google Cloud services.

For production environments, use a more tightly scoped service account with the following predefined roles:

Predefined roles:

- Artifact Registry Reader (*roles/artifactregistry.reader*)
- Storage Object Viewer (*roles/storage.objectViewer*)
- Kubernetes Engine Cluster Viewer (*roles/container.clusterViewer*)
- Cloud Functions Viewer (*roles/cloudfunctions.viewer*)

Also, create custom role with the following permissions, and attach it to your serivce account.

- *compute.instances.list*
- *compute.zones.list*
- *compute.projects.get*
- *cloudfunctions.functions.sourceCodeGet* # Required for serverless function scanning

Configuring cloud platforms discovery

Set up Prisma Cloud to scan your cloud platform accounts for cloud-native resources and services. Then configure Prisma Cloud to protect them with a single click.

Prerequisites: You created service accounts for your cloud providers that provide the minimum required permissions, as described here.

STEP 1 | Log in to Prisma Cloud Compute Console.

STEP 2 | Select **Compute > Manage > Cloud Accounts**.

STEP 3 | Select the accounts to scan. If there are no accounts in the table, use the **+ Add account** button to onboard your cloud accounts.

- *On GCP: If you select organization level GCP credentials, for an organization with hundreds of projects, the performance of the Google Cloud Registry discovery might be affected due to long query time from GCP. The best approach to reduce scan time and avoid potential timeouts is to divide the projects in your organization into multiple GCP folders. Then create a service account and credential for each folder, and use these credentials for cloud discovery.*
- *On Azure: If you create a credential in the credentials store under* ***Manage > Authentication > Credentials store****, your service principal authenticates with a password. To authenticate with a certificate, create a cloud account.*

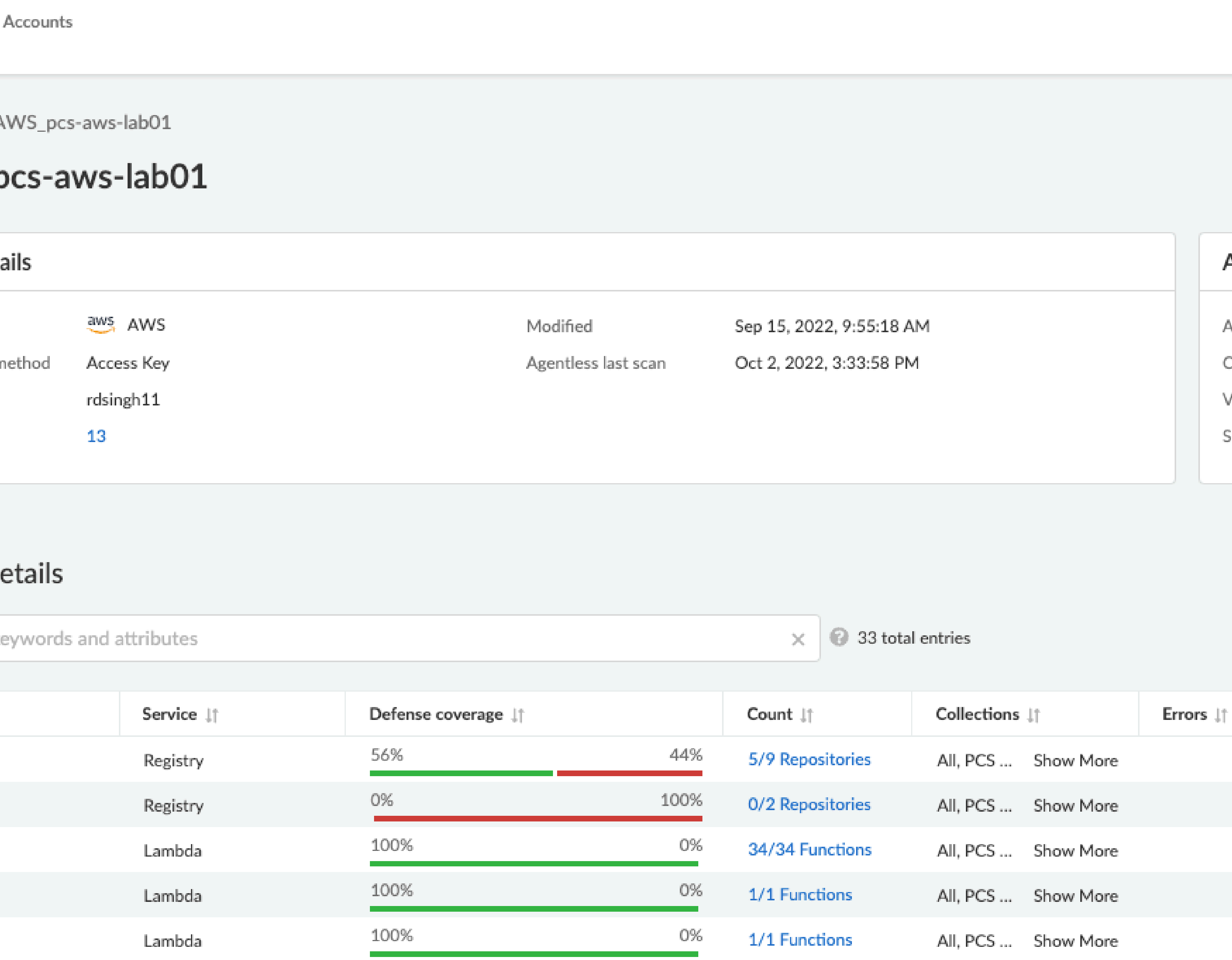
STEP 4 | Enable **Cloud discovery**.

STEP 5 | Click **Add account** to save the changes.

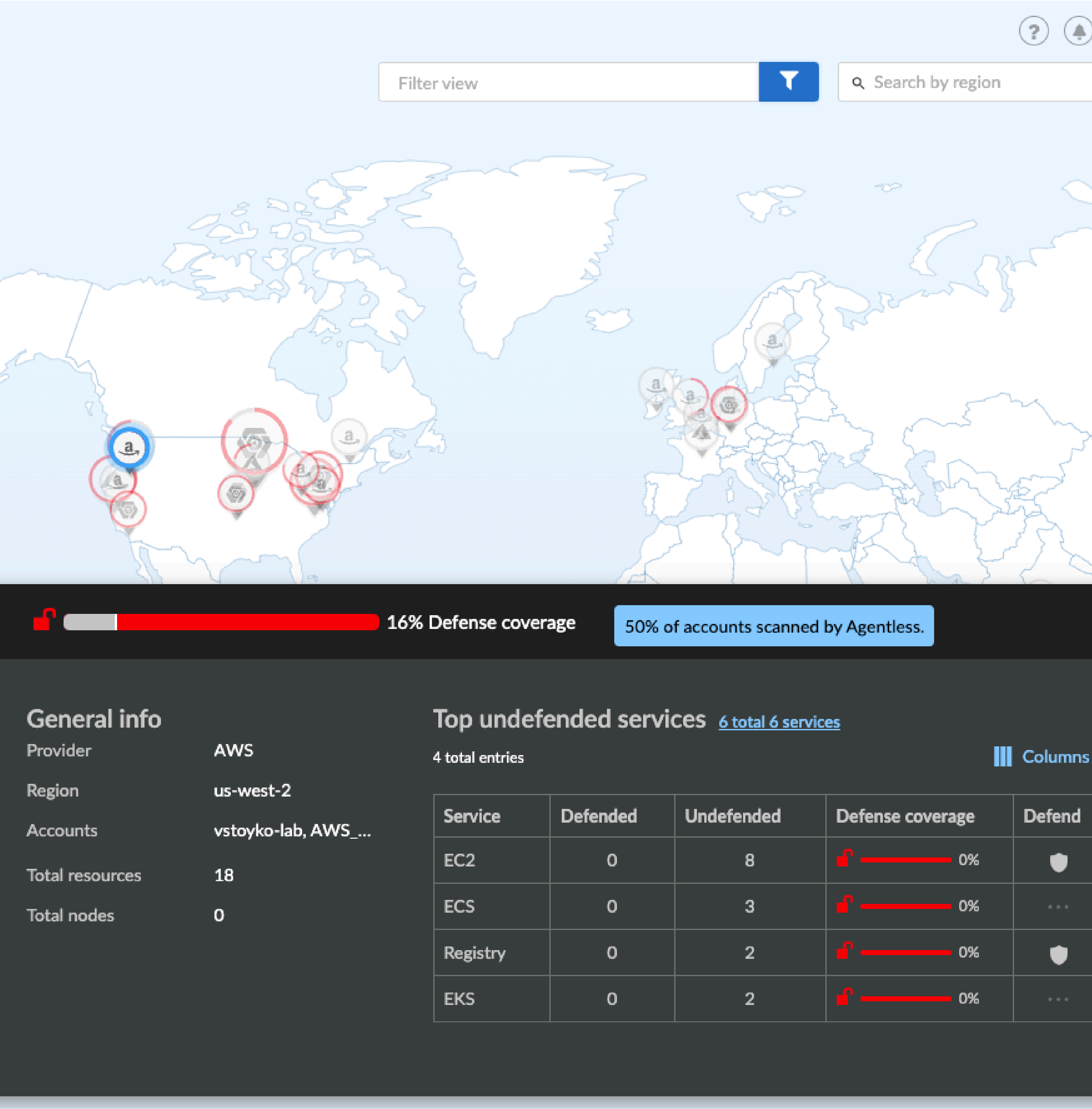
STEP 6 | Review the scan report.

1. Go to **Compute > Manage > Cloud Accounts** to view the scan report as a table.

   1. Select the **Show account details** icon to see the discovery scan results for resources within the cloud account.



2. Go to **Radar** and select **Cloud** to view the scan report as a graphic.



3. Click **Defend** for the entities you want Prisma Cloud to scan for vulnerabilities.

When you click **Defend**, a new scan rule is proposed. Select the appropriate credential, tweak the scan rule as desired, then click **Add**.

4. Go to the scan reports under **Monitor > Vulnerabilities**
5. Select **Hosts**, **Registry**, or **Functions** to see the pertinent report.

Troubleshooting

Ensure you have the right permissions for the account before you start with cloud discovery.

Empty results from Cloud Discovery

Cloud discovery results are visible per account. If you have multiple credentials associated with the same account, the results are only displayed for one credential to avoid duplication. The other credentials for the same account will show empty results. To view comprehensive results for all credentials, navigate to Cloud Radar **Radars > Cloud**.

OSS license management

Edit on GitHub

Prisma Cloud can detect licenses for package dependencies in code repositories. It can scan code repos hosted by service providers (currently GitHub only). It can also scan build folders constructed by CI build jobs.

A license policy defines the criticality of a license. For example, you might specify consider any package with a GPL license as a critical issue. Depending on your license policy, Prisma Cloud can raise alerts and block builds.

Create a license compliance policy

Compliance policies consist of one or more rules.

*Prisma Cloud ships with a default rule named **Default - alert all components**. This rule ships with alerts disabled, so the policy is effectively disabled. As a starting point, consider cloning this rule, and reconfiguring it for your own purposes. Set a threshold, and declare licenses you consider critical. Rule order is important, so be sure your custom rule sits above the default rule.*

STEP 1 | Open Console.

STEP 2 | Go to **Defend > Compliance > Code repositories**.

STEP 3 | Choose the target of your policy.

If your policy targets GitHub, go to the **Repositories** tab.

If your policy targets your CI pipeline, go to the **CI** tab.

STEP 4 | Click **Add rule**.

STEP 5 | Specify a rule name.

STEP 6 | In **Scope**, select one or more collections to apply your policy to specific repos.

Use the default **All** collection to apply it to all repos.

STEP 7 | Set the rule thresholds.

STEP 8 | Specify the severity of each license of interest.

Each field offers SPDX license identifiers as suggestions. Pattern-matching expressions are supported (e.g., *GPL-**).

Scan with twistcli

To scan a folder with twistcli, use the following command:

```
twistcli coderepo scan [FOLDER_PATH] --details
```


Contents of the repo are assessed according to the policy in **Defend > Compliance > Code repositories > CI**. Scan results are published in **Monitor > Compliance > Code repositories > CI**

For CI only, a status column indicates if twistcli passed or failed the build according to the defined policy.

Review scan results.

Go to **Monitor > Compliance > Code repositories**. Each row in the results table has a meter which shows the number of compliance issues at each severity level. Click on a row to drill into the details of the scan report.

Runtime defense

Edit on GitHub

Runtime defense is the set of features that provide predictive protection for containers and threat based active protection for running containers, hosts and serverless functions.

Predictive protection includes capabilities like determining when a container runs a process not included in the origin image or creates an unexpected network socket.

Threat based protection includes capabilities like detecting when malware is added to a workload or when a workload connects to a botnet.

- Runtime defense for containers
- Runtime defense for hosts
- Runtime defense for serverless functions
- Runtime defense for App-Embedded
- Custom runtime rules
- Import and export individual rules
- ATT&CK Explorer
- Runtime Audits
- Event Aggregation
- Image analysis sandbox
- Incident Explorer
- Incident types

Runtime defense for containers

Edit on GitHub

Runtime defense is the set of features that provide both predictive and threat-based active protection for running containers. For example, predictive protection includes capabilities like determining when a container runs a process not included in the origin image or creates an unexpected network socket. Threat-based protection includes capabilities like detecting when malware is added to a container or when a container connects to a botnet.

Prisma Cloud Compute has distinct sensors for file system, network, and process activity. Each sensor is implemented individually, with its own set of rules and alerting. The runtime defense architecture is unified to both simplify the administrator experience and to show more detail about what Prisma Cloud automatically learns from each image. Runtime defense has two principle object types: models and rules.

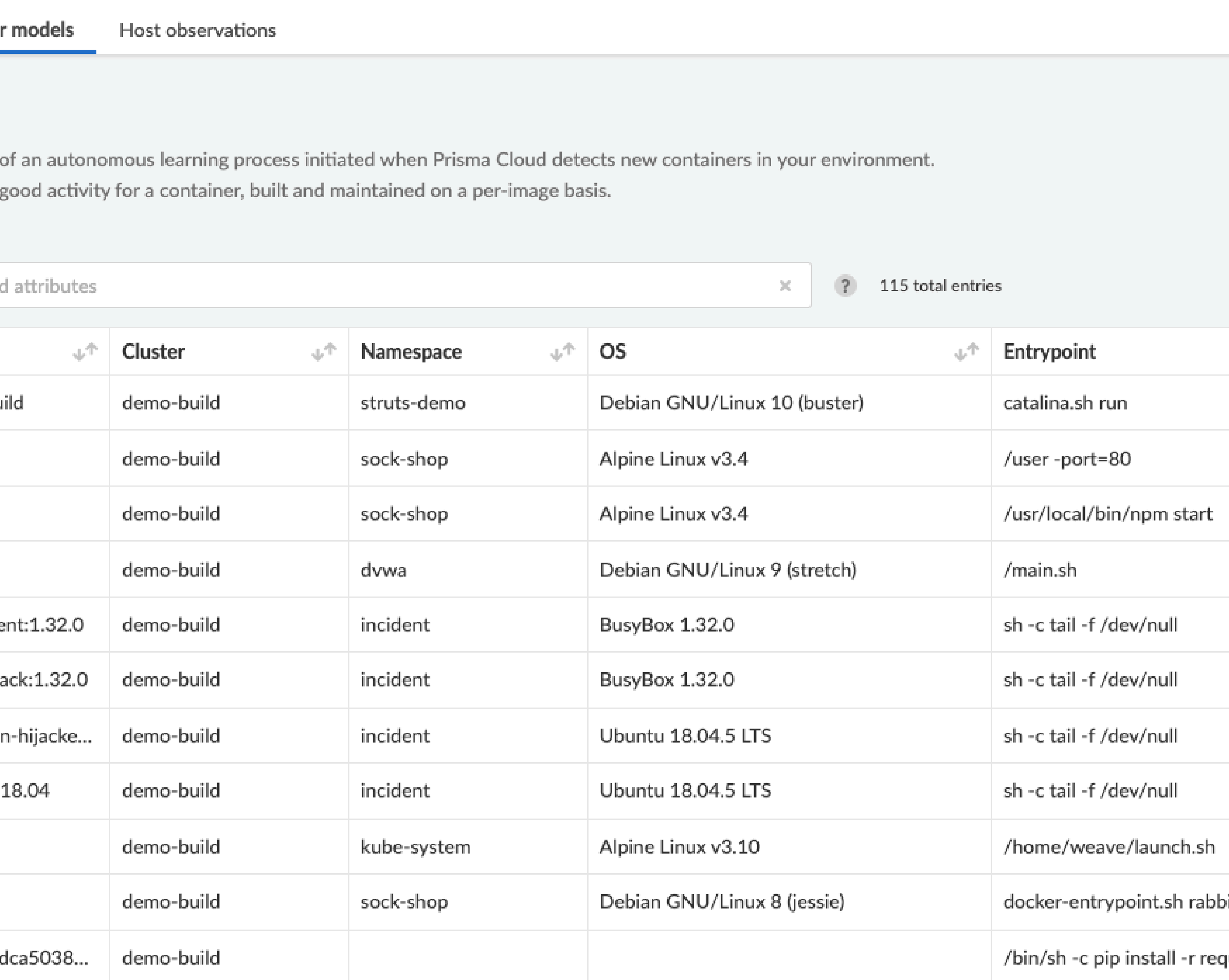
Container Models

Models are the results of the autonomous learning that Prisma Cloud performs every time we see a new image in an environment. A model is the "allow list" for what a given container image should be doing, across all runtime sensors. Models are automatically created and maintained by Prisma Cloud and provide an easy way for administrators to view and understand what Prisma Cloud has learned about their images. For example, a model for an Apache image would detail the specific processes that should run within containers derived from the image and what network sockets should be exposed.

Navigate to **Monitor > Runtime > Container Models**. Click on the image to view the model.

There is a 1:1 relationship between models and images; every image has a model and every model applies to a single unique image. For each image, a unique model is created and mapped to the image digest. So, even if there are multiple images with the same tags, Prisma Cloud will create unique models for each image.

Models are built from both static analysis (such as building a hashed process map based on parsing an init script in a Dockerfile ENTRYPOINT) and dynamic behavioral analysis (such as observing actual process activity during early runtime of the container). Models can be in one of 3 modes: Active, Archived, or Learning.



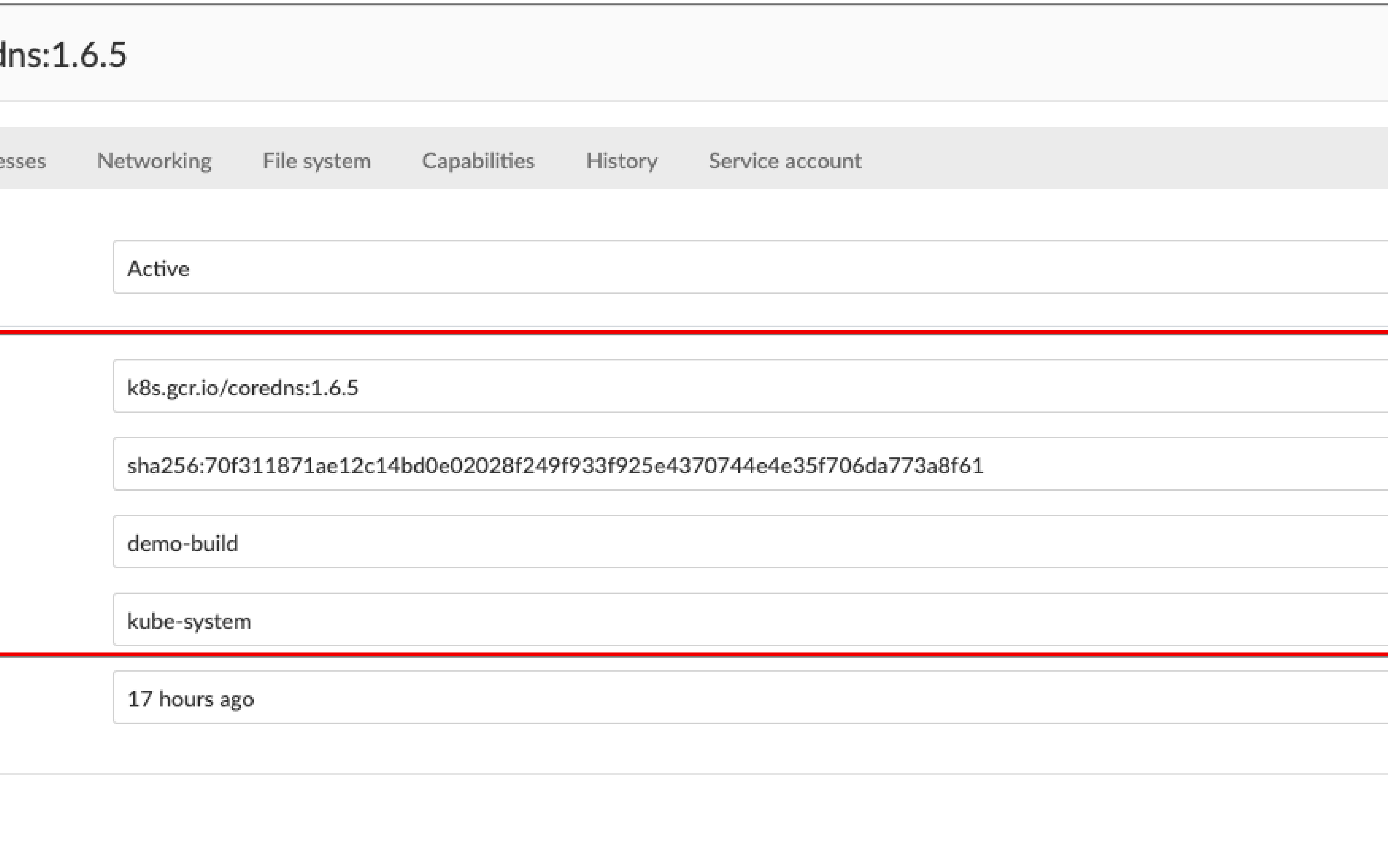
	Cluster	Namespace	OS	Entrypoint
uild	demo-build	struts-demo	Debian GNU/Linux 10 (buster)	catalina.sh run
	demo-build	sock-shop	Alpine Linux v3.4	/user -port=80
	demo-build	sock-shop	Alpine Linux v3.4	/usr/local/bin/npm start
	demo-build	dvwa	Debian GNU/Linux 9 (stretch)	/main.sh
ent:1.32.0	demo-build	incident	BusyBox 1.32.0	sh -c tail -f /dev/null
ack:1.32.0	demo-build	incident	BusyBox 1.32.0	sh -c tail -f /dev/null
n-hijacke...	demo-build	incident	Ubuntu 18.04.5 LTS	sh -c tail -f /dev/null
18.04	demo-build	incident	Ubuntu 18.04.5 LTS	sh -c tail -f /dev/null
	demo-build	kube-system	Alpine Linux v3.10	/home/weave/launch.sh
	demo-build	sock-shop	Debian GNU/Linux 8 (jessie)	docker-entrypoint.sh rabbi
dca5038...	demo-build			/bin/sh -c pip install -r req

For containers in Kubernetes clusters, Prisma Cloud considers the image, namespace, and cluster when creating models.

- When the same image runs in multiple different clusters, Prisma Cloud creates separate models for each image in each cluster.
- When the same image runs in multiple different namespaces, Prisma Cloud creates separate models for each image in each namespace.
- When there are multiple running instances of an image in the same namespace, Prisma Cloud creates a single model.

- When clusters are not applicable, Prisma cloud considers the image and namespace to create models.

Prisma Cloud shows you how models map to specific images. Go to **Monitor > Runtime > Container Models**, click a model in the table, and click the **General** tab.

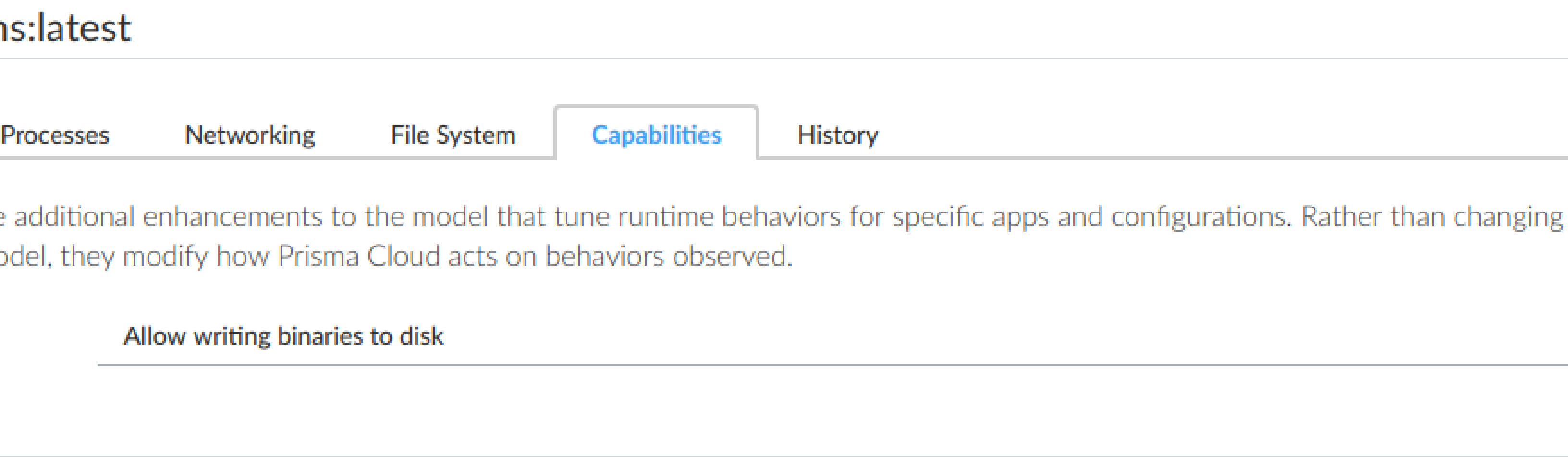


Capabilities

Some containers are difficult to model. For example, Jenkins containers dynamically build and run numerous processes, and the profile of those processes changes depending on what's being built. Constructing accurate models to monitor processes in containers that build, run, test, and deploy software is impractical, although other aspects of the model can still have utility. Prisma Cloud automatically detects known containers, and overrides one more aspects of the model with *capabilities*.

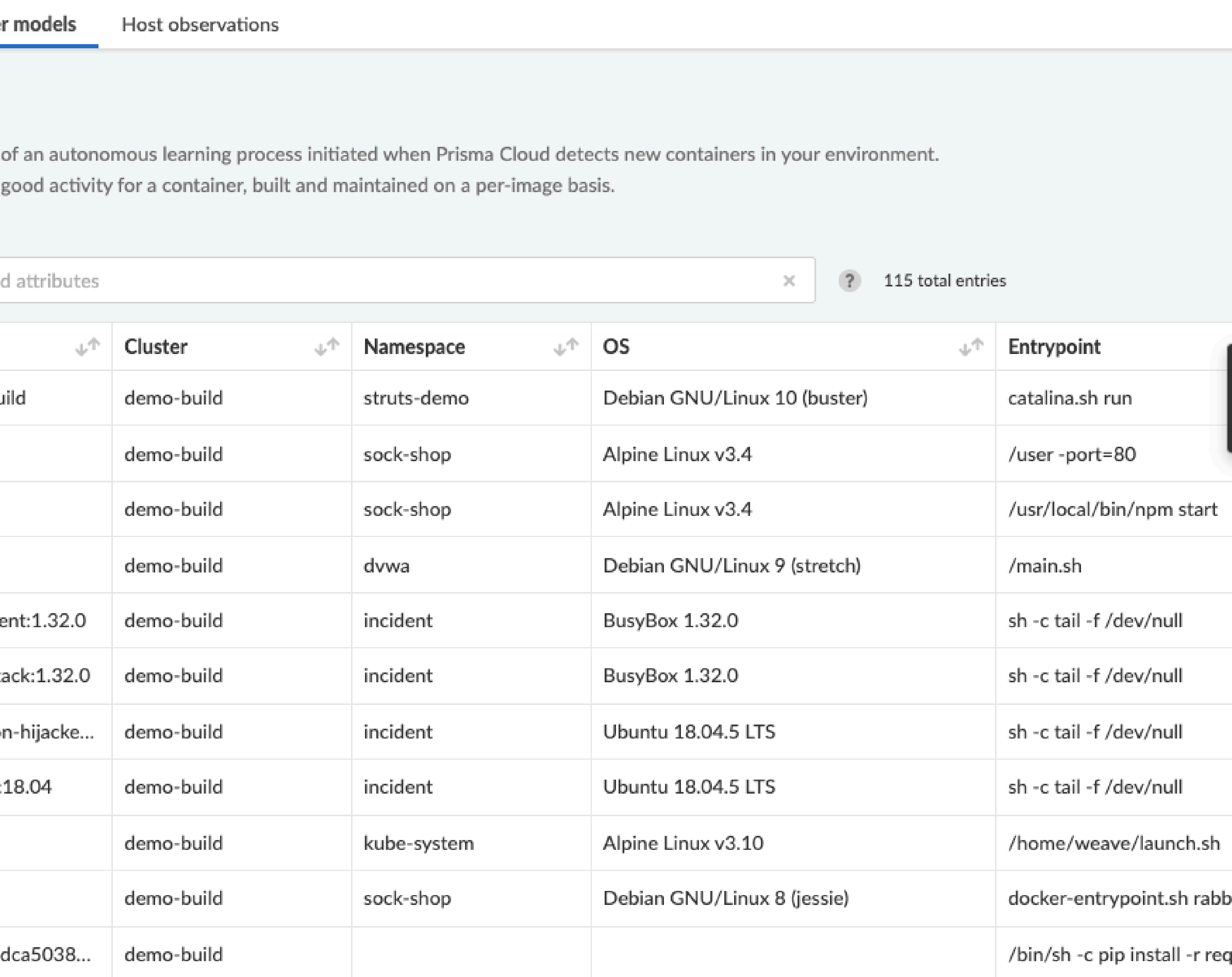
Capabilities are discrete enhancements to the model that tune runtime behaviors for specific apps and configurations. Rather than changing what's learned in the model, they modify how Prisma Cloud acts on observed behaviors.

For example, the following model for the Jenkins container is enhanced with the capability for writing and executing binaries.



Learning mode

Learning mode is the phase in which Prisma Cloud performs either static or dynamic analysis. Because the model depends on behavioral inputs, images stay in learning mode for 1 hour to complete the model. After this 1 hour, Prisma Cloud enters a 'dry run' period for 24 hours to ensure there are no behavioral changes and the model is complete. If during this 24 hours period, behavioral changes are observed, the model goes back to Learning mode for additional 24 hours. The behavioral model uses a combination of machine learning techniques and typically requires less than 1 hour of cumulative observation time for a given image (it might comprise of a single container running the entire learning period or multiple containers running for some time slice where the sum of the slices is 1 hour). During this period, only threat based runtime events (malicious files or connections to high risk IPs) are logged. Prisma Cloud automatically detects when new images are added anywhere in the environment and automatically puts them in learning mode.



- Relearn: You can relearn an existing model by clicking the **Relearn** button in the **Actions** menu. This is an additive process, so any existing static and behavioral modeling remains in place.
- Manual Learning: You can manually alter the duration of learning at any time by starting and stopping the **Manual Learning** option in the **Actions** menu. This should be done with discretion because the model may or may not complete within the time period due to manual interruption. There is no time limit for manual learning. It depends on the user's selection.

Active mode

Active mode is the phase in which Prisma Cloud is actively enforcing the model and looking for anomalies that violate it. Active mode begins after the initial 1 hour that the Learning mode takes to create a model. Because models are explicit allow lists, in enforcing mode, Prisma Cloud is simply looking for variances against the model. For example, if a model predicted that a given image should only run the foo process and Prisma Cloud observes the bar process has spawned, it would be an anomaly. Prisma Cloud automatically transitions models from learning mode into enforcing mode after the model is complete. During this period, runtime events are logged.

During the initial dry run period (the first 24 hours), model may switch automatically from Active mode to Learning mode depending on the behavioral changes observed, as mentioned above. This automatic switching only happens during the first 24 hours of model initiation. If violations are observed later on, they are logged as runtime alerts under Monitor > Runtime.

Archived mode

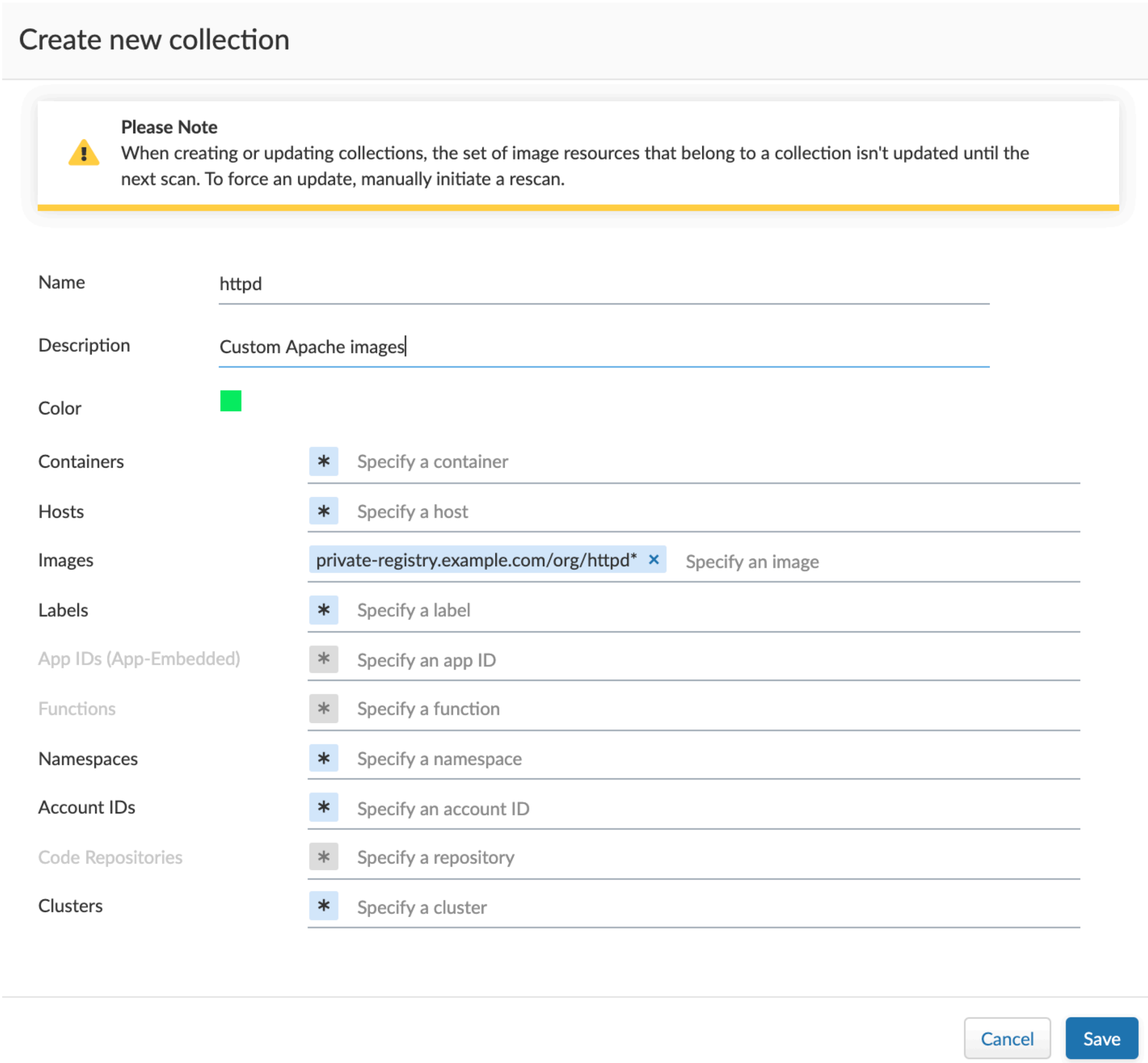
Archived mode is a phase that models are transitioned into when a container is no longer actively running them. Models persist in archived mode for 24 hours after being archived, after which point they're automatically removed. Archived mode serves as a 'recycle bin' for models, ensuring that a given image does not need go through learning mode again if it frequently starts and stops while also ensuring that the list of models does not continuously grow over time.

Models display all the learned data across each of the runtime sensors to make it easy to understand exactly what Prisma Cloud has learned about an image and how it will protect it. However, what if you need to customize the protection for a given image, set of images, or containers? That's the job of rules.

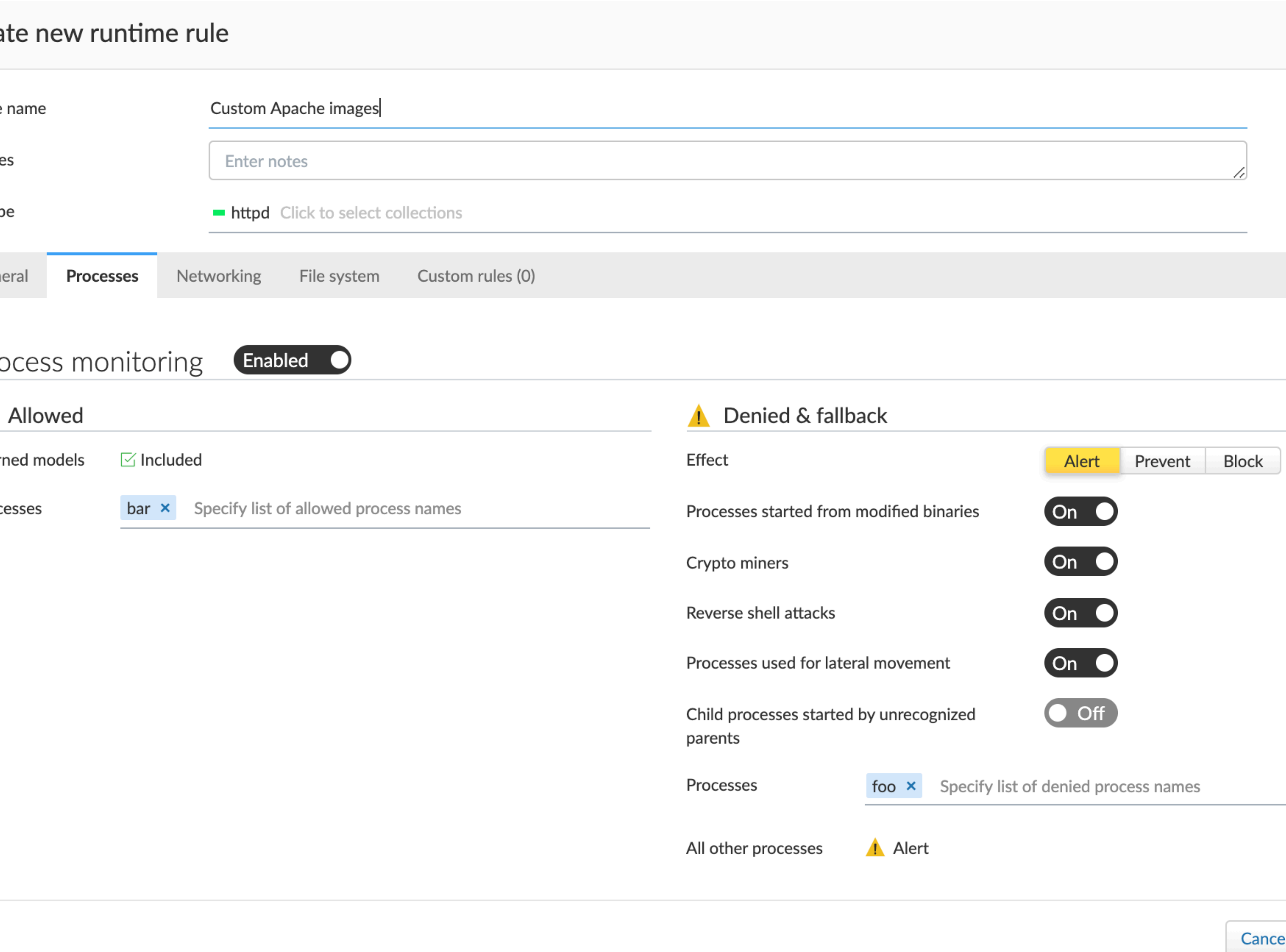
Rules

Rules control how Prisma Cloud uses the autonomously generated models to protect an environment. For example, if Prisma Cloud's model for the Apache image includes the process httpd, but you know that process bar will eventually run and you want to ensure that process foo never runs, you can create a rule that applies to all images named httpd, add bar to the allowed process list, and add foo to the blocked process list.

The following screenshot shows how the scope of the rule is set with collections:



The following screenshot shows how allowed and blocked process activity is set in the rule:



Rules let you explicitly allow and block activity by sensor. Rules and models are evaluated together to create a resultant policy as follows:

model (which contains only allowed activity) + **allowed activity from rule(s) - blocked activity from rule(s)** = **resultant policy**

The resultant policy from the previous example:

model (**httpd**) + allowed activity from rule (**process bar**) - blocked activity from rule (**process foo**) = httpd and bar are allowed and foo always is an anomaly regardless of the model

By default, Prisma Cloud ships with an empty container runtime policy. An empty policy disables runtime defense entirely. To enable runtime defense, create a rule. New runtime rules can be created in Console in **Defend > Runtime > Container policy**.

As with every other subsystem in Prisma Cloud, you can customize how it works by creating rules, scoping rules to desired objects with filtering and pattern matching, and properly ordering the rules in the policy. Rules are evaluated sequentially from top to bottom. Once a match is found for the scope, the actions in the rule are executed and enforced. Only a single rule is ever enforced for a given event. While rules work in conjunction with models as described above, rules themselves are never combined.

Refine your policy by creating rules that target specific resources, enabling or disabling protection features, and defining exceptions to the automatically generated allow-list models.

Discrete blocking

Prisma Cloud lets you create runtime rules that block discrete processes inside a container using the **Prevent** effect. It is an alternative to stopping an entire container when the violation of a runtime rule is detected.

Blocked containers

Prisma Cloud's runtime defense system compares the state of a running container to the predictive model created for it during its learning period. When abnormal activity is detected, such as executing an unknown process, Prisma Cloud can:

- Raise an alert by generating an audit. Audits are shown under **Monitor > Events > Container Audits**. If you have an alert channel configured, such as email or Slack, audits are forwarded there too. Alert is the default action for new runtime rules.
- Block the container by stopping it altogether. To enable blocking, create a new runtime rule.
- Prevent just the discrete process or file system write (not the entire container).

Blocking action

Blocking stops potentially compromised containers from running in your environment.

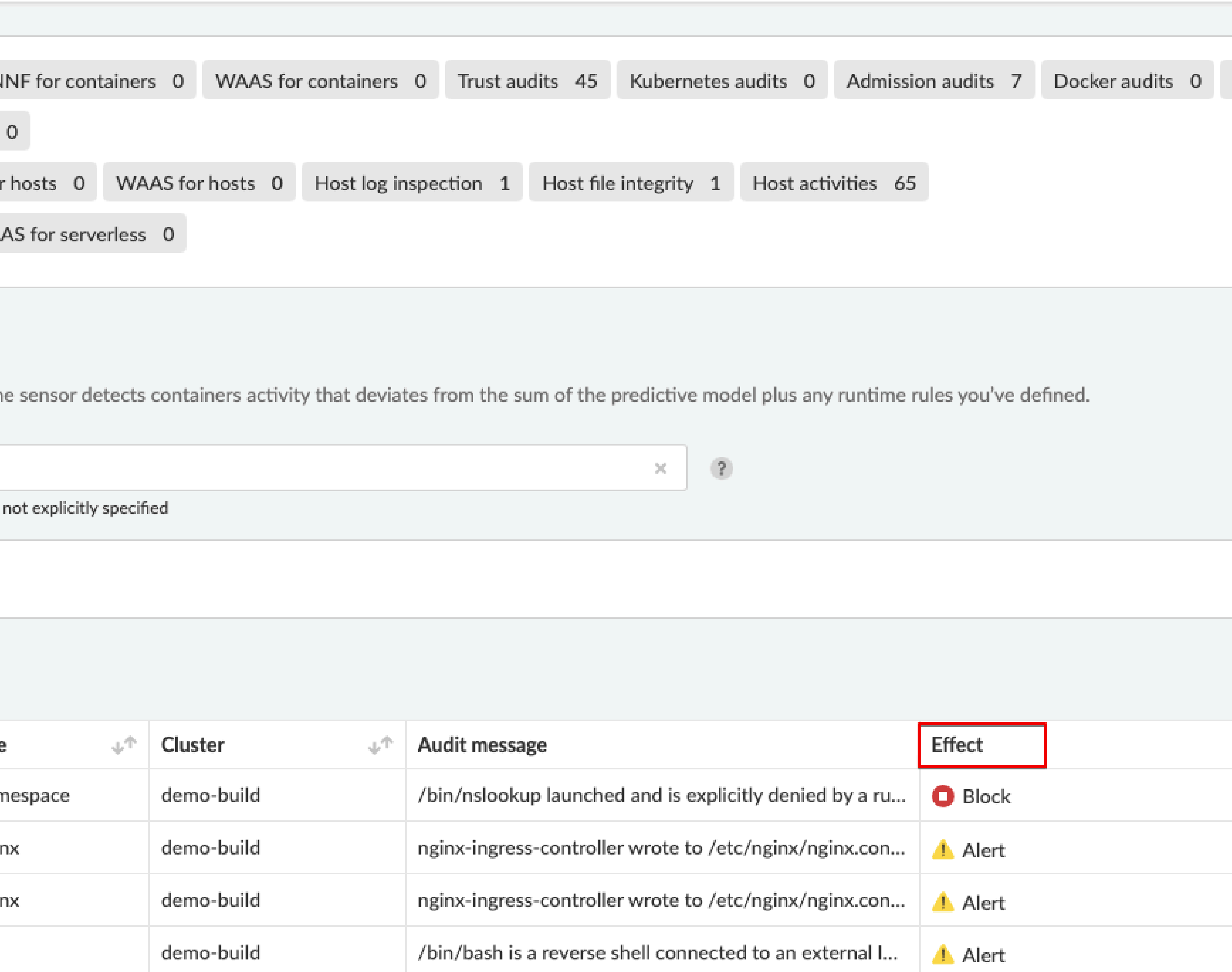
Prisma Cloud blocks containers under the following conditions:

- A container violates its runtime model, and you've installed a runtime rule with the action set to block. For example, if an attacker infiltrates a container and tries to run a port scan using nc, then the container would be blocked if nc weren't a known, allowed process.
- A newly started container violates a vulnerability or compliance rule, and those rules have the action set to block. Prisma Cloud scans all images before they run, to enforce policies about what's allowed to execute in your environment. For example, you policy might call for blocking any container with critical severity vulnerabilities.

Runtime rules can be created under Defend > Runtime > Container Policy. Vulnerability rules can be created under Defend > Vulnerabilities > Policy, and compliance rules can be created under Defend > Compliance > Policy.

Viewing blocked containers

Blocking immediately stops a container, taking it out of service. Blocked containers are never restarted. To see a list of blocked containers, go to the container audits page under **Monitor > Events > Container Audits**.



When a container is stopped, Prisma Cloud takes no further action to keep it stopped. Orchestrators, such as Kubernetes and Openshift, start a fresh container in the blocked container's place. Orchestrators have their own mechanism for maintaining a set point, so they ignore the restart policy defined in the image's Dockerfile.

There is an exception when you run containers in a Docker-only environment (no orchestrator) and Prisma Cloud blocks a container. In this case, Prisma Cloud must take additional action to keep the container blocked. To prevent the container from automatically restarting, Prisma Cloud modifies the container's restart policy to always unless stopped. If you want to unblock a container, connect to the node with the blocked container, and manually modify the container's Docker configuration.

Blocked container artifacts

Forensic investigators can inspect a blocked container's artifacts to determine why it was stopped. You can capture all of the container's contents, including its file system data, with the docker export command. Go to the node with the blocked container and run:

```
$ docker export [container_id] > /path/filename.tar
```

VMware Tanzu Application Service (TAS)

Runtime rules for VMware TAS apps are scoped by app name and space ID. Specify values for app name and space ID in the **Labels** field of the relevant collection. This field is auto-populated with values from your environment.

```
tas-application-name:<value>
tas-space-id:<value>
```

Best practices

One key goal is minimizing the amount of work you're required to do to manage runtime defense. Leverage the models that Prisma Cloud can automatically create and manage. Because behavioral learning for model creation is mature technology for Prisma Cloud, in most cases, you won't need to create auxiliary rules to augment model behavior. There will be some exceptions. For example, a long-running container that changes its behavior throughout its lifecycle might need some manually created rules to fully capture all valid behaviors. This is atypical for most environments, however, as containers that need to be upgraded are typically destroyed and reprovisioned with new images.

If you do need to create runtime rules, here are some best practices for doing so:

Minimize the number of rules — Creating static rules requires time and effort to build and maintain; only create rules where necessary and allow the autonomous models to provide most of the the protection.

Precisely target rules — Be cautious of creating rules that apply to broad sets of images or containers. Providing wide ranging runtime exceptions can lower your overall security by making rules too permissive. Instead, target only the specific containers and images necessary.

Name rules consistently — Because rule names are used in audit events, choose consistent, descriptive names for any rules you create. This simplifies incident response and investigation. Also, consider using Prisma Cloud's alert profile feature to alert specific teams to specific types of events that are detected.

Container runtime policy

Anti-malware

Anti-malware provides high level control for anti-malware capabilities for containers. More granular configuration for each runtime capability is available through each the other tabs on the rule.

- **Prisma Cloud advanced threat protection** — Use Prisma Cloud advanced threat protection intelligence feed, to apply malware prevention techniques across processes, networking and filesystem.
- **Kubernetes attacks** — Monitors attempts to directly access Kubernetes infrastructure from within a running container, including both usage of the Kubernetes administrative tools and attempts to access the Kubernetes metadata.
- **Suspicious queries to cloud provider APIs** — Monitors access to cloud provider metadata API from within a running container.

Advanced malware analysis

- **Use WildFire malware analysis** — Use WildFire, Palo Alto Networks' malware analysis engine, to detect malware. Currently Wildfire analysis is provided without additional costs, but this may change in future releases. To use Wildfire, it must first be enabled.

Processes

This section discusses runtime protection for processes.

Effect

When behavior is detected that deviates from your runtime policy (resultant from the combination of your container model and your rules), Prisma Cloud Defender takes action. For processes, the Defender can be set into one of four modes.

- **Disable** — Defender doesn't provide any protection for processes.
- **Alert** — Defender raises alerts when it detects process activity that deviates from your defined runtime policy. These alerts are visible in **Monitor > Events > Container Audits**.
- **Prevent** — Defender stops the process (and just the process) that violates your policy from executing. This is known as discrete blocking.

  Prisma Cloud runtime rules let you deny specific processes. When you specify the **Prevent** action in a runtime rule, Prisma Cloud blocks containers from running processes that are not defined in the model or the explicitly allowed processes list. The rest of the container continues to execute without disruption. The alternative to discrete blocking is container blocking, which stops the entire container when a denied process is detected.

  *The **Prevent** action is not supported on Debian 8.*

- **Block** — Defender stops the entire container if a process that violates your policy attempts to run.

Note that besides taking action on processes outside of the allow-list model, Defender also takes action when existing binaries that have been modified are executed. For example, an attacker might replace httpd (Apache) with an older version that can be exploited. Prisma Cloud raises alerts for each of the following cases:

- A modified binary is executed,
- A modified binary listens on a port,
- A modified binary makes an outbound connection.

Detections

Prisma Cloud can detect anomalous process activity. These features can be independently enabled or disabled.

- **Allow all activity in attached sessions** — Bypass runtime rules when attaching to running containers or pods. This control lets developers and DevOps engineers troubleshoot and investigate issues in containers and pods without generating spurious audits or being stymied by block/prevent controls. It applies to all types of attach sessions, including *kubectl exec* and *docker exec*. Only Linux containers are supported; Windows containers aren't supported.

  Note that this control bypasses all runtime activity - process, network, and file system - even though it's situated in the process tab.

  The following event types can't be bypassed by this control: DNS queries, listening ports, and raw sockets. For these types of events, activity in the attach session won't be allowed if set in your policy.
- **Processes started from modified binaries** — Detect when binaries from a container image have been modified and executed.
- **Crypto miners** — Prisma Cloud can detect crypto miners. If detected, a crypto miner incident type is created in Incident Explorer. When this option is enabled, Defender takes action on this type of incident according to the configured effect.
- **Reverse shell attacks** — Detect usage of reverse shell.
- **Detect processes used for lateral movement** — Prisma Cloud can detect processes, such as netcat, known to facilitate lateral movement between resources on a network. If detected, a lateral movement incident type is created in Incident Explorer. When this option is enabled, Defender takes action on this type of incident according to the configured effect.
- **Child processes started by unrecognized parents** — As part of the model, Prisma Cloud learns what processes are invoked, and the parent processes that triggered the invocation. If this option is enabled, Defender can act on processes that are invoked by a parent other than that which is specified by the model. This action may show up as an audit in a number of different incident types in Incident Explorer.
- **Processes started with SUID** — Detect suspicious privilege escalation by watching for binaries with the setuid bit.

  Explictly allowed processes from your runtime policy and learned processes from your runtime models bypass this control. For example, if *ping* is added to the container's runtime model during the learning period, *ping* is permitted to run regardless of how this control is set. However, if *ls* is explicitly permitted by your policy, but *sudo ls* is detected, this control flags the privilege escalation. If you explicitly allow *sudo*, and then run *sudo ls*, this control is bypassed.
- **Explicitly allowed and denied processes** — The fields for **Explicitly allowed processes** and **Explicitly denied processes** let you tailor your runtime models. Processes can be listed by name or MD5 hash.

Runtime container models

Container models are the product of an autonomous learning process initiated when Prisma Cloud detects new containers in your environment. A model is an 'allow list' of known good activity for a container, built and maintained on a per-image basis. You can see the domains in the model by going to **Monitor > Runtime > Container Models**, clicking on a model, then opening the **Process** tab.

- **Static container models**—processes that were scanned in the first scan during the container loading.
- **Behavioral container models**—processes that were scanned in the learning period that are not static.
- **Extended behavioral container models**—processes detected after the learning period, where Prisma Cloud identifies them as "low severity". These types of processes will be also added to the model An alert is raised only once with a message saying there is a low likelihood that this process is malicious and no further alerts for this type of event will be raised. Extended behavioral processes are added to the extended behavioral table in **Monitor > Runtime > Container Models** in the process tab in the extended behavioral section.

Networking

Prisma Cloud can monitor container networking activity for patterns that indicate an attack might be underway. These features can be independently enabled or disabled with runtime rules. The final policy that's enforced is the sum of the container model and your runtime rules.

IP connectivity

When Prisma Cloud detects an outgoing connection that deviates from your runtime policy, Prisma Cloud Defender can take action. Networking rules let you put Defender into one of three modes:

- **Disable**—Defender does not provide any networking protection.
- **Alert**—Defender raises alerts when targeted resources establish connections that violate your runtime policy. The corresponding audits can be reviewed under **Monitor > Events > Container Audits**.
- **Block**—Defender stops the container if it establishes a connection that violates your runtime policy. The corresponding audit can be reviewed under **Monitor > Events > Container Audits**.

The fields for **Explicitly allowed** and **Explicitly denied** let you tailor the runtime models for known good and known bad network connections. These rules define the policy for listening ports, outbound internet ports for Internet destinations, and outbound IP addresses. Defining network policy through runtime rules lets you specify permitted and forbidden behavior for given resources, and instructs Defender on how to handle traffic that deviates from the resultant policy.

- **Detect port scanning**—Port scans are used by attackers to find which ports on a network are open and listening. If enabled, Defenders detect network behavior indicative of port scanning. If detected, a port scanning incident is created in Incident Explorer.
- **Raw sockets**—Prisma Cloud can monitor your environment for raw sockets, which can indicate suspicious activity. Raw sockets let programs manipulate packet headers and implement custom protocols to do things such as port scanning. Raw socket detection is enabled by default in new rules.

DNS

Modern attacks, particularly coordinated, long running attacks, use short lived DNS names to route traffic from the victim's environment to command and control systems. This is common in large scale botnets. When DNS monitoring is enabled (Alert, Prevent, or Block) in your runtime rules, Prisma Cloud analyzes DNS lookups from your running containers. By default, DNS monitoring is disabled in new rules.

Dangerous domains are detected as follows:

- **Prisma Cloud Intelligence Stream** — Prisma Cloud's threat feed contains a list of known bad domains.
- **Behavioral container models** — When learning a model for a container, Prisma Cloud records any DNS resolutions that a container makes. When the model is activated, Defender monitors network traffic for DNS resolutions that deviate from the learned DNS resolutions.

  You can see the domains in the model by going to **Monitor > Runtime > Container Models**, clicking on a model, then opening the **Networking** tab. Known good domains are listed under **Behaviorally learned domains**.
- **Extended behavioral container models** — network traffic detected after the learning period, which Prisma Cloud identifies as "low severity". This traffic will be also added to the model. An alert is raised only once with a message saying there is a low likelihood that this event is malicious and no further alert for this type of event will be raised.
- **Explicit allow and deny lists:** Runtime rules let you augment the Prisma Cloud's Intelligence Stream data and models with your own explicit lists of known good and bad domains. Define these lists in your runtime rules.

In your runtime rules, set **Effect** in the DNS section to configure how Defender handles DNS lookups from containers:

- **Disable:** DNS monitoring is disabled. DNS lookups are not modeled in learning mode. DNS lookups aren't analyzed when models are active.
- **Alert:** DNS monitoring is enabled. DNS lookups are modeled in learning mode. DNS lookups are analyzed when models are active. Anomalous activity generates audits.
- **Prevent:** DNS monitoring is enabled. DNS lookups are modeled in learning mode. DNS lookups are analyzed when models are active. Anomalous activity generates audits. Anomalous DNS lookups are dropped.
- **Block** DNS monitoring is enabled. DNS lookups are modeled in learning mode. DNS lookups are analyzed when models are active. Anomalous activity generates audits. When anomalous DNS lookups are detected, the entire container is stopped.

File system

Prisma Cloud's runtime defense for container file systems continuously monitors and protects containers from suspicious file system activities and malware.

Prisma Cloud monitors and protects against the following types of suspicious file system activity:

- Changes to any file in folders *not* in the runtime model.
- Changes to binaries or certificates anywhere in the container.
- Changes to SSH administrative account configuration files anywhere in the container.
- Presence of malware anywhere in the container.

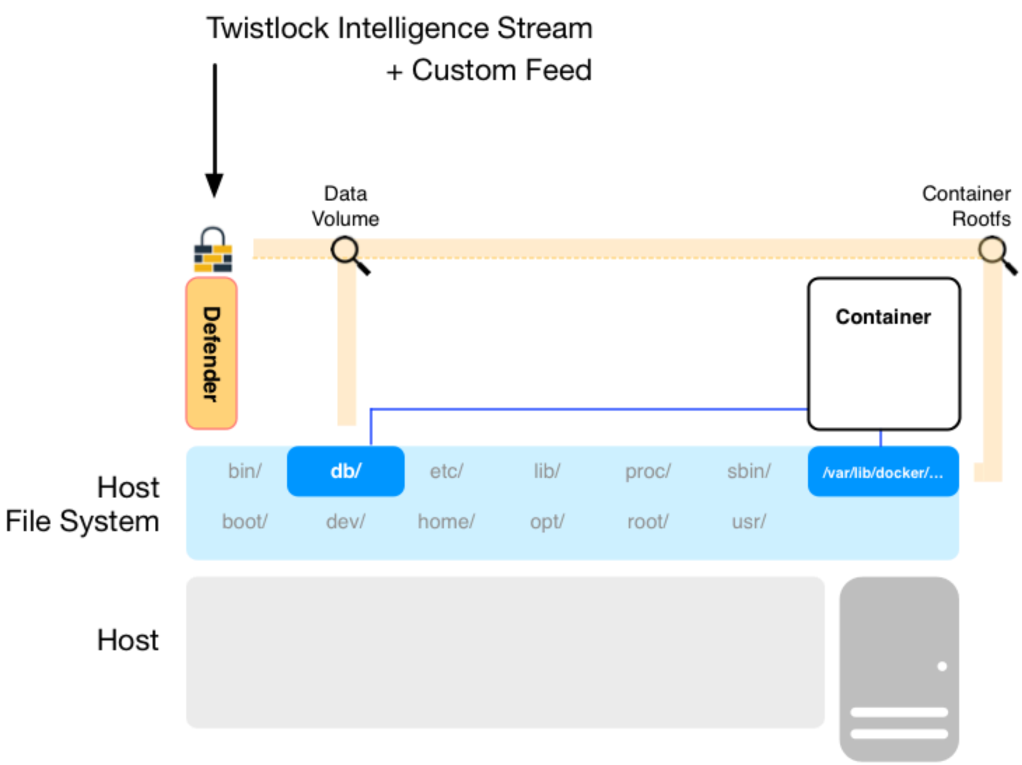
Malware protection

Defender monitors container file systems for malicious certs and binaries using data from the Prisma Cloud Intelligence Stream. Console receives the Prisma Cloud feed, and then distributes it to all deployed Defenders. You can optionally supplement the Prisma Cloud feed with your own custom data.

When a file is written to the container file system, Defender compares the MD5 hash of the file to the MD5 hash of known malware. If there is a match, Defender takes the action specified in your rules. Defender also looks for attributes that make files suspicious, including signs they've been rigged for anti-analysis.

By default, new rules configure Defender to monitor both the container root file system and any data volumes. Container root file systems reside on the host file system. In this diagram, the running container also has a data volume. It mounts the db/ directory from the host file system into its own root file system. Both locations are monitored by Defender.

The following diagram shows how Prisma Cloud protects containers from malicious files:



Effect

When behavior is detected that deviates from your runtime policy (resultant from the combination of your container model and your rules), Prisma Cloud Defender takes action. For processes, the Defender can be set into one of four modes.

- **Disable** — Defender doesn't provide any protection for file system.
- **Alert** — Defender raises alerts when it detects file system activity that deviates from your defined runtime policy. These alerts are visible in **Monitor > Events > Container Audits**.
- **Prevent** — Defender stops the process (and just the process) that violates your policy from executing. This is known as discrete blocking. Prisma Cloud also lets you deny file system writes to specific directories. Like the process rule, file system rules can be configured with the **Prevent** action, which blocks the creation and modification of any files in the specified directories. This mechanism is designed to prevent bad actors from writing certificates or

binary attack tools to disk, all without killing the process that initiated the write or stopping the entire container.

*The **Prevent** action in file system rules is not supported for some kernel types. If you specify a **Prevent** action, but the kernel does not support it, you will be alerted with an audit but the activity will not be prevented. The audit message will state that Prevent is not supported.*

*The **Prevent** action in file system rules is not supported when the Docker storage driver is set to aufs. It is supported for other storage drivers, such as devicemapper and overlay2. If you specify a **Prevent** action, but the storage driver does not support it, Prisma Cloud will respond with an alert and log the following message in Defender's log: Docker storage driver on host doesn't support discrete file blocking.*

*For the "Changes to binaries", "Detection of encrypted/packed binaries", and "Binaries with suspicious ELF headers" detections, the **Prevent** effect is only supported for existing files that are being modified. This is because these detections rely on the file content. When the file is new, it is empty so it cannot be identified by with one of these detections. On such cases, you are alerted with an audit but the activity is not prevented. The audit message will state that Prevent is not supported.*

- **Block** — Defender stops the entire container if a process that violates your policy attempts to run.

Detections

Prisma Cloud can detect anomalous file system activity. These features can be independently enabled or disabled.

- **Changes to binaries** — Detect when binaries from a container image are modified.
- **Detection of encrypted/packed binaries** — Detect usage of encrypted/packed binaries. Such files are alerted on as encrypted and packed binaries may be used as a method to deploy malware undetected.
- **Changes to SSH and admin account configuration files**
- **Binaries with suspicious ELF headers**
- **Explicitly allowed and denied system paths** — The fields for **Explicitly allowed paths** and **Explicitly denied paths** let you tailor your runtime models, by explicitly denying paths in the model or explicitly allowing paths that aren't in the model.
- **Extended behavioral container models** — Suspicious file system activities that are detected after the learning period, which Prisma Cloud algorithm identifies as "low severity". These activities are also added to the model. An alert will be only raised once with a message saying there is a low likelihood that this event is malicious, and no further alerts for this type of event will be raised.

Custom rules

For details on custom rules policy refer to this section.

Runtime defense for hosts

Edit on GitHub

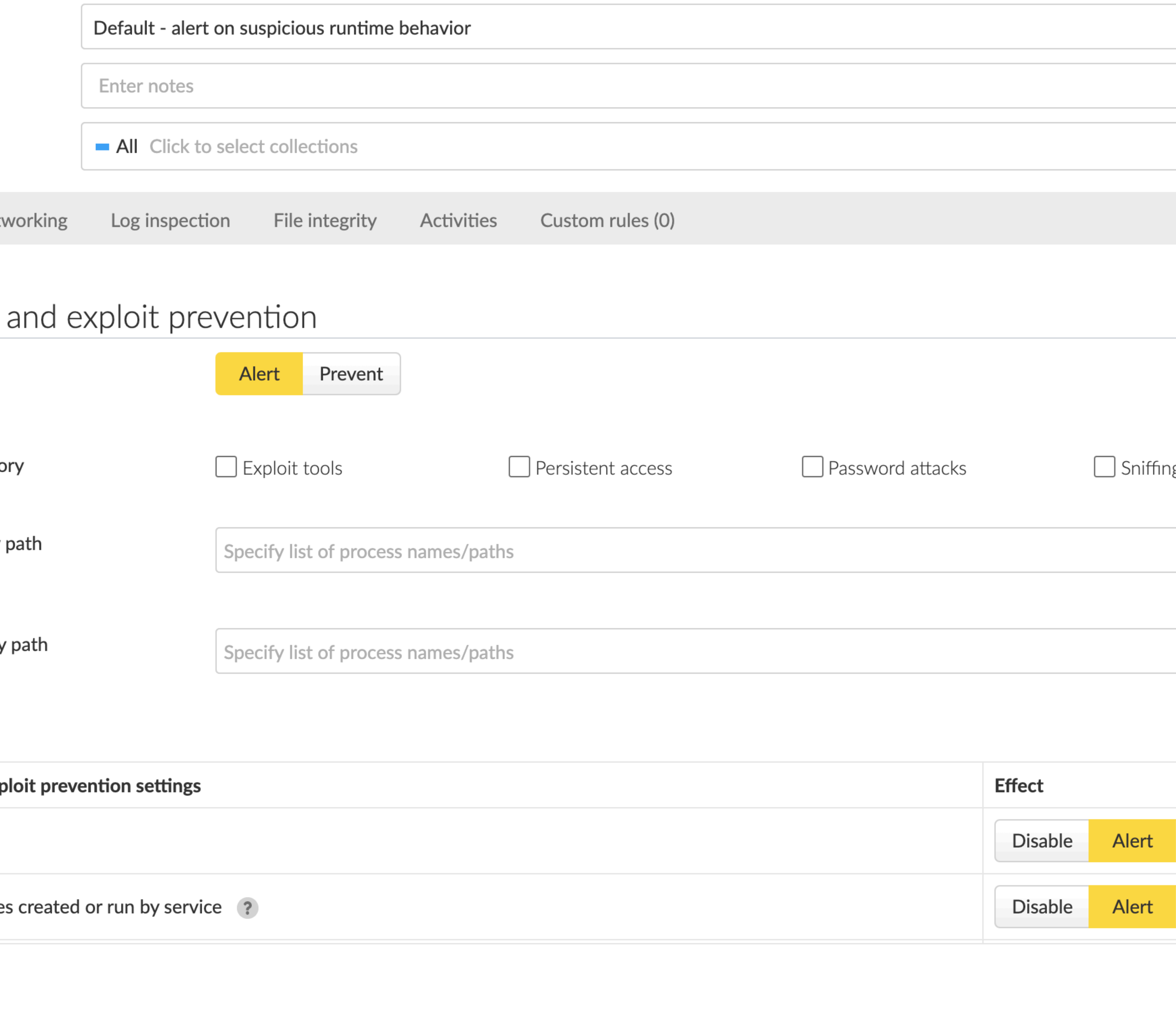
Without secure hosts, you cannot have secure containers. Host machines are a critical component in the container environment, and they must be secured with the same care as containers. Prisma Cloud Defender collects data about your hosts for monitoring and analysis.

Runtime host protection is designed to continuously report an up-to-date context for your hosts. You can set detection for malware, network, log inspection, file integrity, activities and custom events. Some of the detected events can only be alerted on, while others can be prevented.

Host runtime policy

By default, Prisma Cloud ships with an empty host runtime policy. An empty policy disables runtime defense entirely.

Creating a new rule enables runtime defense. When Defender is installed, it automatically starts collecting data about the underlying host. To create a rule, open Console, go to **Defend > Runtime > Host Policy**, and click **Add rule**. Create new rules to enhance host protection.



- Rules are assigned with names to provide an indication of target of each rules.
- The scope of each rule is determined by the collection assigned to that rule.
- Prisma Cloud uses rule order and pattern matching to determine which rule to apply for each workload.

Anti-malware provides a set a capabilities that lets you alert or prevent malware activity and exploit attempts.

*The **Prevent** action for detection of file system events requires a Linux kernel version 4.20 or later.*

Anti-malware

Global settings

- **Alert/prevent processes by path** — Provides the ability to alert on or prevent execution of specific processes based on the processes name or the full path of binary from which the process is executed. Some of the common tools are available for easy addition by selecting their category.
- **Allow processes by path** — Provides the ability to mark processes as safe to use based on the process name or full path. Processes added to this list will not be alerted on or prevented by any of the Malware runtime capabilities.

  The above two fields are evaluated together to create a resultant policy: **Final allowed paths** = **Allow paths** - **Alert/prevent paths**

Anti-malware and exploit prevention settings

- **Crypto miners** — Apply specific techniques for detection of crypto miners, alert on file creation, and alert or prevent their execution.
- **Non-packaged binaries created or run by service** — Detect binaries created by a service without a package manager. Alert on file creation, and alert or prevent their execution.

  Defender must be running when a file is written to detect its source.

  To detect binaries that have been deployed without a package manager, Prisma Cloud depends on the package manager on the host. Currently, apt, yum, and dnf are supported.
- **Non-packaged binaries created or run by user** — Detect binaries created by a user without a package manager. Alert on file creation, and alert or prevent their execution.

  Defender must be running when a file is written to detect its source.

  To detect binaries that have been deployed without a package manager, Prisma Cloud depends on the package manager on the host. Currently, apt, yum, and dnf are supported.
- **Processes running from temporary storage** — Detect processes running from temporary storage (unexpected behavior for legitimate processes). Alert/prevent on file creation or execution.
- **Webshell attacks** — Detect abuse of web servers vulnerabilities to create a webshell. Alert on webshell creation and and alert or prevent execution of linux command line tools from web servers.
- **Reverse shell attacks** — Detect usage of reverse shell and generate an alert.
- **Execution flow hijack** — Detect execution flow hijack attempt and generate an alert.

- **Encrypted/packed binaries** — Detect usage of encrypted/packed binaries and generate an alert. Such files are alerted on as encrypted and packed binaries may be used as a method to deploy malware undetected.
- **Binaries with suspicious ELF headers** — Detect suspicious binaries for ELF headers and generate an alert.
- **Malware based on custom feeds** — Generate alerts for files classified as malware by their MD5
- **Malware based on Prisma Cloud advanced threat** — Generate alerts for files classified as malware by Prisma Cloud advanced intelligence feed

Advanced malware analysis

- **Malware based on WildFire analysis** — Use WildFire, Palo Alto Networks' malware analysis engine, to detect malware and generate alerts. Currently Wildfire analysis is provided without additional costs, but this may change in future releases. To use Wildfire, it must first be enabled.

Host observations

- **Track SSH events** — As part of the host observation capability, we are also full tracking all SSH activities, which is enabled by default in new rules. Tracking can be disabled via this toggle.

Networking

Networking provides customers high level of granularity in controlling network traffic based on IP, port and DNS. Customers can use their own custom rules or use Prisma Cloud advanced threat protection to alert on or prevent access to malicious sites.

IP connectivity

- *Allowed IPs: — create an approved list of IPs which access to will not generate an alert.
- **Denied IPs and ports** — Create lists of listening ports, outbound internet ports and outbound IPs which access to would generate an alert.
- **Suspicious IPs based on custom feed** — Generate alerts based on entries added to the list of suspicious or high risk IP endpoints under **Manage > System > Custom feeds > IP reputation lists**
- **Suspicious IPs based on Prisma Cloud advanced threat protection** — Generate alerts based on the Prisma Cloud advanced threat protection intelligence stream.

DNS

When DNS monitoring is enabled, Prisma Cloud filters DNS lookups. By default, DNS monitoring is disabled in new rules.

- **Allowed domains** — Create an approved list of domains which access to will not generate an alert or be prevented.
- **Denied domains** — Create a list of denied domains which access to will be alerted or prevented.
- **Suspicious domains based on Prisma Cloud Advanced threat protection** — Generate alerts or prevent access to domains based on the Prisma Cloud advanced threat protection intelligence stream.

Log inspection

Prisma Cloud lets you collect and analyze logs from operating systems and applications for security events. For each inspection rule, specify the log file to parse and any number of inspection expressions. Inspection expressions support the RE2 regular expression syntax.

A number of predefined rules are provided for apps such as sshd, mongod, and nginx.

Regardless of the specified inspection expression, log inspection has the following boundaries.

- The maximum amount of bytes read per second is *100*.
- The maximum amount of bytes in a chunk read per second is *2048*.

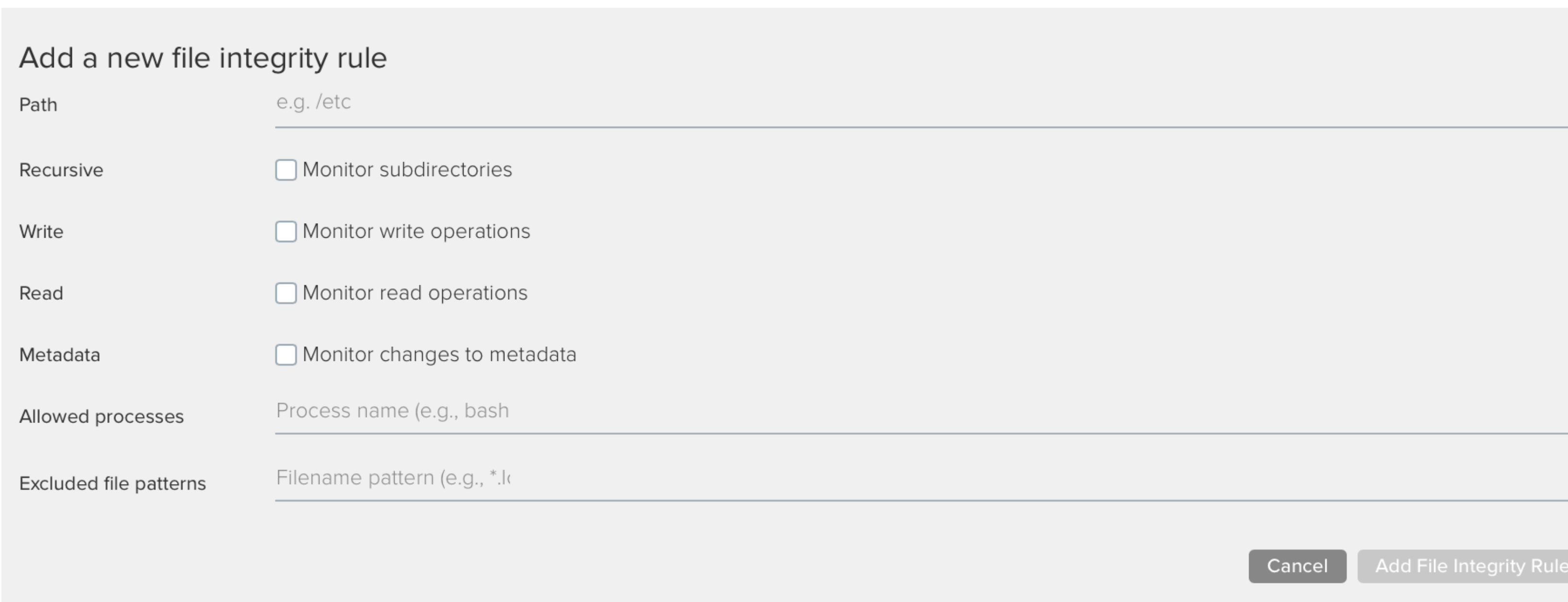
These boundaries are non-customizable.

File integrity management (FIM)

Changes to critical files can reduce your overall security posture, and they can be the first indicator of an attack in progress. Prisma Cloud FIM continually watches the files and directories in your monitoring profile for changes. You can configure to FIM to detect:

- Reads or writes to sensitive files, such as certificates, secrets, and configuration files.
- Binaries written to the file system.
- Abnormally installed software. For example, files written to a file system by programs other than apt-get.

A monitoring profile consists of rules, where each rule specifies the path to monitor, the file operation, and exceptions.



The file operations supported are:

- Writes to files or directories. When you specify a directory, recursive monitoring is supported.
- Reads. When you specify a directory, recursive monitoring isn't supported.

- Attribute changes. The attributes watched are permissions, ownership, timestamps, and links. When you specify a directory, recursive monitoring isn't supported.

Activities

Set up rules to audit host events.

Custom rules

For details on custom rules policy refer to this section.

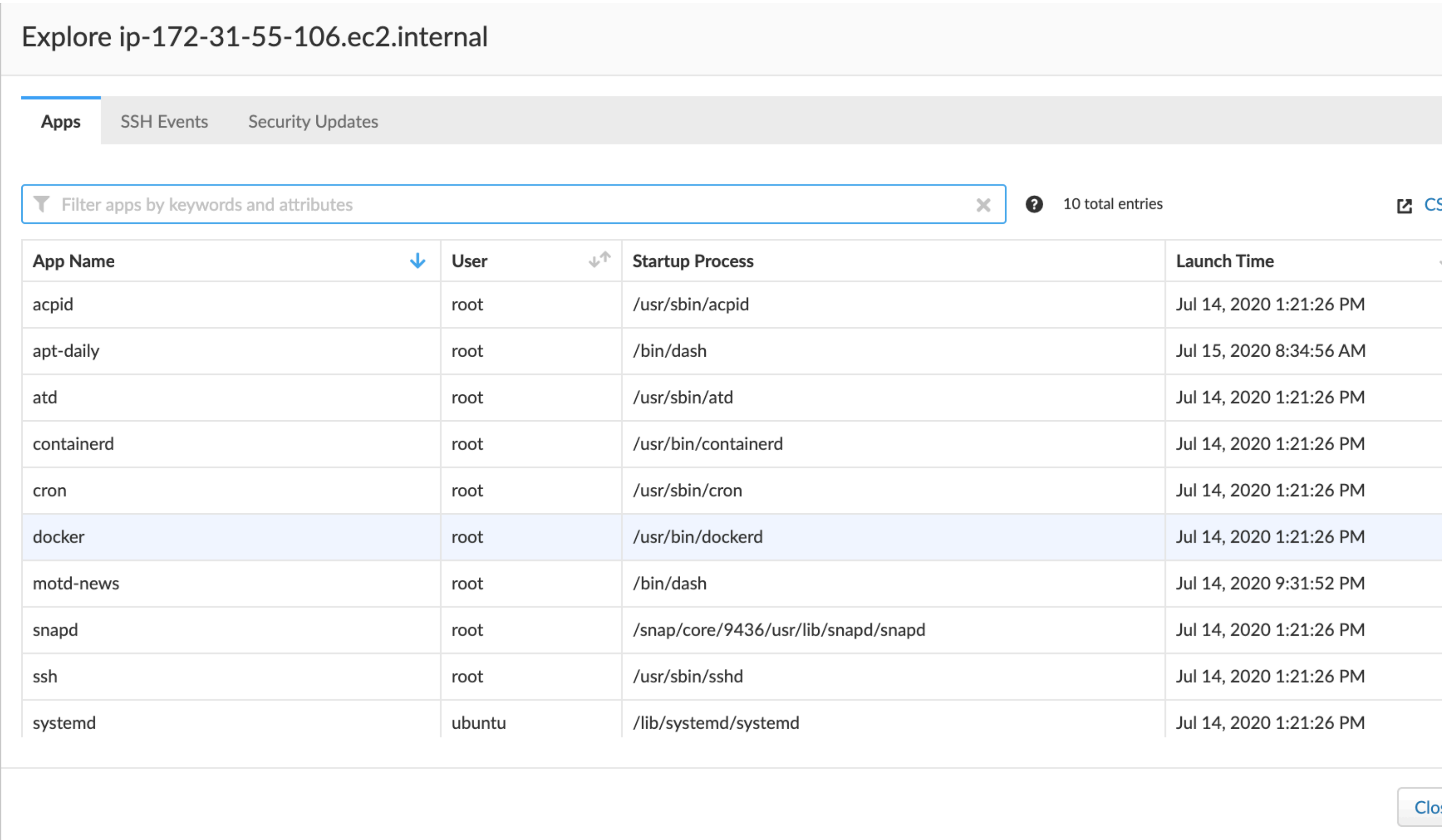
Monitoring

To view the data collected about each host, go to **Monitor > Runtime > Host Observations**, and select a host from the table.

Apps

The **Apps** tab lists the running programs on the host. New apps are added to the list only on a network event.

Prisma Cloud automatically adds some important apps to the monitoring table even if they don't have any network activity, including cron and systemd.



For each app, Prisma Cloud records the following details:

- Running processes (limited to 10).
- Outgoing ports (limited to 5).
- Listening ports (limited to 5).

Prisma Cloud keeps a sample of spawned processes and network activity for each monitored app, specifically:

- Spawned process — Processes spawned by the app, including observation timestamps, user name, process (and parent process) paths, and the executed command line (limited to 10 processes).
- Outgoing ports — Ports used by the app for outgoing network activity, including observation timestamps, the process that triggered the network activery, IP address, port, and country resolution for public IPs (limited to 5 ports).
- Listening ports — Ports used by the app for incoming network activity, including the listening process and observation timestamps (limited to 5 ports).

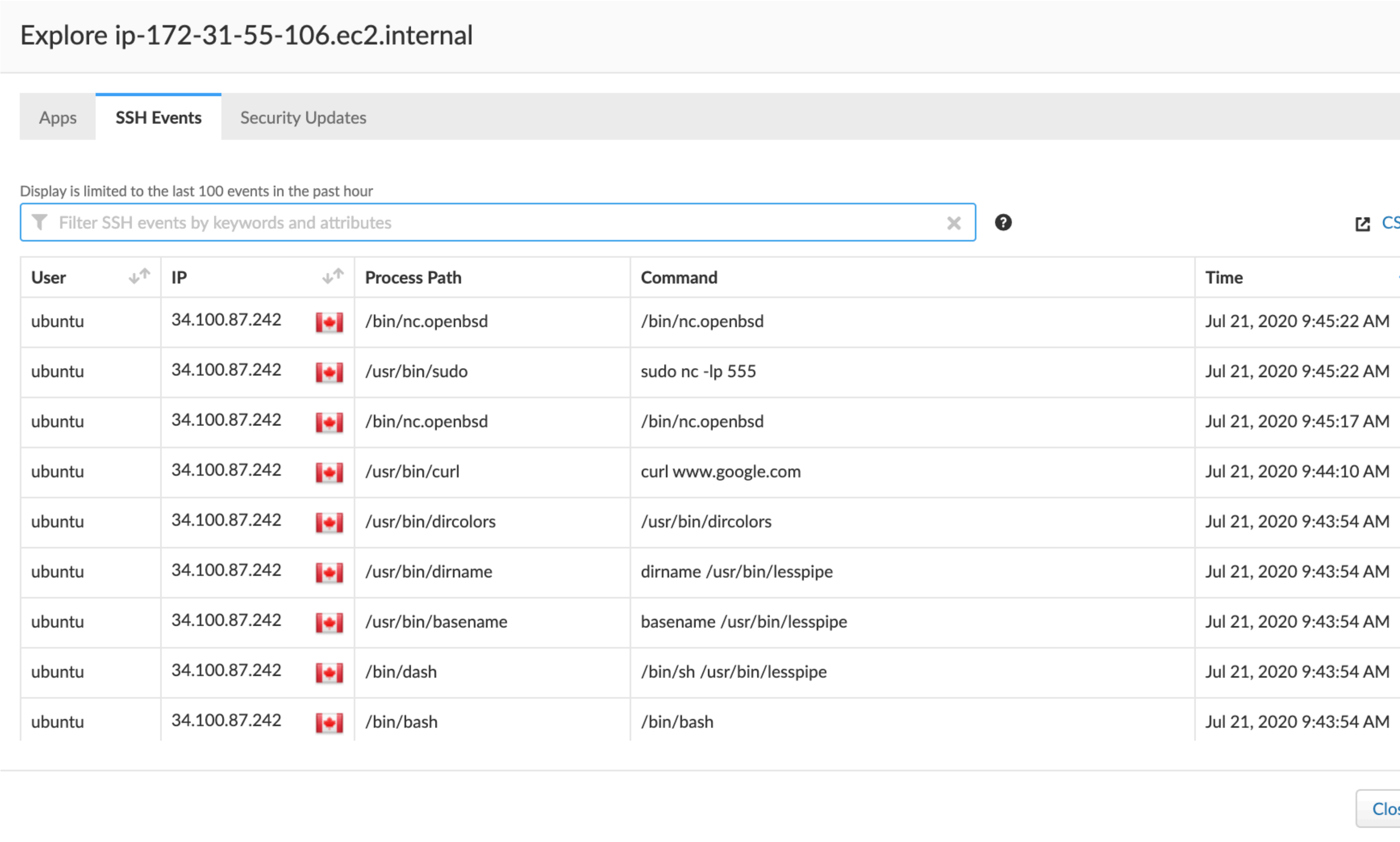
Proc events will add the proc only to existing apps in the profile. Defender will cache the runtime data, saving timestamps for each of the 10 processes last spawn time.

Limitations:

- Maximum of 100 apps.
- Last 10 spawned processes for each app.

SSH session history

The SSH events tab shows ssh commands run in interactive sessions, limited to 100 events per hour.



Explore ip-172-31-55-106.ec2.internal

Apps | SSH Events | Security Updates

Display is limited to the last 100 events in the past hour

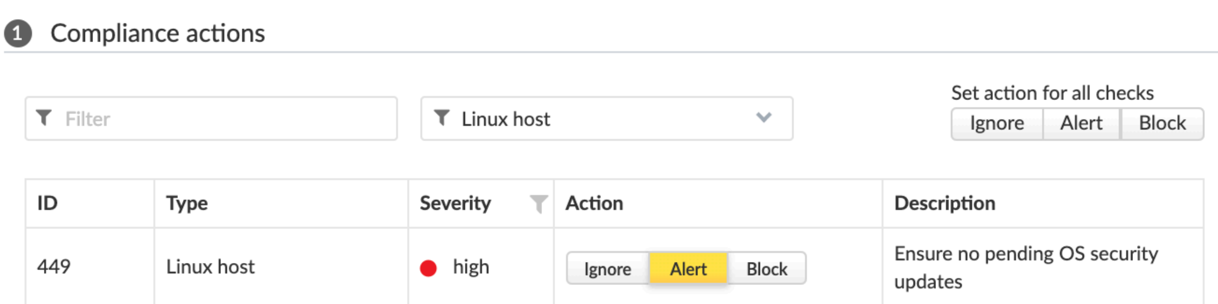
Filter SSH events by keywords and attributes

User	IP	Process Path	Command	Time
ubuntu	34.100.87.242	/bin/nc.openbsd	/bin/nc.openbsd	Jul 21, 2020 9:45:22 AM
ubuntu	34.100.87.242	/usr/bin/sudo	sudo nc -lp 555	Jul 21, 2020 9:45:22 AM
ubuntu	34.100.87.242	/bin/nc.openbsd	/bin/nc.openbsd	Jul 21, 2020 9:45:17 AM
ubuntu	34.100.87.242	/usr/bin/curl	curl www.google.com	Jul 21, 2020 9:44:10 AM
ubuntu	34.100.87.242	/usr/bin/dircolors	/usr/bin/dircolors	Jul 21, 2020 9:43:54 AM
ubuntu	34.100.87.242	/usr/bin/dirname	dirname /usr/bin/lesspipe	Jul 21, 2020 9:43:54 AM
ubuntu	34.100.87.242	/usr/bin/basename	basename /usr/bin/lesspipe	Jul 21, 2020 9:43:54 AM
ubuntu	34.100.87.242	/bin/dash	/bin/sh /usr/bin/lesspipe	Jul 21, 2020 9:43:54 AM
ubuntu	34.100.87.242	/bin/bash	/bin/bash	Jul 21, 2020 9:43:54 AM

Security updates

Prisma Cloud periodically checks for security updates. It’s implemented as a compliance check. This feature is supported only for Ubuntu/Debian distributions with the "apt-get" package installer.

Prisma Cloud probes for security updates every time the scanner runs (every 24 hours, by default). The check is enabled by default in **Defend > Compliance > Hosts** in the **Default - alert on critical and high** rule.



1 Compliance actions

Filter | Linux host

Set action for all checks: Ignore | Alert | Block

ID	Type	Severity	Action	Description
449	Linux host	high	Ignore Alert Block	Ensure no pending OS security updates

The security updates tab shows pending security updates (based on a new compliance check that was added for this purpose). Supported for Ubuntu and Debian

On each host scan, Prisma Cloud checks for available package updates marked as security updates. If such updates are found, they're listed under the security updates tab.

Audits

Audits can be viewed under **Monitor > Events**.

Runtime defense for serverless functions

Edit on GitHub

Prisma Cloud lets you monitor process, network and filesystem activity within your serverless functions and enforce policies to allow or deny these activities. Policies let you define:

- Process activity - enables specifying specific whitelisted processes, blocking all processes except the main process and detecting cryptomining attempts.
- Network activity - enables monitoring and enforcement of DNS resolutions, inbound and outbound network connections.
- Filesystem activity - enables defining specific paths in an allowed or denied list.

In addition to runtime policy, you can also configure multiple WAAS application firewall protections to defend your functions from application layer attacks.

Securing serverless functions

To secure Serverless functions:

1. Verify that you have installed Serverless Defenders on your functions.

   You must install Serverless Defenders before you can create serverless runtime policy.
2. Log in to the Prisma Cloud Console and select **Defend > Runtime > Serverless policy** to add policies.
3. Embed the Serverless Defender into your function either manually or with Auto-defend:
   - Manually embed a Serverless Defender
   - Use a Lambda layer to embed a Serverless Defender
   - Use Auto-defend to deploy Serverless Defenders

Defining your policy

Add runtime protection for your serverless function by defining a runtime rule for it in the Prisma Cloud Console.

Prisma Cloud ships without a Serverless runtime policy. Serverless Defenders fetch the policy from the TW_POLICY environment variable and dynamically during runtime from the console (every 2 minutes).

By default, new rules apply to all functions (*), but you can target them to specific functions and/or regions using pattern matching. For Azure Functions only, you can additionally scope rules by account ID.

STEP 1 | Log into Prisma Cloud Console.

STEP 2 | Go to **Defend > Runtime > Serverless Policy**.

STEP 3 | Click **Add rule**.

1. Enter a rule name.
2. By default, the rule applies to all functions in all regions and accounts.

   Target the rule to specific functions.
3. Click the **Networking** tab.
4. Enable **DNS** toggle
5. Set **Effect** to **Prevent**.
6. Add **amazon.com* to the **DNS allow list**

   *By default, rules are set to allow traffic. When adding a domain to the allow list, then everything outside the allow list is denied by default. The above rule will block all traffic except to *amazon.com.*
7. Click **Save**.

View runtime audits

To view the security audits, go to **Monitor > Events > Serverless Audits**. You should see audits with the following messages:

```
DNS resolution of domain name yahoo.com triggered by /usr/bin/wget
 explicitly denied by a runtime rule.
```

To refine the view, use filters. For example, to see Azure Functions only, use the *Provider: Azure* filter.

Runtime defense for App-Embedded

Edit on GitHub

App-Embedded Defenders monitor and protect your containers at runtime, ensuring they execute as designed, and securing them against suspicious activity.

App-Embedded Defender runtime rules let you control:

- Process activity.
- Network connections.
- File system activity.

App-Embedded Defenders also support custom runtime rules.

For front-end containers, deploy the WAAS application firewall for additional runtime protection.

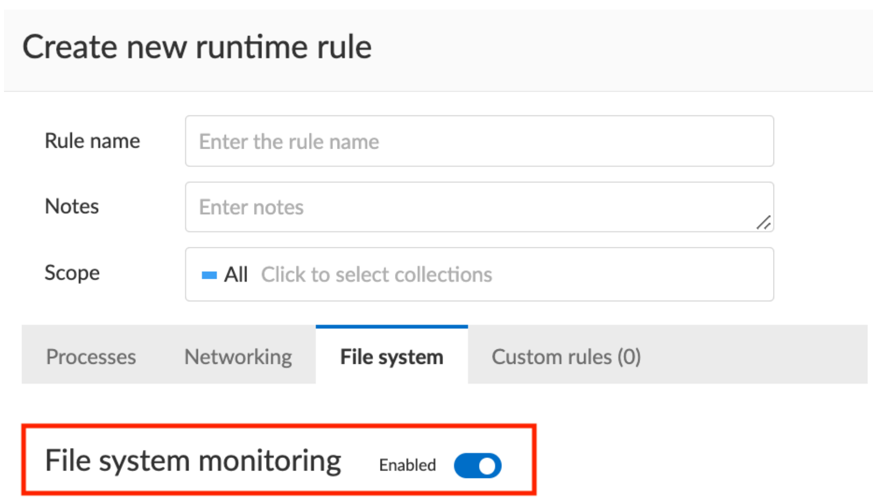
App-Embedded runtime policy

App-Embedded Defenders have distinct process, network, and file system sensors to monitor a workload's activity at runtime. Each sensor is implemented individually, with its own set controls and configurations. After deploying App-Embedded Defender, customize runtime protection for the workload by creating rules. Rules let you control process, network, and file system activity.

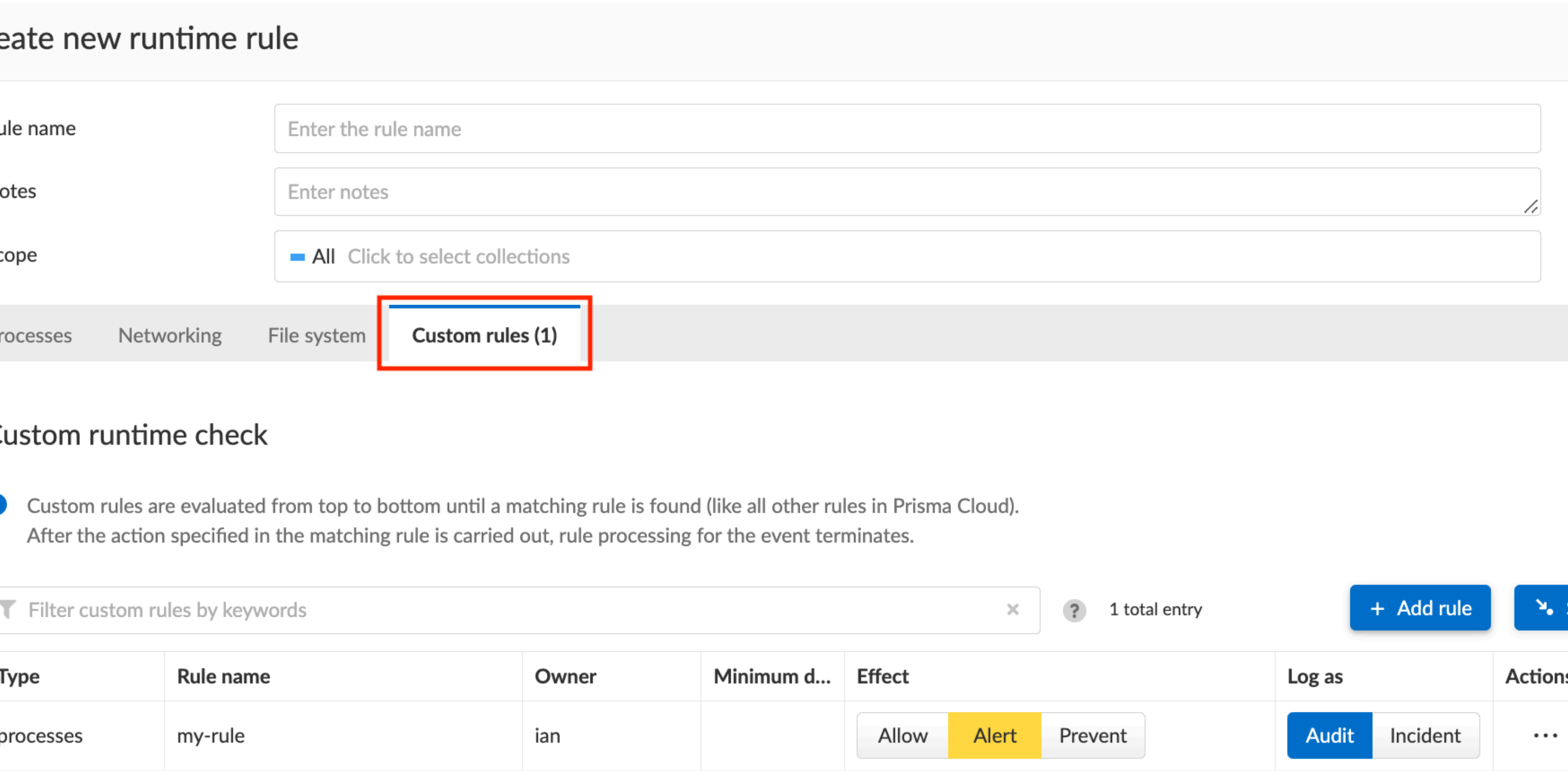
App-Embedded Defenders dynamically retrieve policies from Console as they are updated. You can embed App-Embedded Defender into a workload with a very simple initial policy, and refine it later, as needed.

Audits can be reviewed under **Monitor > Events > App-Embedded Audits** App-Embedded Defender generates audits and incidents when one of the following conditions applies:

- The sensor is enabled. For example, the following screenshot shows that the file system sensor is enabled:



- A custom rule is attached to an App-Embedded runtime rule. For example:



> *Unlike Container Defenders, App-Embedded Defenders don't support* learning and models.

Effect

App-Embedded Defender runtime protectionn can be configured to operate in one of the following modes:

- **Disable** — Defender doesn't provide any protection.
- **Alert** — Defender generates audits when it detects runtime activity that violates your defined policy. If alerts are configured, they're also generated and sent. Audits can be reviewed under **Monitor > Events > App-Embedded audits**.
- **Prevent** — Prevents the runtime activity. For example, file system defense prevents the creation or modification of files.

App-Embedded Defenders don't support the **Block** action, where Defender stops the entire container. Blocking isn't feasible when workloads run on Containers-as-a-Service platforms, where neither the workload nor the embedded Defender have access to the underlying container runtime (e.g. Docker Engine).

Process monitoring

App-Embedded Defender can detect anomalous process activity. Each control can be independently enabled or disabled.

- **Processes started from modified binaries** — Detects when binaries from a container image have been modified and then subsequently executed.
- **Crypto miners** — Detects crypto miners and creates a crypto miner incident.
- **Explicitly allowed and denied processes** — Controls which processes can run. If you specify an allow list, then everything outside the allow list is denied by default. If you specify a deny list, then everything outside the deny list is allowed by default. Processes can be specified by name or MD5 hash.

Network monitoring

App-Embedded Defender can monitor container networking activity for patterns that indicate an attack might be underway. Each control can be independently enabled or disabled.

- **Allowed** and **Denied** — Specifies known good or bad network connections. You can define policy for listening ports, outbound internet ports for Internet destinations, and outbound IP addresses. If you specify an allow list, then everything outside the allow list is denied by default. If you specify a deny list, then everything outside the deny list is allowed by default.

DNS

DNS monitoring analyzes DNS lookups from your running containers. Dangerous domains are detected as follows:

- **Prisma Cloud Intelligence Stream** — Prisma Cloud's threat feed contains a list of known bad domains.
- **Explicit allow list:** Runtime rules let you augment the Prisma Cloud's Intelligence Stream data with your own explicit lists of known good domains.

File system monitoring

App-Embedded Defender's runtime defense for container file systems continuously monitors and protects containers from suspicious file system activities and malware.

By default, App-Embedded Defender monitors both the container's root file system and any mounted data volumes.

Enabling file system monitoring

The file system sensor evaluates changes to the file system. File system monitoring is disabled by default because it can impact the protected workload's performance.

When you embed App-Embedded Defender into a workload, the state of the file system monitoring subsystem is set. Once the state is set, it cannot be changed dynamically at runtime. You must re-embedd Defender with a different setting.

When the file system monitoring subsystem is enabled in App-Embedded Defender, the sensor captures file system events in the background, regardless of the settings in your runtime rules. In particular, file system forensics (binary created event) are collected and reported regardless of how your runtime policy is configured.

Security teams can globally specify the default setting for file system monitoring. During the Defender embed (deployment) flow, individual teams can then see and accept the organization's recommended setting. They can also override the default setting as they see fit for the efficient operation of their own applications.

For more information, see configuring the default setting for file system protection.

Detections

Prisma Cloud can detect anomalous file system activity. Each control can be independently enabled or disabled.

Defender also looks for attributes that make files suspicious, including signs they've been rigged for anti-analysis.

- **Changes to binaries or certificates** — Detects when these types of files from a container image are modified.
- **Detection of encrypted/packed binaries** — Detects usage of encrypted/packed binaries. Such files are suspicious because it's a sign they've been rigged for anti-analysis to deploy malware undetected.
- **Changes to SSH and admin account configuration files**
- **Binaries with suspicious ELF headers**
- **Custom feed for malware detection**
- **Use WildFire malware analysis** — Use WildFire, Palo Alto Networks' malware analysis engine, to detect malware. To use Wildfire, it must first be enabled.
- **Explicitly allowed and denied system paths** — Controls where files can be written. If you specify an allow list, then everything outside the allow list is denied by default. If you specify a deny list, then everything outside the deny list is allowed by default.

*The **Prevent** effect is supported for "Changes to SSH and admin account configuration files" and denied system paths only. For all other detections, you are alerted with an audit but the activity is not prevented.*

Malware protection

App-Embedded Defender monitors container file systems for malicious binaries and certs using data from:

- Your custom malware feed.
- Wildfire.

When a file is written to the container file system, Defender compares the MD5 hash of the file to the MD5 hashes configured under **Manage > System > Custom feeds > Malware signatures**. If there is a match, Defender creates an audit.

Custom rules

Custom rules offer another mechanism to protect running software. Custom rules are expressions that give you a precise way to describe and detect discrete runtime behaviors. Expressions let you examine various facets of an event in a programmatic way, then take action when they evaluate to true.

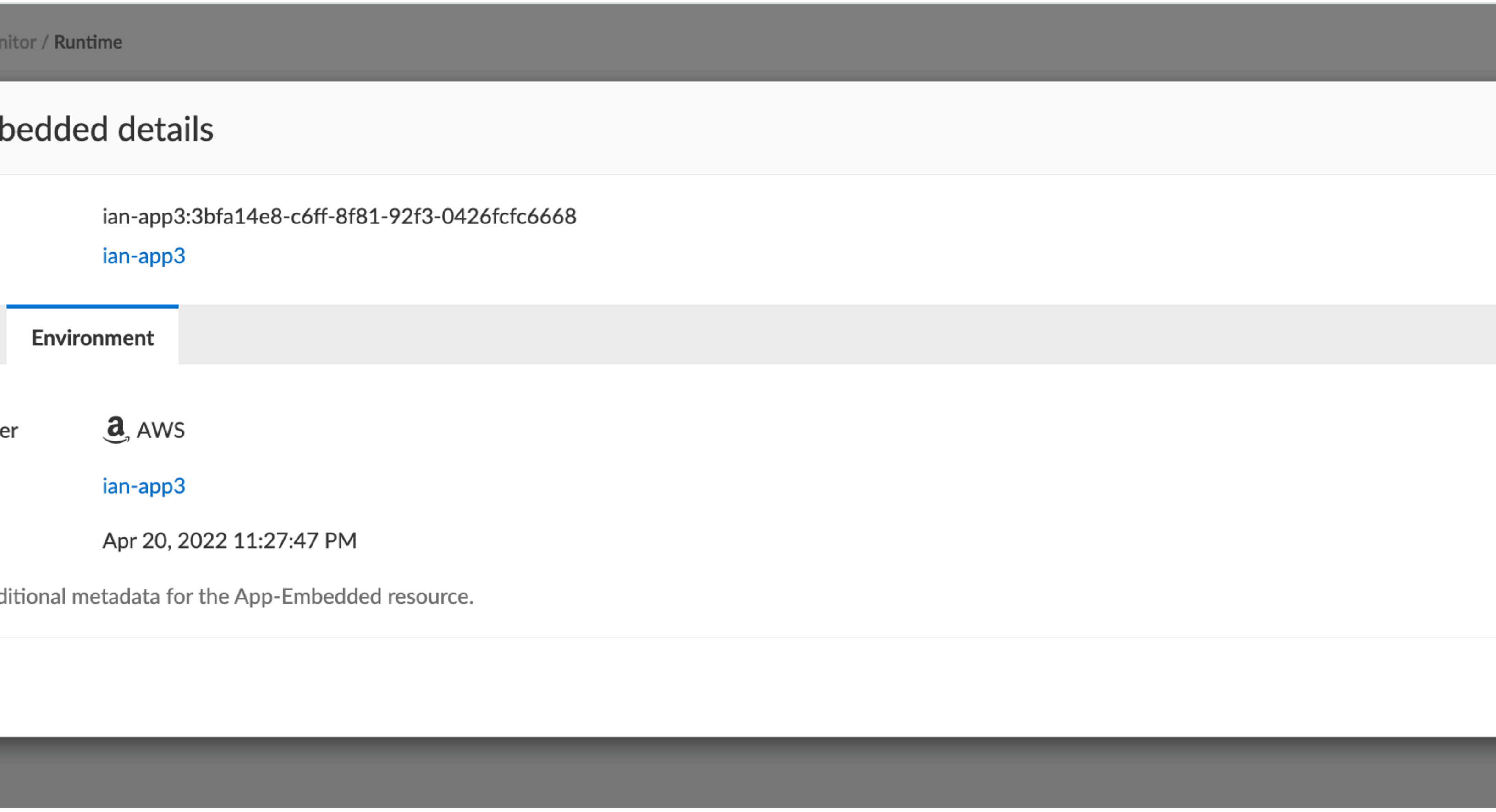
For more information, see custom rules.

*The **Prevent** effect isn't supported when using the file.type or file.md5 properties in custom rules for App-Embedded Defenders.*

Monitoring workloads at runtime

Go to **Monitor > Runtime > App-Embedded observations** to monitor and manage workloads protected by App-Embedded Defender. This page aggregates and reports runtime audits, forensics, and environment metadata for each workload. You can filter the workloads in the table by a number of facets, including collections and App ID.

App-Embedded Defenders collect and report metadata about the environment in which they run. From the **App-Embedded observations** page, click on a protected workload to open the report, and the click on the **Environment** tab.



The metadata App-Embedded Defenders collect depends on what's available from the underlying cloud provider. App-Embedded Defenders can collect and report the following metadata when running on the following cloud provider services:

Metadata	AWS - Fargate with ECS	AWS - Fargate with EKS	Google Cloud Run	Azure ACI
Cloud provider	Y	Y	Y	Y
Region	Y		Y	

Metadata	AWS - Fargate with ECS	AWS - Fargate with EKS	Google Cloud Run	Azure ACI
Account ID	Y		Y	
Cluster	Y			
Instance ID	Y (task ID)		Y	
Resource name (e.g., pod name)	Y (container name)			
Image name	Y	Y	Y	Y
Container name	Y			
App ID	Y	Y	Y	Y

When App-Embedded Defender runs in Fargate on Amazon EKS and Azure ACI, it emits an error that says Defender failed to fetch cloud metadata. This is by design, and the error message can safely be ignored.

For AWS Fargate on Amazon EKS, Prisma Cloud doesn't report any cloud metadata because AWS doesn't support the instance metadata service for pods that are deployed with Fargate. Similarly for images running on ACI, no cloud metadata is available for Prisma Cloud to report.

Securing your App-Embedded containers

To secure App-Embedded containers, including Fargate tasks, embed the Prisma Cloud App-Embedded Defender into it. The steps are:

1. Define your policy in Prisma Cloud Console under **Defend > Runtime > App-Embedded policy**.
2. Embed the App-Embedded Defender into your container or task definition using one of the following procedures:
   - Install App-Embedded Defender
   - install App-Embedded Defender for Fargate
3. Start the service that runs your container.

Custom runtime rules

Edit on GitHub

Prisma Cloud's approach to scaling runtime defense in big, fluid environments is to model runtime behavior with machine learning. Machine learning reduces the effort required to manually create and maintain loads of rules to secure running software. When machine learning doesn't fully capture the range of acceptable runtime behaviors, rules provide a way to declaratively augment models with exceptions and additions.

Custom rules offer another, additional mechanism to protect running software. Custom rules are expressions that give you a precise way to describe and detect discrete runtime behaviors. Runtime sensors in your environment already detect process, file system, and network activity, then pass those events to Prisma Cloud for processing. Expressions let you examine various facets of an event in a programmatic way, then take action when they evaluate to true. Custom rules can be applied to both hosts and containers.

For example, the expression grammar supports the following logic:

```
"If user Jake runs binary netcat with parameter -l, log an alert"
```

Rule library

Custom rules are stored in a central library, where they can be reused. Besides your own rules, Prisma Cloud Labs also distributes rules via the Intelligence Stream. These rules are shipped in a disabled state by default. You can review, and optionally apply them at any time.

Custom rules are written and managed in Console under **Defend > Custom Rules > Runtime**. Click **Add rule** to bring up the online editor. The compiler checks for syntax errors when you save the rule.

There are four types of rules, but only three are relevant to runtime:

- processes
- filesystem
- networking-outgoing

Expression grammar

Expressions let you examine the contents of process, file system, and network events.

For example, any time a process is forked on a host protected by Container Defender or Host Defender, a process event fires. The following very simple expression looks for processes named netcat:

```
proc.name = "netcat"
```

Expressions have the following grammar:

*expression: term (op term | in)**

- **term** --

  integer | string | keyword | event | '(' expression ')' | unaryOp
- **op** --

  and | or | > | < | >= | # | = | !=
- **in** --

  '(' integer | string (',' integer | string)*)?
- **unaryOp** --

  not
- **keyword (similar to wildcards)** --

  startswith | contains
- **string** --

  strings must be enclosed in double quotes
- **integer** --

  int
- **event** --

  process, file system, or network

Expressions examples:

```
net.outgoing_ip = "169.254.169.254" or net.outgoing_ip =
 "169.254.170.2"
```

```
proc.pname in ("mysql", "sqlplus", "postgres") and proc.pname !=
 proc.name
```

```
file.path startswith "/etc"
```

Process events

Process events fire when new processes are forked. Expressions can examine the following attributes of a new process.

Attribute	Type	Description
proc.name	string	Process name.
proc.pname	string	Parent process name.
proc.path	string	Full path to the program.
proc.user	string	User to whom the process belongs.

Attribute	Type	Description
proc.interactive	bool	Interactive process. *Not supported in App-Embedded runtime*
proc.cmdline	string	Command line.
proc.service	string	Only for host rules.

File system events

Filesystem events fire when there are writes to disk. All properties of the process doing the writes are accessible from this context. Expressions can examine the following attributes of file system write activity.

Attribute	Type	Description
file.path	string	Path of the file being written.
file.dir	string	Directory of the file being written.
file.type	enum	File type. Supported types are: elf, secret, regular, and folder.
file.md5	string	MD5 hash of the file. Supported only for ELF files. For other types of files, this property will be empty.

Networking events

Network events fire when a process tries to establish an outbound connection. Expressions can examine the following attributes when network events fire:

Attribute	Type	Description
proc.name	string	Name of process initiating the outbound network connection.
net.outgoing_port	string	Outbound port.
net.outgoing_ip	string	Outgoing IP address. The following expression looks for outbound connections to a range of IP addresses: net.outgoing_ip # "1.1.1.1" and net.outgoing_ip # "1.1.1.9"

Attribute	Type	Description
net.private_subnet	bool	Private subnet.

Example expressions

The Prisma Cloud Labs rules in the rule library are the best place to find examples of non-trivial expressions.

STEP 1 | In Console, go to **Defend > Custom configs > Runtime**.

STEP 2 | In the **Type** column, add a filter for processes, filesystem, or network outgoing.

STEP 3 | Click on any rule that starts with **Prisma Cloud Labs** to see the implementation.

Activating custom rules

Your runtime policy is defined in **Defend > Runtime > {Container Policy | Host Policy | App-Embedded Policy}**, and it's made up of models and rules. Your expressions (custom rules) can be added to runtime rules, where you further specify what action to take when expressions evaluate to true. Depending on the event type, the following range of actions are supported: allow, alert, prevent, or block. Also, you can deteremine whether you want to log the raised event as an audit or as an incident.

Custom rules are processed like all other rules in Prisma Cloud: the policy is evaluated from top to bottom until a matching rule is found. After the action specified in the matching rule is performed, rule processing for the event terminates.

Within a runtime rule, custom rules are processed first, and take precedence over all other settings. Be sure that there is no conflict between your custom rules and other settings in your runtime rule, such as allow and deny lists.

STEP 1 | Open Console, and go to **Defend > Runtime > {Container Policy | Host Policy | App-Embedded Policy}**.

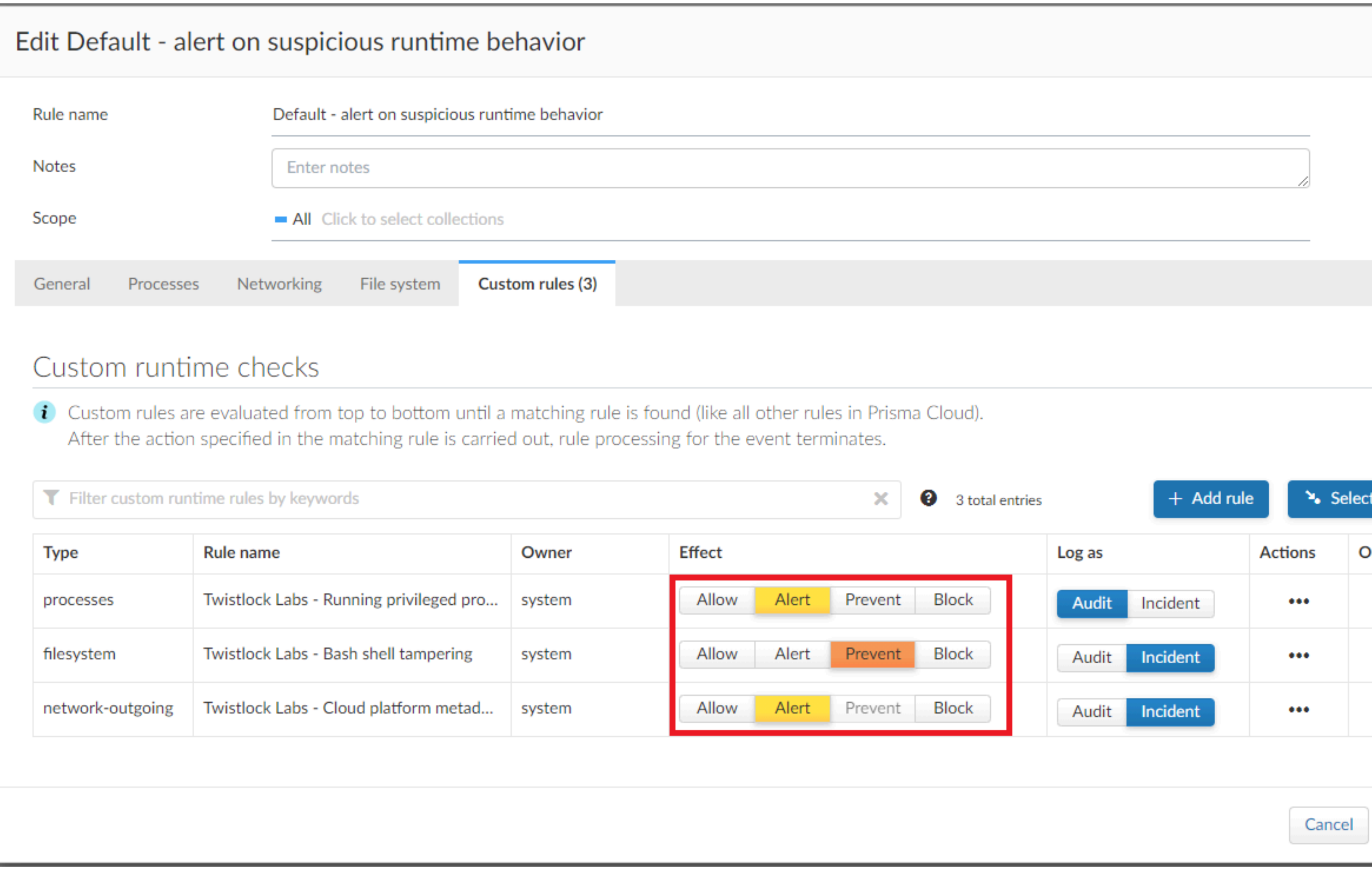
STEP 2 | Click **Add rule**.

STEP 3 | Enter a name for the rule.

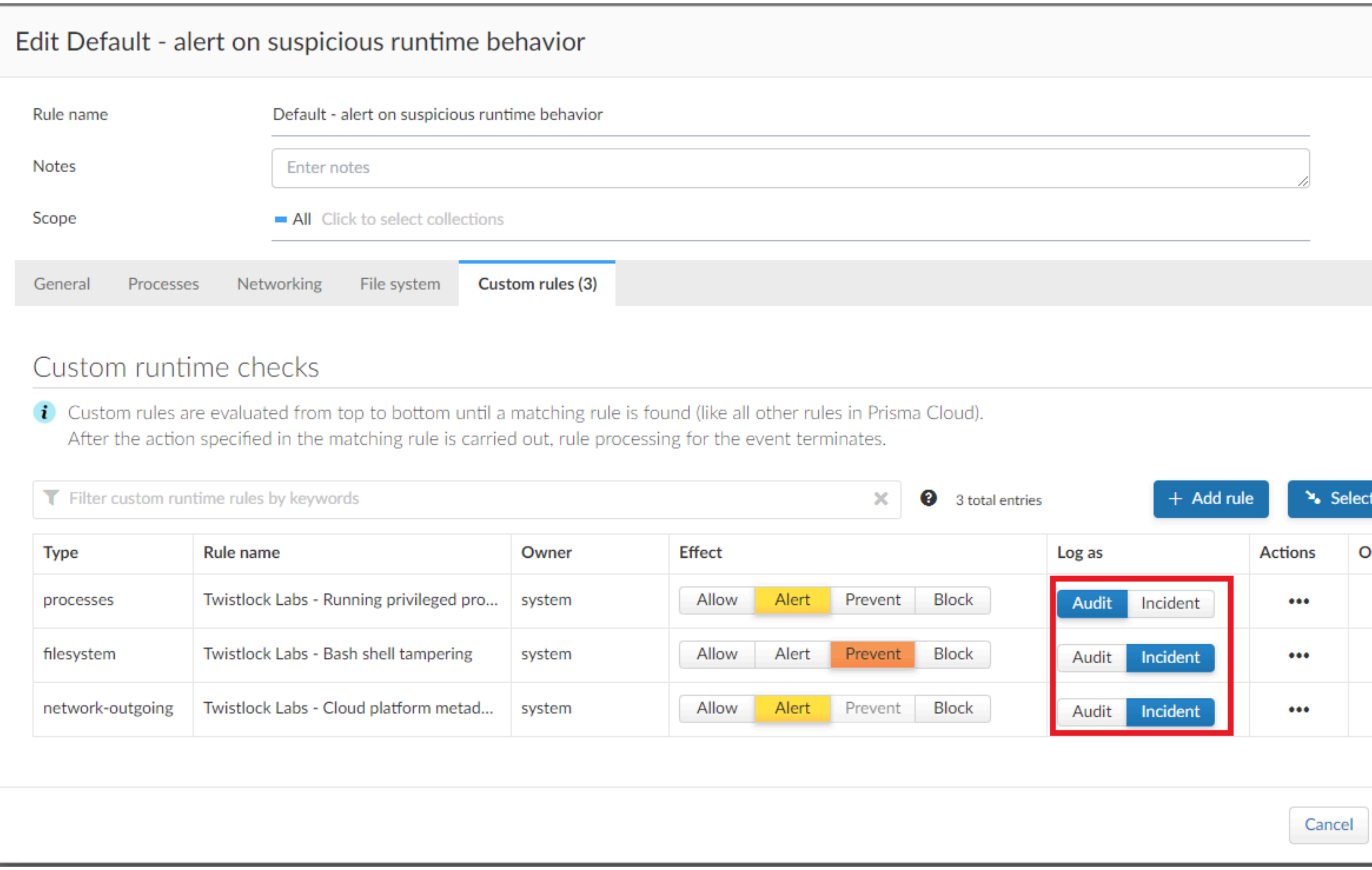
STEP 4 | Click the **Custom Rules** tab.

STEP 5 | Click **Select rules**, choose the rules to add, and click **Apply**.

STEP 6 | Specify an effect for each rule.



STEP 7 | Specify how to log the event for each rule.



STEP 8 | Click **Save**.

Limitations

There are a number of things that custom rules cannot do:

- The proc.cmdline and file.type fields are not supported in prevent mode. You'll get an error if you try to attach a custom rule to a runtime rule with these fields and the action set to prevent.
- Prisma Cloud cannot inspect command line arguments before a process starts to run. If you explicitly deny a process and set the effect to **Prevent** in the **Process** tab of a runtime rule, the process will never run, and Prisma Cloud cannot inspect it's command line arguments. The same logic applies to custom rules that try to allow processes that are prevented by other policies. For example, consider process 'foo' that is explicitly denied by a runtime rule, with the effect set to **Prevent**. You cannot allow 'foo -bar' in a custom runtime rule by analyzing proc.cmdline for '-bar'.

- Prisma Cloud doesn't support prevent on write operations to existing files. For example, consider the following expression:

  ```
  file.path = "/tmp/file"
  ```

  If this expression is added to a runtime rule, and the effect is set to prevent, then Prisma Cloud will prevent the creation of such a file. If the file already exists, however, Prisma Cloud won't prevent any write operation to it, but will raise an alert.
- App-Embedded custom rules support Processes and Outbound Connection rule types. The Block action is not supported, while Prevent is supported for both Processes and Outbound Connection rule types.
- The **Prevent** effect isn't supported when using the *file.type* or *file.md5* properties in custom rules for App-Embedded Defenders.

Import and export individual rules

Edit on GitHub

Prisma Cloud lets you import and export rules from one Console to another. Every rule created in Prisma Cloud under the **Defend** section has copy and export buttons in the **Actions** menu. An import button is located at the bottom of every rule table.

Copying rules

To copy a rule:

STEP 1 | Go to **Defend > Runtime > {Vulnerabilities | Compliance | Access}**.

STEP 2 | Click **Actions > Copy** for the rule you want to copy.

A dialog box named **Edit copy of....** opens.

STEP 3 | Make any desired changes to the copied rule.

STEP 4 | Click **Save**.

Exporting rules

Click **Actions > Export** next to any rule to export it in json format.

Example

```
{
  "name": "Default - ignore Prisma Cloud components",
  "owner": "system",
  "modified": "2017-05-31T20:47:21.573Z",
  "effect": "alert",
  "resources": {
    "hosts": [
      "*"
    ],
    "images": [
      "docker.io/twistlock/private:console*"
    ],
    "labels": [
      "*"
    ],
    "containers": [
      "twistlock_console"
    ],
    "services": []
  },
  .
  .
  .
}
```


Importing rules

A rule can be imported into Console in JSON format. To capture a rule in JSON format, use the export function described above.

ATT&CK Explorer

Edit on GitHub

Prisma Cloud's monitoring section includes an Att&CK Explorer dashboard providing a framework that helps you to contextualize runtime audits, manage them, and generate risk reports.

ATT&CK Explorer is a knowledge base of tactics and techniques that adversaries use to attack applications and infrastructure. It's a useful framework for threat-informed defense, where a deep understanding of adversary tradecraft can help protect against attacks.

The ATT&CK framework has two key concepts:

- **Tactics** - An adversary's technical goals.
- **Techniques** - How those goals are achieved or What they acheive

The relationship between tactics and techniques is presented as a matrix. One tactic in the matrix is called *Persistence*. After establishing a foothold in your environment, adversaries want to reliably return to it. Adversaries use a number of techniques to achieve persistence, such as *Account Manipulation* and *Event Triggered Execution*.

Cloud Native threat matrix

Prisma Cloud protects cloud native applications running in Kubernetes clusters, serverless functions, Containers-as-a-Service offerings, and virtual machines. The Cloud Native threat matrix covers the different techniques that impact cloud native applications across all these environments. It's composed from ATT&CK for Linux, recent community efforts around ATT&CK for Containers and Kubernetes, and a few techniques from Prisma Labs. The Cloud Native threat matrix is the foundation for the ATT&CK dashboard.

ATT&CK dashboard

The ATT&CK dashboard serves as a portal to the raw events in the **Monitor > Events** view. All Prisma Cloud audits are mapped to the tactics and techniques in the ATT&CK framework. For example, when Defender detects a crypto miner in your environment, we map the audit to the *Resource Hijacking* technique under the *Impact* tactic.

The ATT&CK dashboard collates audits, maps them to the tactics and techniques, and presents the data visually in the ATT&CK matrix. Each card in the matrix shows a count of events. Higher counts represent a higher severity issues. Filters let you slice and dice the data to inspect specific segments of your environment. The dashboard:

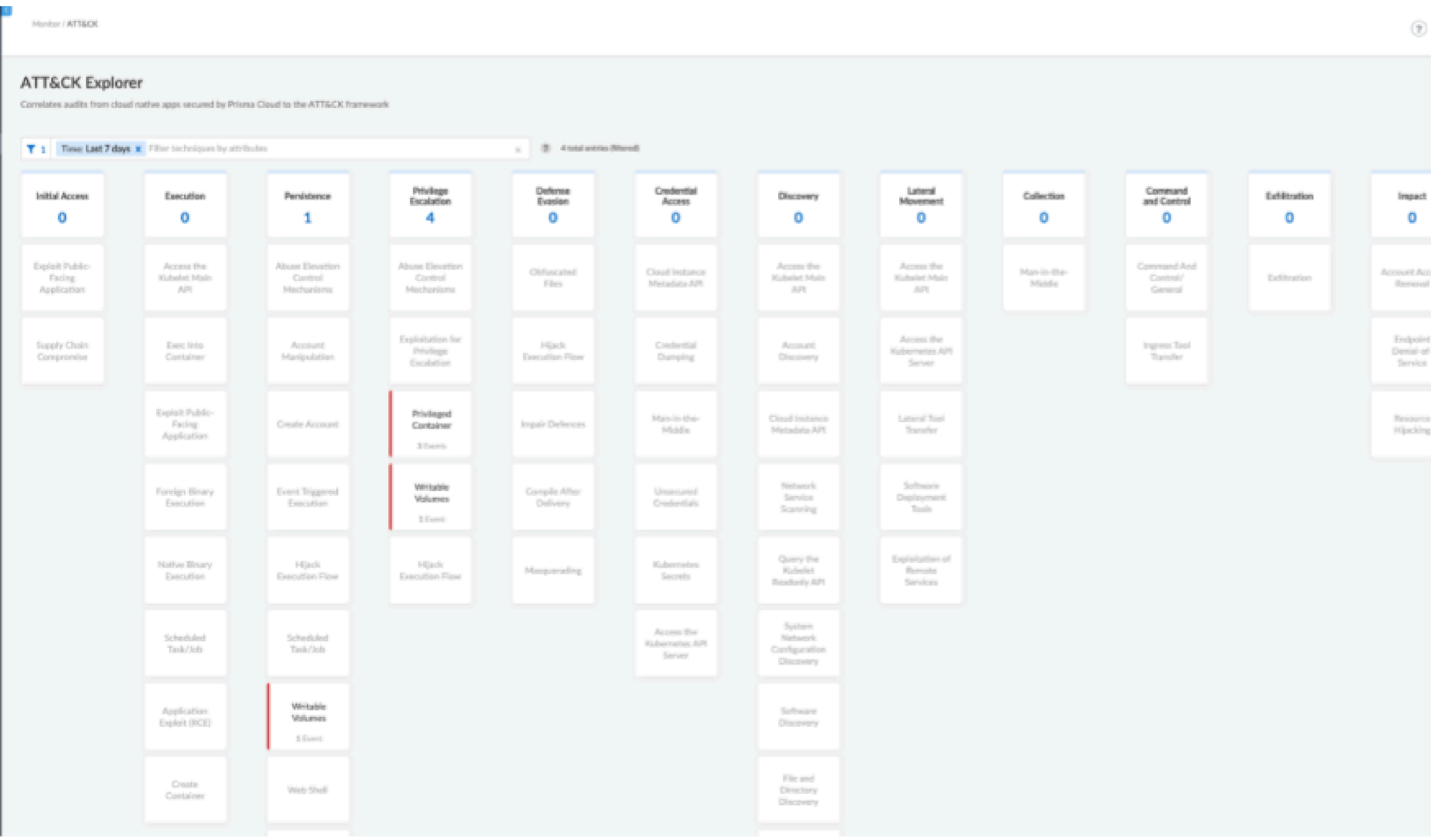
- Presents a real-time view of tactics and techniques being employed by adversaries.
- Identifies weaknesses in your defenses. Use the counts to prioritize work to fortify defenses for the techniques favored by adversaries.
- Provides raw data for risk reports for management.

Audits from the following subsystems flow into the ATT&CK dashboard:

- Container runtime audits.
- Host runtime audits.

- Serverless runtime audits.
- App-Embedded runtime audits.
- WAAS audits.
- Kubernetes audits.
- Admission (OPA) audits.
- Custom runtime rule audits for builtin system checks only. Currently, you cannot specify tactic and technique for user-defined custom runtime rules.

To see the ATT&CK dashboard, open Console, and go to **Monitor > ATT&CK**. The following screenshot highlights the main components in the dashboard:



1. Filter - Filter data in the dashboard by:

- **Impacted technique**.
- **Date**: View events that occurred in the past 24 hours, 7 days, 30 days, or 3 months.
- **Collection**: View data for just some segment of your environment (e.g., a production cluster).

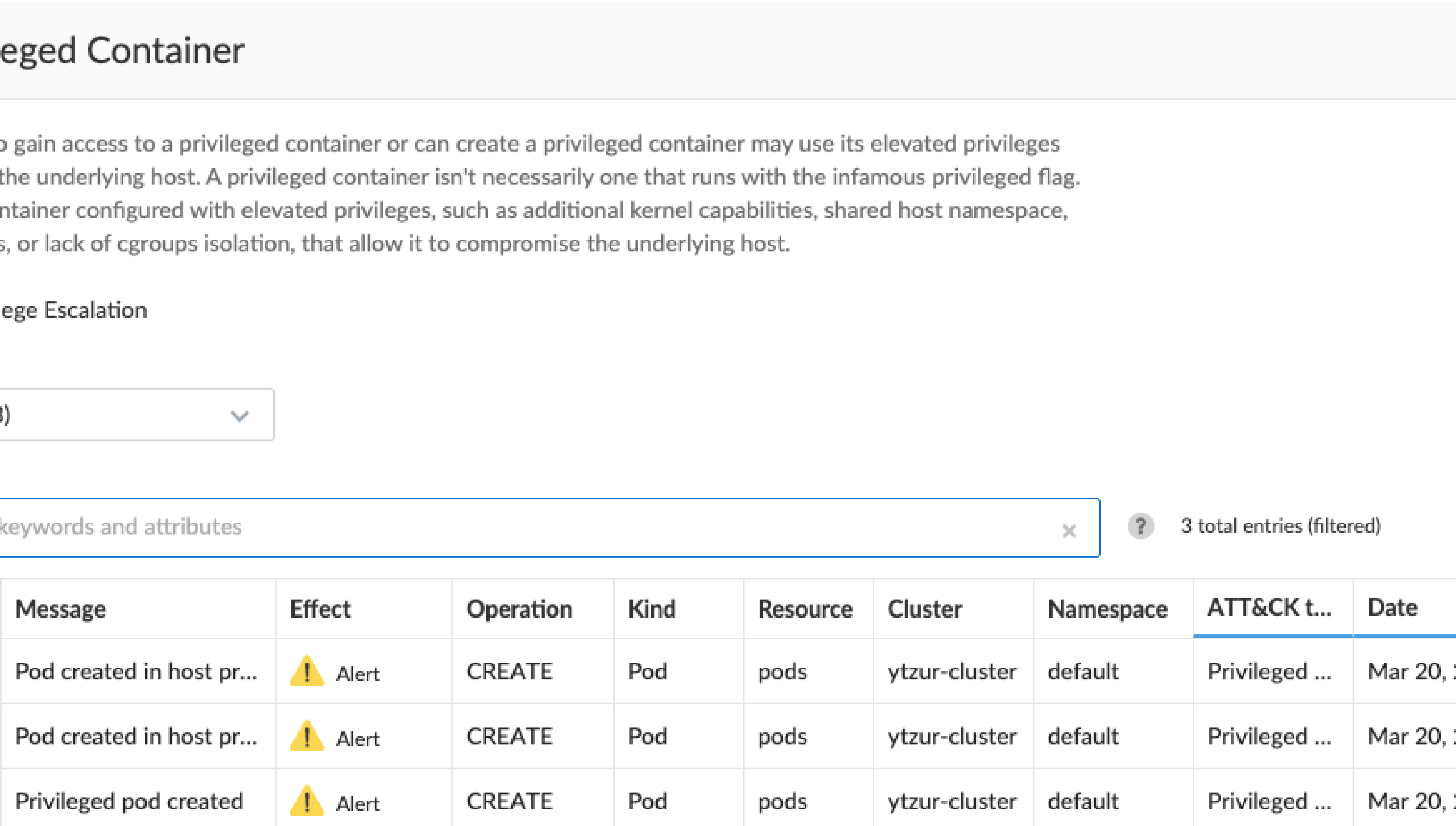
2. Tactics - Tactics are listed across the top row of the matrix. A count shows the sum of all events for all corresponding techniques in the category. Each column lists the techniques that can be used to achieve the tactic.

3. Techniques - Lists of techniques that can be used to achieve a tactic. The color of the card is based on the event count for a technique. If there is one or more events for a technique, the card is colored red. Otherwise, if there are no events, the card is gray. All techniques are fully described here.

Clicking on an impacted technique card opens a dialog that shows all relevant audits for the technique.

The following screenshot shows the dialog for the *Privileged Container* card. The dialogs are organized as follows:

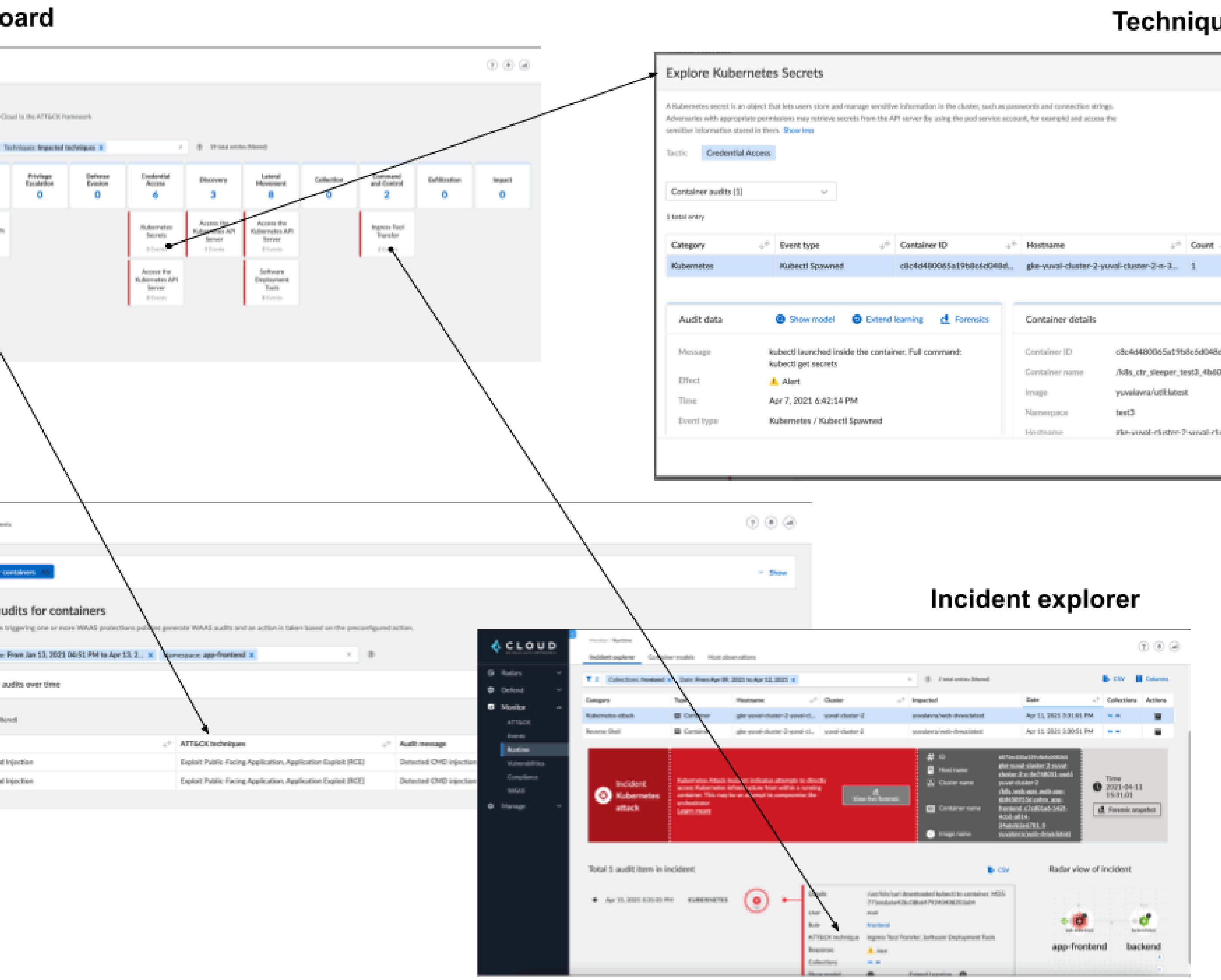
- Description.
- Audit source filter (pick from the drop-down list).
- Table of relevant audits.



Syslog messages contain tactic and technique information for all relevant audits.

Investigating incidents

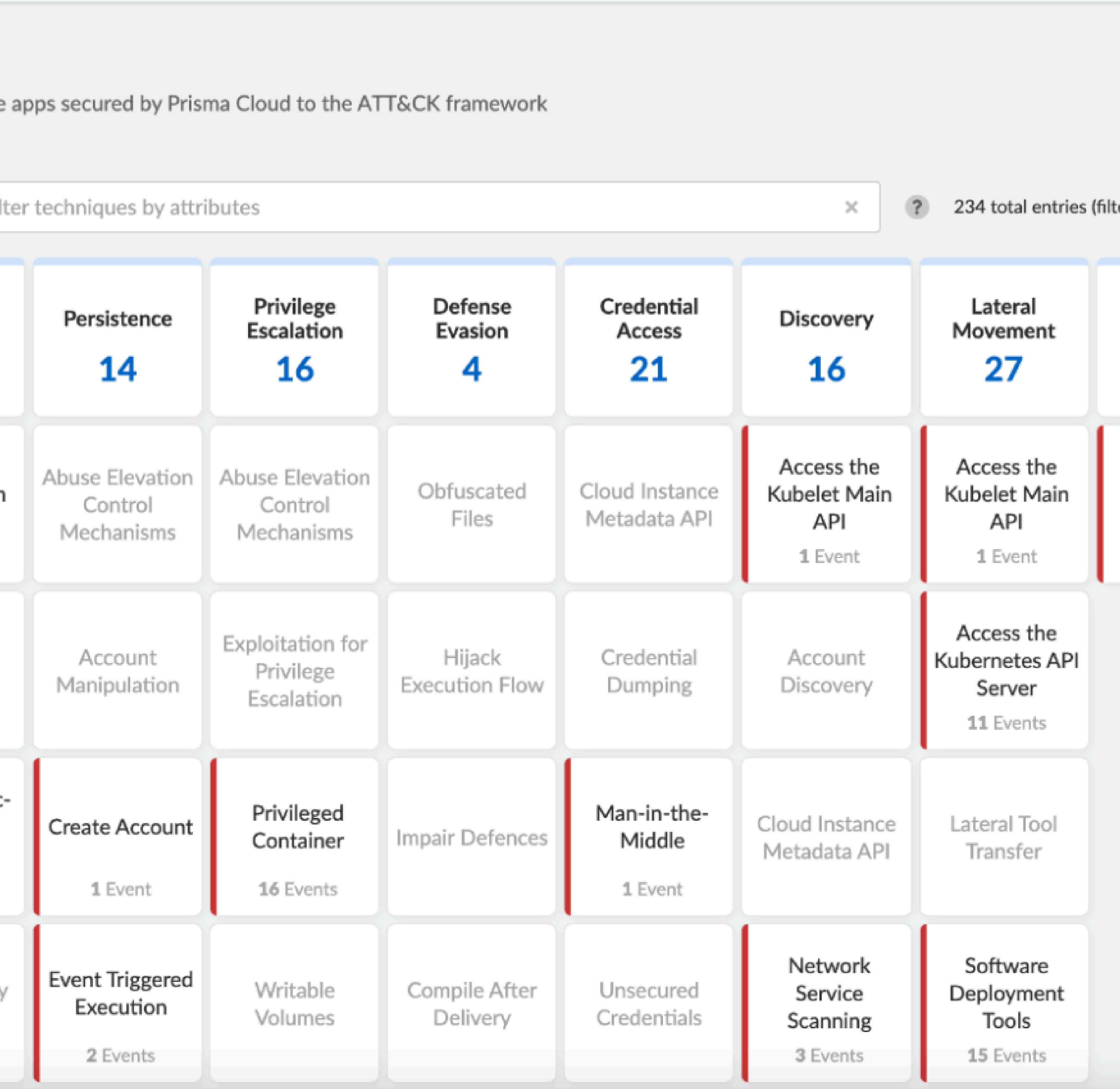
As you monitor your environment, you'll see tactics and techniques are applied consistently across views. Tactics and techniques are shown in **Monitor > ATT&CK**, **Monitor > Events**, and **Monitor > Runtime > Incident explorer**.



Surfacing impacted techniques

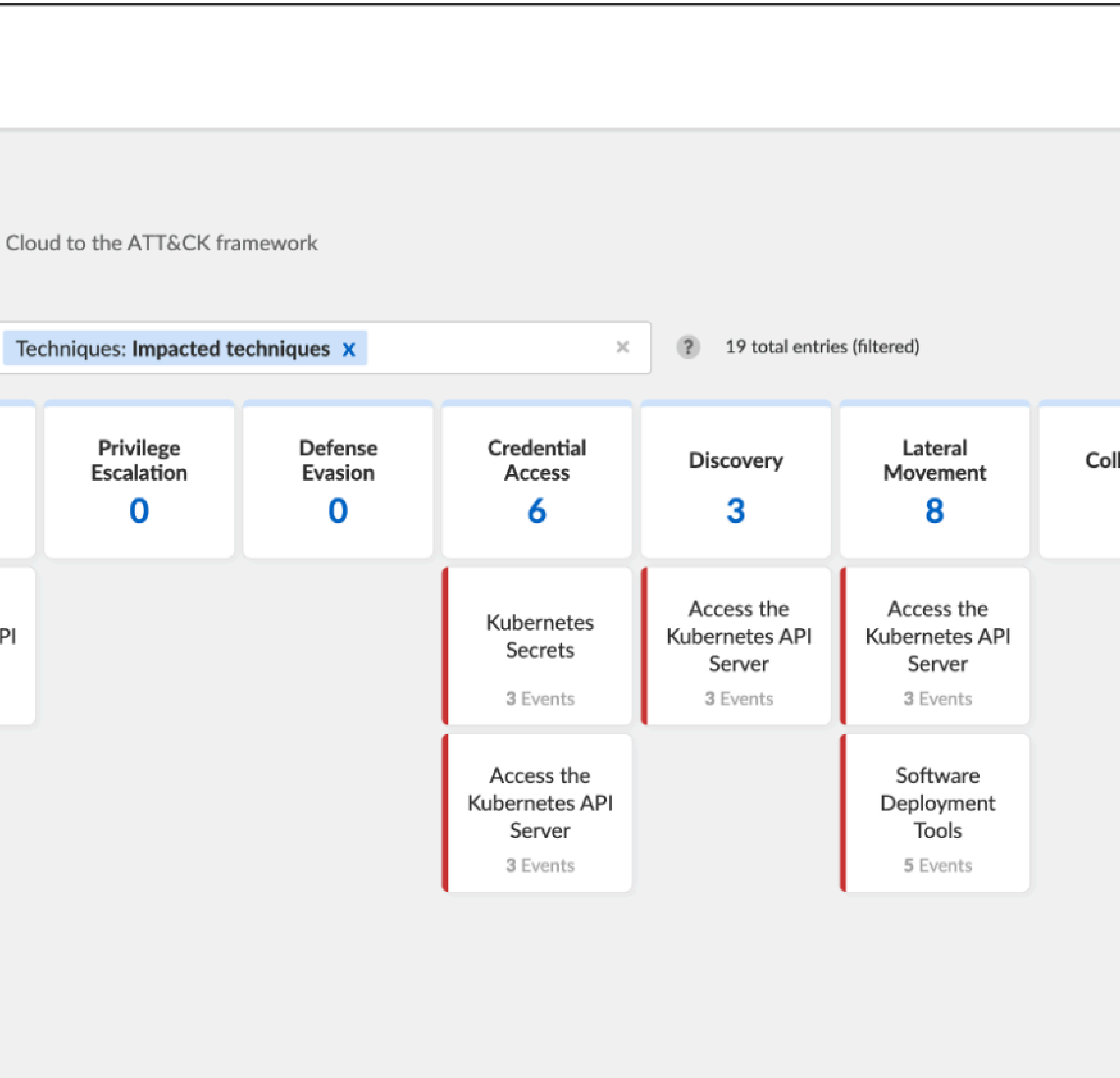
When investigating an incident, you'll want to focus on the segment of your environment that has been impacted. Use the filter box to focus your view of the data.

One important filter is **Impacted techniques**. Without the filter, all technique cards are displayed.


e apps secured by Prisma Cloud to the ATT&CK framework
lter techniques by attributes
234 total entries (filt
Persistence 14
Privilege Escalation 16
Defense Evasion 4
Credential Access 21
Discovery 16
Lateral Movement 27
Abuse Elevation Control Mechanisms
Abuse Elevation Control Mechanisms
Obfuscated Files
Cloud Instance Metadata API
Access the Kubelet Main API 1 Event
Access the Kubelet Main API 1 Event
Account Manipulation
Exploitation for Privilege Escalation
Hijack Execution Flow
Credential Dumping
Account Discovery
Access the Kubernetes API Server 11 Events
Create Account 1 Event
Privileged Container 16 Events
Impair Defences
Man-in-the-Middle 1 Event
Cloud Instance Metadata API
Lateral Tool Transfer
Event Triggered Execution 2 Events
Writable Volumes
Compile After Delivery
Unsecured Credentials
Network Service Scanning 3 Events
Software Deployment Tools 15 Events

With the filter, only techniques that have been detected are displayed. In the following screenshot, we've narrowed the data to:

- Audits within the past seven days.
- Containers in the frontend collection, which are exposed to the Internet, and likely where the attack started.
- Attack techniques used by the adversary.



Mapping audits to techniques

Every audit (for example, runtime, admission, and so on) maps to one or more techniques. The following table shows the mappings.

The **Techniques** column shows the technique to which an audit is always mapped.

The **Possible Additional Techniques** column shows the techniques that to which an audit can be optionally mapped, depending on changing information from the audit. For example, for some audits, on new files being created, we will check if the process that created the files is a compiler. If so, we also map the audit to the **Compile After Delivery** technique.

	Category	Audit Type	Techniques (techniques that the audit is always mapped to)	Possible Additional Techniques (techniques that the audit can optionally be mapped to, depending on changing information from the audit)
Runtime Audits	Cloud	cloudMetadataProbing	Cloud Instance Metadata API	-
Runtime Audits	Kubernetes	kubeletAPIAccess	Access Kubelet Main API	-
Runtime Audits	Kubernetes	kubeletReadonlyAccess	Query Kubelet Readonly API	-
Runtime Audits	Kubernetes	kubectlSpawned	Access the Kubernetes API Server Software Deployment Tools	Lateral Tool Transfer Exec Into Container Create Container Kubernetes Secrets
Runtime Audits	Kubernetes	kubectlDownloaded	Ingress Tool Transfer Software Deployment Tools,	-
Runtime Audits	Network	horizontalPortScanning	Network Service Scanning	-
Runtime Audits	Network	verticalPortScanning	Network Service Scanning	-
Runtime Audits	Network	explicitlyDeniedIP	-	-

	Category	Audit Type	Techniques (techniques that the audit is always mapped to)	Possible Additional Techniques (techniques that the audit can optionally be mapped to, depending on changing information from the audit)
Runtime Audits	Network	customFeedIP	-	-
Runtime Audits	Network	feedIP	Command and Control / General Resource Hijacking	-
Runtime Audits	Network	unexpectedOutboundPort	Exfiltration Command and Control / General	-
Runtime Audits	Network	suspiciousNetworkActivity	-	Man In The Middle Network Service Scanning
Runtime Audits	Network	unexpectedListeningPort	-	-
Runtime Audits	Network	explicitlyDeniedListeningPort		-
Runtime Audits	Network	explicitlyDeniedOutboundPort	-	-
Runtime Audits	Network	listeningPortModifiedProcess	Command and Control / General	-
Runtime Audits	Network	outboundPortModifiedProcess	Exfiltration Command and Control / General	-
Runtime Audits	DNS	feedDNS	Command and Control / General Resource Hijacking	

	Category	Audit Type	Techniques (techniques that the audit is always mapped to)	Possible Additional Techniques (techniques that the audit can optionally be mapped to, depending on changing information from the audit)
Runtime Audits	DNS	explicitlyDeniedDNS	-	-
Runtime Audits	DNS	dnsQuery	-	-
Runtime Audits	Processes	unexpectedProcess	Native Binary Execution	
Runtime Audits	Processes	portScanProcess	Network Service Scanning	
Runtime Audits	Processes	explicitlyDeniedProcess	Native Binary Execution	
Runtime Audits	Processes	modifiedProcess	Foreign Binary Execution	
Runtime Audits	Processes	cryptoMinerProcess	Resource Hijacking	
Runtime Audits	Processes	lateralMovementProcess	-	-
Runtime Audits	Processes	tmpfsProcess	-	-
Runtime Audits	Processes	policyHijacked	Impair Defences	-
Runtime Audits	Processes	reverseShell	Native Binary Execution	-
Runtime Audits	Processes	SuidBinaries	Abuse Elevation Control Mechanisms	-
Runtime Audits	Processes	ProcUnknownOriginBinary	Foreign Binary Execution	-
Runtime Audits	Filesystem	administrativeAccount	-	Account Manipulation

	Category	Audit Type	Techniques (techniques that the audit is always mapped to)	Possible Additional Techniques (techniques that the audit can optionally be mapped to, depending on changing information from the audit)
				Create Account Abuse Elevation Control Mechanisms
Runtime Audits	Filesystem	sshAccess	-	Account Manipulation
Runtime Audits	Filesystem	explicitlyDeniedFile	-	-
Runtime Audits	Filesystem	malwareFileCustom	-	-
Runtime Audits	Filesystem	malwareFileFeed	-	-
Runtime Audits	Filesystem	execFileAccess	-	Masquerading IngressToolTransfer Compile After Delivery
Runtime Audits	Filesystem	elfFileAccess	-	IngressToolTransfer Compile After Delivery
Runtime Audits	Filesystem	secretFileAccess	-	-
Runtime Audits	Filesystem	regFileAccess	-	-
Runtime Audits	Filesystem	fileIntegrity	-	-
Runtime Audits	Filesystem	alteredBinary	Supply Chain Compromise	-
Runtime Audits	Filesystem	malwareDownloaded	Ingress Tool Transfer	
Runtime Audits	Filesystem	suspiciousELFHeader	Obfuscated Files	

	Category	Audit Type	Techniques (techniques that the audit is always mapped to)	Possible Additional Techniques (techniques that the audit can optionally be mapped to, depending on changing information from the audit)
Runtime Audits	Filesystem	executionFlowHijackAttempt	Hijack Execution Flow	
Runtime Audits	Filesystem	RuntimeAttackTypeFSEncryptedBinary	Obfuscated Files	
Runtime Audits	Filesystem	WildFireMalware	-	Masquerading IngressToolTransfer Compile After Delivery
Runtime Audits	Filesystem	webShell	Web Shell Ingress Tool Transfer	-
Runtime Audits	Filesystem	FSUnknownOriginBinary	-	Masquerading IngressToolTransfer Compile After Delivery
Runtime Custom Rule	Processes	Running privileged process within container	Software Deployment Tools	-
Runtime Custom Rule	Processes	Running cron app	Scheduled Task / Job	-
Runtime Custom Rule	Processes	Database app spawned process	Application Exploit (RCE) Exploitation Of Remote Services	-
Runtime Custom Rule	Processes	Suspicious networking tool	-	-

	Category	Audit Type	Techniques (techniques that the audit is always mapped to)	Possible Additional Techniques (techniques that the audit can optionally be mapped to, depending on changing information from the audit)
Runtime Custom Rule	Processes	Suspicious networking scaning tool	Network Service Scanning	-
Runtime Custom Rule	Processes	User creation (Container)	Create Account	-
Runtime Custom Rule	Processes	User deletion (Container)	Account Access Removal	-
Runtime Custom Rule	Processes	User modification (Container)	Account Manipulation	-
Runtime Custom Rule	filesystem	Bash shell tampering	Event Triggered Execution	-
Runtime Custom Rule	filesystem	Linux user management files	CreateAccount Account Manipulation	-
Runtime Custom Rule	filesystem	Configuration file changes (Host)	-	-
Runtime Custom Rule	filesystem	Configuration file changes (Container)	-	-
Runtime Custom Rule	network-outgoing	Common data exfiltration ports	Exfilitration	-
Runtime Custom Rule	network-outgoing	Common crypto mining pool ports	Resource Hijacking	-

	Category	Audit Type	Techniques (techniques that the audit is always mapped to)	Possible Additional Techniques (techniques that the audit can optionally be mapped to, depending on changing information from the audit)
Runtime Custom Rule	network-outgoing	Cloud platform metadata API access (Container)	Cloud Instance Metadata API	-
WAAS	-	xss	ExploitationForPrivilegeEscalation	
WAAS	-	sqli	Exploit Public-Facing Application Application Exploit (RCE)	
WAAS	-	cmdi	Exploit Public-Facing Application Application Exploit (RCE)	
WAAS	-	lfi	Exploit Public-Facing Application Application Exploit (RCE)	
WAAS	-	codeInjection	Exploit Public-Facing Application Application Exploit (RCE)	
WAAS	-	deniedIP	-	-
WAAS	-	deniedCountry	-	-
WAAS	-	header	-	-

	Category	Audit Type	Techniques (techniques that the audit is always mapped to)	Possible Additional Techniques (techniques that the audit can optionally be mapped to, depending on changing information from the audit)
	-			
WAAS	-	attackTools	NetworkServiceScanning	-
WAAS	-	shellshock	Exploit Public-Facing Application Application Exploit (RCE)	
WAAS	-	disallowedFile	-	-
WAAS	-	malformedRequest	-	-
WAAS	-	informationLeak	Exfilitration	System Credential Dumping System Account Discovery File And Directory Discovery System Unsecured Credentials Network Configuration Discovery Software Discovery
WAAS	-	unexpectedAPI	-	-
WAAS	-	dos	Endpoint Denial-of-Service	-
WAAS	-	searchEngineCrawler		-

	Category	Audit Type	Techniques (techniques that the audit is always mapped to)	Possible Additional Techniques (techniques that the audit can optionally be mapped to, depending on changing information from the audit)
WAAS	-	businessAnalyticsBot	-	-
WAAS	-	educationalBot	-	-
WAAS	-	newsBot	-	-
WAAS	-	financialBot	-	-
WAAS	-	contentFeedClient	-	-
WAAS	-	archivingBot	-	-
WAAS	-	careerSearchBot	-	-
WAAS	-	mediaSearchBot	-	-
WAAS	-	genericBot	-	-
WAAS	-	webAutomationTool	-	-
WAAS	-	webScraper	-	-
WAAS	-	apiLibrary	-	-
WAAS	-	httpLibrary	-	-
WAAS	-	sessionValidation	-	-
WAAS	-	javascriptTimeout	-	-
WAAS	-	missingCookie	-	-
WAAS	-	browserImpersonation	-	-
WAAS	-	requestAnomalies	-	-
WAAS	-	userDefinedBot	-	-

	Category	Audit Type	Techniques (techniques that the audit is always mapped to)	Possible Additional Techniques (techniques that the audit can optionally be mapped to, depending on changing information from the audit)
Kubernetes Audits	-	GKE - pod created in host process namespace	Privileged Container	
Kubernetes Audits	-	GKE - pod created with host file system mount	-	-
Kubernetes Audits	-	GKE - pod created without security context	-	-
Kubernetes Audits	-	GKE - pod created on host network	Privileged Container	-
Kubernetes Audits	-	GKE - privileged pod creation	Privileged Container	-
Kubernetes Audits	-	GKE - Forbidden request	-	-
Kubernetes Audits	-	GKE - exec or attach to a pod	Exec Into Container	-
Kubernetes Audits	-	Twistlock Labs - GKE - Tampering with Twistlock configuration	Impair Defences	-
Kubernetes Audits	-	Pod created in host process namespace	Privileged Container	-

	Category	Audit Type	Techniques (techniques that the audit is always mapped to)	Possible Additional Techniques (techniques that the audit can optionally be mapped to, depending on changing information from the audit)
Kubernetes Audits	-	Pod created with host file system mount	-	-
Kubernetes Audits	-	Pod created without security context	-	-
Kubernetes Audits	-	Pod created on host network	Privileged Container	-
Kubernetes Audits	-	Privileged pod creation	Privileged Container	-
Kubernetes Audits	-	Forbidden request	-	-
Kubernetes Audits	-	Exec or attach to a pod	Exec Into Container	-
Kubernetes Audits	-	Twistlock Labs - Tampering with Twistlock configuration	Impair Defences	-
Kubernetes Admission	-	CIS - Privileged pod created	Privileged Container	-
Kubernetes Admission	-	CIS - Pod created in host process ID namespace	Privileged Container	-
Kubernetes Admission	-	CIS - Pod created on host IPC namespace	Privileged Container	-

	Category	Audit Type	Techniques (techniques that the audit is always mapped to)	Possible Additional Techniques (techniques that the audit can optionally be mapped to, depending on changing information from the audit)
Kubernetes Admission	-	CIS - Pod created on host network	Privileged Container	-
Kubernetes Admission	-	CIS - Privilege escalation pod created	Privileged Container	-
Kubernetes Admission	-	Pod created with sensitive host file system mount	Writable Volumes	-
Kubernetes Admission	-	Exec or attach to a pod	Exec Into Container	-

Runtime Audits

Edit on GitHub

This document summarizes all the runtime audits (detections) that are available in Prisma Cloud Compute. For each detection, you can learn more about what it actually detects, how to enable or disable it, avoid false positives, relevant workloads (Containers, Hosts, Serverless and App-embedded), and if the audit also generates an incident.

Runtime detections for processes

Detection	Context	Audit message	Trig an inci	Workloads
Unexpected Process	Indicates when a process that is not part of the runtime model was spawned. • Avoid audits for specific known and allowed processes, by adding the process name to the runtime rules processes **Allowed** list. • In order to add the processes to the model, navigate to the relevant model under **Monitor > Runtime > Container** models, then click on **...** and select **Extend learning**	• <process> launched but is not found in the runtime model • <process> launched from <parent process> but is not found in the runtime model		Containers
Port Scanning Process	Indicates a process was spawned, that is identified as being used for port scanning. • Enable and disable this detection via the **Port scanning** toggle, under the Runtime rule Processes tab • Avoid audits on specific known and allowed processes, by adding process names to the runtime rule processes **Allowed** list.	<process> launched and is identified as a process used for port scanning	Port scanning	Containers
Explicitly Denied Process	Indicates that a process listed in the **Denied & fallback** list was spawned. • For App-embedded and Serverless, this indicates that a process that is not listed in the **Allowed** list was spawned	<process> launched and is explicitly denied by runtime rule. Full command <command>		Containers, Host, Serverless, App-embedded

Detection	Context	Audit message	Trig an inci	Workloads
Modified Process	Indicates a modified process was spawned. A modified process is a process whose binary was created or modified after the container was started. • Enable and disable this detection via the **Processes started from modified binaries** toggle, under the Runtime rule Processes tab • Avoid audits on specific known and allowed processes, by adding process names to the runtime rules processes **Allowed** list.	A modified executable <process> was launched		Containers, App-embedded
Altered Binary	Indicates that a package binary file was replaced during image build. This detection will generate an audit when a process is started from an altered binary. • Enable and disable this detection via the **Processes started from modified binaries** toggle, under the Runtime rule Processes tab • Avoid audits on specific known and allowed processes, by adding process names to the runtime rules processes **Allowed** list.	<process path> launched and is detected as an altered or corrupted package binary. The file metadata doesn't match what's reported by the package manager.	Altered binary	Containers, App-embedded
Crypto Miner Process	Indicates a process that is identified as a crypto miner was spawned. • Enable and disable this detection via the **Crypto miners** toggle, under the Runtime rule Processes / Anti-malware tab. • Avoid audits on specific known and allowed processes, by adding process names to the runtime rules processes **Allowed** list.	<process> launched and is identified as a crypto miner. Full command: <path>	Crypto miners	Containers, Hosts, Serverless, App-embedded
Lateral Movement Process	Indicates a process that is used for lateral movement was spawned. • Enable and disable this detection via the **Processes used for lateral**	<process> launched and is identified as a process used for lateral movement.	Lateral movement	Containers

Detection	Context	Audit message	Trig an inci	Workloads
	movement toggle, under the Runtime rule Processes tab. • Avoid audits on specific known and allowed processes, by adding process names to the runtime rules processes **Allowed** list.	Full command: <path>		
Temporary File System Process	Indicates that a process is running from a temporary file system. • Enable and disable this detection via the **Processes running from temporary storage** toggle, under the Runtime rule Anti-malware tab. • Avoid audits on specific known and allowed processes, by adding process names to the runtime rules processes **Allowed** list.	<process> launched from a temporary file storage, which usually indicates malicious activity.		Hosts
Policy Hijacked	Indicates that the Prisma Cloud process policy was hijacked	Possible tampering of Defender policy detected.		Serverless
Reverse Shell	Indicates that a process was identified as running a reverse shell • Enable and disable this detection via the **Reverse shell attacks** toggle, under the Runtime rule Processes / Anti-malware tab. • Avoid audits on specific known and allowed processes, by adding process names to the runtime rules processes **Allowed** list.	<processes> is a reverse shell . Full command: <path>	Reverse shell	Containers, Hosts
Suid Binaries	Indicates that a process is running with high priviliges, by watching for binaries with the setuid bit that are executed. • Enable and disable this detection via the **Processes started with SUID** toggle, under the Runtime rule Processes tab.	<process> launched and detected as a process started with SUID. Full command: <path>		Containers

Detection	Context	Audit message	Trig an inci	Workloads
Unknown Origin Binary by service	Indicates detection of binaries created by a service without a package manager. • Enable and disable this detection via the **Non-packaged binaries created or run by service** toggle, under the Runtime rule Anti-malware tab. • You can also select to **Suppress detection for binaries created by compilation tools**, to ignore binaries that are created by a specific compilation tool. • Avoid audits on specific known and allowed processes, by adding process names to the runtime rules processes **Allowed** list.	<process path> launched from a binary file which was written by <creating process path> that is not known OS distribution package manager.		Hosts
Unknown Origin Binary by user	Indicates detection of a binary created by a user without a package manager. • Enable and disable this detection via the **Non-packaged binaries created or run by user** toggle, under the Runtime rule Anti-malware tab. • Avoid audits on specific known and allowed processes, by adding process names to the runtime rules processes **Allowed** list.	<process path> launched from a binary file which was written by <creating process path> that is not known OS distribution package manager.		Hosts
Web Shell	Indicates that the process was launched by a web shell • Enable and disable this detection via the **Webshell attacks** toggle, under the Runtime rule Anti-malware tab. • Avoid audits on specific known and allowed processes, by adding process names to the runtime rules processes **Allowed** list.	<process path> suspected to be launched by a webshell at <path>		Hosts

Container general runtime detections

Detection	Context	Audit message	Trig an inci	Workloads
Cloud Metadata Probing	Indicates the container is trying to access a cloud provider metadata server. • Enable and disable this detection via the **Suspicious queries to cloud provider APIs** toggle, under the Runtime rule Anti-malware tab	Container queried provider API at <address>		Containers
Kubelet API Access	Indicates that a container is trying to access the Kubelet main API. • Enable and disable this detection via the **Kubernetes attacks** toggle, under the Runtime rule Anti-malware tab. • Avoid audits on specific known and allowed processes, by adding process names to the runtime rules processes **Allowed** list.	Container queried kubelet API at <address>	Kubernetes attacks	Containers
Kubelet Readonly Access	Indicates that a container is trying to access the Kubelet readonly API. • Enable and disable this detection via the **Kubernetes attacks** toggle, under the Runtime rule Anti-malware tab. • Avoid audits on specific known and allowed processes, by adding process names to the runtime rules processes **Allowed** list.	Container queried kubelet readonly API at <address>	Kubernetes attacks	Containers
Kubectl Spawned	Indicates the kubectl process was spawned from the container. • Enable and disable this detection via the **Kubernetes attacks** toggle, under the Runtime rule Anti-malware tab. • Avoid audits on specific known and allowed processes, by adding process names to the runtime rules processes **Allowed** list.	kubelet launched inside a container	Kubernetes attacks	Containers

Detection	Context	Audit message	Trig an inci	Workloads
Kubectl Downloaded	Indicates that the kubectl binary was downloaded and written to the disk. • Enable and disable this detection via the **Kubernetes attacks** toggle, under the Runtime rule Anti-malware tab. • Avoid audits on specific known and allowed processes, by adding process names to the runtime rules processes **Allowed** list.	<process path> downloaded kubectl to container.	Kubernetes attacks	Containers

Runtime detections for Network activities

Detection	Context	Audit message	Trig an inci	Workloads
Horizontal Port Scanning	Indicates horizontal port scanning detected • Enable and disable this detection via the **Port scanning** toggle, under the Runtime rule Networking tab.	Horizontal port scanning <process> to target IP <IP address> detected. Target ports <ports>	Port scanning	Containers
Vertical Port Scanning	Indicates vertical port scanning detected • Enable and disable this detection via the **Port scanning** toggle, under the Runtime rule Networking tab.	Vertical port scanning <process> to target IP <IP address> detected. Target ports <ports>	Port scanning	Containers
Explicitly Denied IP	Indicates that access to an IP address listed in the **Denied & fallback** list was detected. For App-embedded and Serverless, this indicates that access was detected to an IP address that is not listed in the **Allowed** list	Outbound connection <process> to IP <ip address> is explicitly denied by a runtime rule		Containers, Hosts, Serverless, App-embedded
Custom Feed IP	Indicates detection of a connection to a high risk IP, based on a custom feed. • Enable and disable this detection for **Containers** via the **Prisma Cloud advanced threat protection**	Connect to <address> is high risk, based on custom IP feed.		Containers, Hosts

Detection	Context	Audit message	Trig an inci	Workloads
	toggle, under the Runtime rule Anti-malware tab. • Enable and disable this detection for **Hosts** via the **Suspicious IPs based on custom feed** toggle, under the Runtime rule Networking tab.			
Feed IP	Indicates a connection to a high risk IP, based on intelligence feed data. • Enable and disable this detection for **Containers** via the **Prisma Cloud advanced threat protection** toggle, under the Runtime rule Anti-malware tab. • Enable and disable this detection for **Hosts** via the **Suspicious IPs based on Prisma Cloud advanced threat protection** toggle, under the Runtime rule Networking tab.	Connect to <address> is high risk. Intelligence stream categorizes <address> as <malware>.		Containers, Hosts
Unexpected Outbound Port	Indicates detection of an outbound connection on a port that is not part of the runtime model. • To avoid audits on specific ports, add the port to the runtime rule's Networking **Outbound internet ports** Allowed list, under **Defend > Runtime > Container policies** rules. • In order to add the processes to the model, navigate to the relevant model under **Monitor > Runtime > Container** models, click on **...** and select Extend learning	Outbound connection to an unexpected port: <destination port> IP: <destination ip>		Containers
Unexpected Listening Port	Indicates a container process is listening on a port that is not part of the runtime model. • To avoid audits on specific ports, add the port to the runtime rule's Networking **Listening ports** Allowed list, under **Defend > Runtime > Container policies** rules.	Process <process path> is listening on unexpected port <port>		Containers

Detection	Context	Audit message	Trig an inci	Workloads
	• In order to add the processes to the model, navigate to the relevant model under **Monitor > Runtime > Container** models, click on the **...** and select Extend learning			
Suspicious Network Activity	Indicates detection of a process performing raw socket usage. • Enable and disable this detection via the **Raw sockets** toggle, under the Runtime rule Networking tab.	Process <process name> performed suspicious raw network activity, <attack> • The <attack> could indicate an ARP spoofing attempt or a port scanning attempt		Containers
Explicitly Denied Listening Port	Indicates a container process is listening on a port that is explicitly listed in the **Listening ports** list, under **Denied & fallback**. For App-embedded and Serverless, this indicates ports that are not listed in the Allowed Listening ports list.	Process <process name> is listening on port <port> explicitly denied by a runtime rule		Containers, Hosts, Serverless, App-embedded
Explicitly Denied Outbound Port	Indicates a container process uses an outbound port that is explicitly listed in the **Outbound internet ports** list under **Denied & fallback**. For App-embedded and Serverless, this indicates ports that are not listed in the **Outbound ports** list under **Allowed**.	Outbound connection <process> to port <destination port> (IP: <destination ip>) is explicitly denied by a runtime rule.		Containers, Hosts, Serverless, App-embedded
Listening Port Modified Process	Indicates a container modified process is listening on an unexpected port. • Enable and disable this detection via the **Networking activity from modified binaries** toggle, under the Runtime rule Networking tab. • To avoid getting such an event for an allowed port, add the port to the Runtime rule's **Allowed Listening ports** list.	Container process <process> was modified and is listening on unexpected port		Containers

Detection	Context	Audit message	Trig an inci	Workloads
Outbound Port Modified Process	Indicates a container modified process opened an outbound port. • Enable and disable this detection via the **Networking activity from modified binaries** toggle, under the Runtime rule Networking tab. • To avoid getting such an event for an allowed port, add the port to the Runtime rule's **Allowed Outbound internet ports** list.	Outbound connection by modified process <process> to port: <destination port> IP: <destination IP>		Containers
Feed DNS	Indicates a DNS resolution query for a high risk domain, based on an intelligence stream. • Enable and disable this detection for **Containers** via the **Prisma Cloud advanced threat protection** toggle, under the Runtime rule Anti-malware tab. • Enable and disable this detection for **Hosts** via the **Suspicious domains based on Prisma Cloud advanced threat protection** toggle, under the Runtime rule Networking tab. • Make sure that the DNS toggle in the Runtime rule Networking tab is enabled as well • To avoid getting such an event for a known and allowed domain, add the domain name to the Runtime rule's **Domains** list under **Allowed** in the Networking tab.	<domain name> identified as high risk. Intelligence feed categorizes this domain as <malicious category>		Containers, Hosts
Explicitly Denied DNS	Indicates a DNS resolution query for a blacklisted domain, that is explicitly listed in the **Domains** list, under **Denied & fallback** in the Networking tab. For App-embedded and Serverless, this indicates domains that are not listed in the Allowed Domains list.	DNS resolution of domain name <domain name> triggered by <process path> explicitly denied by runtime rule.		Containers, Hosts, Serverless, App-Embedded

Detection	Context	Audit message	Trig an inci	Workloads
	• Make sure that the DNS toggle in the Runtime rule Networking tab is enabled as well.			
DNS Query	Indicates a DNS resolution query of a domain name that is not part of the runtime model. • To avoid getting such an event for a known and allowed domain, add the domain name to the Runtime rule's **Domains** list, under **Allowed** in the Networking tab.	DNS resolution of suspicious name <domain name>, type <domain type>		Containers

Runtime detections for File system activities

Detection	Context	Audit message	Trig an inci	Workloads
Administrative Account	Indicates that an administrative account file was accessed. Changes to such files can be related to backdoor attacks. • Enable and disable this detection via the **Changes to SSH and admin account configuration files** toggle, under the Container/App-Embedded Runtime rule's File system tab. • To ignore such a detection for a known and allowed process, create a Runtime custom rule that allows these file changes by a specific process.	<process name> wrote to administrative accounts configuration file <path>	Back admi acco	Containers, App-Embedded
SSH Access	Indicates that a ssh config file was accessed • Enable and disable this detection via the **Changes to SSH and admin account configuration files** toggle, under the Container/App-Embedded Runtime rule's File system tab.	<process name> wrote to SSH configuration file <path>	Back SSH acces	Containers, App-Embedded

Detection	Context	Audit message	Trig an inci	Workloads
	• To ignore such a detection for a known and allowed process, create a Runtime custom rule that allows these file changes by a specific process.			
Encrypted Binary	Indicates that an encrypted binary was written to disk, by checking the binary entropy. • Enable and disable this detection via the **Detection of encrypted/ packed binaries** toggle, under the **Container/App-Embedded** Runtime rule File system tab. • Enable and disable this detection via the **Encrypted/packed binaries** toggle, under the **Host** Runtime rule Anti-malware tab. • To ignore such a detection for a known and allowed process, create a Runtime custom rule that allows these file changes by a specific process.	<process name> wrote a suspicious packed/ encrypted binary to <path>. Packing/ encryption can conceal malicious executables.		Containers, Hosts, App-Embedded
Explicitly Denied File	Indicates that a file listed in the File system **Denied & fallback** list was accessed.	<process name> changed explicitly monitored file <path>		Containers, App-Embedded
Malware File Custom	Indicates that a file that is identified as malware, based on a custom feed, was accessed. • Enable and disable this detection for **Containers** via the **Prisma Cloud advanced threat protection** toggle, under the Runtime rule Anti-malware tab. • Enable and disable this detection for **Hosts** via the **Malware based on custom feed** toggle, under the Runtime rule Anti-malware tab. • Enable and disable this detection for App-embedded via the Custom feed	<process name> created <file path> which was detected as <malware name> malware in the custom malware feed	Malware	Containers, Hosts, App-Embedded

Detection	Context	Audit message	Trig an inci	Workloads
	for malware detection toggle, under the Runtime rule File system tab.			
Malware File Feed	Indicates that a file that is identified as malware, based on the intelligence stream, was accessed. • Enable and disable this detection for **Containers** via the **Prisma Cloud advanced threat protection** toggle, under the Runtime rule Anti-malware tab. • Enable and disable this detection for **Hosts** via the Malware based on **Prisma Cloud advanced threat protection** toggle, under the Runtime rule Anti-malware tab.	Process <process name> created the file <file path> which was detected as malicious. Intelligence feed identifies the file as <malware name>	Malware	Containers, Hosts
Executable File Access	Indicates that an executable file was written. • Enable and disable this detection via the **Changes to binaries and certificates** toggle, under the Runtime rule File system tab. • To ignore such a detection for a known and allowed process, create a Runtime custom rule that allows these file changes by a specific process	<process name> changed the binary <file path>		Containers, App-Embedded
ELF File Access	Indicates that an ELF file, that is not part of the runtime model, was modified. • This detection works automatically when using a Container runtime model. • To disable this detection, disable the **Enable automatic runtime learning** toggle under the **Defend > Runtime > Container policy** tab.	<process name> changed the binary <file path>		Containers, App-Embedded
Secret File Access	Indicates that a file containing sensitive key material, that is not part of the runtime model, was written.	<process name> created a key file at <file path>		Containers, App-Embedded

Detection	Context	Audit message	Trig an inci	Workloads
	• This detection works automatically for containers when using a Container runtime model. • To disable this detection for containers, disable the **Enable automatic runtime learning** toggle under the **Defend > Runtime > Container policy** tab. • Enable and disable this detection for app-embedded via the Changes to binaries and certificates toggle, under the Runtime rule File system tab.			
Regular File Access	Indicates that a regular file, that is not part of the runtime model, was created. • This detection works automatically when using a Container runtime model. • For Serverless, this works when adding the path to the **Denied & fallback** list under File System. • To disable this detection, disable the **Enable automatic runtime learning** toggle under the **Defend > Runtime > Container policy** tab.	• Container: <process name> wrote suspicious file to <file path> • Serverless: <process name> access a suspicious path of <file path>		Containers, Serverless, App-Embedded
WildFire Malware detection	Indicates that a file detected by WildFire as malware was written to the file system. To enable or disable WildFire: • Open the **Manage > system > WildFire** page and configure the desired settings • Open the Runtime rule for Containers, Hosts, or App-Embedded, and enable/disable the **Use WildFire malware analysis**, under the Anti-malware tab	Process <process name> created the file <file name> with MD5 <MD5>. The file created was detected as malicious. Report URL: <report url>	Malware	Containers, Hosts, App-Embedded

Detection	Context	Audit message	Trig an inci	Workloads
Unknown Origin Binary	Indicates that a binary file was written by a process that is not a known OS distribution package manager. • Enable and disable this detection via the **Non-packaged binaries created or run by user** and **Non-packaged binaries created or run by service** toggles, under the Runtime rule Anti-malware tab. • To ignore such a detection for a known and allowed process, create a Runtime custom rule that allows these file changes by a specific process	<process name> which is not a known OS distribution package manager wrote the binary <path>		Hosts
Web Shell	Indicates that a file written to disk was detected as a web shell. • Enable and disable this detection via the **Webshell attacks** toggle, under the **Host** Runtime rule Anti-malware tab • To ignore such a detection for a known and allowed process, create a Runtime custom rule that allows these file changes by a specific process	<process name> wrote the file <file path> that was detected as a web shell.		Hosts
File Integrity	Indicates that file integrity detection was audited. • To configure File integrity detections, open the Host runtime rule, navigate to the File integrity tab, and create rules to add specific detections.			Hosts
Malware Downloaded	Indicates when a binary that has an architecture not supported by PC Compute Defender, is written to disk by a file download utility ("wget", "curl", etc.). PC Compute Defender supports the x86_64 architecture. • Enable and disable this detection via the **Binaries with suspicious**	Suspected malicious ELF file <file path> downloaded by process <process name> that is spawned by service <service name> [For interactive	Susp binary	Containers, Hosts, App-Embedded

Detection	Context	Audit message	Trig an inci	Workloads
	ELF headers toggle, under the **Containers/App-Embedded** Runtime rule File system tab, or under the **Host** Runtime rule Anti-malware tab. • To ignore such a detection for a known and allowed process, create a Runtime custom rule that allows these file changes by a specific process	audits, should include: <audit message> and user <user>] <audit message>. Incompatible process architecture <architecture>.		
Suspicious ELF Header	Indicates that an ELF file with suspicious malware indicators in the header was created. The ELF header can indicate that the file was modified with anti-analysis techniques, which is used often by malware to avoid detection. • Enable and disable this detection via the **Binaries with suspicious ELF headers** toggle, under the **Containers/App-Embedded** Runtime rule File system tab, or under the **Host** Runtime rule Anti-malware tab. • To ignore such a detection for a known and allowed process, create a Runtime custom rule that allows these file changes by a specific process	Suspected malicious ELF file <file path>. File headers indicate anti-analysis techniques have been used to modify the file, which is used often by malware to avoid detection.	Suspicious binary	Containers, Hosts, App-Embedded
Execution Flow Hijack Attempt	Indicates a possible attempt to hijack program execution flow. For example, an audit will be generated when a process writes to /etc/ld.so.preload. • Enable and disable this detection via the **Execution flow hijacking** toggle, under the Host Runtime rule Anti-malware tab • To ignore such a detection for a known and allowed process, create a Runtime custom rule that allows these file changes by a specific process	Binary <process name> wrote to <file path>. File /etc/ld.so.preload is a special Linux system file that impacts the entire system. Libraries specified in this file are preloaded for all programs that are executed in the system.	Execution flow hijack attempt	Hosts

Event Aggregation

Edit on GitHub

This document explains logic behind runtime event aggregation in Prisma Cloud Compute.

When a high number of events of the same event type are reported on the same image in specific host, Prisma Cloud Compute starts aggregating them to avoid Console inflating with large number of similar audits.

For event aggregation to start, the following conditions must take place:

- Events are generated on the same resource. Example: a specific image on a host.
- Events generated are the same "Event Type" category (not specific audit). Example: Network / Unexpected Listening Port, Filesystem / Reg File Access etc.
- More than 5 events satisfying the above conditions are reported within a 15 minutes timeframe.

When such report aggregation starts, a message with the same is recorded in the Events table.

- **Example** --

  High rate of reg file access events, reporting aggregation started; last event: /sbin/apk wrote a suspicious file to /usr/lib/node_modules/npm/.apk.f64bd79770d6df713fa07ddeabb044bb3eb76ffb554c2dab. Command: apk add npm

Aggregation happens for 10 minutes, after which a message with the most recent audit is displayed.

- **Example** --

  In the past 10 minutes, container /aqsa_high_alerts had 3 events of type unexpected listening port; The most recent event was: Container process /usr/local/bin/np is listening on unexpected port 8888

After aggregation period is completed, any new events that occur, are identified uniquely and go through the same logic above for aggregation as applicable.

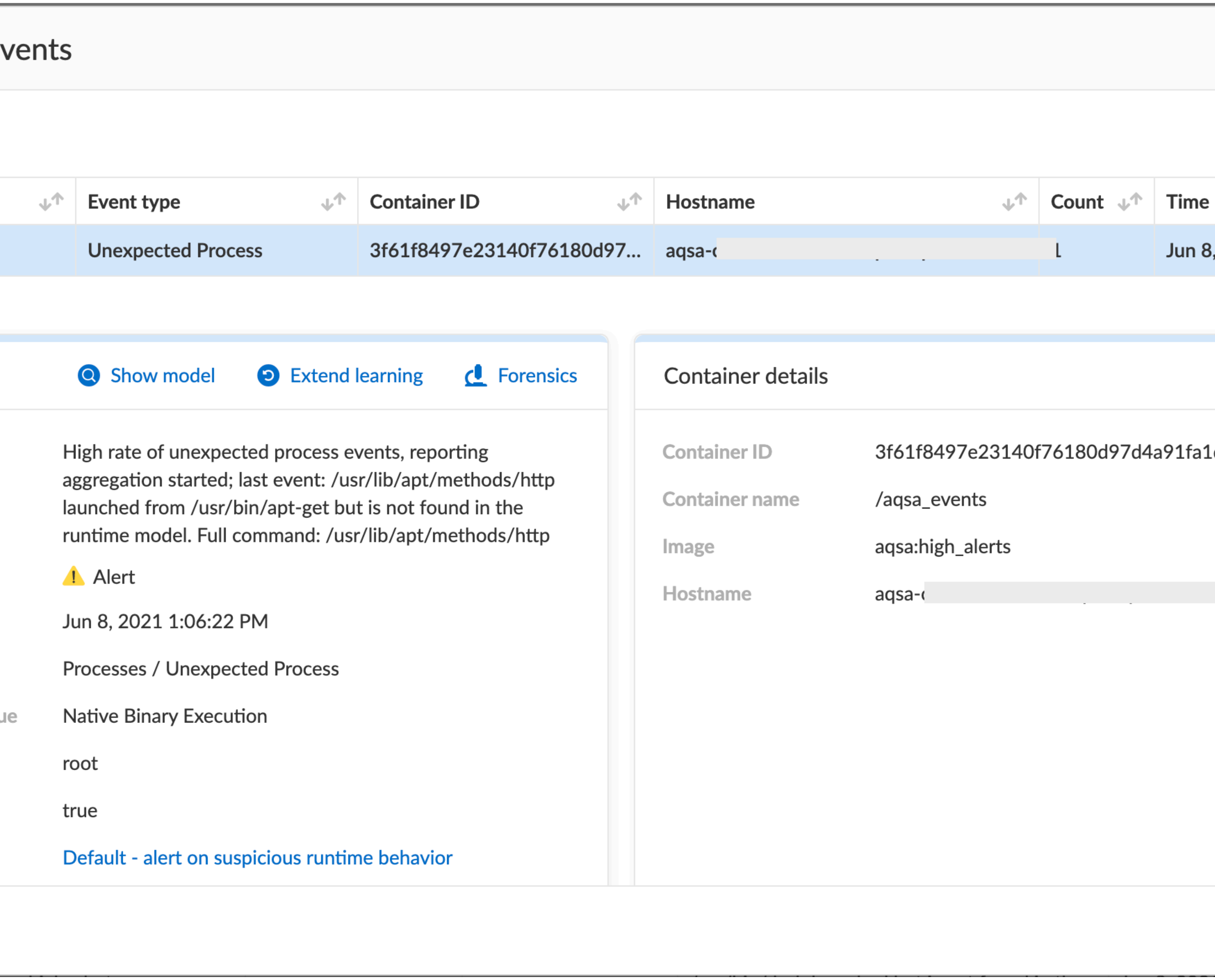

vents
Event type
Container ID
Hostname
Count
Time
Unexpected Process
3f61f8497e23140f76180d97...
aqsa-
Jun 8,
Show model
Extend learning
Forensics
Container details
High rate of unexpected process events, reporting aggregation started; last event: /usr/lib/apt/methods/http launched from /usr/bin/apt-get but is not found in the runtime model. Full command: /usr/lib/apt/methods/http
Alert
Jun 8, 2021 1:06:22 PM
Processes / Unexpected Process
ue
Native Binary Execution
root
true
Default - alert on suspicious runtime behavior
Container ID
3f61f8497e23140f76180d97d4a91fa1
Container name
/aqsa_events
Image
aqsa:high_alerts
Hostname
aqsa-

Image analysis sandbox

Edit on GitHub

The image analysis sandbox lets you dynamically analyze the runtime behaviour of images before running them in your development and production environments.

The analysis mechanism collects and displays container behaviours by safely exercising the image in a sandbox machine. It also exposes risks and identifies suspicious dependencies buried deep in your software supply chain that would otherwise be missed by static analysis for vulnerabilities and compliance issues.

Running the analysis is supported for Linux images on Docker container runtime.

Setup the sandbox machine

In order to run a sandbox analysis for an image, you first need to set up a dedicated sandbox virtual machine.

Prerequisites:

- The twistcli tool should exist on the machine.
- The sandbox machine should have connectivity to Prisma Cloud Compute Console.
- The machine must be a Linux VM.
- Docker should be installed on the machine.

When setting up the VM, follow the guidelines below to make sure potential malware doesn't exploit your sandbox:

- Make sure that the kernel is up to date.
- Make sure that Docker and Runc are up to date.
- Make sure all the software components on the machine are up to date (to make sure there is no other vulnerable component on the machine).
- The VM should be as isolated as possible. Run the VM in a dedicated network, separate from production. If other services run alongside the sandbox VM in the same local network, set up firewall rules to ensure the sandbox VM cannot reach them.
- If the VM runs in the cloud, it shouldn't run with any service account.

It is recommended to avoid running a Defender on the same machine used as the sandbox VM. Running a Defender on this machine might cause the image that is being analyzed in the sandbox to also be presented under ***Monitor > Vulnerabilities/Compliance > Images > Deployed images*** *as an image running in the environment.*

Setup the sandbox user

Create a dedicated, least-privileged user for running the image analysis sandbox.

Running the sandbox with a privileged role (Admin, Operator) is a risk in case a malware escapes (by using a zero day, one day, exploit misconfiguration, etc.), and can potentially use this role to take over Prisma.

1. Create a custom role under **Manage > Authentication > Roles** with Write permissions for Container Runtime Results and Read permissions for CI Results. For roles created via the API, also add Write permission for User.
2. Create a sandbox user and assign it with the new custom role you created.
3. When triggering the sandbox analysis via twistcli, use the sandbox user credentials. It is recommended to use a short-lived token (available under **Manage > System > Utilities**) rather then a username and password.

Running the *sandbox* command

Description

Triggering a sandbox analysis is done by executing the *twistcli sandbox* command on an image. After the command is triggered, Prisma Cloud's sandbox mechanism runs the container, and starts tracing its behaviour. The events occuring on the running container are collected, and are later being analyzed to discover suspicious behaviours.

Synopsis

The usage of the *twistcli sandbox* command is very similar to running a container image using docker:

```
$ twistcli sandbox [OPTIONS] IMAGE [COMMAND] [ARG...]
```

For example:

```
$ twistcli sandbox --address https://<console-address>:8083 --token
 'your-api-token' --analysis-duration 2m -v "$PWD":/app python:3
 python3 /app/server.py
```

To specify an image to scan, use either the image ID, or repository name and tag. The image should be present on the sandbox machine, having either been built or pulled there. If a repository is specified without a tag, *twistcli* looks for an image tagged *latest*.

The entrypoint and arguments should be specified after the image. If an entrypoint isn't specified, the default entrypoint of the image will be used.

Options

- ***--address URL*** --

  Complete URL for Console, including the protocol and port. Only the HTTPS protocol is supported. By default, Console listens to HTTPS on port 8083, although your administrator can configure Console to listen on a different port. Defaults to https://127.0.0.1:8083.

  Example: --address https://console.example.com:8083
- ***-u, --user USERNAME*** --

  Username to access Console. If not provided, the *TWISTLOCK_USER* environment variable will be used if defined, or "admin" is used as the default.

- ***-p, --password PASSWORD*** --

  Password for the user specified with *-u, --user*. If not specified on the command-line, the *TWISTLOCK_PASSWORD* environment variable will be used if defined, or otherwise will prompt for the user's password before the scan runs.

- ***--project PROJECT NAME*** --

  Interface with a specific supervisor Console to publish the results.

  Example: *--project "Tenant Console"*

- ***--output-file FILENAME*** --

  Write the results of the analysis to a file in JSON format.

  Example: *--output-file analysis-results.json*

- ***--analysis-duration DURATION*** --

  The duration of the analysis in a Go duration string format. The default duration is 1 minute.

  Adjust the duration according to your image. A longer duration may allow detecting more behaviours. An analysis duration that is too short might cause missing some of the suspicious findings that could have been detected on the container.

  Example: *--analysis-duration 2m30s*

  The analysis duration can be shorter than the duration you specified, if the container exits before the analysis time ends.

  When WildFire integration is enabled, the analysis duration can be longer than specified, since the communication with WildFire may take longer than the analysis duration. When the specified duration is met, Prisma Cloud stops the container, so no more events are collected, but is waiting for WildFire verdict to publish the results.

- ***-e, --env ENVIRONMENT VAR*** --

  A key=value pair to define an environment variable in the running container. Repeat flag for each environment variable.

  Example: *-e "GOROOT=/usr/local/go" -e "HTTPS_PORT=4443"*

- ***-v, --volume VOLUME*** --

  A src:dst pair to mount a volume to the running container. Repeat flag for each mount.

  Example: *-v "/home/developer/app:/app" -v "/var/lib/mongo:/data"*

  Any volume that is shared with the sandbox will be accessible to a potential malware exists on the container. Therefore, carefully consider the usage of volumes.

- ***-w, --workdir DIRECTORY*** --

  Working directory inside the container.

  Example: *-w "/usr/src/myapp"*

- ***--port PORT*** --

  A host_port:container_port[/tcp|udp] pair to bind a host port the running container's port. Repeat for each port. Port ranges are not supported.

  Example: *--port "80:123/tcp"*
- ***--tlscacert PATH*** --

  Path to Prisma Cloud CA certificate file. If no CA certificate is specified, the connection to Console is insecure.
- ***--token TOKEN*** --

  Token to use for Prisma Cloud Console authentication. Tokens can be retrieved from the API endpoint *api/v1/authenticate* or from the **Manage > System > Utilities** page in Console.
- ***--exit-on-error TRUE/FALSE*** --

  Immediately exit the analysis if an error is encountered.
- ***-h, --help*** --

  Show help

Return value

The exit code is 0 if the sandbox analysis verdict is "Passed". If the verdict is "Failed", the exit code is 1.

The criteria for passing or failing the sandbox analysis is determined by the severity of the suspicious findings detected during the analysis. The analysis verdict is "Failed" when there is at least one finding with Critical or High severity. Otherwise, the verdict is "Passed".

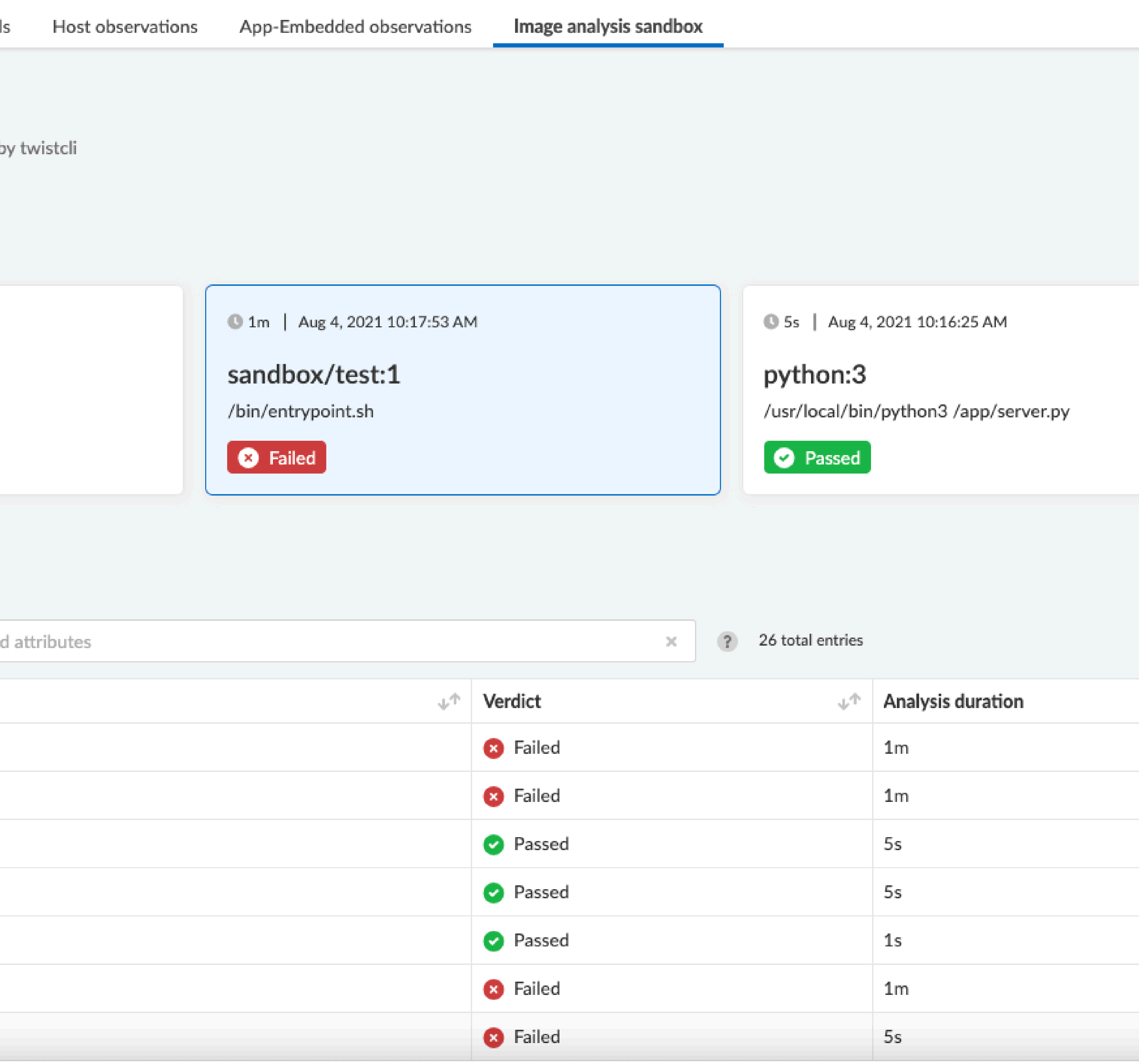
Another reason why *twistcli sandbox* might return an exit code of 1 is if the analysis failed due to an error.

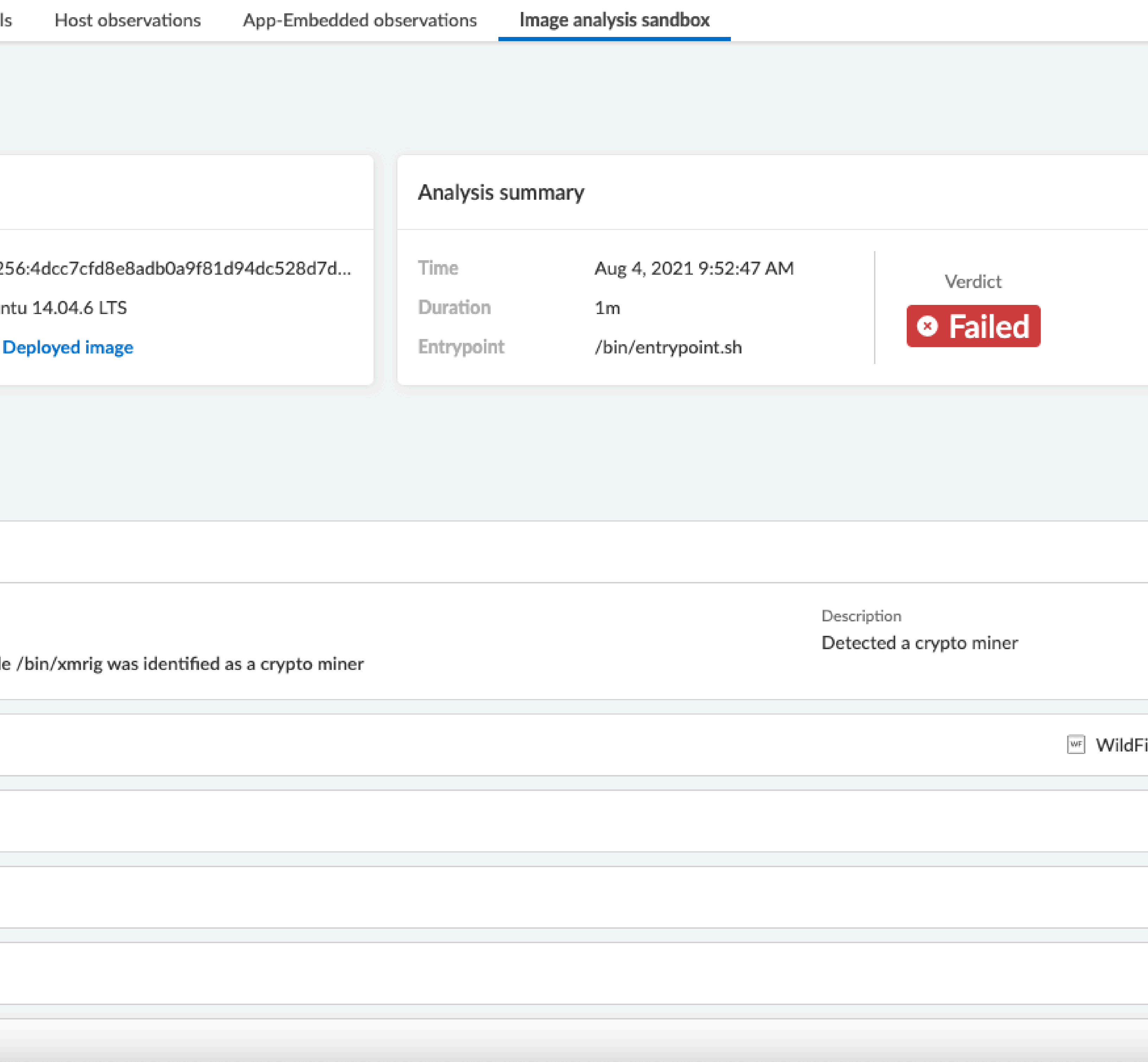
Sandbox analysis results

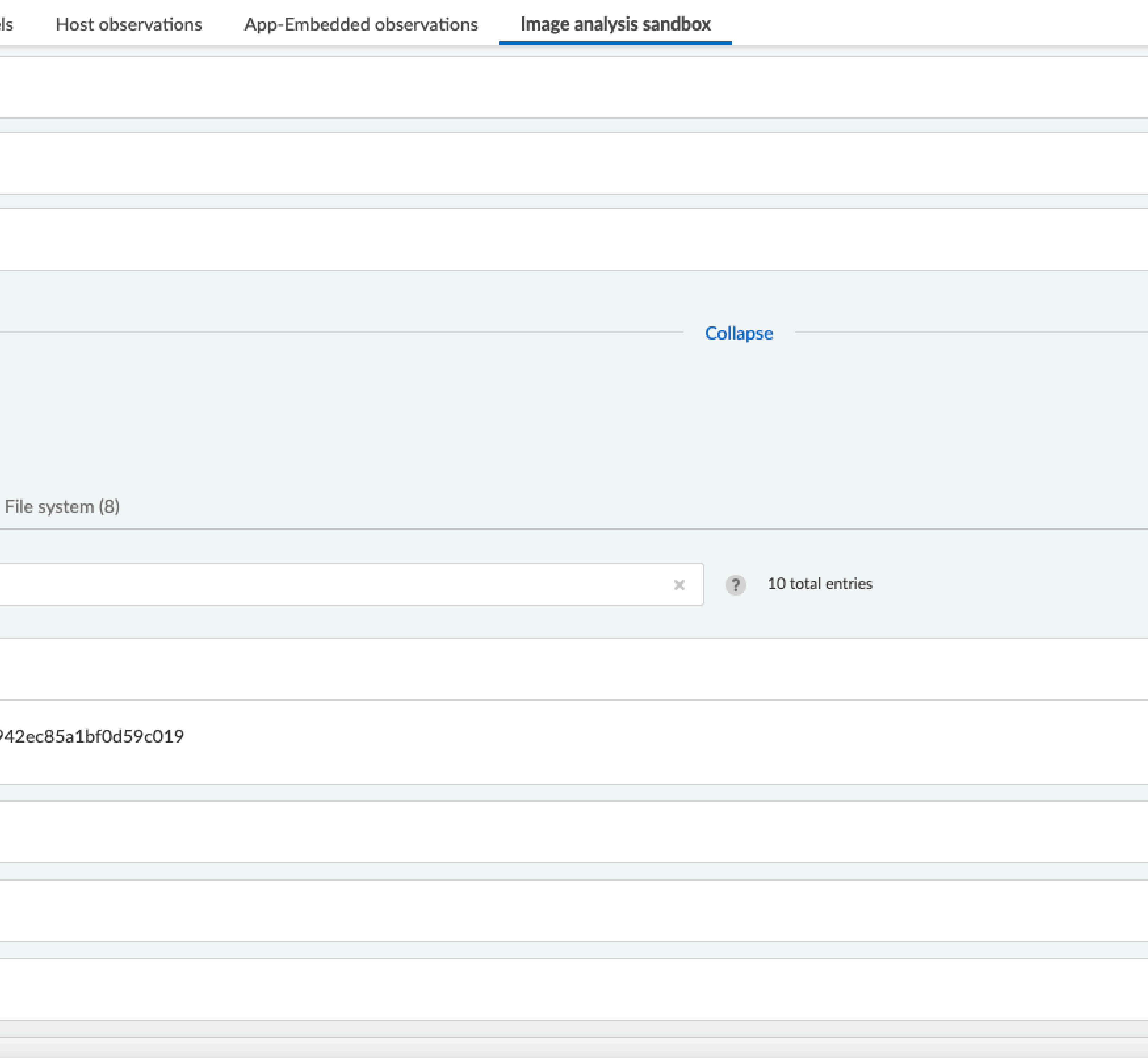
After *twistcli* dynamically analyzes the image, *twistcli*:

- Exits with a return value.
- Outputs a summary of the results, including a verdict.
- Outputs a link to the results report in the Console UI.

The results report in the Console UI includes the analysis summary and verdict, a list of suspicious detections found on the image, and the entire container behaviour events occurred during container runtime.


ls
Host observations
App-Embedded observations
Image analysis sandbox
by twistcli
1m | Aug 4, 2021 10:17:53 AM
sandbox/test:1
/bin/entrypoint.sh
Failed
5s | Aug 4, 2021 10:16:25 AM
python:3
/usr/local/bin/python3 /app/server.py
Passed
d attributes
26 total entries
Verdict
Analysis duration
Failed
1m
Failed
1m
Passed
5s
Passed
5s
Passed
1s
Failed
1m
Failed
5s


ls
Host observations
App-Embedded observations
Image analysis sandbox
Analysis summary
256:4dcc7cfd8e8adb0a9f81d94dc528d7d...
ntu 14.04.6 LTS
Deployed image
Time
Aug 4, 2021 9:52:47 AM
Duration
1m
Entrypoint
/bin/entrypoint.sh
Verdict
Failed
Description
Detected a crypto miner
le /bin/xmrig was identified as a crypto miner
WildFi



Analysis summary

The analysis summary contains the following main parts:

- Verdict - whether the image passed or failed the analysis.

  The criteria for passing or failing the sandbox analysis is determined by the severity of the suspicious findings detected during the analysis. The analysis verdict is "Failed" when there is at least one finding with Critical or High severity. Otherwise, the verdict is "Passed".
- Highest severity - the severity of the most severe suspicious finding.
- Suspicious findings count - the number of suspicious findings detected.
- Analysis metadate - analysis time, duration, and the container entrypoint.
- Image details - the details of the analyzed image.

  The image details also include an indication of an additional scan that may have been performed on the image. If the image was scanned for vulnerabilities and compliance as a part of the CI process, registry scanning, or as a deployed image, it will be displayed in the **Additional scan** field. You will also be able to click on its value to see the scan results. Only the furthest stage is reported in the following order: CI → Registry → Deployed.

Suspicious findings

The sandbox analysis mechanism detects the following suspicious behaviours:

Detection	Description	Severity
Malware	Malware detected by WildFire. Detecting malware using WildFire requires the WildFire integration to be enabled. Go to **Manage > System > WildFire** and turn on the "Enable runtime protection" toggle. You can also choose to upload files with unknown verdicts to WildFire using the matching toggle.	Critical
Crypto miners	Crypto miner was detected.	Critical
Suspicious ELF headers	ELF file with a suspicious header was detected. The binary is either incompatible with the system architecture or the ELF header was manipulated to hinder analysis. For ELF header tampering, Prisma Cloud identifies overlapping headers, deleted headers, and improperly specified section sizes as suspicious.	High
Vertical port scanning	Vertical port scanner was detected.	High
Kernel module modifiction	Kernel module was being loaded or unloaded.	High
Dropper	A binary that wasn't included in the original image (dropped on disk) was executed.	High

Detection	Description	Severity
Modified binary	A process modified a binary.	High
Modified binary execution	Execution of a binary that was included in the original image but has been modified.	High
Fileless Execution	Execution from a memory file descriptor was detected.	High
Fileless executable creation	An executable was written into a memory file descriptor.	High
Executable creation	A new executable file created on the disk.	Medium

Container behaviour

The sandbox analysis mechanism collects Processes, Networking, and Filesystem events that occurred while the container was running in the sandbox. The events are displayed in the Console UI analysis report, in order to provide you with an overview of the container behaviour at runtime.

There are two display modes for viewing the container behaviour events:

- By Type - the events are aggregated by the main event properties, to give you an overview of which process run on the container, what were the network destinations it was trying to reach, what are its listening ports, etc. For example, if a process was running three times, only a single row will appear for this process, with the common properties only (MD5), and without the properties that are changing between events (command, parent process, etc).
- By Time - all the events are presented ordered by the time they occurred. For example, if a process was running three times, three rows with the same process will appear, with different time, and with all the event details for each one of them (command, parent process, etc).

Filesystem events

For container filesystem, Prisma Cloud collects Open, Create, and Modify file events.

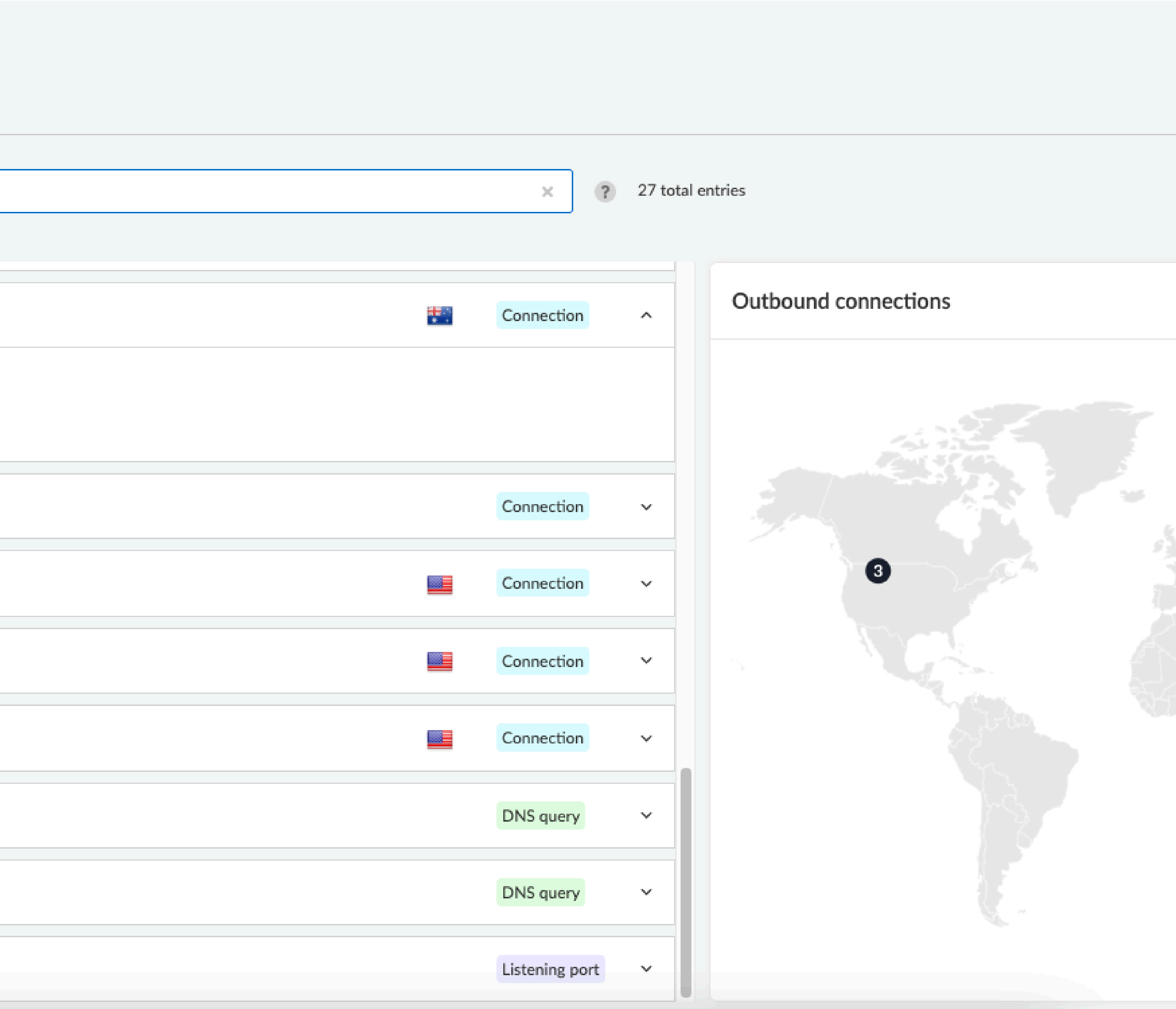
Network events

There are three event types collected for container networking:

- Listening port
- Outbound connection
- DNS query

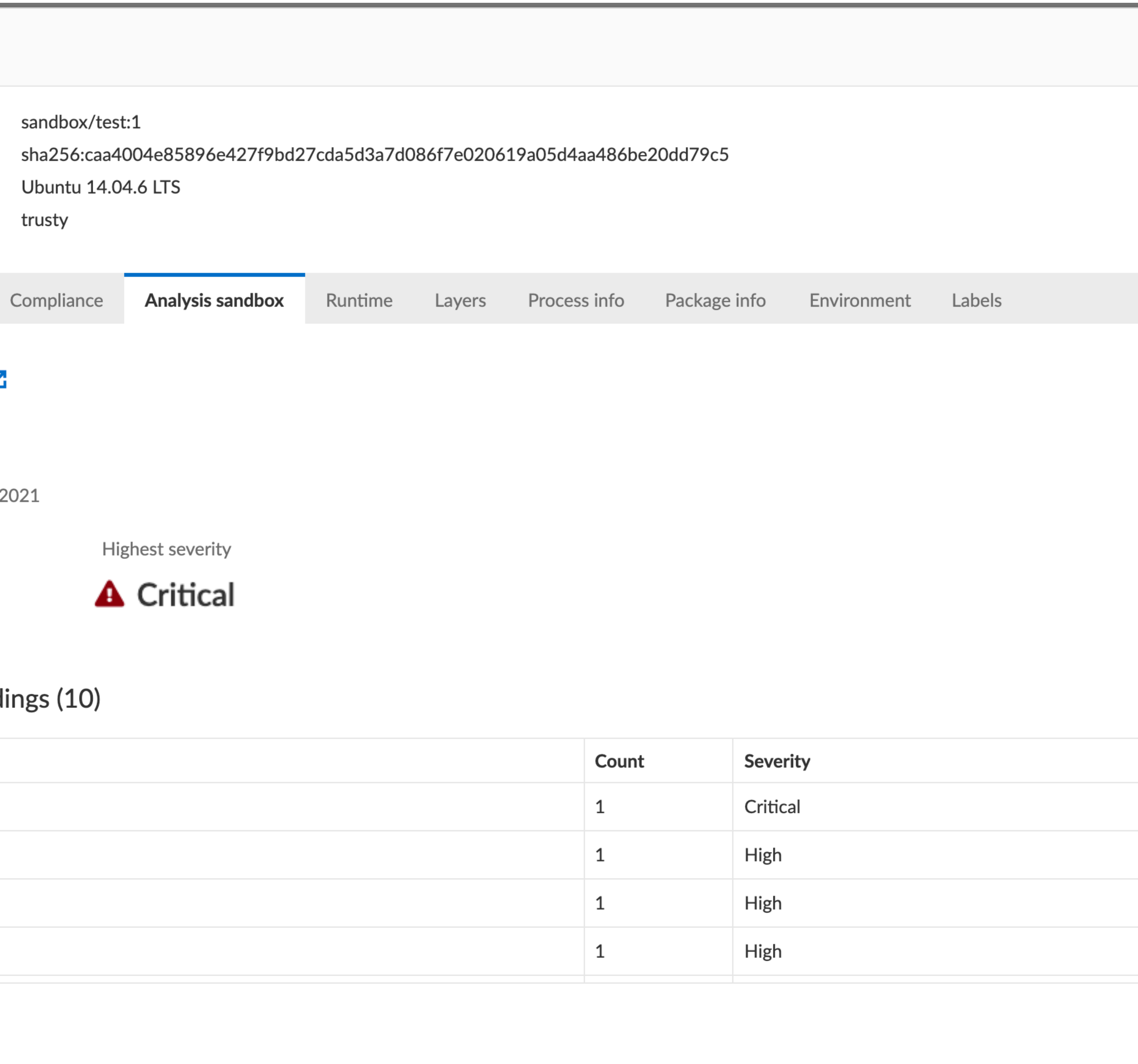
All three types are presented together under the **Networking** tab, but each has its own properties.

Outbound connection events are also displayed on a world map according to the country matching their IP. Clicking on a connection event will mark it on the map. Hovering a country on the map will show you how many connections were detected for this country.



View sandbox results on image details

When reviewing image details, you can look at its latest sandbox analysis results in a dedicated section. The **Anaysis sandbox** section contains an analysis summary, including the verdict and the suspicious findings counts by type. Click on the link at the top to move to the full report page.


sandbox/test:1
sha256:caa4004e85896e427f9bd27cda5d3a7d086f7e020619a05d4aa486be20dd79c5
Ubuntu 14.04.6 LTS
trusty
Compliance
Analysis sandbox
Runtime
Layers
Process info
Package info
Environment
Labels
2021
Highest severity
Critical
lings (10)
Count
Severity
1
Critical
1
High
1
High
1
High

Actions

Add to trust group

After reviewing the analysis results of an image, you can decide whether you trust this image to run in your development and production environments. Optionally, you can add the image repository to a single or multiple trust groups using the **Add to trust group** action. This way it is possible for you to get notified or block images that are not trusted. See Trusted Images to learn more.

Export to JSON file

To export the analysis results, use the **Export to JSON** action at the top of the page. This action will download a file in a JSON format with the analysis results for the image.

Incident Explorer

Edit on GitHub

Incident Explorer elevates raw audit data to actionable security intelligence, enabling a more rapid and effective response to incidents. Rather than having to manually sift through reams of audit data, Incident Explorer automatically correlates individual events generated by the firewall and runtime sensors to identify unfolding attacks.

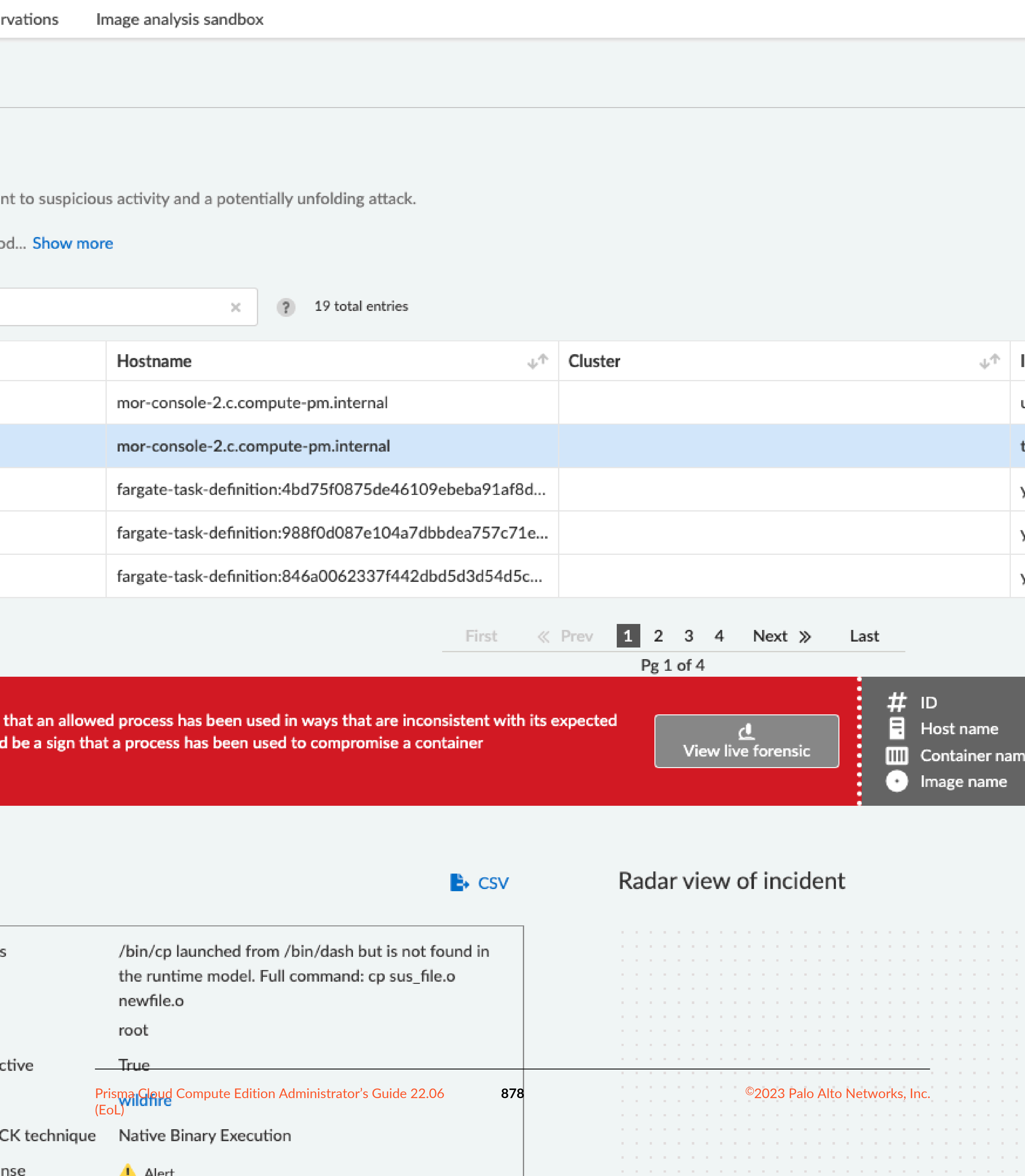
Audit events generated as a byproduct of an attack rarely occur in isolation. Attackers might modify a configuration file to open a backdoor, establish a new listener to shovel data out of the environment, run a port scan to map the environment, or download a rootkit to hijack a node. Each of these attacks is made up of a sequence of process, file system, and network events. Prisma Cloud's runtime sensors generate an audit each time an anomalous event outside the allow-list security model is detected. Incident Explorer sews these discrete events together to show the progression of a potential attack.

To learn more about the challenges of incident response in cloud native environments, and how Prisma Cloud can help, see this webinar recording.

Viewing incidents

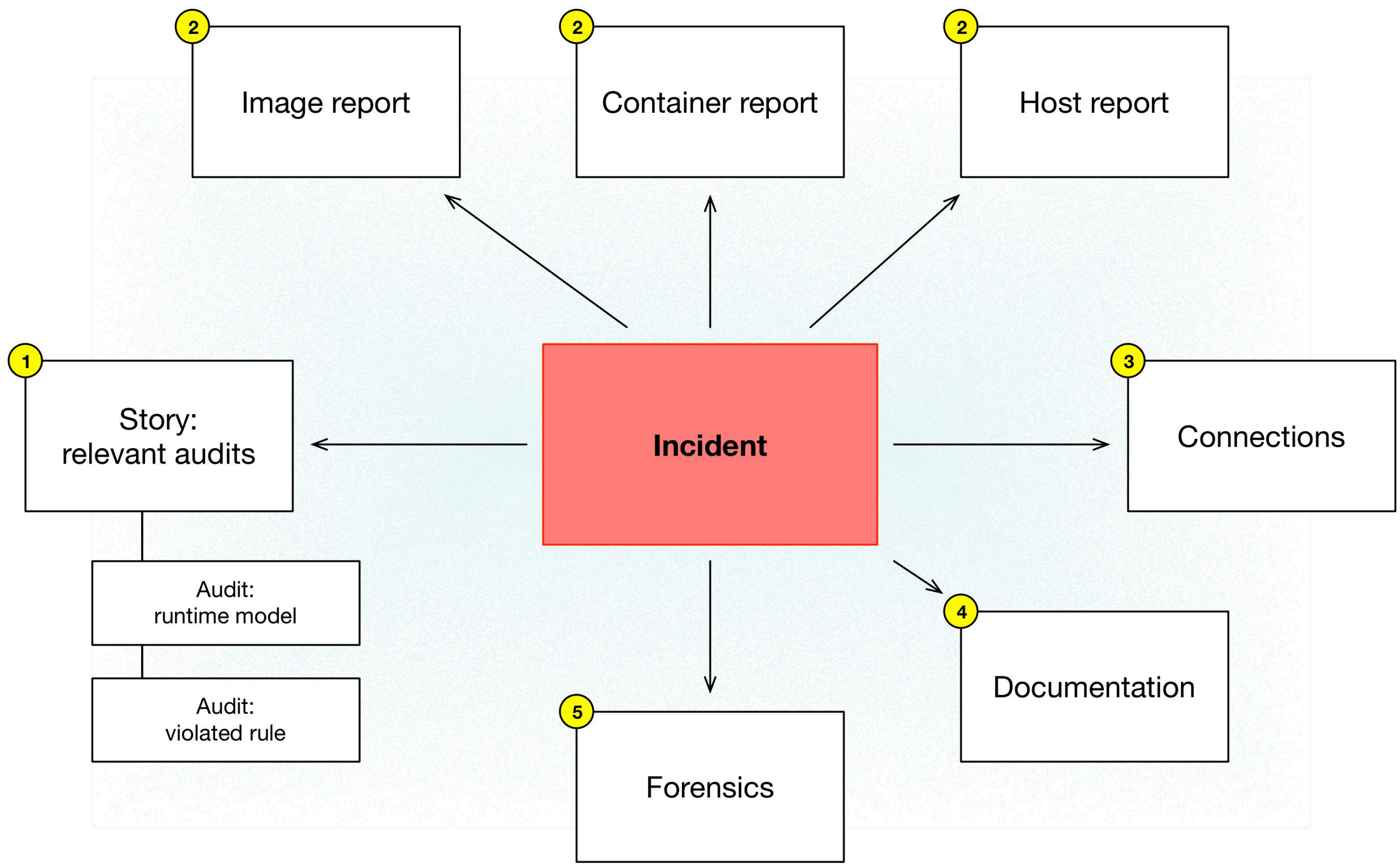
To view incidents, go to **Monitor > Runtime > Incident Explorer**. Click on an incident to examine the events in the kill chain. Clicking on individual events shows more information about what triggered the audit. After you have examined the incident, and have taken any necessary action, you can declutter your workspace by archiving the incident.

Only one incident from the same type (port scanning, altered binary, etc.) will be initiated for the same resource (container, host, etc.) every 24 hours. Further incidents from this type for the same resource will be automatically suppressed for 24 hours.


rvations
Image analysis sandbox
nt to suspicious activity and a potentially unfolding attack.
od... Show more
19 total entries
Hostname
Cluster
mor-console-2.c.compute-pm.internal
mor-console-2.c.compute-pm.internal
fargate-task-definition:4bd75f0875de46109ebeba91af8d...
fargate-task-definition:988f0d087e104a7dbbdea757c71e...
fargate-task-definition:846a0062337f442dbd5d3d54d5c...
First
Prev
1
2
3
4
Next
Last
Pg 1 of 4
that an allowed process has been used in ways that are inconsistent with its expected
d be a sign that a process has been used to compromise a container
View live forensic
ID
Host name
Container nam
Image name
CSV
Radar view of incident
/bin/cp launched from /bin/dash but is not found in the runtime model. Full command: cp sus_file.o newfile.o
root
ctive
True
wildfire
CK technique
Native Binary Execution
nse
Alert

All the raw audit events that comprise the incident can be found in the audit data tab. To see the individual events and export the data to a CSV file, go to **Monitor > Events > {Container audits | Host audits | App-Embedded audits}**.

Incident Explorer is organized to let you quickly access the data you need to investigate an incident. The following diagram shows the contextual data presented with each incident:



- **(1) Story** — Sequence of audits that triggered the incident.
- **(2) Image, container, and host reports** — Scan reports for each resource type. Scan reports list vulnerabilities, compliance issues, and so on.
- **(3) Connections** — Incident-specific radar that shows all connections to/from the container involved in the incident. Its purpose is to help you assess risk by showing you a connection graph for the compromised asset.
- **(4) Documentation** — Detailed steps for investigating and mitigating every incident type.
- **(5) Forensics** — Supplemental data collected and stored by Defender to paint a better picture of the events that led to an incident.

Forensics

Prisma Cloud Forensics is a lightweight distributed data recorder that runs alongside all the containers in your environment. Prisma Cloud continuously collects detailed runtime information to help incident response teams understand precisely what happened before, during, and after a breach.

Forensic data consists of additional supplemental runtime events that complement the data (audits) already captured by Prisma Cloud's runtime sensors. It provides additional context when trying to root cause an incident. Each Defender collects and stores forensic data in a fixed-sized first-in-first-out log file on the host where it runs. Forensic data is only downloaded to Console when it's needed for an incident investigation. This architecture enables Defender to store large amounts of data without any impact on network bandwidth or server processing (on the host where Console runs).

Forensics data is retrieved:

- After Prisma Cloud detects an incident. A minute after an incident occurs, Prisma Cloud collects forensic data from the relevant Defenders, and archives the data in Console. By default, Console stores up to 100 incident snapshots, which are managed on a FIFO basis.
- On-demand. Forensics data can be retrieved for review at any time from the Console UI.

Forensics event types

Containers:

- Process spawned — Process was run in the container. Fields: timestamp, container ID, PID, PPID, path, command, arguments.
- Container started — Container was started. Fields: timestamp, container ID.
- Binary created — Executable file or binary blob was created (file system event). Fields: timestamp, container ID, user, PID, path.
- Listening port — Container is listening on a network port. Fields: timestamp, container ID, PID, path to executable that's listening, listening start time, port.
- Connection established — Connection was established (incoming or outgoing) between the container and another entity. Fields: timestamp, container ID, source, destination, destination port.
- DNS query — DNS query was sent by the container. Fields: timestamp, container ID, domain name, domain type. Collecting DNS query events for container forensics depends on enabling DNS monitoring in the container runtime policy.
- Runtime profile — Runtime action was allowed for the container image while it was in learning mode. Fields: timestamp, container ID, user, PID, PPID, path, command.
- Runtime audit — Event occurred in a container that violates your runtime policy (model + runtime rules). Fields: timestamp, container ID, user, audit message, attack type, effect (alert or block).
- Incident — Incidents detected in a container that violates your runtime policy. Fields: timestamp, container ID, audit message, Category.

App-Embedded:

- Process spawned — Process was run in the container. Fields: timestamp, PID, PPID, path, command, arguments.
- Container started — Container was started. Fields: timestamp.
- Binary created — Executable file or binary blob was created (file system event). Fields: timestamp, container ID, user, PID, path.

- Listening port—Container is listening on a network port. Fields: timestamp, PID, path to executable that's listening, listening start time, port.
- Connection established—Connection was established (incoming or outgoing) between the container and another entity. Fields: timestamp, source, destination, destination port.
- DNS query—DNS query was sent by the container. Fields: timestamp, container ID, domain name.
- Runtime audit—Event occurred in a container that violates your runtime policy (runtime rules). Fields: timestamp, user, audit message, attack type, effect (alert or prevent).
- Incident—Incident detected in a container that violates your runtime policy. Fields: timestamp, audit message, category.

Hosts:

- Process spawned—Process was run on the host. Fields: timestamp, hostname, path, PID, parent PID, parent path, user, command, interactive (true or false), program name.
- Binary created—Executable file or binary blob was created (file system event). Fields: timestamp, app, user, PID, path.
- DNS query—DNS query was sent from the host. Fields: timestamp, domain name, domain type. Collecting DNS query events for host forensics depends on enabling DNS monitoring in the host runtime policy.
- Runtime profile—Runtime action was allowed for an app while it was in learning mode. Fields: timestamp, app, user, capabilities, command.
- Runtime audit—Event occurred in a container that violates your runtime policy (model + runtime rules). Fields: timestamp, app, user, audit message, attack type, effect (alert or block).

Configuring data collection

To configure Forensics, go to **Manage > System > Forensics**. By default, forensic data collection is enabled.

With forensic data collection enabled, Defender requires an additional 1 MB of memory and 110 MB of storage space (100 MB for containers forensics and 10 MB for host forensics). If enabled, you can specify the desired amout of storage space allocated to each Defender, see the suggested values below:

- Container forensics, 100 MB per Defender
- Host forensics, 10 MB per Defender
- App-Embedded forensics, 10 MB per Defender. Note that each AWS Fargate task has one Defender that monitors all the task's containers.

You can specify a minimum of 10 MB and a maximum of 1000 MB for each category.

Several settings dictate what type of data is collected and for how long:

- **Max number of incident snapshots Console can store**—After an incident occurs, Prisma Cloud collects and saves the relevant forensic data set in Console. To control the amount of data Console stores, Prisma Cloud caps the number of data sets and mananges them on a FIFO basis.
- **Collect network snapshots**—When this option is enabled, the forensic package that you can download from Console includes a netstat-style snapshot of the current connections.

- **Collect network firewall snapshots** — When this option is enabled, the forensic data includes the *Connection established* event type, which shows incoming and outgoing connection details, including source IP, destination IP, and destination port.

Viewing forensic data

Forensic data is associated with incidents.

STEP 1 | Open Console, and go to **Monitor > Runtime > Incident Explorer**.

STEP 2 | In the **Active** tab, select an incident.

STEP 3 | Click on **View forensic data**.

If you configure Prisma Cloud to send out alerts on channels, such as email or Slack, when incidents occur, the alert messages will contain a direct link for downloading the forensics data.

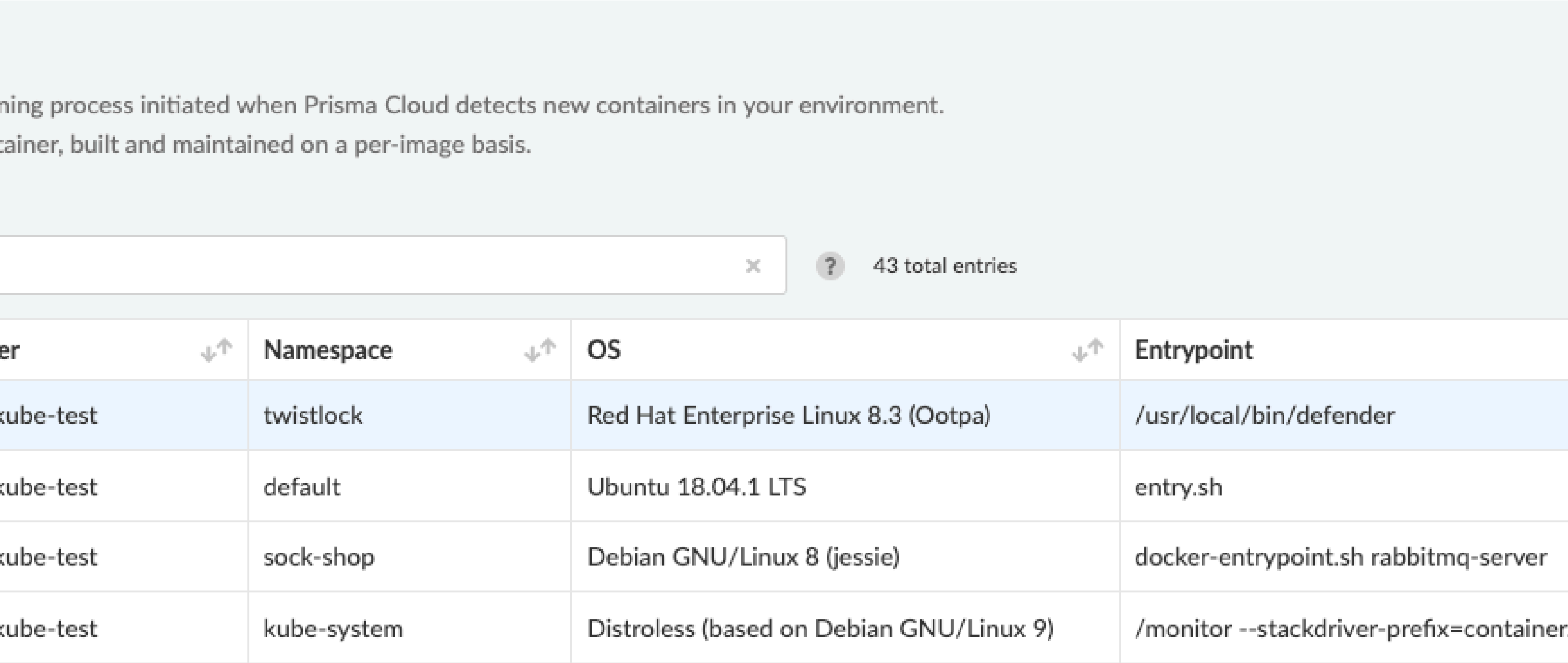
Viewing container forensic data

While Incident Explorer presents forensic data relevant to specific incidents, you can also view all available forensic data at anytime outside the scope of an incident.

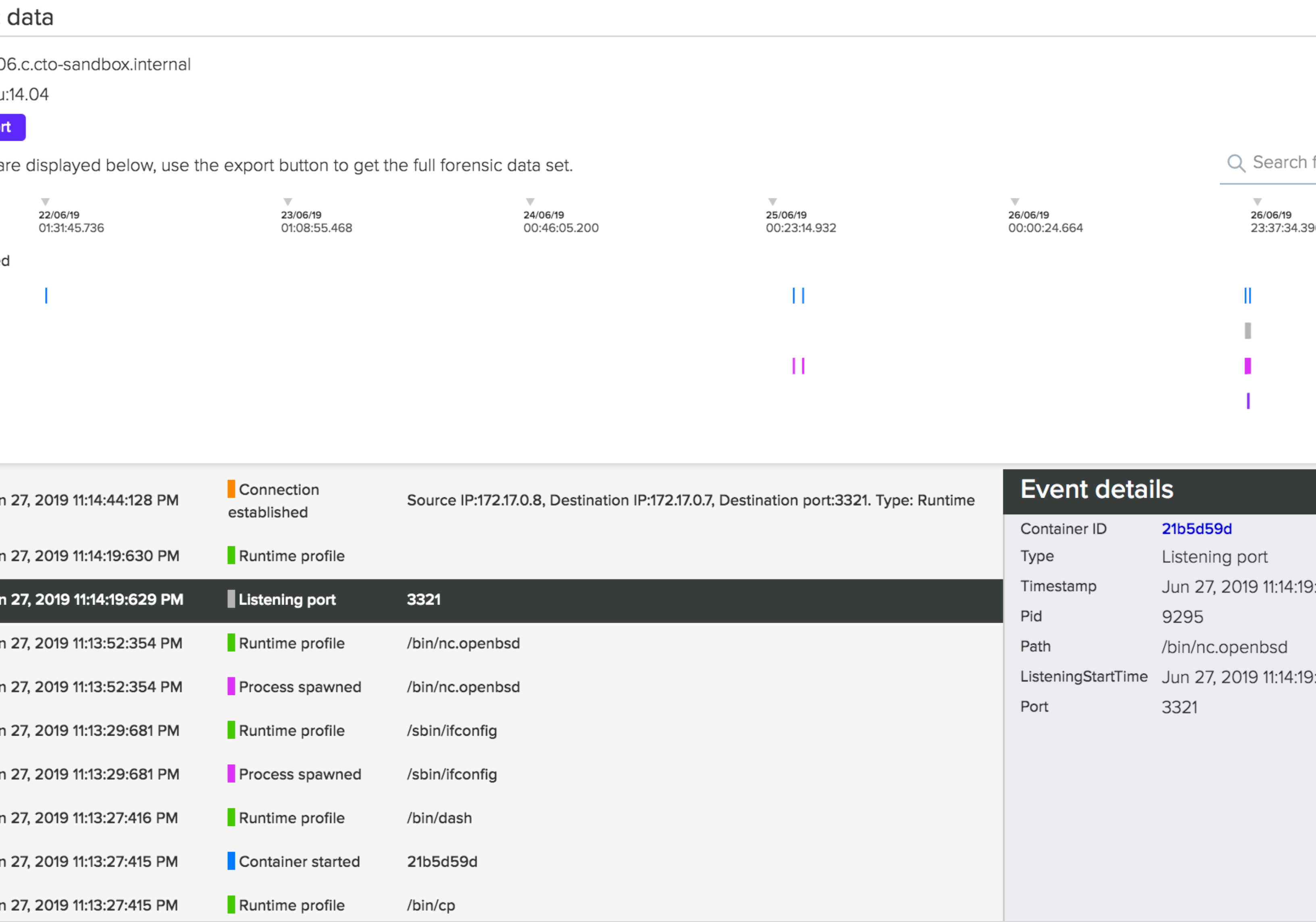
For containers, forensic data is collected on a per-model basis. To retrieve and review the forensic data for a container:

STEP 1 | Open Console, and go to **Monitor > Runtime > Container Models**.

STEP 2 | In the table, click the microscope icon for the container of interest.



Events are displayed in a coordinated timeline-table interface.



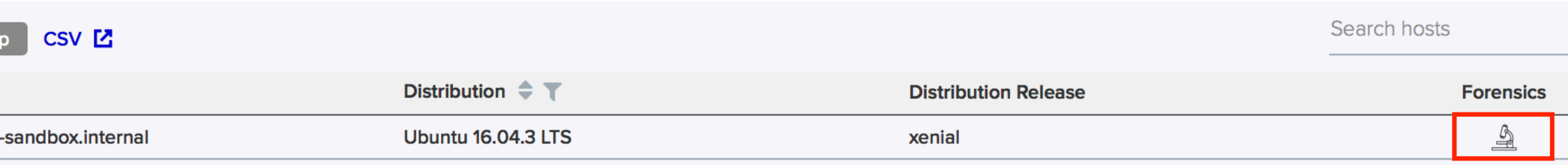
Viewing host forensic data

To retrieve and view the forensic data for a host:

STEP 1 | Open Console, and go to **Monitor > Runtime > Host Models**.

STEP 2 | Click the **Host** toggle button.

STEP 3 | In the table, click the microscope button for the host of interest.

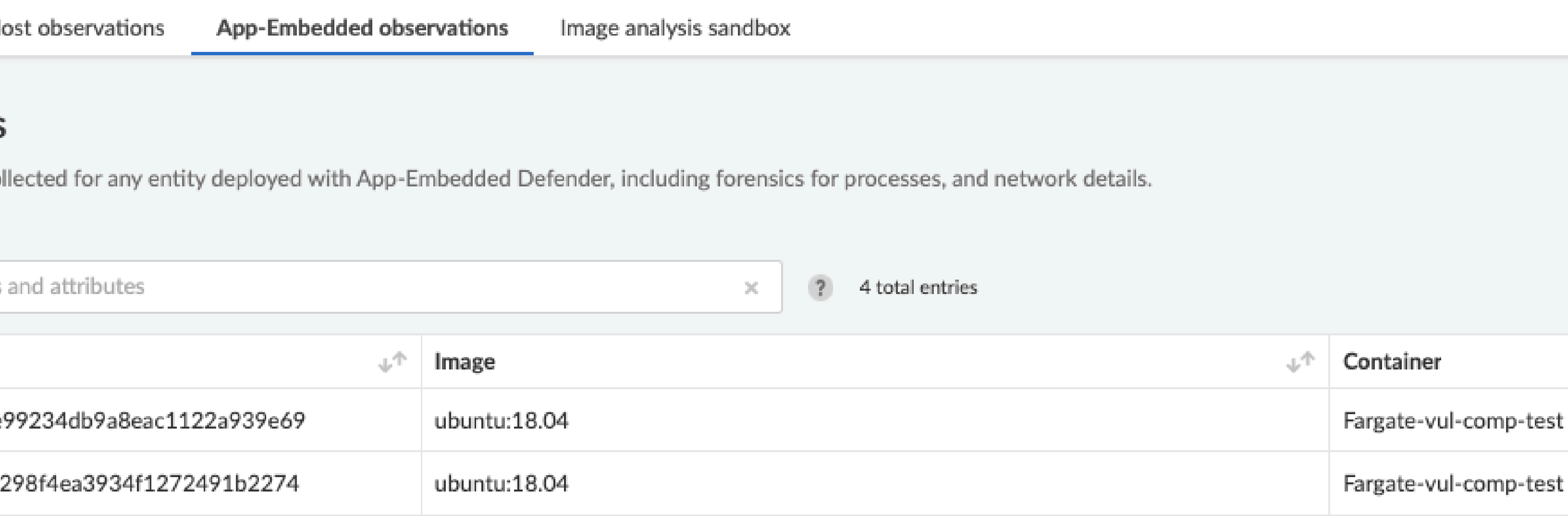


Viewing App-Embedded forensic data

To retrieve and view the forensic data for App-Embedded:

STEP 1 | Open Console, and go to **Monitor > Runtime > App-Embedded observations**.

STEP 2 | In the table, click the microscope button for the App-Embedded instance of interest.



Since the table allows querying live forensics, the App-Embedded observations table will remove inactive App-Embedded instances once an hour.

Incident types

Edit on GitHub

This section describes the incident types surfaced in Incident Explorer.

- Altered binary
- Backdoor admin accounts
- Backdoor SSH access
- Brute force
- Crypto miners
- Execution flow hijack attempt
- Kubernetes attack
- Lateral movement
- Malware
- Port scanning
- Reverse shell
- Suspicious binary

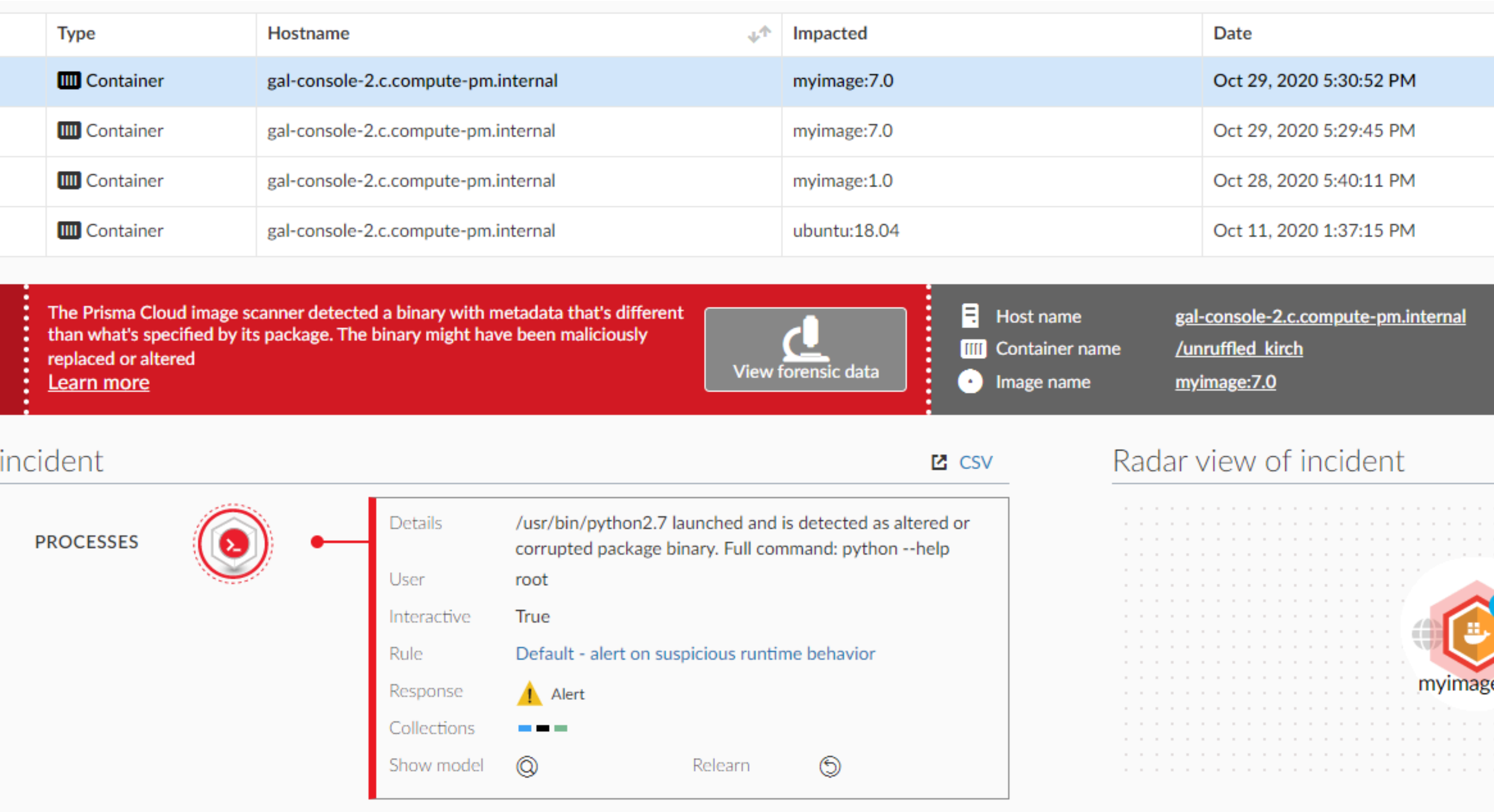
Altered binary

Edit on GitHub

An altered binary incident indicates that a binary that during image scanning was found with different metadata than what is specified by its package was executed. This binary might have been maliciously replaced or altered.

Investigation

The following incident shows that the process *python2.7* was launched, but it seems to be altered or corrupted.



Your investigation should focus on locating the source of the affected image.

If the image was pulled from a remote repository, you should confirm the image hash is as expected given the repository image metadata. You should make sure the image repository and the author are valid.

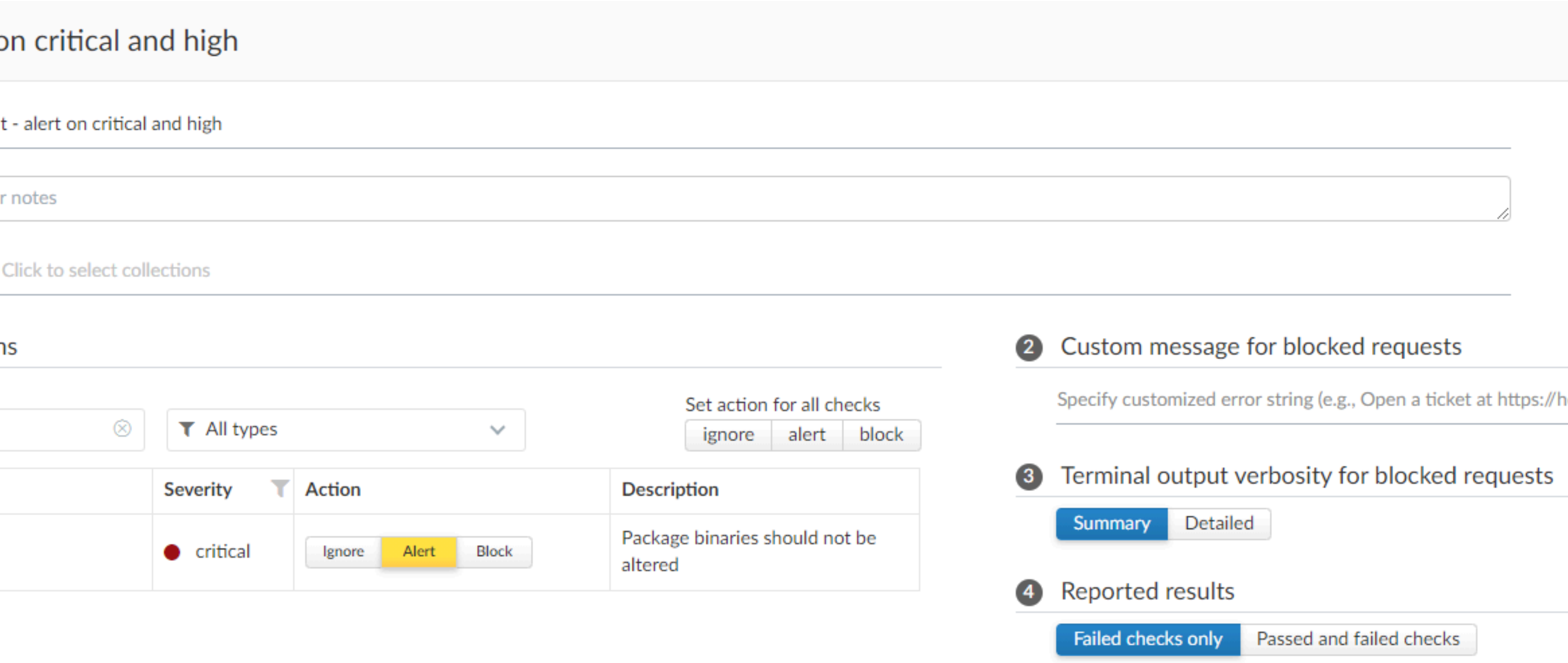
If the image was built locally, you must examine the build process. Inspect your supply chain to understand if any binary from signed sources (such as a package manager) is changed or modified throughout the build.

Mitigation

A full mitigation strategy for this incident begins by resolving the issues that allowed to pull or build an image including an altered binary.

Ensure that compliance benchmarks are appropriately applied to the affected images and containers. Use Trusted Images, to avoid image pulls from untrusted sources.

For additional protection, Enable the *block* action in the applicable compliance check to take action when altered binaries are found in an image during a scan.



Backdoor admin accounts

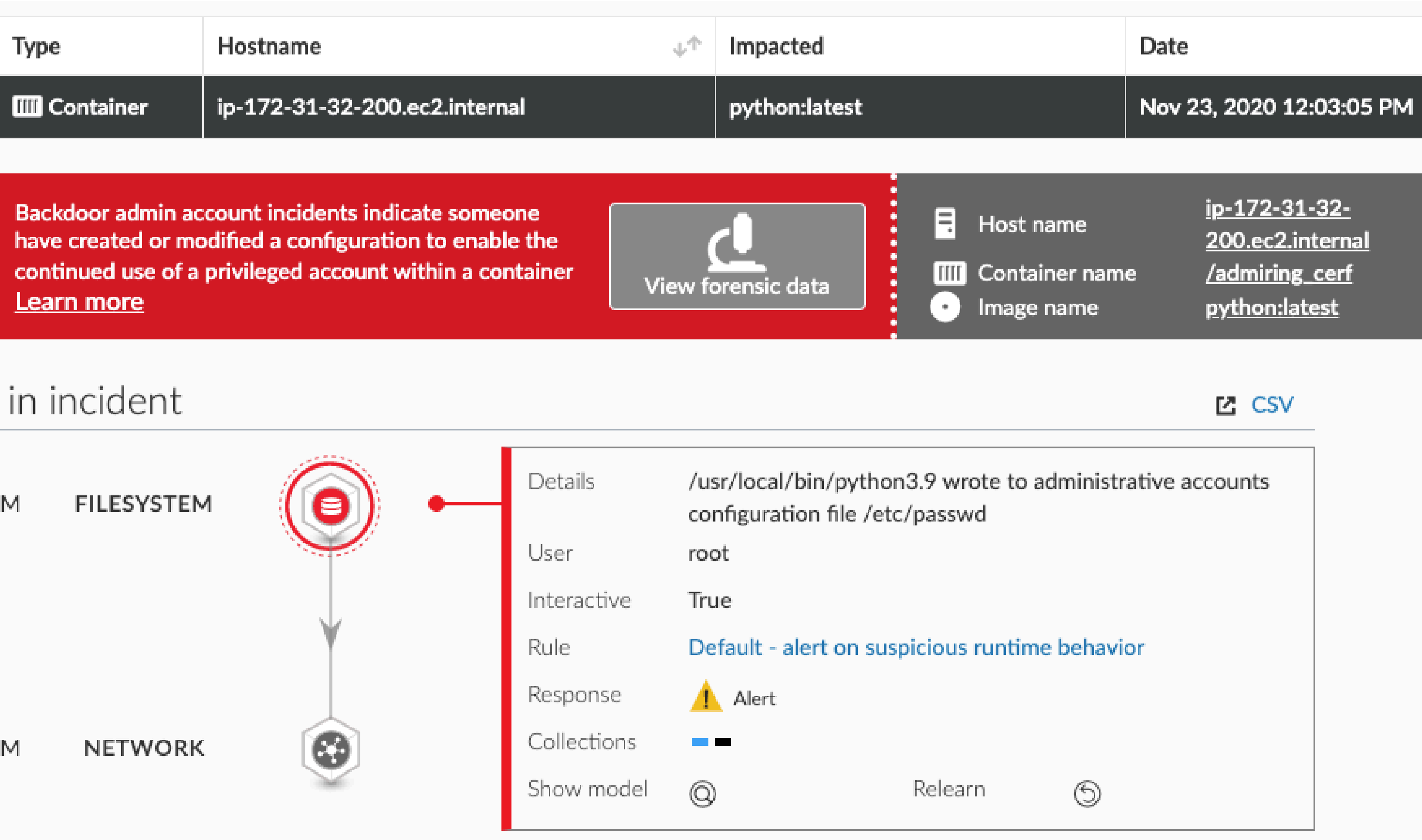
Edit on GitHub

Backdoors are a method for bypassing normal authentication systems, and are used to secure remote access to a system.

Backdoor admin account incidents surface event patterns that indicate an actor might have created or modified a configuration to enable the continued use of a privileged account.

Investigation

In the following incident, you can see that a python script was used to modify */etc/passwd*, potentially enabling an attacker to add or change a user account. In addition, there was other suspicious network activity that was made by the same python process.



The first step in an investigation is to validate that the changes represent a bona fide security incident. In this case, the events that led to the incident seem to indicate a valid security incident, but you should examine the changes to */etc/passwd* to see if they represent the potential for an attacker to maintain persistence.

Having determined that this is a bona fide incident, then the next steps focus on determining how an attacker was able to modify the system configuration. This would, generally, be a post-compromise approach to maintain access to the compromised systems. Check Incident Explorer for additional incidents. Review additional runtime audits for the source to see if there are other clues.

Review access to the container and ensure that the affected account(s) weren't subsequently used for further access to systems and data.

Mitigation

A full mitigation strategy for this incident begins with resolving the issues that allowed the attacker to modify the system configuration.

Ensure that compliance benchmarks are appropriately applied to the affected resources. For example, if the critical file systems in the container are mounted read-only, it will be more difficult for an attacker to change a configuration to their advantage.

Backdoor SSH access

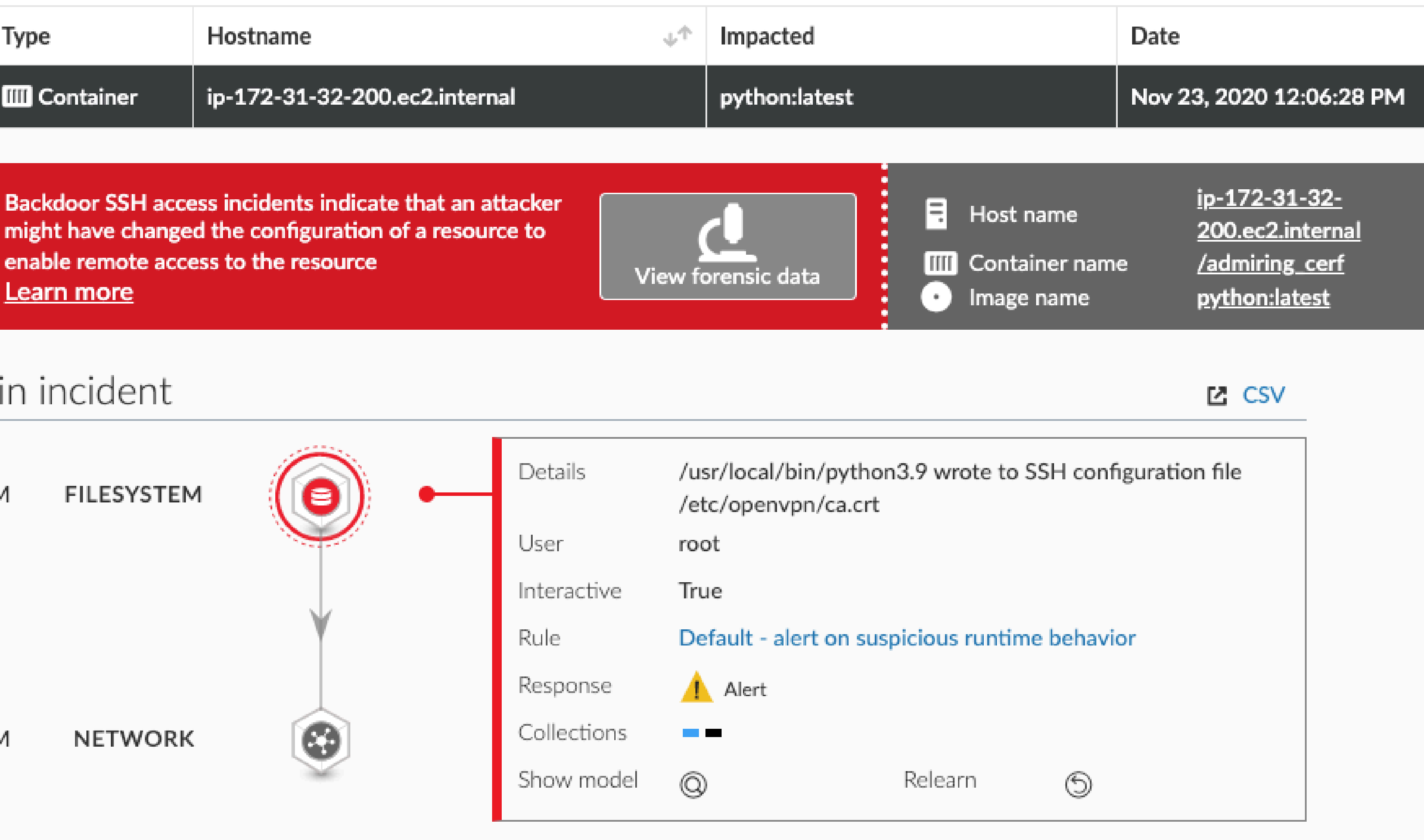
Edit on GitHub

Backdoors give attackers a way to bypass normal authentication systems, and are used to secure remote access to a system.

Backdoor SSH access incidents indicate that an attacker might have changed the configuration of a resource to enable remote access to the resource.

Investigation

In the following incident, you can see two audits. The first audit is a file system event that shows a new certificate was created in */etc/openvpn*. An attacker could use this certificate for follow-on access to the container.



The first step in an investigation is to validate that the changes represent a bona fide security incident. In this example, it's unlikely that a change in the ca cert file is a valid one, but it might not always be so clear.

After validating that this is a security incident, the next step is determining how an attacker was able to modify the system configuration. This would, generally, be a post-compromise approach to maintain access to the compromised systems. Check Incident Explorer for other potentially related incidents. Review additional runtime audits for the source to see if there are other clues.

Review access to the container and ensure that accesses weren't subsequently used for further access to systems and data.

Mitigation

A full mitigation strategy for this incident begins with resolving the issues that allowed the attacker to modify the system configuration.

Ensure that compliance benchmarks are appropriately applied to the affected resources. For example, if the critical file systems in the container are mounted read-only, it will be more difficult for an attacker to change a configuration to their advantage.

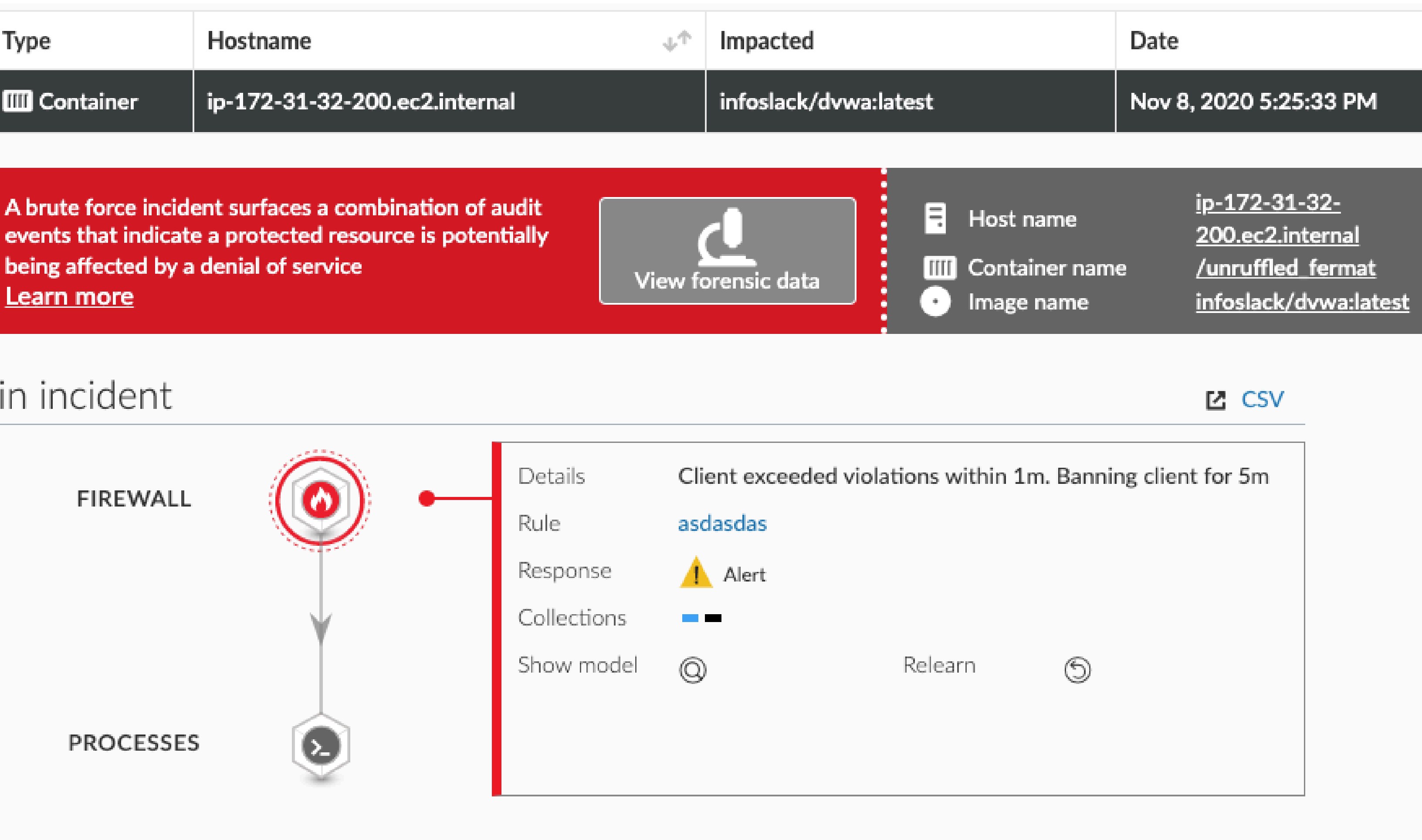
Brute force

Edit on GitHub

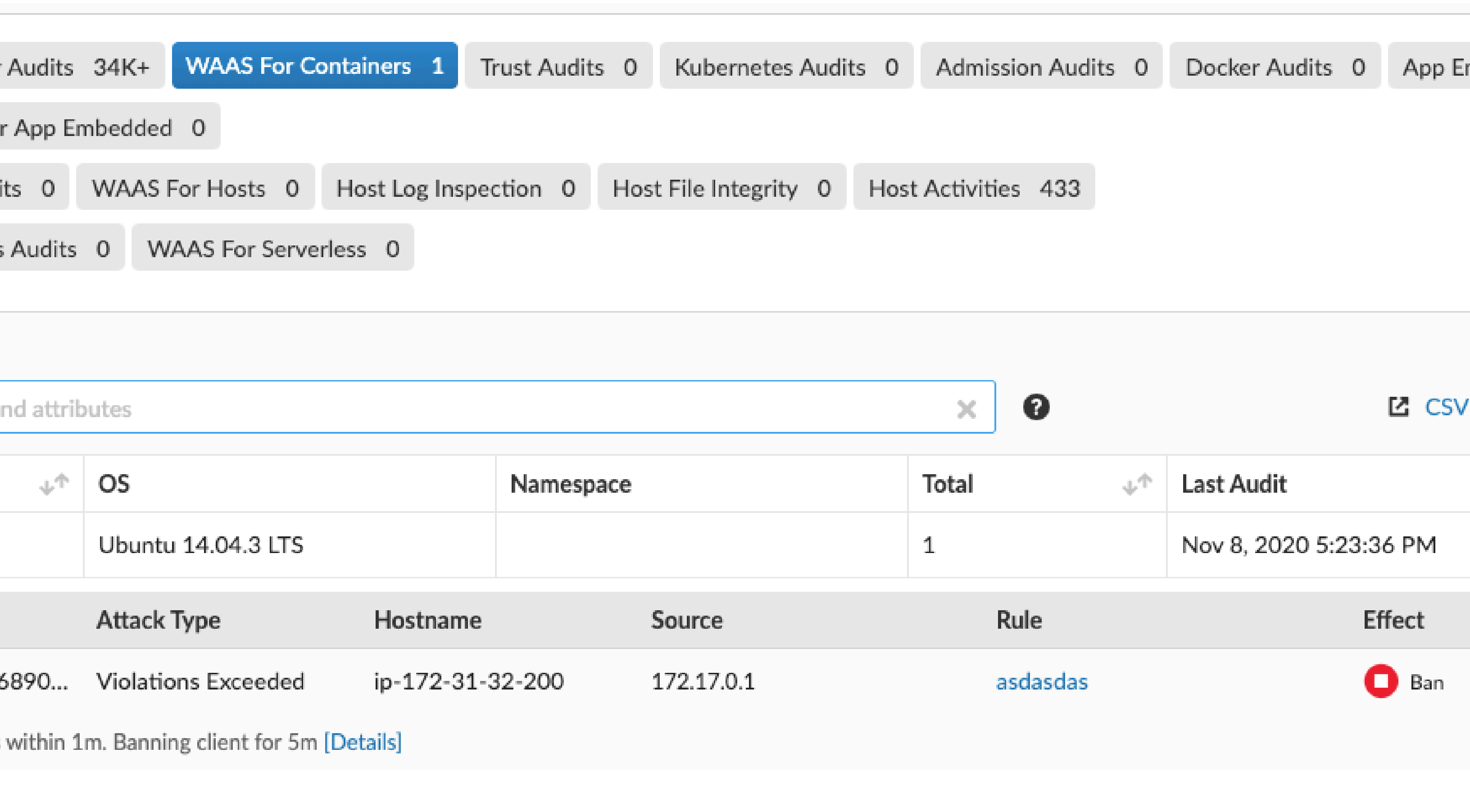
A Brute Force incident surfaces a combination of audit events that indicate a protected resource is potentially being affected by an attempted DoS.

Investigation

In the following incident, you can see that a container received a flood of attempted actions to the extent that the Web Application and API Security (WAAS) blocked the source.



Review the WAAS audit logs to determine any further impact:



Additionally, review the logs of potentially affected applications to determine if there was any further impact.

Mitigation

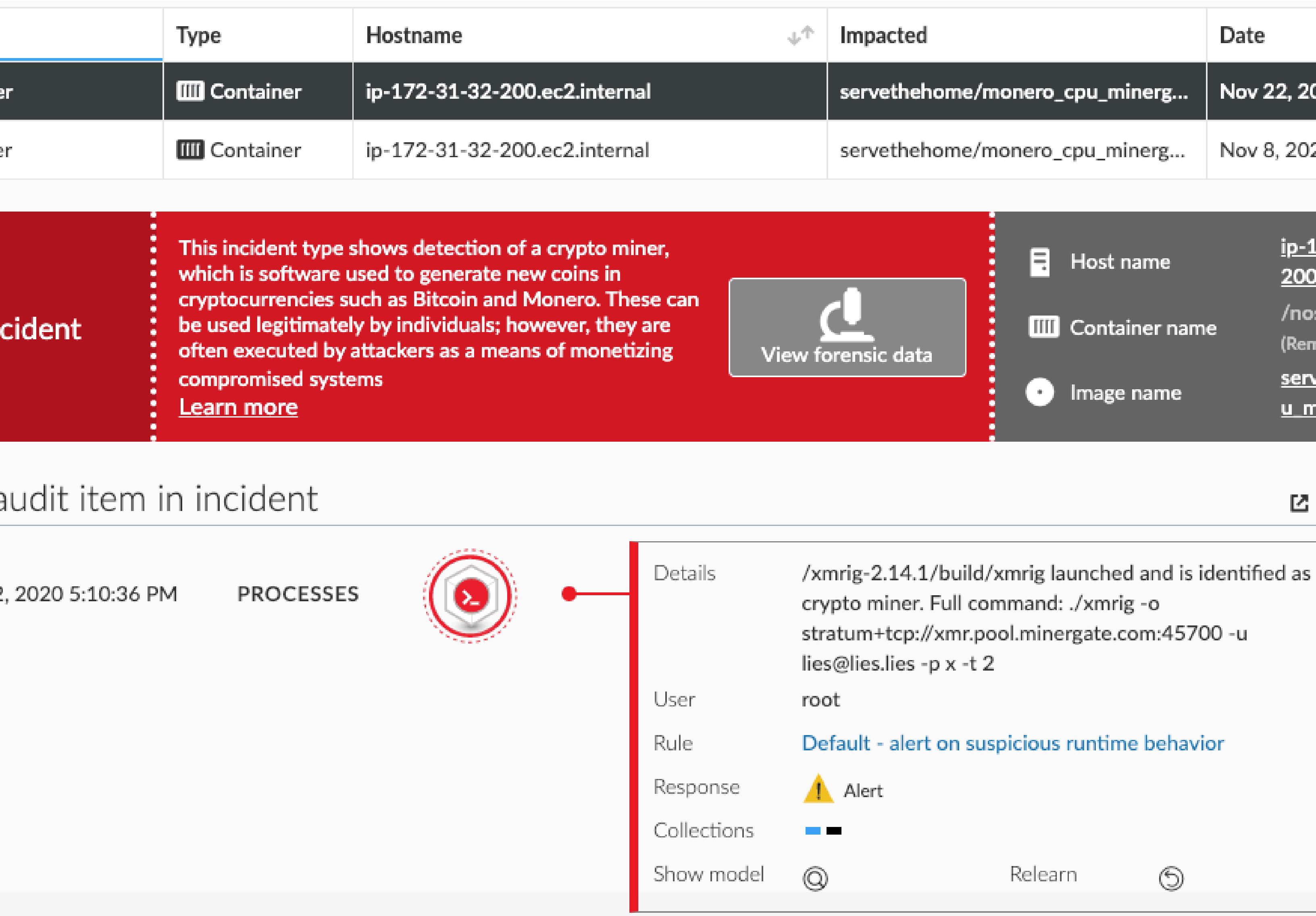
Ensure that WAAS rules provide protection for exposed services.

Cryptominers

Edit on GitHub

Cryptominers are software used to generate new coins in cryptocurrencies such as Bitcoin and Monero. These can be used legitimately by individuals; however, in containerized environments, they are often executed by attackers as a means of monetizing compromised hosts.

Unless you are intentionally running a cryptominer, this alert most likely indicates a security incident in which an attacker was able to introduce a cryptominer into your infrastructure and execute it.



Our research indicates that the potential attack vectors include:

- A Kubernetes or Docker endpoint exposed to the Internet that allows unauthenticated access, or that is protected with weak credentials.
- A registry exposed to the Internet that allows unauthenticated users, or users with weak or common passwords, to make changes to stored images.
- Vulnerable code in a containerized service that has been exploited, followed by lateral movement and remote code execution.

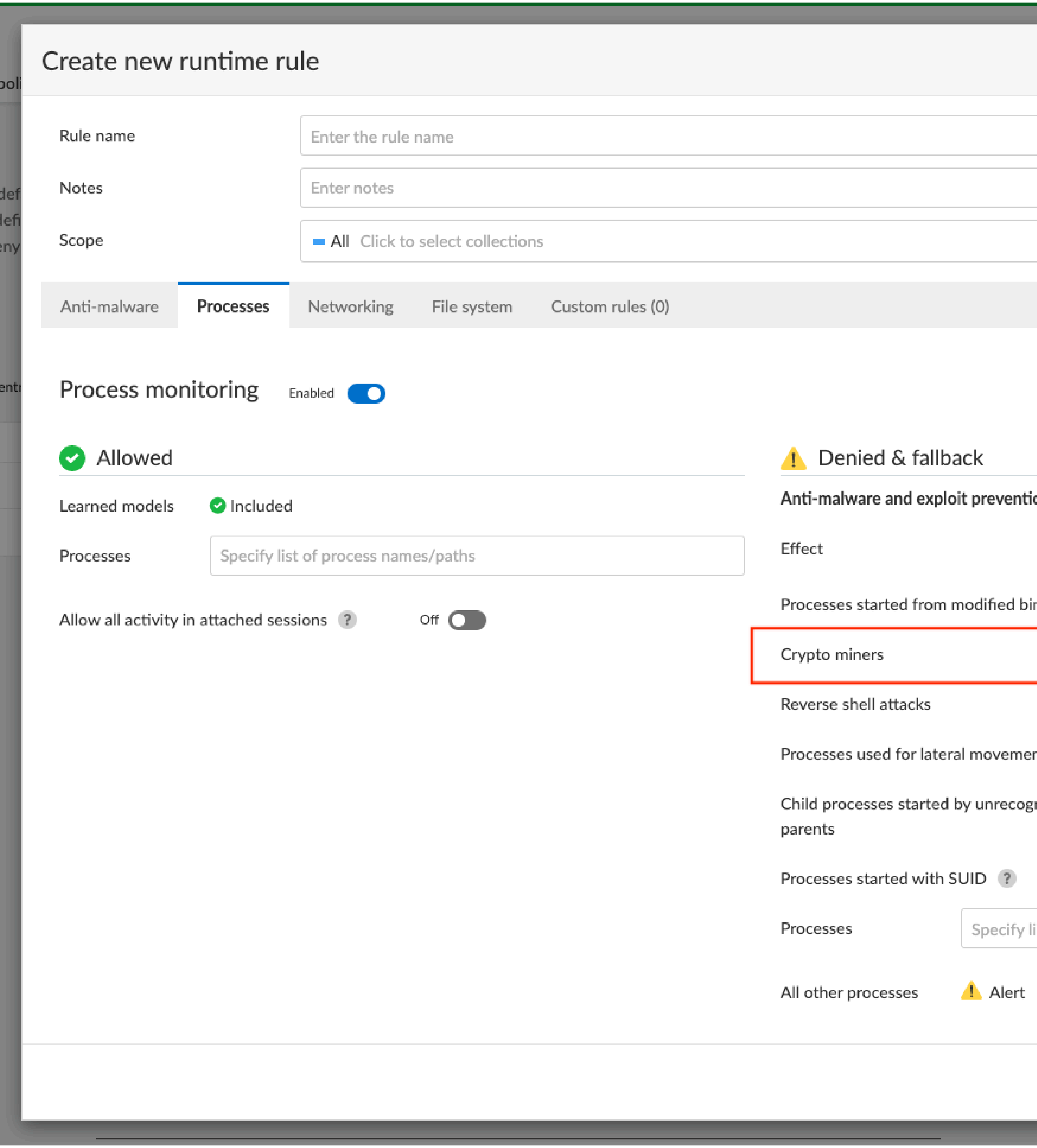
Enable Runtime Monitoring for Detection of Cryptominers

1. Ensure **Defend > Runtime > Container policy > Enable automatic runtime learning** is set to **on.**

   Known cryptominers are part of the IS feed on Prisma Cloud and they detect high CPU consumption and network communications.

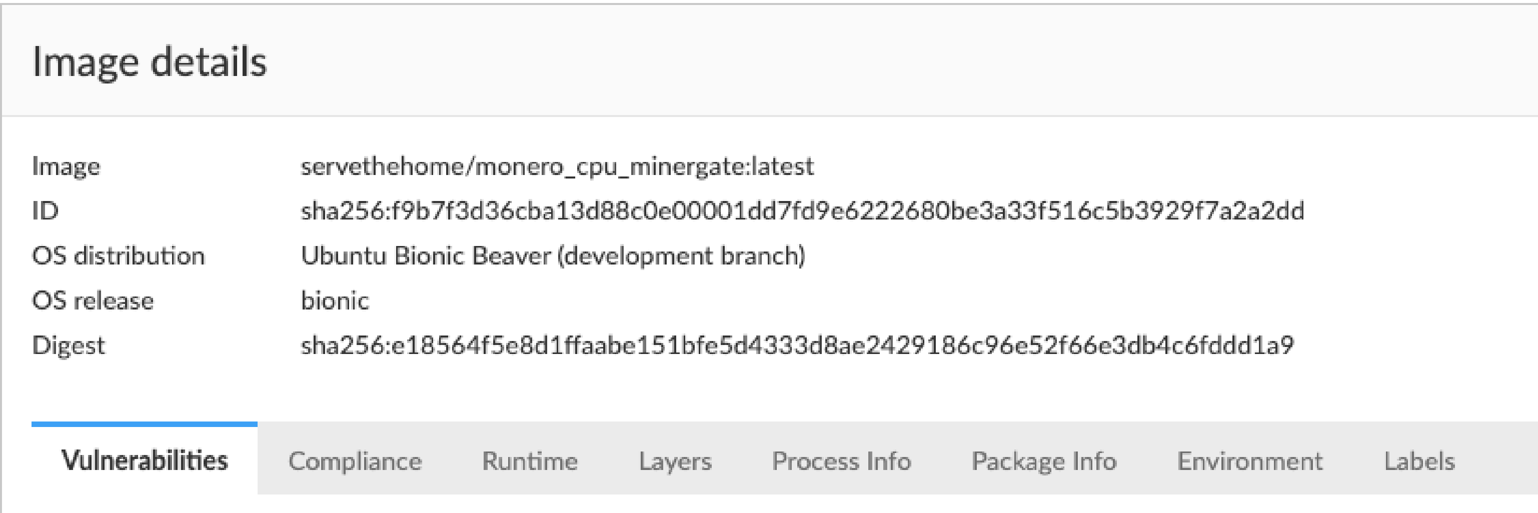
2. Add a new runtime rule.

   When you add a new rule for container policy or serverless policy, it is enabled to alert you for cryptominer detection. You can modify this to block the activity.


Create new runtime rule
Rule name
Enter the rule name
Notes
Enter notes
Scope
All Click to select collections
Anti-malware
Processes
Networking
File system
Custom rules (0)
Process monitoring
Enabled
Allowed
Learned models
Included
Processes
Specify list of process names/paths
Allow all activity in attached sessions
Off
Denied & fallback
Effect
Crypto miners
Reverse shell attacks
parents
Processes started with SUID
Processes
All other processes
Alert

Investigation

The first step in determining how the crypto miner was introduced is to determine if this is an existing image which has had unwanted processes introduced into it or if this is an entirely unwanted image. You can inspect the image itself in the Prisma Cloud Console.



We can see that this image comes from Docker Hub and that it is not an image that was developed internally. In this case, we would want to dig deeper into how the image was pulled and the container executed. You may have many sources of this information including the Prisma Cloud Docker access logs (Monitor/Access/Docker), which have been exported to CSV and filtered here:

370	alice	docker run -e username=lies@li	docker			Local audit	prod-54.docker_cluster.int		TRUE	2018-02-27 18:19:41.939 +0000 UTC
371	alice	docker_ping	docker			Allow all	neilcar.c.cto-	Local	TRUE	2018-02-27 18:19:42.036 +0000 UTC
372	alice	container_create	docker	e2216c423707494301ba7	servethehome/monero_cpu_minergate	Allow all	neilcar.c.cto-	Local	TRUE	2018-02-27 18:19:42.117 +0000 UTC
373	alice	container_attach	docker	/quirky_payne	servethehome/monero_cpu_minergate@sha256:da9cf52e008abe7da86839b	Allow all	neilcar.c.cto-	Local	TRUE	2018-02-27 18:19:42.131 +0000 UTC
374	alice	container_wait	docker	/quirky_payne	servethehome/monero_cpu_minergate@sha256:da9cf52e008abe7da86839b	Allow all	neilcar.c.cto-	Local	TRUE	2018-02-27 18:19:42.141 +0000 UTC
375	alice	container_start	docker	/quirky_payne	servethehome/monero_cpu_minergate@sha256:da9cf52e008abe7da86839b	Allow all	neilcar.c.cto-	Local	TRUE	2018-02-27 18:19:42.63 +0000 UTC

This shows that a user account, 'alice', was used to run 'docker exec' and start the container, and that the command was run locally. From here, we would want to review authentication logs on the system to determine how 'alice' was able to logon and to review other data to determine what else 'alice' was able to accomplish.

If the image was an existing one that the enterprise legitimately uses, the next steps in the investigation would be to determine how the image was modified to include the crypto miner. Start by reviewing the image in any registry where it is stored and looking at a history of changes made to the image. It may be necessary to walk through the entire CI/CD pipeline to determine if changes were made prior to being pushed to the registry.

Mitigation

As soon as the investigation is complete, remove all instances of the running container (docker stop quirky_payne | docker rm quirky_payne in this case). If the container(s) were started with an orchestrator like Kubernetes, it may be necessary to remove any configuration that would cause them to restart.

If the image was pushed to a registry, take steps to remove affected versions from the registry.

Secure all access, starting with any point of entry that was found. Ensure that only needed endpoints are exposed to the Internet and that authentication is required at each endpoint that could, directly or indirectly, result in remote code execution. Ensure accounts have strong passwords and, where possible, two-factor authentication.

Investigate any successful attack vectors that were found in the investigation. This may not be the only successful attack to have used this approach; instead, it may just be the most visible one.

Execution flow hijack attempt

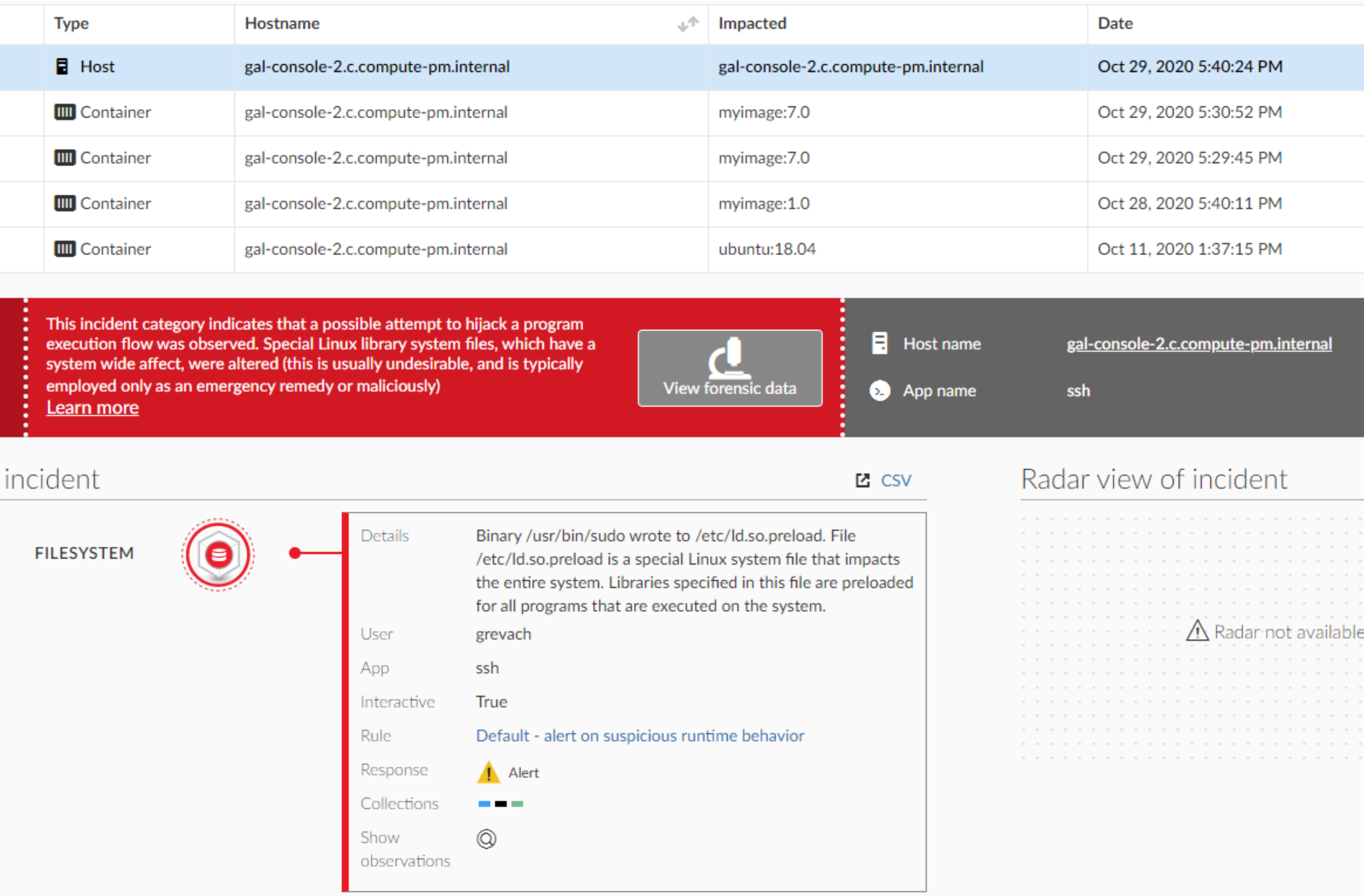
Edit on GitHub

An execution flow hijack attempt incident indicates that a possible attempt to hijack a program execution flow was observed. Special Linux library system files, which have a system-wide effect, were altered (this is usually undesirable, and is typically employed only as an emergency remedy or maliciously).

Investigation

The following incident shows that the binary *sudo* wrote to *ld.so.preload* file, which is a special Linux system file that impacts the entire system. By editing the Linux dynamic loader or files relied upon by the loader such as *ld.so.preload*, the attacker can inject malicious code to any binary execution.

For further information about these files, see the following link.



Your investigation should focus on:

- Determining the process that opened the Special Linux file.
- If the source of the alteration was an interactive process (such as shell), determine how an attacker gained access to that process.
- Review the forensics date for the host, other entries in the Incident Explorer, and audits from the source, looking for unusual process execution, hijacked processes, and explicit execution of commands.

Mitigation

A full mitigation strategy for this incident begins by resolving the issues that allowed the attacker to access and modify the system file.

In addition, track the change that was done to the configuration in the system file. For example, in case of detected modification to the *ld.so.preload* file, look for the shared library that was added to the file and determine the source of this malicious shared library.

Ensure that compliance benchmarks are appropriately applied to the affected resources. For example, if the critical file systems in the host are mounted read-only, it will be more difficult for an attacker to change system files and configurations to their advantage.

Kubernetes attacks

Edit on GitHub

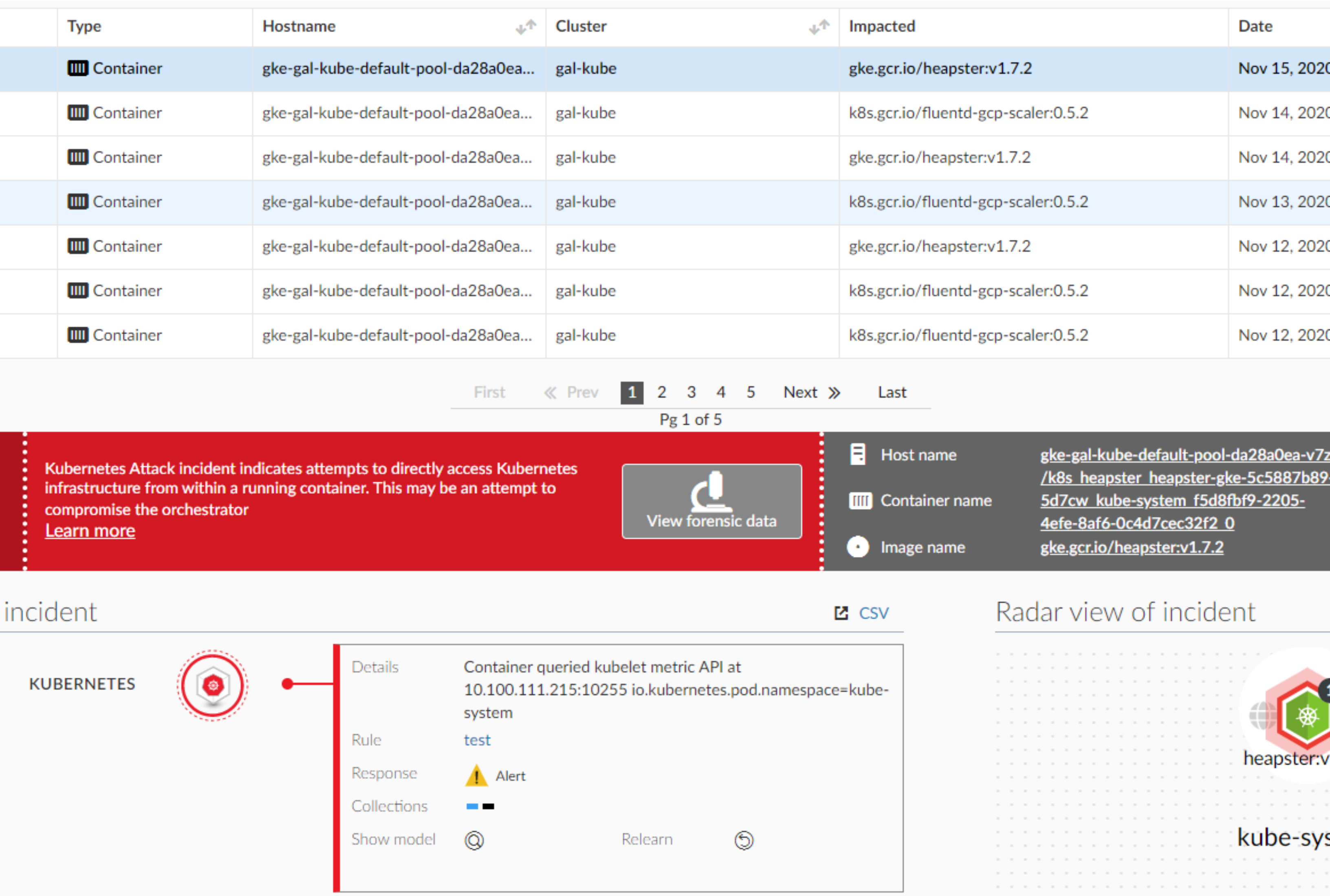
Exploiting weaknesses in the container orchestrator to manipulate cluster settings is known as a Kubernetes attack. This incident indicates attempts to directly access Kubernetes infrastructure from within a running container. This may be an attempt to compromise the orchestrator.

Actions that can trigger this incident include attempts to download and use Kubernetes administrative tools within a container, in addition to any attempts to access Kubernetes metadata.

To detect Kubernetes attacks, you must have a runtime rule with the **Detect Kubernetes attacks** option enabled.

Investigation

The following incident shows that a container queried kubelet metric API, which might be an attempt to compromise the orchestrator.



The first step in an investigation is to validate that the changes represent a bona fide security incident. Having determined that this is a bona fide incident, then the next steps focus on determining how an attacker would have gained access to the resources with access to the Kubernetes cluster. Also, it is important to restrict access to your cluster by following best practices regarding access control.

Review your Kubernetes cluster to ensure that no actions were taken to compromise your cluster. In addition, closely review the audit actions and the forensic data available through incident explorer to understand the scope of the incident.

Mitigation

A full mitigation strategy for this incident begins with resolving the issues that allowed the attacker to attempt to access the Kubernetes infrastructure.

For additional protection, customize your runtime rules to *prevent* or *block* actions that access the metadata services or the open local kubelet port. Compliance rules should include checks set to *alert* or *block* to ensure your containers and hosts are following the best practices for Kubernetes.

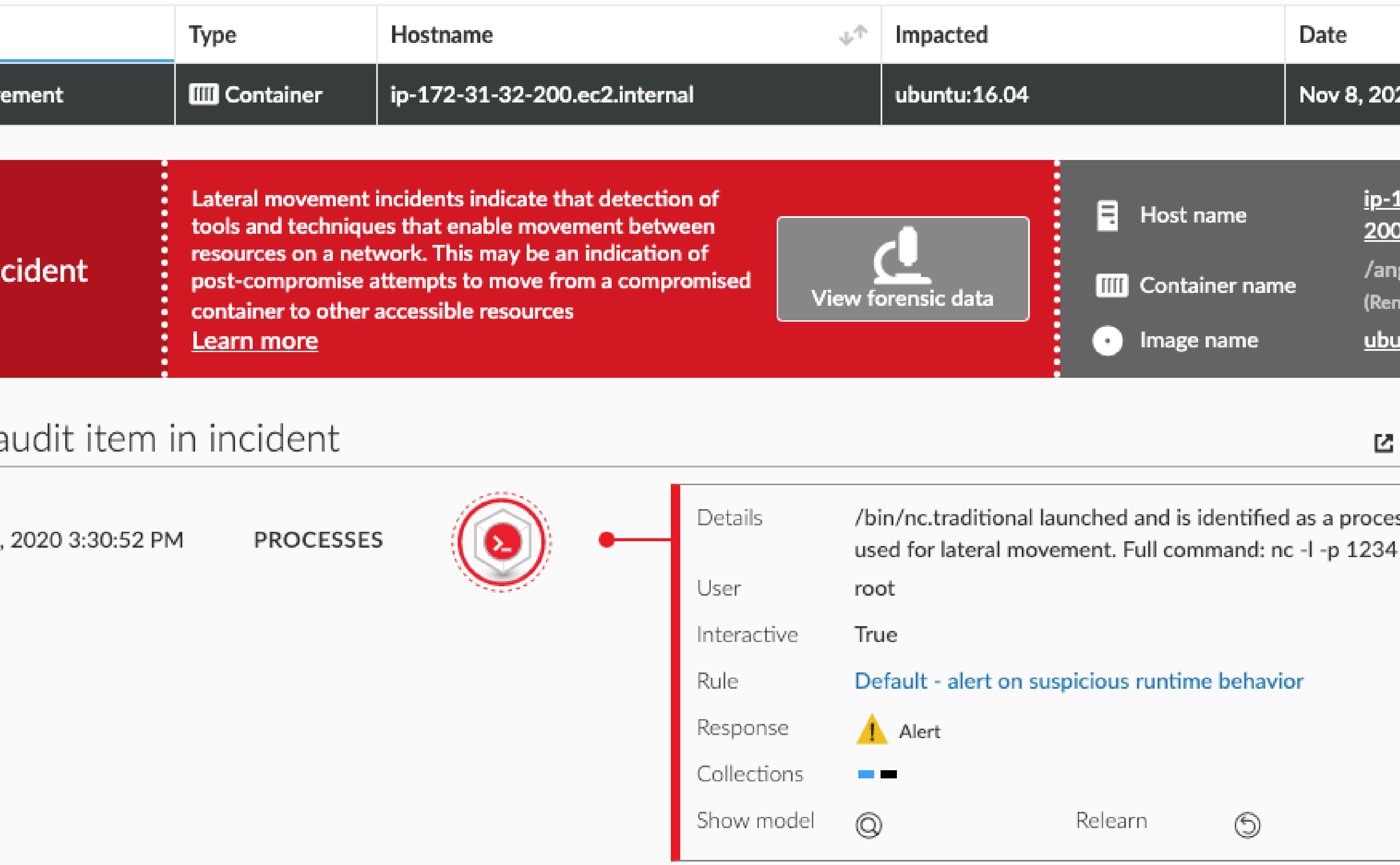
Lateral movement

Edit on GitHub

Lateral movement incidents indicate that an attacker is using tools and techniques that enable movement between resources on a network.

Investigation

The following incident shows that netcat was used to establish a listener on port 9000.



This behavior is a probable precursor to creating a reverse shell, allowing network-based remote control of another resource.

Your investigation should focus on:

- Determining how the process in the alert, such as *nc.openbsd*, was executed. Review additional entries in Incident Explorer and other audits from the source, looking for unusual process execution, and explicit execution of commands.
- Reviewing container runtime audits to determine if the target successfully connected.
- If the target did successfully connect, determine what the attacker was able to do and if they were able to move further through the network.

Mitigation

After determining the cause of the process execution, resolve the problem, whether it be an exposed vulnerability, a configuration issue, or something else.

For additional protection, enable the *prevent* or *block* actions in the applicable runtime rules to take action when anomalous processes, such as *netcat*, are executed.

Malware

Edit on GitHub

A malware binary incident indicates that a malware binary was written to the file system. A binary can be identified as malware using WildFire, Prisma Cloud advanced intelligence stream or based on a custom feed.

Investigation

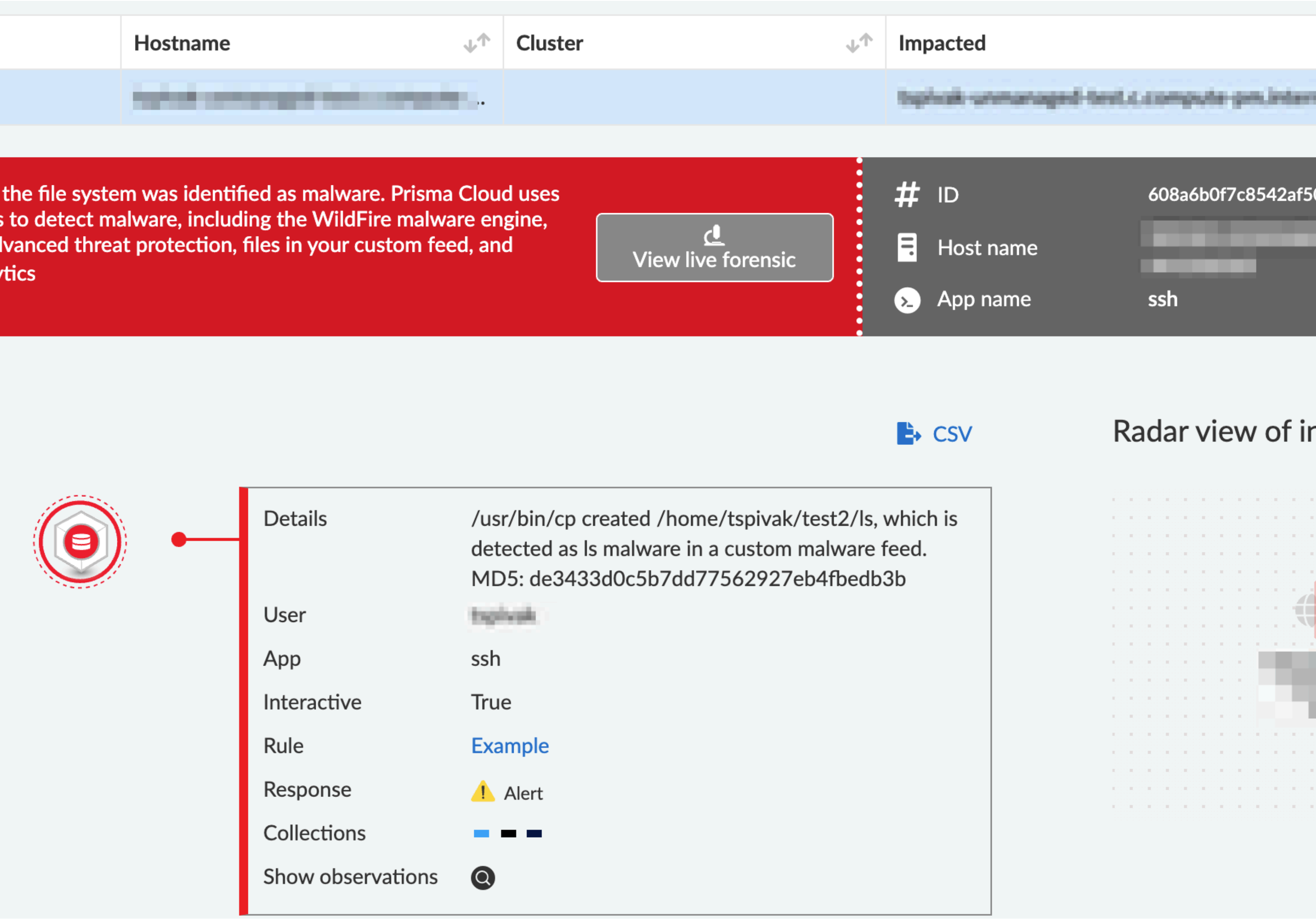
File can be identified as malware by WildFire, Prisma Cloud advanced threat intelligence feed and custom feeds.

For files identified as malware by Wildfire, the WildFire report should be examined for additional details on the malware behavior.

A malware incident indicates an attacker has access to writing or modifying files in a container/host and might have gained full code execution.

Therefore, for investigating this incident you must first determine the source of the file write call. The process that called the file write is likely malicious, or a user may have downloaded the malware unaware of the risk.

You should further investigate how this process gained execution. Review the forensics date for the container/host, other entries in the Incident Explorer, and audits from the source, looking for unusual process execution, hijacked processes, and explicit execution of commands.



Mitigation

A full mitigation strategy for this incident begins by resolving the issues that allowed the attacker to write or modify the file.

Ensure that compliance benchmarks are appropriately applied to the affected resources. For example, if the critical file systems in the host are mounted read-only, it will be more difficult for an attacker to change system files and configurations to their advantage.

Port scanning

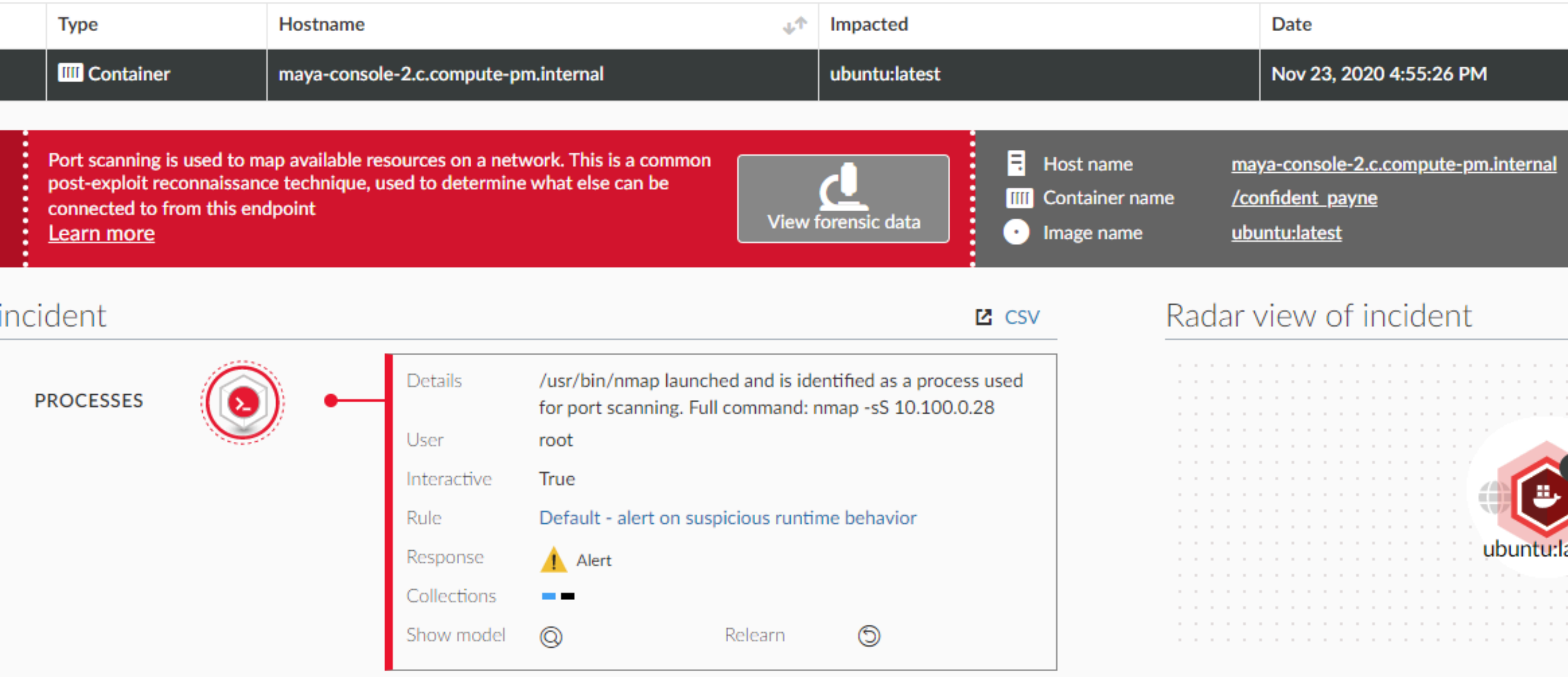
Edit on GitHub

Port scans are a method for finding which ports on a network are open and listening. It is a reconnaissance technique that gives attackers a map of where they can further probe for weaknesses.

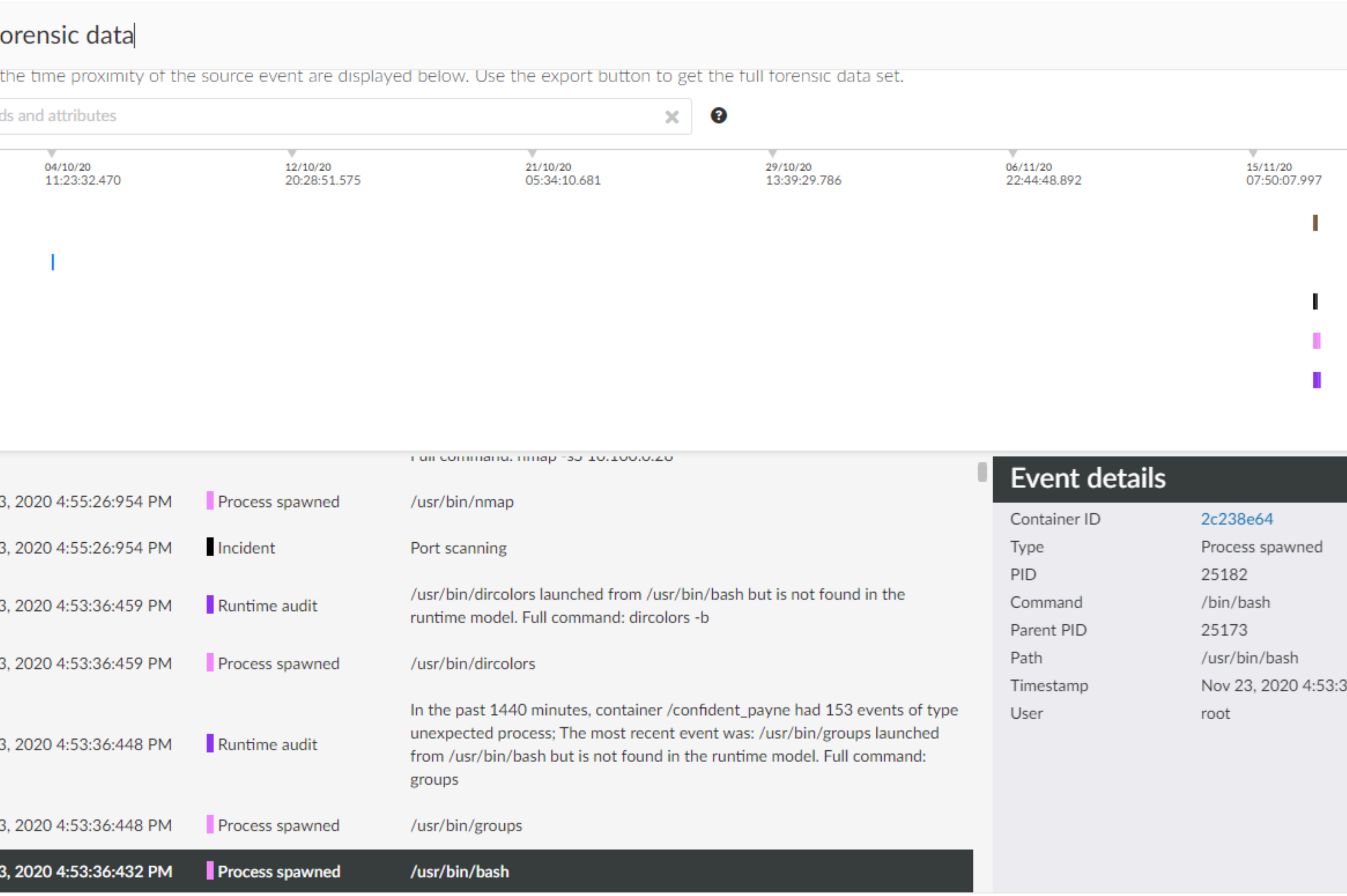
Port scanning incidents indicate that a container is attempting to make an unusual number of outbound network connections to hosts and ports to which it does not normally connect. Port scanning could be a post-compromise attempt to use the container to find other resources on the network as a precursor to lateral movement.

Investigation

The following screenshot shows a port scanning incident.



The first step in an investigation is to determine whether the source of the outbound network activity was an otherwise-valid process that was misused or a newly introduced process. Prisma Cloud forensics are a great place to start. In Incident Explorer, click **View Live Forensics**. It shows that *bin/bash* was launched immediately before the port scan, and that the shell was used to launch nmap. Nmap is a popular network scanning tool.



The next step in the investigation is to determine how nmap was introduced and executed. Some plausible scenarios include:

- A user account was used to execute nmap via a Docker command from the host. If enabled, Prisma Cloud access logs would show which user ran the command and when it was run.
- A remote code execution vulnerability was used to run nmap remotely. If the Prisma Cloud Web Application and API Security (WAAS) was configured to protect this container's inbound traffic, the WAAS logs may help with your investigation. Additionally, logs from the services in the container, such as Apache access logs, may shed additional light on the incident.

Once the cause has been identified, the next step in the investigation is to review the services that the actor may have discovered via port scanning and to inspect those containers to ensure that there hasn't been additional lateral movement. Container runtime audits may show specific connection attempts.

Mitigation

Mitigation and remediation for a port scanning incident should focus on resolving the issue that allowed execution of the responsible process.

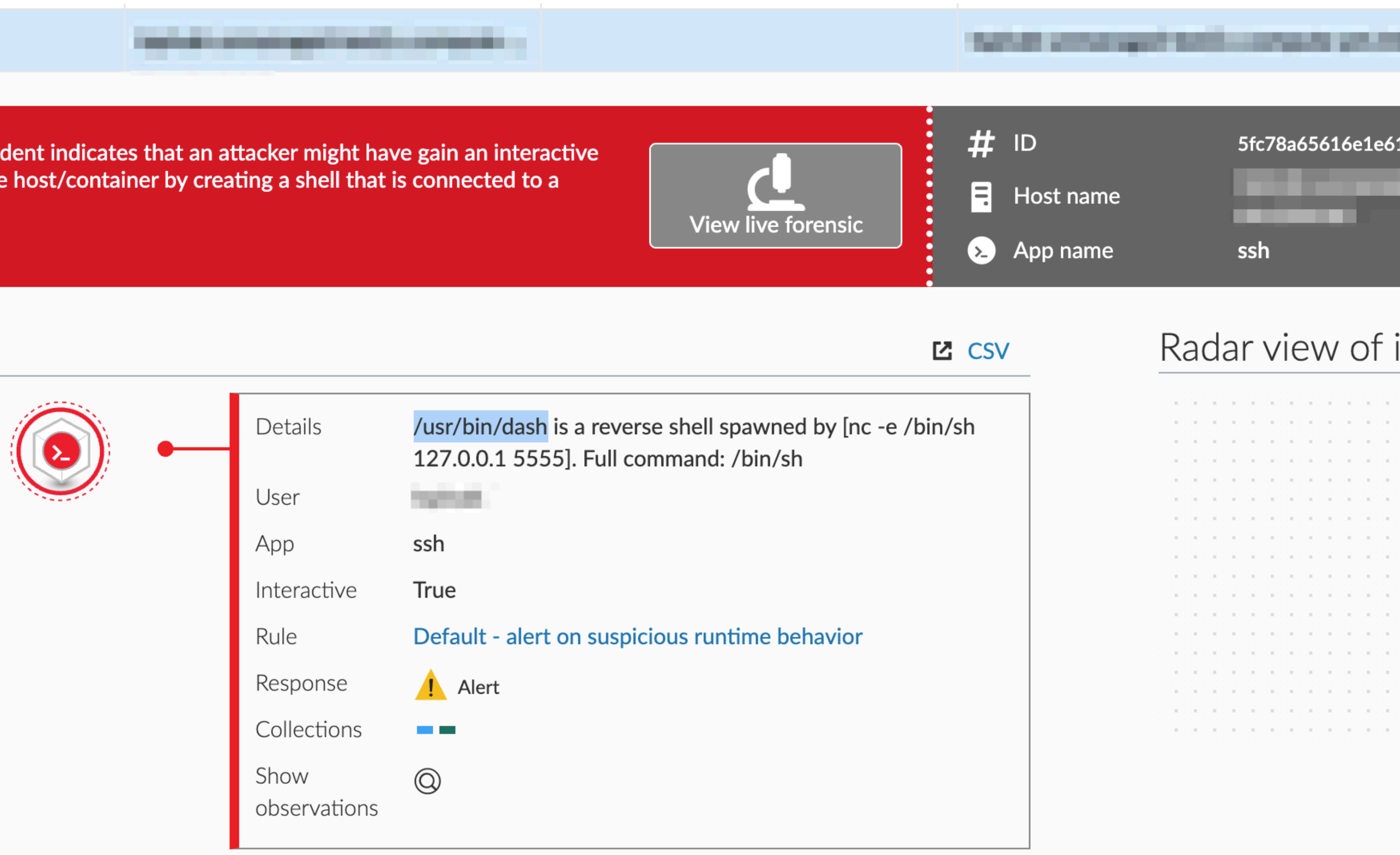
Reverse shell

Edit on GitHub

Reverse shell is a method used by attackers for gaining access to a victim's system. A reverse shell is a established by a malicious payload executed on a targeted resource which connects to a pre-configured host and provides an attacker the means to execute interactive shell commands through that connection.

Investigation

In the following incident, you can see that a reverse shell was used to provide a remote user interactive shell on this host, potentially enabling an attacker to execute any command that the user used to launch the reverse shell is authorized to execute.



The first step in an investigation is to validate that the reverse shell represent a bona fide security incident. While it is unlikely that a legitimate application or user is using a reverse shell for legitimate reasons, the first step should be validation that the reported application and user have not used reverse shell intentionally.

In this case it appears that a user used nc in order to allow a remote shell via ssh. "View forensics data" can be used to gain better understanding on what was done via the shell and understand whether this was for legitimate activity.

Having determined that this is a bona fide incident, the next steps focus on determining how an attacker managed to execute the process that allowed them to initiate the remote shell.

Check Incident Explorer for additional incidents. Review additional runtime audits for the source to see if there are other clues.

Review access to the resources and ensure that the affected account(s) weren't subsequently used for further access to systems and data.

Mitigation

A full mitigation strategy for this incident begins with resolving the issues that allowed the attacker to execute the process that initiated the remote shell.

Ensure that compliance benchmarks and patches are appropriately applied to the affected resources. For example, an unpatched critical vulnerability can be abused to execute a process that allows for the remote shell to be triggered remotely.

Suspicious binary

Edit on GitHub

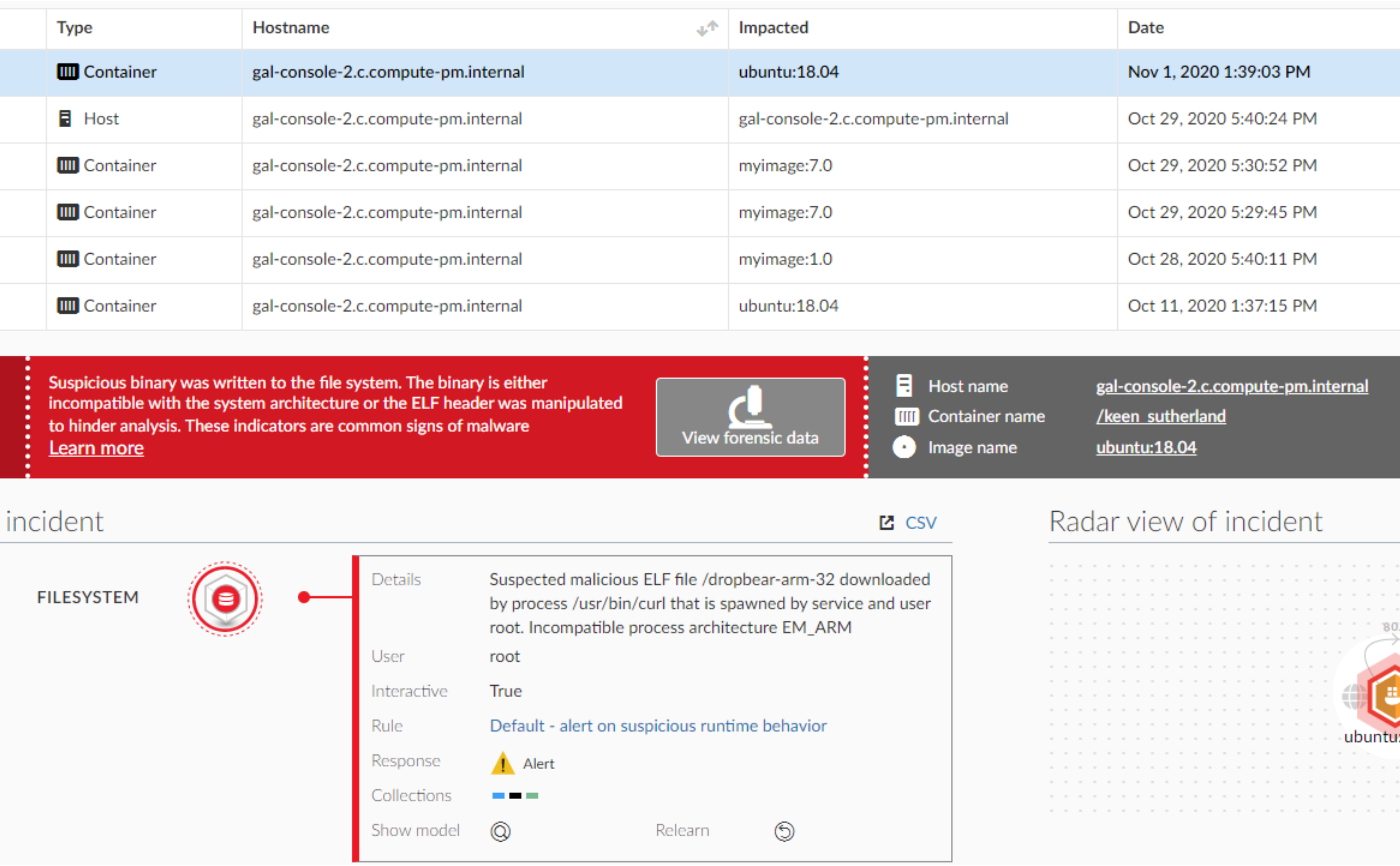
A suspicious binary incident indicates that a suspicious binary was written to the file system. The binary is either incompatible with the system architecture or the ELF header was manipulated to hinder analysis. These indicators are common signs of malware.

Investigation

The first indicator of a suspicious binary would be that the binary is incompatible with the system architecture.

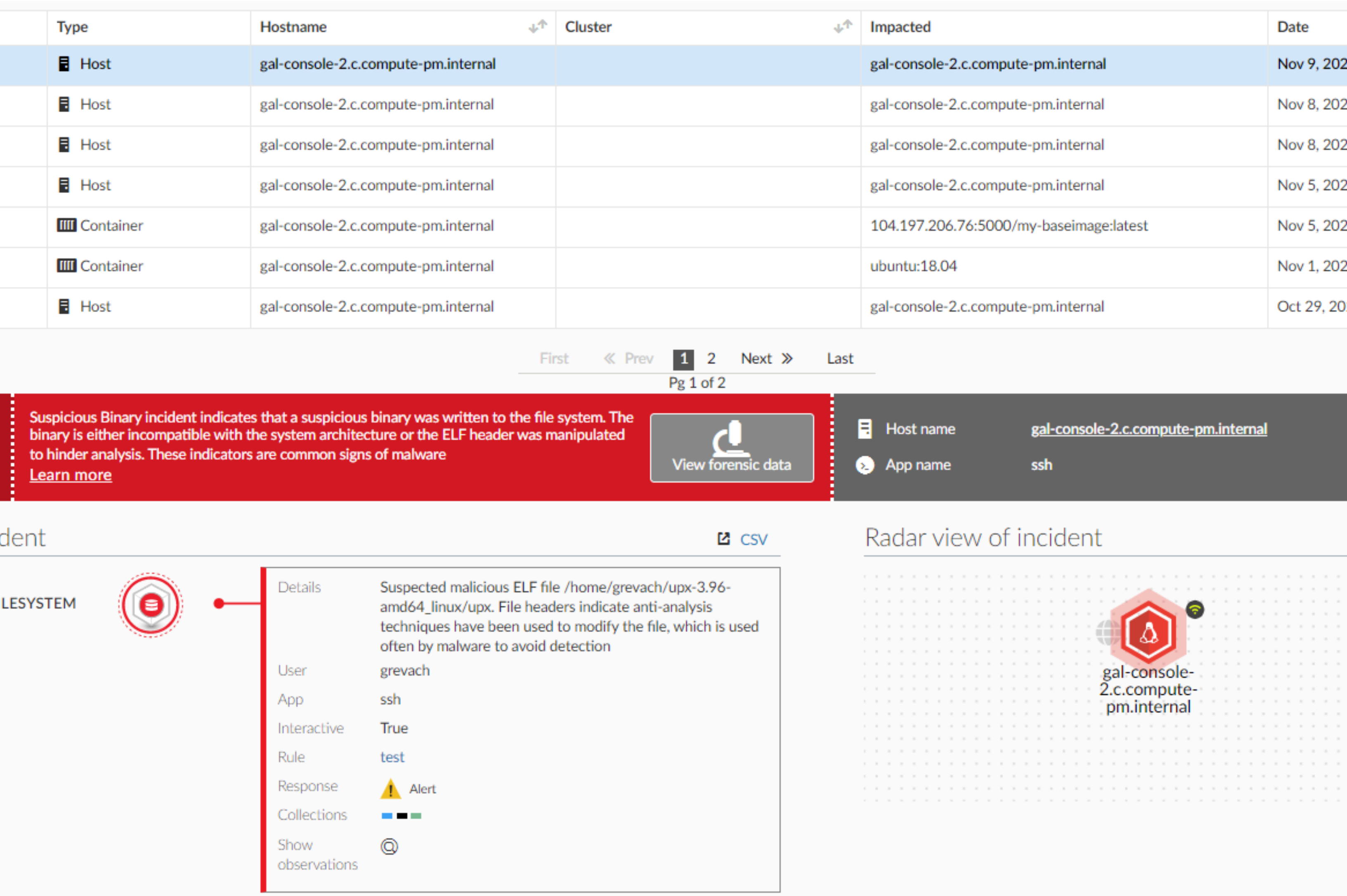
Attackers use automated tools frequently to download multiple binaries of different architectures when the target architecture is not known. Process *curl* downloading an ARM binary, for example, strongly indicates a breach had taken place.

The following incident shows that the process *curl* downloaded the ELF file */dropbear-arm-32*, which is incompatible with the system architecture.



The second indicator of a suspicious binary would be ELF headers with non-typical content. This indicates that the binary might have been compiled by attacking tools or otherwise hindered to avoid detection.

The following incident shows that the ELF file *upx* was written to the file system and is suspected as malicious. Its ELF header indicates using anti-analysis techniques to modify the file.



When triggered in a container, the suspicious binary incident indicates an attacker has access to writing or modifying files in the container and might have gained full code execution in the container.

Therefore, for investigating this incident you must first determine the source of the file write call. The process that called the file write is likely malicious.

You should further investigate how this process gained execution. Review the forensics date for the container/host, other entries in the Incident Explorer, and audits from the source, looking for unusual process execution, hijacked processes, and explicit execution of commands.

Mitigation

A full mitigation strategy for this incident begins by resolving the issues that allowed the attacker to write or modify the file.

Ensure that compliance benchmarks are appropriately applied to the affected resources. For example, if the critical file systems in the host are mounted read-only, it will be more difficult for an attacker to change system files and configurations to their advantage.

Other incident types

Edit on GitHub

- **Hijacked Process:** Indicates that an allowed process has been used in ways that are inconsistent with its expected behavior. This type of incident could be a sign that a process has been used to compromise a container.
- **Data exfiltration:** Indicates the unauthorized transfer of data from one system to another. These incidents are triggered when a pattern of audits indicate attempts to move data to an external location. For example: High rate of DNS query events, reporting aggregation started in a container, DNS resolution of suspicious name (www.<WEBSITE_NAME>.com).
- **Cloud Provider:** Indicates attempts to abuse a provider's service to extract sensitive information. For example: Container A queried provider API at <IP_ADDRESS>.

Access control

Edit on GitHub

Establish and monitor access control measures for cloud workloads and cloud native applications.

- Role-based access control for Docker Engine
- Admission control with Open Policy Agent

Role-based access control for Docker Engine

Edit on GitHub

Prisma Cloud lets you control access to Docker commands based on group membership.

Prisma Cloud lets you:

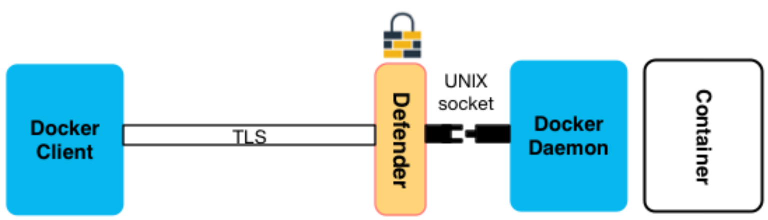
- Secure access to remote Docker Engine instances.
- Control access to Docker commands on a user-by-user basis.

After integrating Prisma Cloud with Active Directory, OpenLDAP, or SAML, you could create a group called Dev Team. Then in Console, you could grant all users in Dev Team permission to remotely run any Docker commands on hosts in the development environment, but deny permission to create, start, or stop containers on hosts in the production environment.

The groups specification is not applicable for Prisma Cloud Enterprise (SAAS) Console.

Securing remote access

The following diagram shows how Docker commands are routed from a user's workstation over the network to a host protected by Defender:



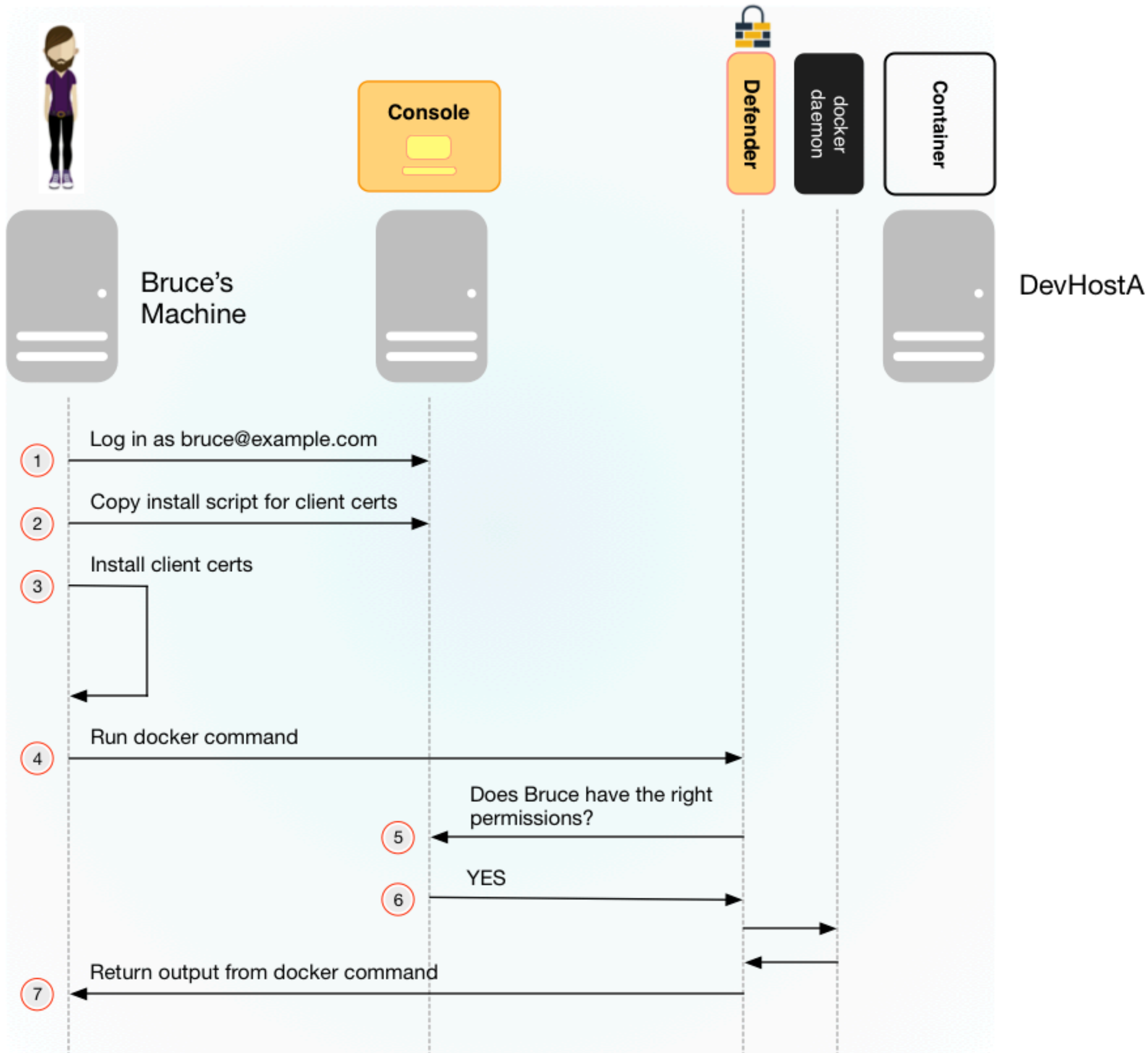
The Docker client securely transmits the command over the network to Defender using the Transport Layer Security (TLS) protocol. Defender acts as a proxy to the Docker daemon. If the installed policy permits the command to be executed, it is forwarded to the Docker daemon over the UNIX socket. The UNIX socket is created when the Docker daemon first starts, and it exposes a REST API through which Docker commands can be run.

Controlling access to resources

The following sequence diagram shows how users gain access to Docker resources, and how your access policies are enforced.

In this flow, it is assumed that:

- User Bruce has been added to the AD group Prisma Cloud Devs.
- You have already configured your access policy rules in the Prisma Cloud Console.



1. Bruce logs into Console with his LDAP credentials. He's directed to the single page user view.
2. From the single page user view, he copies a command that installs certs on his machine. These certs identify him as Bruce. Group memberships for the user are embedded in the certificate.
3. Bruce runs the install command on his machine. It copies the certs into the *$HOME/.docker* directory. He can now use TLS to communicate securely with hosts that run Defender.
4. Bruce runs a Docker command on DevHostA (protected by Defender) from his local machine. He specifies the hostname for DevHostA and the port number where Defender listens. By default, Defender listens for TLS traffic on port 9998.
5. Defender acts as a gateway to the Docker daemon. It uses the certificate to determine the user's identity and group memberships. Defender allows or blocks the command, depending on the access policies specified in Console.
6. In this case, Bruce has the right permissions to run this docker command. The command is forwarded to the docker daemon.
7. The response from the Docker daemon is routed back to Bruce through Defender.

Note that Defender does not talk to the identity provider (IdP). Instead, it relies on the user certificate generated from the initial authentication flow, when the user first tries to log into Console. The validity period for the certificate is controlled by the IdP, which embeds the login expiration into its response.

Setting Defender's listener type

To enforce role-based access control, Defender's listener type must be set to TCP.

Clients connect to the Docker socket and use the Engine API to manage and control containers on a host. The best known client is the docker command line tool (docker run, docker ps, etc).

In TCP mode, Defender intercepts traffic to the Docker socket and assesses it against the policies you have installed in Console. With this setup, Defender can block Docker commands and prevent them from reaching the Docker socket for execution by the Docker daemon.

In TCP mode, Defender listens for Docker traffic on port 9998 (this value can be configured). Defender runs as a Docker client with non-exclusive access to the Docker socket. Anyone who gains direct access to the Docker daemon will be able to bypass Defender and your policies. To prevent attackers from circumventing Defender, you should lock down your hosts and harden them for least privilege access.

Docker commands should only be run from remote machines through Defender on port 9998. Any user running Docker commands on port 9998 must be authenticated and authorized. Console generates certificates for users to authenticate to Defender. Any command run against Defender must also be explicitly allowed. Prisma Cloud ships with a default deny-all rule that blocks all commands for all users.

You can dynamically change Defender's listening type from Console, even after Defender is installed.

STEP 1 | Open Console, and go to **Manage > Defenders > Manage**.

STEP 2 | Click on a Defender listed in the table to open a dialog with more details.

STEP 3 | In the **Choose the socket type** drop-down list, select **TCP**.

STEP 4 | Click **Save**. The socket type for the Defender is updated in the Defenders status table.

Version	Type	Socket Type	Roles	Status	Restart	Upgrad
2.1.77	Docker on Linux	TCP Socket	Registry Scanner Docker Proxy	Connected for 3 hours		

Authentication and identity

Prisma Cloud can authenticate users against its internal local database. The initial admin user created when you first access Console, for example, is a local user. Prisma Cloud can also authenticate users against external services, such as Active Directory or SAML Identity Providers.

Users are identified with client certificates. These certs are automatically generated by Prisma Cloud for each user. Users log into Console with their credentials, then download a script that installs the certs on their machine. Client certs should be installed on any host where the *docker* client can be run.

To install the initial client certs on your host:

STEP 1 | Open Console.

STEP 2 | Log in with your credentials.

STEP 3 | Go to **Manage > Authentication > User Certificates**.

Users with the *Access User* role are directed to this page by default.

STEP 4 | Install your client certs, which are used to authenticate commands sent from the Docker client through Prisma Cloud.

Copy the curl-bash command under **Client certificate installation**, then run it on your host. Your client certificate, client private key, and the certificate authority certificate are installed in *$HOME/.docker/*.

If you're using custom certificates for authentication, then the above commands only install the certificate authority in the default Docker folder. The other two user certificates must be manually copied to this location.

Configuring Docker client variables

For access control to work, all Docker commands must be routed through Defender. You can configure your environment to shorten the Docker commands that target remote hosts protected by Defender. You should have already installed your client certificates.

To access Docker daemon through Defender, explicitly specify the host and the port of the Defender. For example:

```
$ docker -H <defender_host_address>:9998 run alpine
```

To simplify and shorten the Docker command, set up the following environment variables to route management traffic to Defender by default.

```
$ export DOCKER_HOST=tcp://<defender_host_address>:9998
$ export DOCKER_TLS_VERIFY=1
```

These environment variables can be set on a local machine (such as a dev laptop) that accesses Docker daemon on some remote host (such as a corporate cloud), or they can set directly on the host that runs Defender, for users who do not have root priviledges (which should be the majority of the users on such a host).

Creating access control rules

Admins can create policies that control which users can run what commands on what hosts.

For example, an admin could create an access control rule called that limits members of the "Dev team" group to a handful of read-only operations so they can debug issues in the production environment. The admin might decide that *docker ps*, *docker logs*, and *docker inspect* are sufficient for devs to do their job, and he could limit the scope of the rule to hosts named *prod**. When this rule is activated, users that are part of the Dev Team group can only run these Docker client commands on production hosts. All other commands are blocked.

Modify the parameters in this example to meet your own specific requirements.

Prerequisites:

- For the purposes of example scenario, you have integrated Prisma Cloud with Active Directory. You could also integrate with OpenLDAP or SAML, or have Prisma Cloud manage your users and groups.
- You have created AD groups for the different types of users that need access to Docker services. This procedure assumes you have a group called Prisma Cloud Devs, and that it has at least one user.

STEP 1 | Set up a user access rule.

1. Log into Console as an admin user.
2. Go to **Defend > Access > Docker**.
3. Click **Add rule**.
4. Enter a name for your rule.
5. Set **Effect** to **Allow**.
6. Deselect **All**, then select the **Actions** to allow:
   - **container_list** to allow access to the *docker ps* command.
   - **container_logs** to allow access to the *docker logs* command.
   - **container_inspect** to allow access to the *docker inspect* command.
7. In the **Groups** field, delete the wildcard (*) and enter the group(s) for which this rule applies.

   For example, enter **Dev team**.
8. Click **Save**.
9. Verify that your new rule is at the top of the list.

   Console ships with a default rule that blocks all Docker commands from remote clients.

   Rules are enforced according to the order that they are listed in Console. Rules at the top of the list have a higher priority than rules lower down.

STEP 2 | Verify that your policy is being enforced.

1. If you're logged in to Console as an admin user, log out.
2. Log into Console as a user from your group.
3. On the **Manage> Authentication > Credentials** page, copy the install command for the client certificate.
4. On your local machine, paste the install command into a shell window and run it.
5. Run a Docker command that's not allowed.

```
$ docker -H <HOST>:9998 --tlsverify pull nginx
Error response from daemon: [Prisma Cloud] The command
 'image_create' denied for user 'bruce@example.com' by rule
 'devs_rule'
```


Troubleshooting

You cannot run Docker commands

First remove Prisma Cloud from the equation. Verify that you can communicate with Docker locally without Defender in the middle. After you have verified this setup, review the parameters you pass to the docker client.

Your policies are not being properly enforced.

Verify your user is in the AD group by following the below steps on the Docker host(s) where you're trying to execute a command:

1. Install ldap-utils:

```
$ sudo apt-get install ldap-utils
```

2. Query Active Directory to verify that your user belongs to your AD group. Use the same parameters that you specified in your integration configuration.

```
$ ldapsearch \
  -x -H [LDAP_URL] \
  -D [LDAP_ADMIN_UPN] \
  -W \
  -b [LDAP_SEARCH_BASE]\
  -s sub (&(userPrincipalName=[UPN])(memberof=[LDAP_GROUP_DN]))
```

Where:

- ***UPN*** --

  User Principal Name of the user

- ***LDAP_GROUP_DN*** --

  Full DN of the LDAP group. For example: *CN=group1,DC=USERS,DC=TWISTLOCK,DC=LOCAL*

Admission control with Open Policy Agent

Edit on GitHub

Prisma Cloud provides a dynamic admission controller for Kubernetes and OpenShift that is built on the Open Policy Agent (OPA). In Console, you can manage and compose rules in Rego, which is OPA's native query language. Rules can allow or deny (alert or block) pods. Console pushes your policies to Defender, which enforces them. Decisions made by the system are logged.

There is currently no support for Windows.

Open Policy Agent

The Open Policy Agent is an open source, general-purpose policy engine that lets you consolidate policy enforcement in a single place. OPA can enforce policies in microservices, Kubernetes clusters, CI/CD pipelines, API gateways, and so on. OPA provides a high-level declarative language called Rego, which lets you specify policy as code. The OPA APIs let you offload policy decision-making from your software.

OPA decouples policy decision-making from policy enforcement. When your software needs to make policy decisions, it queries OPA and supplies structured data, such as JSON, as input. The data can be inspected and transformed using OPA's native query language Rego. OPA generates policy decisions by evaluating the query input and against policies and data.

Prisma Cloud operationalizes OPA by:

- Extending Console to manage and compose policies in Rego.
- Integrating OPA's decision-making library into Defender.
- Connecting Defender's enforcement capabilities to OPA's decisions.

Admission webhook

An admission controller is code that intercepts requests to the API server for creating objects. There are two types of admission controllers: built-in and dynamic. Prisma Cloud implements a dynamic admission controller.

Dynamic admission controllers are built as webhooks. After registering to intercept admission requests, they assess requests against policy, and then accept or reject those requests. In Kubernetes terms, these are known as *validating admission webhooks*.

The Prisma Cloud validating admission webhook handles the API server's AdmissionReview requests, and returns decisions in an AdmissionReview object. When configuring Prisma Cloud, you'll create a ValidatingWebookConfiguration object, which sets up the Defender service to intercept all create, update, and connect calls to the API server.

The default ValidatingWebookConfiguration provided here sets failurePolicy to Ignore. The failure policy specifies how your cluster handles unrecognized errors and timeout errors from the admission webhook. When set to Ignore, the API request is allowed to continue.

Configuring the webhook

Configure the API server to route AdmissionReview requests to Prisma Cloud.

Prerequisites:

- You have a running instance of Prisma Cloud Compute Console.
- You have a Kubernetes cluster. Minimum supported version is v1.16.
- Defender has been deployed to your cluster as a DaemonSet. In Console, you can verify Defenders are running and connected under **Manage > Defenders > Manage**.

STEP 1 | Go to **Defend > Access > Admission**

STEP 2 | Enable admission control.

STEP 3 | Click **Go to settings**.

1. Copy the configuration provided to a file named *webhook.yaml*

> *If the Defender CA has been rotated and the old certificate still hasn't expired, you may have Defenders using an old certificate. For daemonset which its Defenders are using an old certificate, you need to retrieve the old Defender CA certificate from the daemonset yaml file you deployed this daemonset with.*
>
> *Search for defender-ca.pem within the daemonset yaml, copy its content, then paste it to replace the content of the caBundle field of the webhook. If defender-ca.pem doesn't exist in the daemonset yaml, use the content of the ca.pem field.*
>
> *If you don't have the yaml file you used to deploy the daemonset, you can retrieve the old CA bundle from the Console certificates folder under old-defender-ca.pem.*
>
> *To identify whether your Defenders are using an old certificate, see* Console-Defender communication certificates.

STEP 4 | Click **Save**

STEP 5 | Create the webhook configuration object.

```
$ kubectl apply -f webhook.yaml
```

After creating the object, the Kubernetes API server directs AdmissionReview requests to Defender.

Validating your setup

Validate that your webhook has been properly set up with one of the predefined admission rules.

The order in which the rules appear is the order in which they are evaluated. Higher rules take precedence over lower rules. Rules can be reordered. Use the hamburger icon to drag and drop rules into the right place.

Notice that the processing of rules stops at the first match. To make sure the severe action will be taken in a case of more than one rule match, place the rules with action "Block" first.

STEP 1 | Navigate to **Defend > Access > Admission** and verify there exist default admission rules and they are all enabled by default.

STEP 2 | Create the following YAML file to test the **Twistlock Labs - CIS - Privileged pod created** rule.

1. Create the following YAML file: **priv-pod.yaml**

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  containers:
  - name: nginx
    image: nginx
    ports:
    - containerPort: 80
    securityContext:
      privileged: true
```

STEP 3 | Create the privileged pod.

```
$ kubectl apply -f priv-pod.yaml
```

STEP 4 | Verify an audit is created under **Monitor > Events > Admission Audits**.

STEP 5 | Clean up. Delete the pod.

```
kubectl delete -f priv-pod.yaml
```

Creating custom admission rules

Use Rego syntax to create custom rules. To learn more about the syntax, review the predefined rules that ship with Prisma Cloud. Rules scripts are based on the admission review input JSON structure. For more information, see: https://github.com/kubernetes/api/blob/master/admission/v1beta1/types.go.

Examples

The following examples should give you some ideas about how you can create your own policies by using the Rego language.

Do not allow new namespaces to be created:

```
match[{"msg": msg}] {
 input.request.operation == "CREATE"
 input.request.kind.kind == "Namespace"
```

```
 msg := "It's not allowed to create new namespace!"
}
```

Do not allow a specific image (for example nginx) in new pods:

```
match[{"msg": msg}] {
 input.request.operation == "CREATE"
 input.request.kind.kind == "Pod"
 input.request.resource.resource == "pods"
 input.request.object.spec.containers[_].image == "nginx"
 msg := "It's not allowed to use the nginx Image!"
}
```

Do not allow new pods to expose TCP port 80:

```
match[{"msg": msg}] {
 input.request.operation == "CREATE"
 input.request.kind.kind == "Pod"
 input.request.resource.resource == "pods"
 input.request.object.spec.containers[_].ports[_].containerPort == 80
 msg := "It's not allowed to use port 80 (HTTP) with a Pod
 configuration!"
}
```

Control the scope of your the policy rules by checking the object's metadata, such as namespace or labels.

Do not allow new pods in namespace *sock-shop* without the *owner* label:

```
match[{"msg": msg}] {
 input.request.operation == "CREATE"
 input.request.kind.kind == "Pod"
 input.request.resource.resource == "pods"
     input.request.object.metadata.namespace == "sock-shop"
     not input.request.metadata.labels.owner
 msg := "Pod in namespace sock-shop is missing the owner label"
}
```

Continuous integration

Edit on GitHub

Prisma Cloud integrates security into your continuous integration workflows so you can find and fix problems before they enter production. Prisma Cloud's CI plugins surface vulnerability and compliance issues directly in the build tool every time developers build their container images and serverless functions. Security teams can set policies that only allow compliant and fully remediated images to progress down the pipeline.

- Jenkins plugin
- Jenkins Freestyle project
- Jenkins Maven project
- Jenkins Pipeline project
- Run Jenkins in a container
- Jenkins pipeline on Kubernetes
- CI plugin policy
- Code repo scanning

Jenkins plugin

Edit on GitHub

The Jenkins plugin for Prisma Cloud enables you to scan container images and serverless functions for security vulnerabilities and compliance issues within your continuous integration pipeline.

You can download the Jenkins plugin directly from Console (**Manage > System > Utilities**). It's also delivered with the release tarball that you download from Releases.

In order to interoperate, both Console and the Jenkins plugin must be from the same release.

The Jenkins plugin is built for Jenkins on Linux. To scan images with Jenkins on other operating systems, use a platform-specific twistcli binary.

The Jenkins plugin doesn't currently support scanning Windows images for vulnerability and compliance issues on hosts with the containerd runtime. However, the Jenkins plugin does support scanning when running on hosts with Docker Engine.

Build and scan flow

After Jenkins builds a container image or serverless function package, the Prisma Cloud Jenkins plugin scans it for vulnerabilities and compliance issues.

Prisma Cloud can pass or fail builds, depending on the types of issues discovered, and the policies you have defined in Console. By incorporating scanning into the build phase of the development workflow, developers get immediate feedback about what needs to be fixed. The scan report provides all the information required to fix the vulnerabilities that were identified in the scan.

The sequence of events is described below:

1. An developer commits a change, which triggers a build.
2. Jenkins builds the container image.
3. Jenkins calls the Prisma Cloud plugin for scanning. The plugin collects data about the image, including the packages and binaries in the image, and submits it to Console for analysis.
4. Console returns a list of vulnerabilities and compliance issues.
5. The Prisma Cloud plugin passes or fails the build depending upon your policy.

   For more information about configuring a scan, see: Setting up a Freestyle project, Setting up a Maven project, or Setting up a Pipeline project.

   For more information about targeting rules created in Console to the Jenkins plugin, see Set policy in the CI plugins.
6. Scan results can be reviewed in the following locations:
   - Directly in the Jenkins tool, including the project/job page and dashboard view.
   - In Prisma Cloud Console, in the **Monitor > Vulnerabilities > {Images | Functions} > CI** pages.

When scanning multiple images in a single build, results do not appear correctly in the Jenkins dashboard view or vulnerability trends table/graph. Only trend data for the last image scanned is shown. Instead, go to Console to see scan results for all images in the build.

Installing the Prisma Cloud Jenkins plugin

Install the Jenkins plugin.

The build console output in Jenkins may show the message - "No CA cert was specified, using insecure connection". This message is generated because twistcli, which the Jenkins plugin wraps, checks the Console's trust chain by default. When twistcli is run directly, the --tlscacert parameter can be passed to specify the signer, so this message is not shown. To simplify configuration, the Jenkins plugin doesn't provide this option, hence why the message is shown. The connection between Jenkins and Console is still fully encrypted with TLS.

The Prisma Cloud Jenkins plugin uses the proxy settings specified in your Jenkins HTTP proxy configuration, which can be found in ***Manage Jenkins > Manage Plugins > Advanced****.*

Prerequisites:

- Your version of Jenkins meets Prisma Cloud's minimum requirements.
- You have installed Prisma Cloud Console on a host in your environment.
- Your Jenkins host can reach Prisma Cloud Console over the network.
- We recommend adding a Prisma Cloud user with the *CI User* role to minimize privileges on Console. For more information, see user roles.

STEP 1 | Validate that the Jenkins host can communicate with Prisma Cloud Console.

STEP 2 | Open the Jenkins top page.

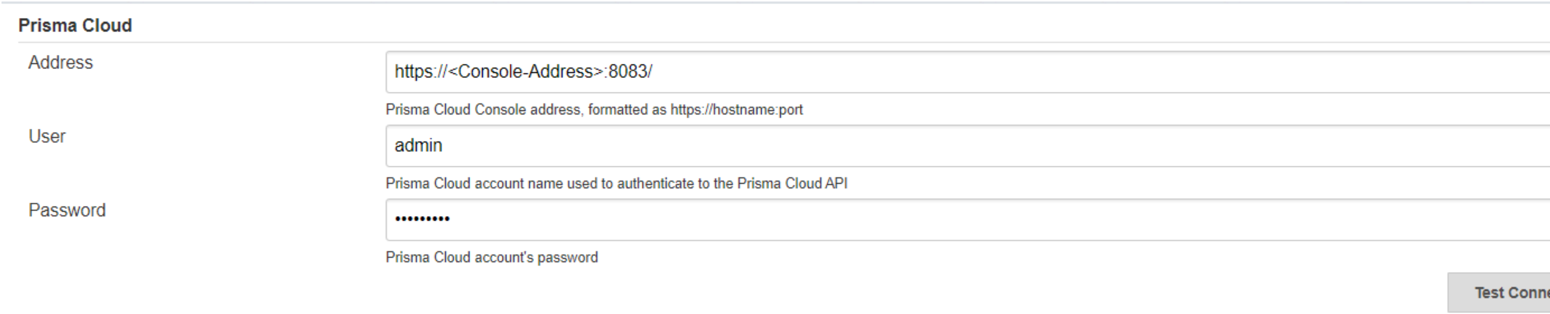
STEP 3 | Install the Prisma Cloud Jenkins plugin.

The Jenkins plugin can downloaded directly from Console (**Manage > System > Utilities**). It's also delivered with the release tarball that you download from Releases.

1. Click **Manage Plugins** (in the left menu bar), and then click the **Advanced** tab.
2. Scroll down to **Upload Plugin**, and click **Choose File**.
3. Navigate to the folder where you unpacked the Prisma Cloud download and select *prisma-cloud-jenkins-plugin.hpi*.
4. Click **Upload**.

STEP 4 | Configure the Prisma Cloud plugin.

1. Go to the Jenkins top page, and then click **Manage Jenkins** > **Configure System**.
2. Scroll down to the Prisma Cloud section.



Configuring a proxy:

3. In the **Address** field, enter the URL for Prisma Cloud Console.
4. In the **User** and **Password** fields, enter the **CI role** user's credentials for Prisma Cloud Console.

   The username is the access key ID and the password is the access key secret of the user with the CI role (Build and Deploy Security permission group with the option to create an access key on Prisma Cloud).
5. In **Choose Proxy Type**, select the proxy option that is to be used for the plugin to communicate with Console.

   Choose either the default global Jenkins proxy, configure a separate one, or choose to skip any Proxy communication with the 'No Proxy' option. If you choose to configure a separate proxy, fill in the proxy's address URL, port, username, password, and CA certificate (if any).
6. Click **Test Connection** to validate that the Jenkins plugin can communicate with Prisma Cloud Console.
7. Click **Save**.

Scan artifacts

When a build completes, you can view the scan results directly in Jenkins. To support integration with other processes and applications in your organization, Prisma Cloud scan reports can be retrieved from several locations.

Full scan reports for the latest build can be retrieved from:

- The scan results file in the project's workspace (by the name configured in the scan steps).
- The Prisma Cloud API. For more information, see the */api/v1/scans* endpoint for downloading Jenkins scan results.

For example, if you use [ThreadFix](https://threadfix.it/) to maintain a consolidated view of vulnerabilities across all your organization's applications, you could create a post-build action which triggers ThreadFix's Jenkins plugin to grab Prisma Cloud Compute's scan report from the project workspace and upload it to the ThreadFix server. Contact your ThreadFix support team for details on how to ingest this output.

To download the scan report from Console using the Prisma Cloud API, use the following command:

```
$ curl -k \
  -u <COMPUTE_CONSOLE_USER> \
  https://<COMPUTE_CONSOLE>/api/v1/scans/download?search=<IMAGE_NAME>
 \
  > scan_report.csv
```

Ignore image creation time

A common stumbling point is the "Ignore Image Build Time" option. This option checks the time the image was created against the time your Jenkins build started. If the image was not created after the start of your current build, the scan is bypassed. The plugin, by default, scans any image generated as part of your build process, but ignores images not created or updated as part of the build.

Keep in mind the nature of Docker creation time in regards to images. If nothing changes in the image, the creation time isn't updated. This could lead to a scenario where an image is built and scanned in one job, but not scanned in subsequent jobs because the creation time wasn't updated because the image didn't change.

Post build cleanup

Most pipelines push images to the registry after passing Prisma Cloud's vulnerability and compliance scan step. Pipelines also have a final cleanup step that removes images from the local Docker cache. If your build fails, and the pipeline is halted, use a **post** section to clean up the Docker cache. The **post** section of a pipeline is guaranteed to run at the end of a pipeline's execution.

For more information, see the Jenkins documentation.

What's next?

Set up a build job and configure Prisma Cloud to scan the Docker image generated from the job.

For more information, see:

- Jenkins Freestyle project
- Jenkins Maven project
- Jenkins Pipeline project

Notifications of build failures can be enabled using existing Jenkins plugins, for example:

- Mailer plugin
- Jira plugin
- Slack plugin

Jenkins Freestyle project

Edit on GitHub

Jenkins Freestyle projects let you create general-purpose build jobs with maximum flexibility.

Setting up a Freestyle project for container images

Create a Freestyle project that builds a Docker image and then scans it for vulnerability and compliance issues.

STEP 1 | Go to the Jenkins top page.

STEP 2 | Create a new project.

1. Click **New Item**.
2. In **Enter an item name**, enter a name for your project.
3. Select **Freestyle project**.
4. Click **OK**.

STEP 3 | Add a build step.

1. Scroll down to the **Build** section.
2. In the **Add build step** drop-down list, select **Execute shell**.
3. In the **Command** text box, enter the following:

```
echo "Creating Dockerfile..."
echo "FROM imiell/bad-dockerfile:latest" > Dockerfile
docker build --no-cache -t test/test-image:0.1 .
```

STEP 4 | Add a build step that scans the container image(s) for vulnerabilities.

1. In the **Add build step** drop-down list, select **Scan Prisma Cloud Images**.
2. In the **Image** field, select the image to scan by specifying the repository and tag.

   Use pattern matching expressions. For example, enter *test/test-image**.

   *If the image you want to scan is created outside of this build, or if you want to scan the image every build, even if the build might not generate an new image, then click **Advanced**, and select **Ignore image creation time**. For more information about advanced options, see here.*

STEP 5 | Add a post-build action to publish the scan results in Jenkins directly.

This post-build step depends on a file generated by the previous scan build step, which holds the scan results. This step specifically makes the results available for review in the Jenkins build tool. Note that the previous scan step already published the results in Console, and they're ready to be reviewed there.

1. Scroll down to **Post-build Actions**.
2. In the **Add post-build action** drop-down menu, select **Publish Prisma Cloud analysis results**.
3. In **Scan Result Files**, accept the default.

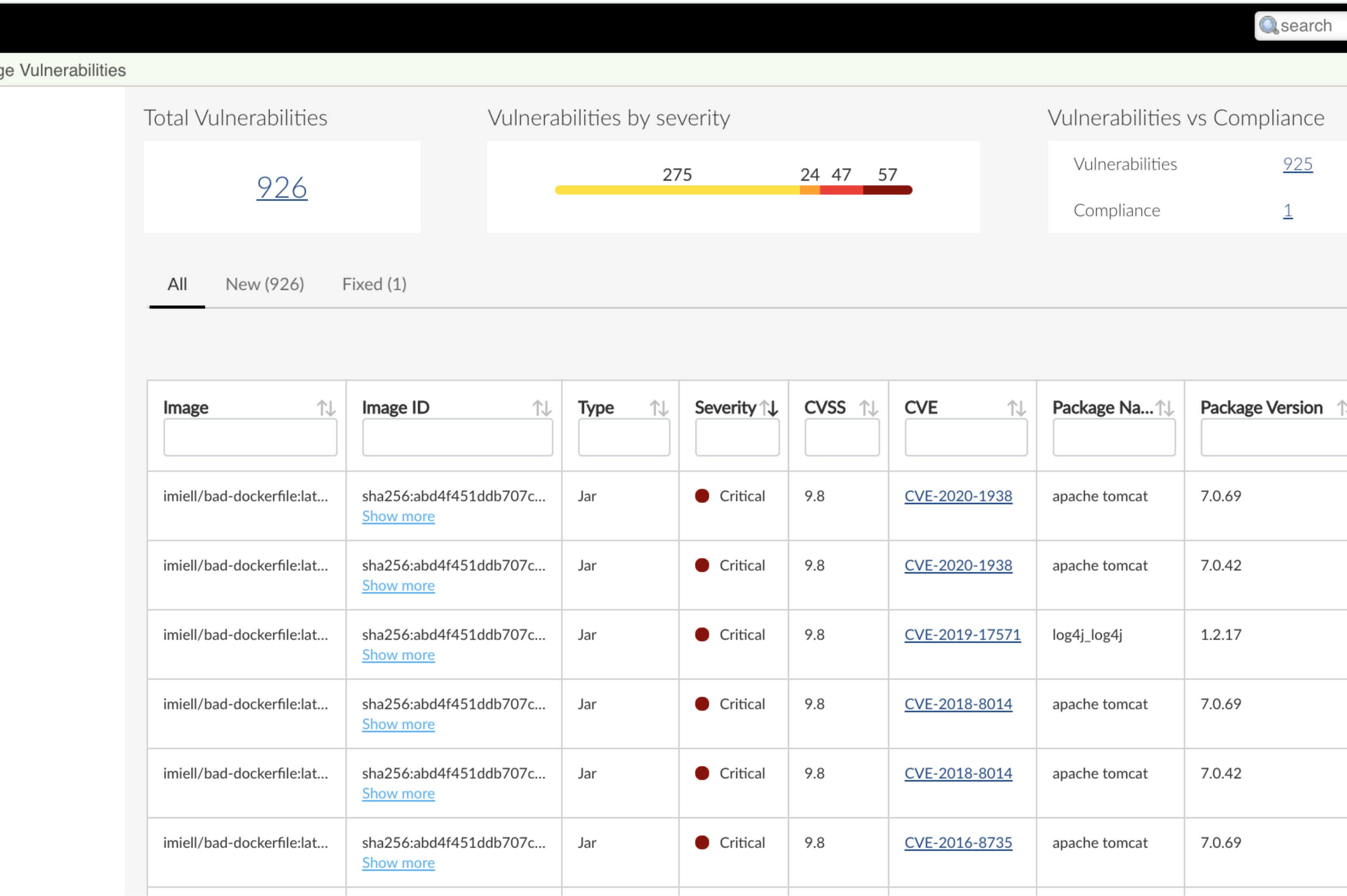
   Scan result files aren't deleted by the publish step. They stay in the workspace.

STEP 6 | Click **Save**.

STEP 7 | Click **Build Now**.

STEP 8 | After the build completes, examine the results. Scan reports are available in the following locations:

- Prisma Cloud Console: Log into Console, and go to **Monitor > Vulnerabilities > Images > CI**.
- Jenkins: Drill down into the build job, then click **Image Vulnerabilities** to see a detailed report.



Setting up a Freestyle project for serverless functions

The procedure for setting up Jenkins to scan serverless functions is similar to the procedure for container images, except you should use the **Scan Prisma Cloud Functions** build step.

Where:

- **Function Path** — Path to the ZIP archive of the function to scan.
- **Function Name** — (Optional) String identifier for matching policy rules in Console with the functions being scanned. When creating policy rules in Console, you can target specific rules to specific functions by function name. If this field is left unspecified, the plugin matches the function to the first rule where the function name is a wildcard.
- **AWS CloudFormation template file** — (Optional) Path to CloudFormation template file in either JSON or YAML format. Prisma Cloud scans the function source code for AWS service APIs being used, compares the APIs being used to the function permissions, and reports when functions have permissions for APIs they don't need.

Jenkins Maven project

Edit on GitHub

Create a Maven project that builds a Docker image and then scans it for vulnerability and compliance issues.

Configuring Maven

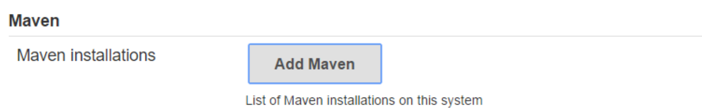
Configure Maven.

STEP 1 | Go to the Jenkins top page.

STEP 2 | Click Manage Jenkins.

STEP 3 | Select Global Tool Configuration.

STEP 4 | Scroll down to the Maven section (Not Maven Configuration), and click Add Maven.



Setting up a Maven project for container images

Set up a Jenkins Maven project.

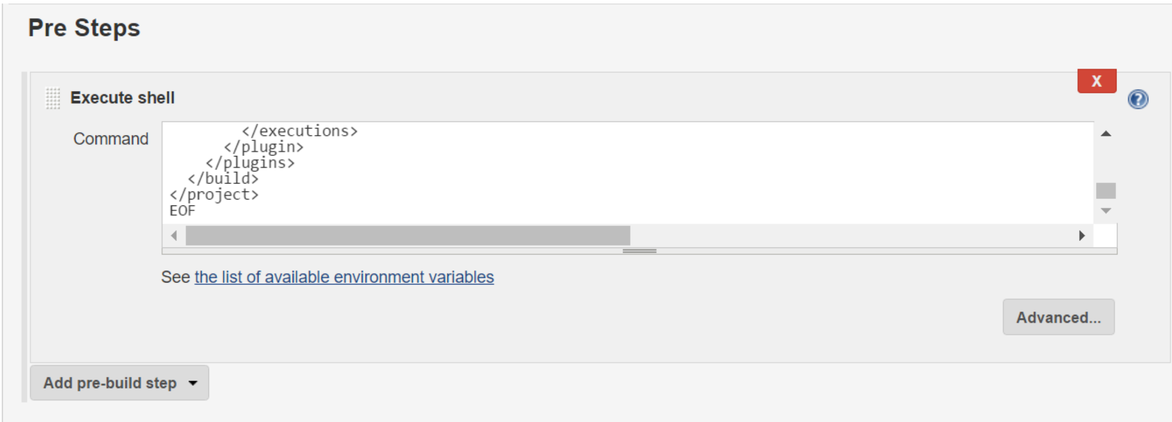
STEP 1 | Go to the Jenkins top page.

STEP 2 | Create a new project.

1. Click **New Item**.
2. In **Item** name, enter a name for your project.
3. Select **Maven project**.
4. Click **OK**.

STEP 3 | Add a build step.

1. Scroll down to the **Pre steps** section.
2. In the **Add pre-build step** drop-down list, select **Execute shell**.



3. In the **Command** text box, enter the following:

```
echo "Creating Dockerfile..."
echo "FROM imiell/bad-dockerfile:latest" > Dockerfile
echo 'docker build --no-cache -t test/test-image:0.1 .' >
 build_image.sh
chmod +x build_image.sh

echo "Creating POM file..."
cat > pom.xml << EOF
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>groupId</groupId>
  <artifactId>artifactid</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>
  <name>projectName</name>
  <properties>
    <project.build.sourceEncoding>UTF-8</
project.build.sourceEncoding>
  </properties>
  <build>

    <plugins>
      <plugin>
        <artifactId>exec-maven-plugin</artifactId>
        <groupId>org.codehaus.mojo</groupId>
        <executions>
          <execution>
            <id>Build Image</id>
            <phase>generate-sources</phase>
            <goals>
              <goal>exec</goal>
            </goals>
```

```
                <configuration>
                  <executable>build_image.sh</executable>
                </configuration>
              </execution>
            </executions>
          </plugin>
        </plugins>
      </build>
    </project>
    EOF
```

STEP 4 | Add a build step that scans the container image(s) for vulnerabilities.

1. In the **Add build step** drop-down list, select **Scan Prisma Cloud Images**.
2. In the **Image** field, select the image to scan by specifying the repository and tag.

   Use pattern matching expressions. For example, enter *test/test-image**.

   > *If the image you want to scan is created outside of this build, or if you want to scan the image every build, even if the build might not generate an new image, then click **Advanced**, and select **Ignore image creation time**.*

STEP 5 | Add a post-build action so that image scan results in Jenkins directly.

This post-build step depends on a file generated by the previous scan build step, which holds the scan results. This step specifically makes the results available for review in the Jenkins build tool. Note that the previous scan step already published the results in Console, and they're ready to be reviewed there.

1. Scroll down to **Post-build Actions**.
2. In the **Add post-build action** drop-down menu, select **Publish Prisma Cloud analysis results**.
3. In the **Scan Result Files** field, accept the default.

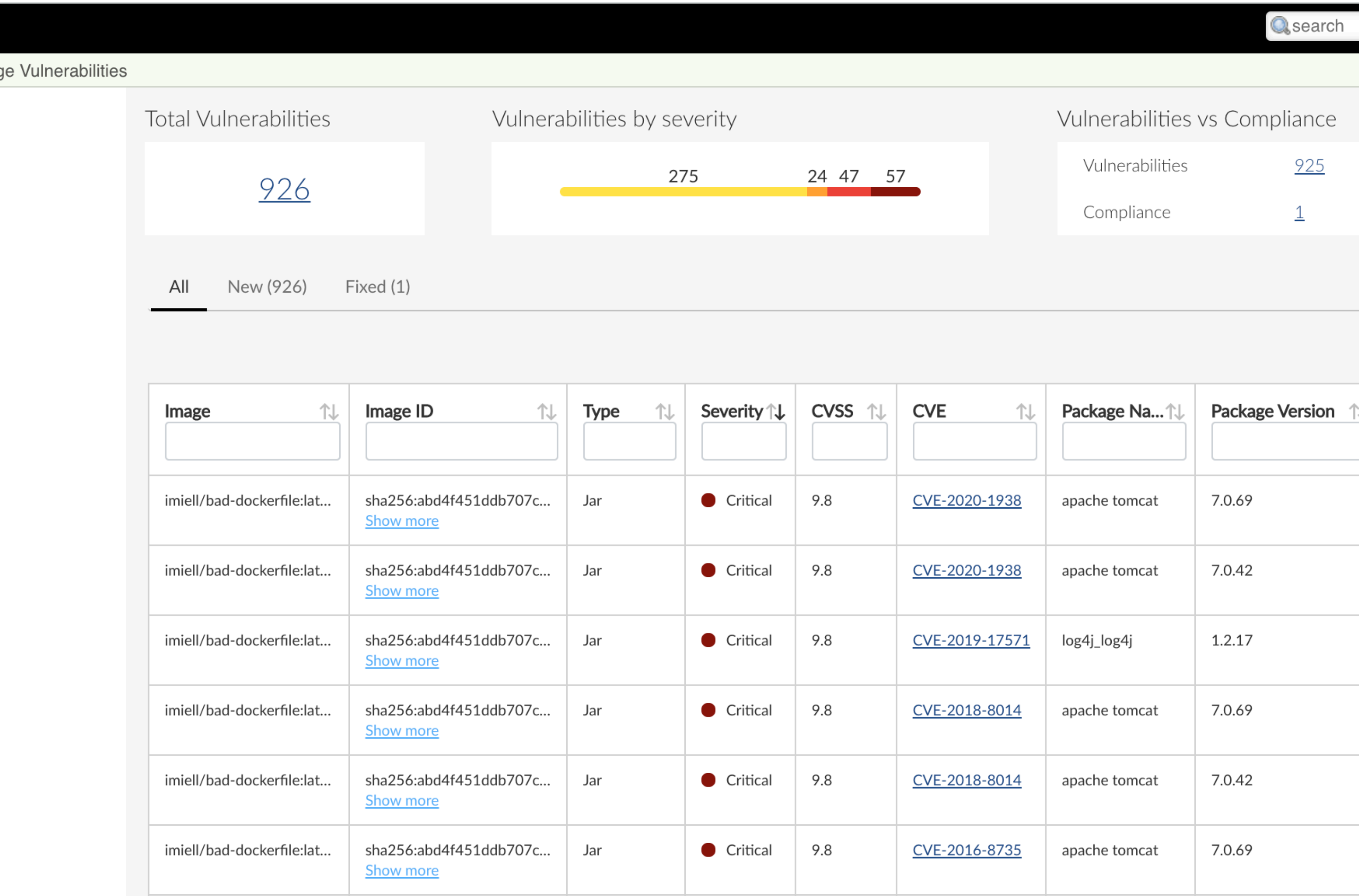
   Scan result files aren't deleted by the publish step. They stay in the workspace.

STEP 6 | Click **Save**.

STEP 7 | Click **Build Now**.

STEP 8 | After the build completes, examine the results. Scan reports are available in the following locations:

- Prisma Cloud Console: Log into Console, and go to **Monitor > Vulnerabilities > Images > CI**.
- Jenkins: Drill down into the build job, then click **Image Vulnerabilities** to see a detailed report.



Setting up a Maven project for serverless functions

The procedure for setting up Jenkins to scan serverless functions is similar to the procedure for container images, except you should use the **Scan Prisma Cloud Functions** build step.

Where:

- **Function Path** — Path to the ZIP archive of the function to scan.
- **Function Name** — (Optional) String identifier for matching policy rules in Console with the functions being scanned. When creating policy rules in Console, you can target specific rules to specific functions by function name. If this field is left unspecified, the plugin matches the function to the first rule where the function name is a wildcard.
- **AWS CloudFormation template file** — (Optional) Path to CloudFormation template file in either JSON or YAML format. Prisma Cloud scans the function source code for AWS service APIs being used, compares the APIs being used to the function permissions, and reports when functions have permissions for APIs they don't need.

After a build completes, you can view the scan reports in the following locations:

- Prisma Cloud Console: Log into Console, and go to **Monitor > Vulnerabilities > Functions > CI**.
- Jenkins: Drill down into the build job, then click **Vulnerabilities** to see a detailed report.

Jenkins Pipeline project

Edit on GitHub

The Prisma Cloud Jenkins plugin supports Jenkins Pipeline. Jenkins Pipeline lets you implement and integrate continuous delivery pipelines into Jenkins.

In this workflow, there are two sequential steps for analyzing scan results. The *publish* build stage depends on the results file generated by *scan* build stage. The results file must be accessible when running the *publish* step. Therefore, it's not possible to run both stages (*scan* and *publish*) on different nodes or in parallel.

For example, a pipeline script that scans a serverless function and publishes the results (assuming the function zip file exists in the current workspace) should look like this:

```
node('master') {
  stage('Scan') {
    prismaCloudScanFunction
  }

  stage('Publish') {
    prismaCloudPublish
  }
}
```

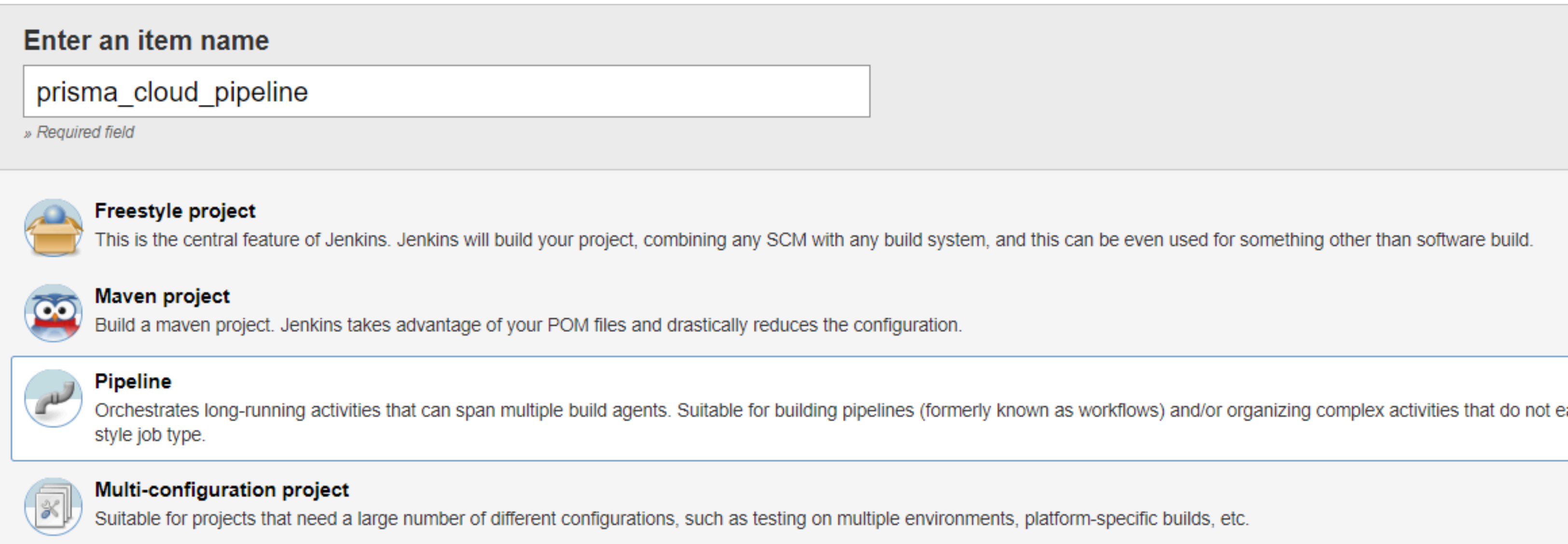
Setting up a Pipeline project for container images

To set up a Jenkins Pipeline:

STEP 1 | Go to the Jenkins top page.

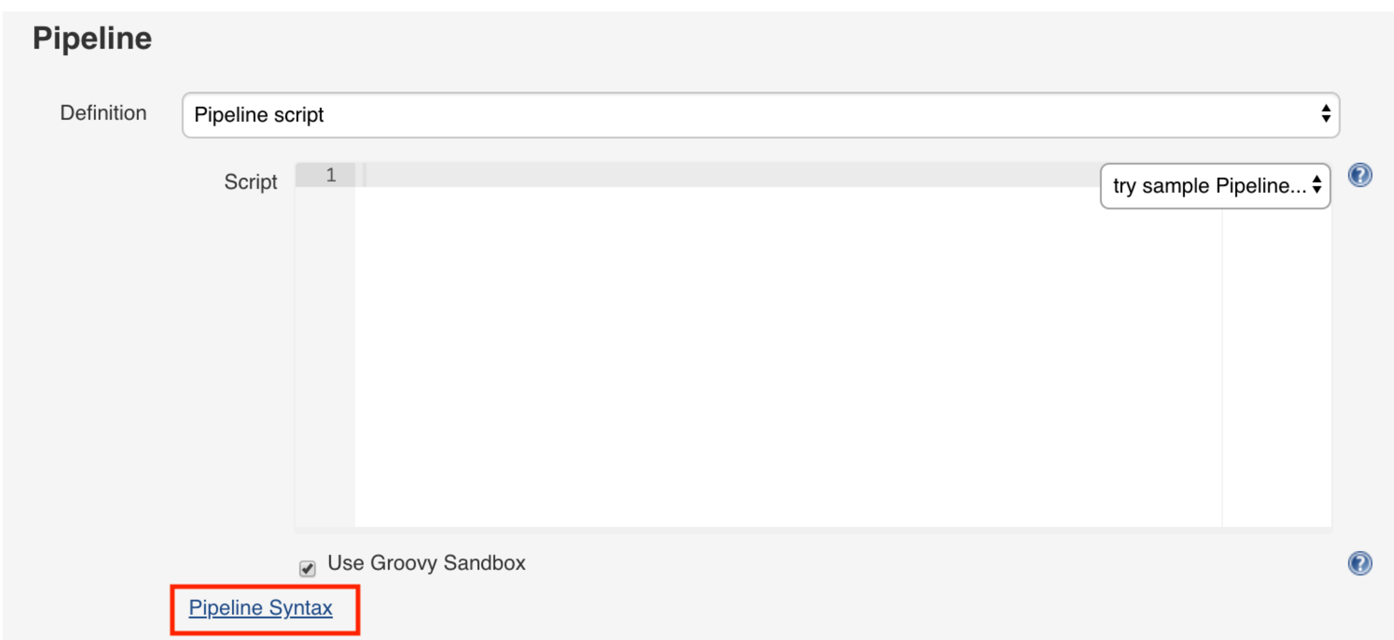
STEP 2 | Create a new pipeline.

1. Click **New Item**.
2. In **Item** name, enter a name for your pipeline.
3. Select **Pipeline**.
4. Click **OK**.



STEP 3 | Use Jenkin's Snippet Generator to generate Pipeline Script for the Prisma Cloud steps.

In the **Pipeline** section, click on the **Pipeline syntax** link, which takes you to *https://<PRISMA_CLOUD_CONSOLE>/job/docs_issue/pipeline-syntax/*.



STEP 4 | Generate Pipeline Script for the scan step.

1. In the **Sample Step** drop-down, select **prismaCloudScanImage - Scan Prisma Cloud Images**.
2. In the **Image** field, select the image to scan by specifying the repository and tag.

   Specify the repository and tag using an exact match or pattern matching expressions. For example, enter *test/test-image**.

   > *If the image you want to scan is created outside of this build, or if you want to scan the image every build, even if the build might not generate an new image, then click* ***Advanced****, and select* **Ignore image creation time**.
3. Click **Generate Pipeline Script**, copy the snippet, and set it aside.

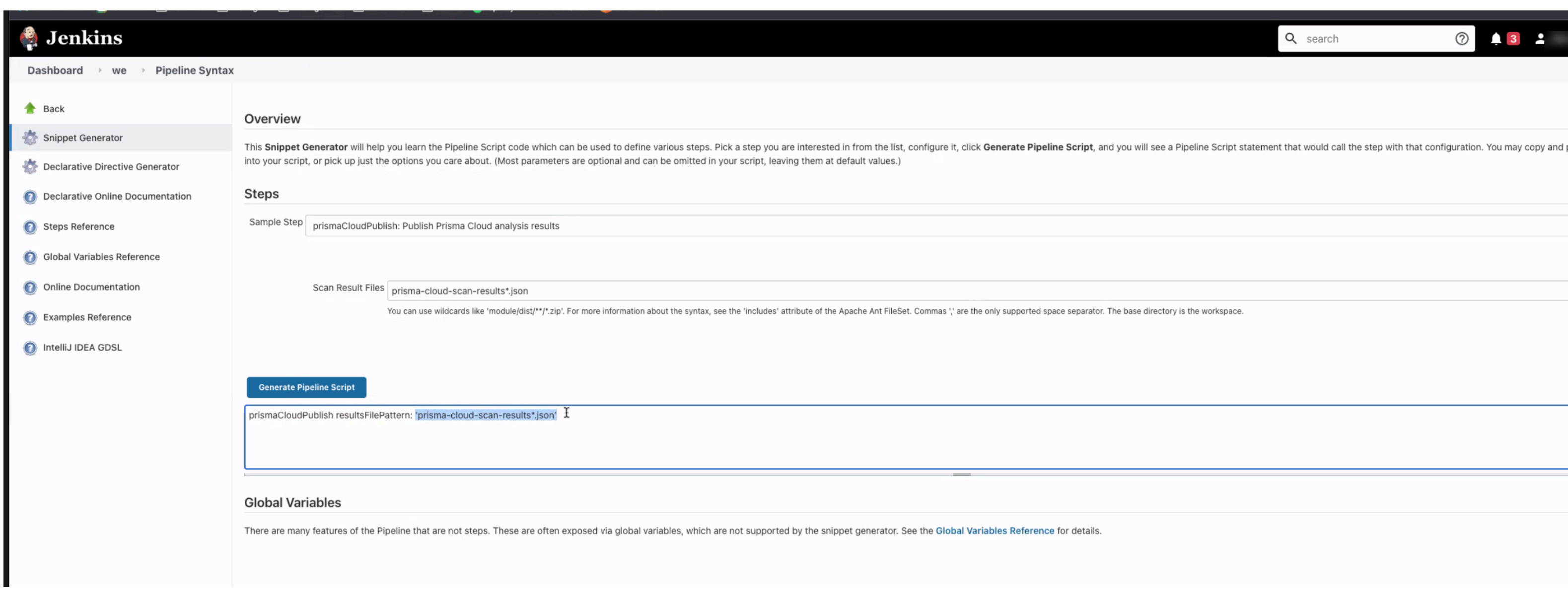
STEP 5 | Generate Pipeline Script to publish the scan results in Jenkins directly.

This post-build step depends on a file generated by the previous scan build step, which holds the scan results. This step specifically makes the results available for review in the Jenkins build tool. Note that the previous scan step already published the results in Console, and they're ready to be reviewed there.

1. In the **Sample Step** drop-down, select **prismaCloudPublish - Publish Prisma Cloud analysis results**.
2. In **Scan Result Files**, review the json filename.

   If you have configured scanning for multiple images and configured unique filenames for each scan in the previous step, you must add a wildcard to the json filename for scan results. For example *prisma-cloud-scan-results*.json*. This ensures that publish command reads all the result files with the same name pattern, and publishes the results so that you can view it. In other cases, accept the default value *prisma-cloud-scan-results.json*.

   Scan result files aren't deleted by the publish step. They stay in the workspace.



3. Click **Generate Pipeline Script**, copy the snippet, and set it aside.

STEP 6 | Return to your project configuration page.

STEP 7 | Paste both snippets into the script section for your project configuration. Use the template below.

The following example template builds a simple image, and runs the scan and publish steps.

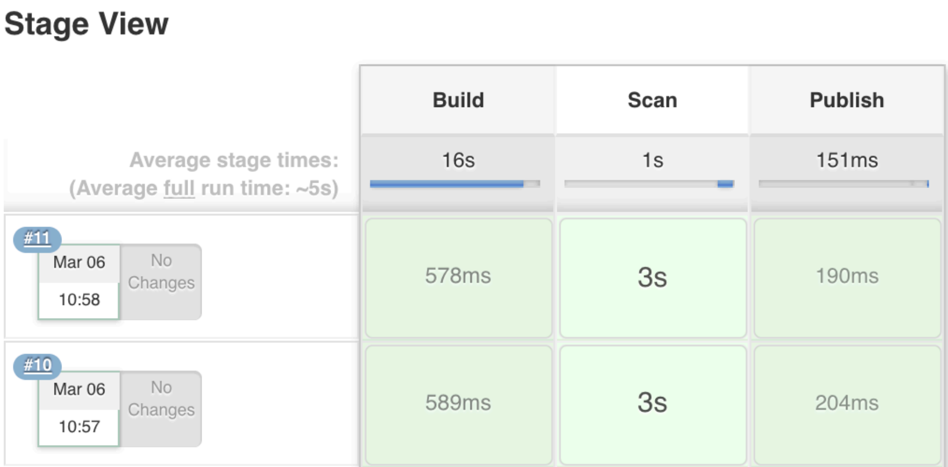
```
pipeline {
    agent any
    stages {
        stage('Build') {
            steps {
                // Build an image for scanning | Input values for
 your image below
                sh 'echo "FROM <registry/repository:tag>
 Dockerfile'
                sh 'docker build --no-cache -t <registry/
repository:tag> .'
            }
        }
        stage('Scan') {
            steps {
                // Scan the image | Input value from first script
 copied below, ''
prismaCloudScanImage - Scan Prisma Cloud Images"
                <PASTE_SCRIPT_HERE>
            }
        }
    }
    post {
        always {
            // The post section lets you run the publish step
 regardless of the scan results | Input value from second script
 copied below, "
prismaCloudPublish - Publish Prisma Cloud analysis results."
            <PASTE_SCRIPT_HERE>
        }
    }
}
```

STEP 8 | Click **Save**.

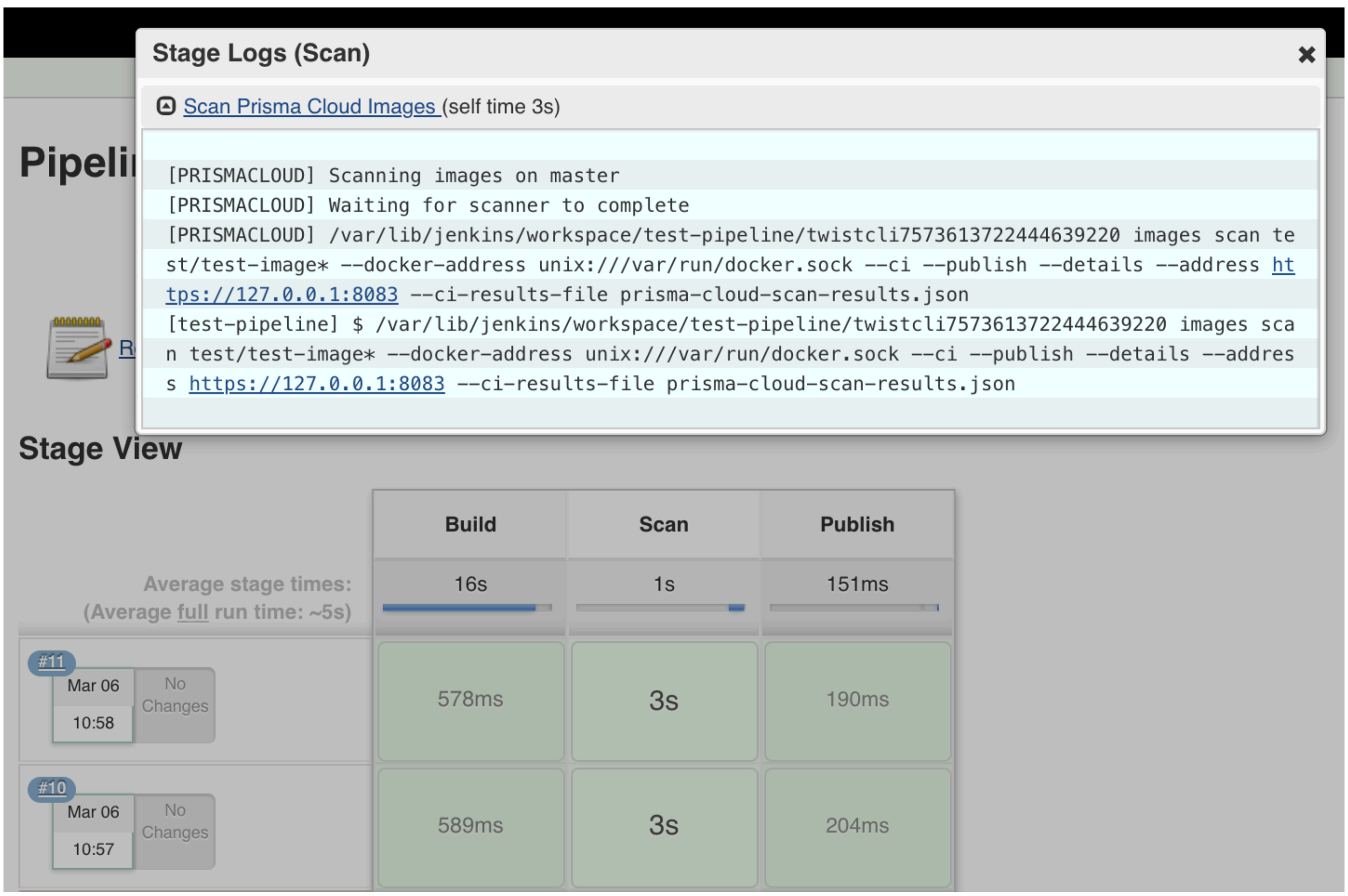
STEP 9 | Click **Build Now**.

STEP 10 | After the build completes, examine the results.

1. The Status page shows a summary of each build step:



2. Click on a step to view the log messages for that step:



3. Scan step returned result:

   The criteria for passing or failing a scan is determined by the CI vulnerability and compliance policies set in Console. The default CI vulnerability policy alerts on all CVEs detected. The default CI compliance policy alerts on all critical and high compliance issues.

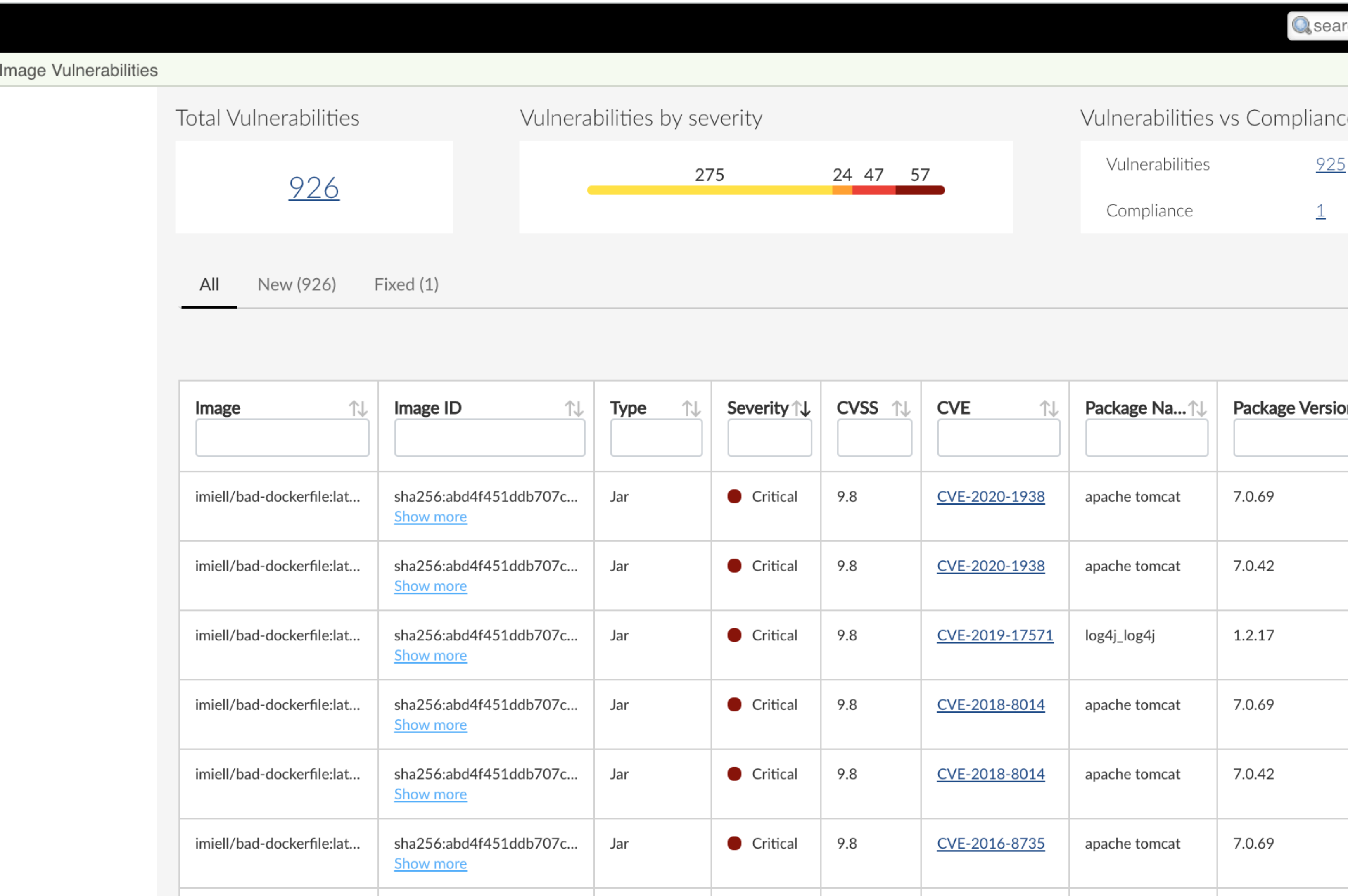
There are two reasons why prismaCloudScanImage scan step might return a failed result.

- *The scan failed because the scanner found issues that violate your CI policy.*
- *Prisma Cloud Compute Jenkins plugin failed to run due to an error.*

In order to understand the reason for the failure, view the step's log messages, or move to the Jenkins Console Output page. Another option that can help you differentiate the reason for the failure could be to create preliminary steps to the scan step in order to check the Console's availability, network connectivity, etc.

Anyhow, although the return value is ambiguous — you cannot determine the exact reason for the failure by just examining the return value — this setup supports automation. From an automation process perspective, you expect that the entire flow will work. If you scan an image, with or without a threshold, either it works or it does not work. If it fails, for whatever reason, you want to fail everything because there is a problem.

4. Scan reports are available in the following locations:
   - Prisma Cloud Console: Log into Console, and go to **Monitor > Vulnerabilities > Images > CI**.
   - Jenkins: Drill down into the build job, then click **Image Vulnerabilities** to see a detailed report.



The **Projects** column in the CI scan results table displays the name of the Jenkins pipeline you created.

Below is the sample code if you'd like to test an image for your Jenkins Pipeline for troubleshooting purposes:

The following example script builds a simple image, and runs the scan and publish steps.

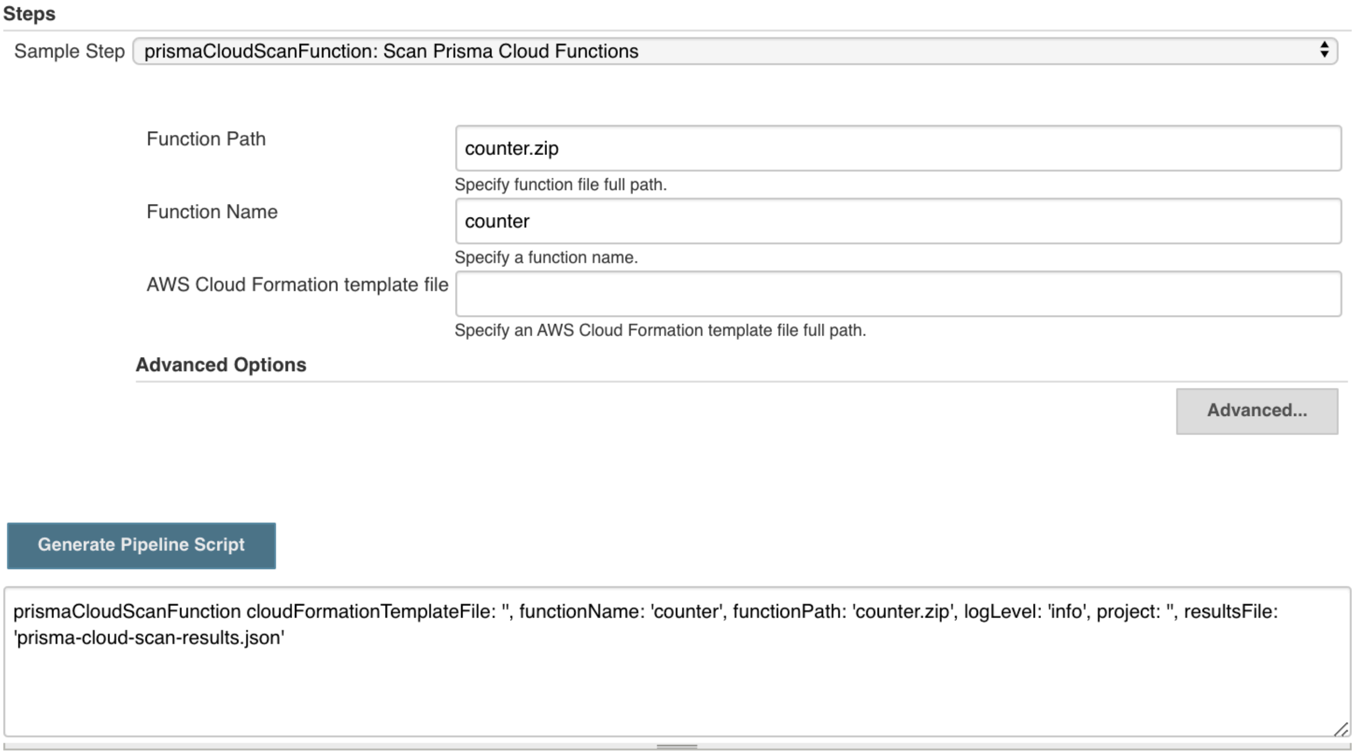
```
pipeline {
    agent any
    stages {
        stage('Build') {
            steps {
                // Build an image for scanning
                sh 'echo "FROM imiell/bad-dockerfile:latest"
 > Dockerfile'
                sh 'docker build --no-cache -t test/test-
image:0.1 .'
            }
```



```
        }
        stage('Scan') {
            steps {
                // Scan the image
                prismaCloudScanImage ca: '',
                cert: '',
                dockerAddress: 'unix:///var/run/
docker.sock',
                image: 'test/test-image*',
                key: '',
                logLevel: 'info',
                podmanPath: '',
                // The project field below is only
 applicable if you are using Prisma Cloud Compute Edition
 and have set up projects (multiple consoles) on Prisma
 Cloud.
                project: '',
                resultsFile: 'prisma-cloud-scan-
results.json',
                ignoreImageBuildTime:true
            }
        }
    }
    post {
        always {
            // The post section lets you run the publish
 step regardless of the scan results
            prismaCloudPublish resultsFilePattern: 'prisma-
cloud-scan-results.json'
        }
    }
}
```

Setting up a Pipeline project for serverless functions

The procedure for setting up Jenkins to scan serverless functions is similar to the procedure for container images, except select **prismaCloudScanFunction: Scan Prisma Cloud Functions** in the snippet generator.



Where:

- **Function Path (functionPath)** — Path to the ZIP archive of the function to scan.
- **Function Name (functionName)** — (Optional) String identifier for matching policy rules in Console with the functions being scanned. When creating policy rules in Console, you can target specific rules to specific functions by function name. If this field is left unspecified, the plugin will use the function zip file name to match against policy.
- **AWS CloudFormation template file (cloudFormationTemplateFile)** — (Optional) Path to CloudFormation template file in either JSON or YAML format. Prisma Cloud scans the function source code for AWS service APIs being used, compares the APIs being used to the function permissions, and reports when functions have permissions for APIs they don't need.

Run Jenkins in a container

Edit on GitHub

Running Jenkins inside a container is a common setup. This article shows you how to set up Jenkins to run in a container so that it can build and scan Docker images.

Setting up and starting a Jenkins container

To set up Jenkins to run in a container:

Prerequisite: You have already installed Docker on the host machine.

STEP 1 | Create the following Dockerfile. It uses the base Jenkins image and sets up the required permissions for the jenkins user.

```
FROM jenkins/jenkins:lts

USER root
RUN apt-get update \
      && apt-get install -y sudo libltdl7 \
      && rm -rf /var/lib/apt/lists/*
RUN echo "jenkins ALL=NOPASSWD: ALL" >> /etc/sudoers
```

STEP 2 | Build the image.

```
$ docker build -t jenkins_docker .
```

STEP 3 | Run the Jenkins container, giving it access to the docker socket.

```
$ docker run -d -v /var/run/docker.sock:/var/run/docker.sock \
  -v $(which docker):/usr/bin/docker -p 8080:8080 jenkins_docker
```

STEP 4 | Open a browser and navigate to <JENKINS_HOST>:8080.

STEP 5 | Install the Prisma Cloud plugin.

For more information, see Jenkins plugin.

Jenkins pipeline on Kubernetes

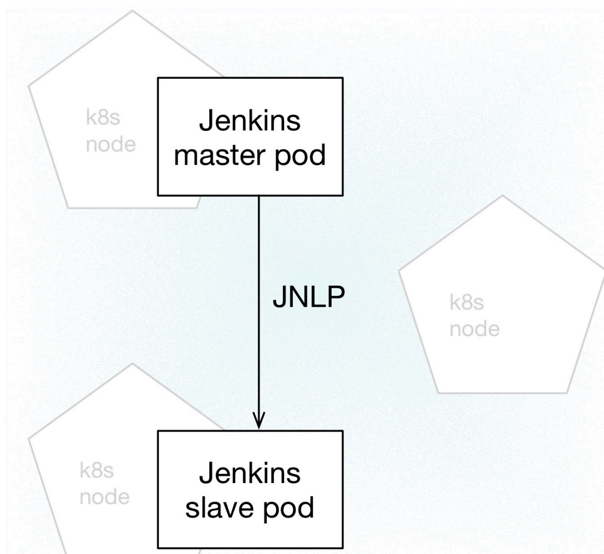
Edit on GitHub

Jenkins is fundamentally architected as a distributed system, with a master that coordinates the builds and agents that do the work. The Kubernetes plugin enables deploying a distributed Jenkins build system to a Kubernetes cluster. Everything required to deploy Jenkins to a Kubernetes cluster is nicely packaged in the Jenkins Helm chart. This article explains how to integrate the Prisma Cloud scanner into a pipeline build running in a Kubernetes cluster.

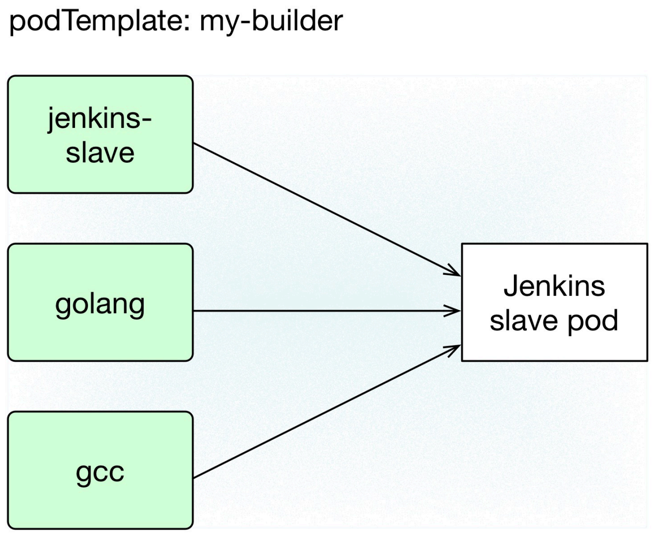
Key concepts

A pipeline is a script that tells Jenkins what to do when your pipeline is run. The Kubernetes Plugin for Jenkins lets you control the creation of the Jenkins slave pod from the pipeline, and add one or more build containers to the slave pod to accommodate build requirements and dependencies.

When the Jenkins master schedules the new build, it creates a new slave pod. Each stage of the build is run in a container in the slave pod. By default, each stage runs in the Jenkins slave (jnlp) container, unless other specified. The following diagram shows a slave pod being launched on a worker node using the Java Network Launch Protocol (JNLP) protocol:



A slave pod is composed of at least one container, which must be the Jenkins jnlp container. Your pipeline defines a podTemplate, which specifies all the containers that make up the Jenkins slave pod. You'll want your podTemplate to include any images that provide the tools required to execute the build. For example, if one part of your app consists of a C library, then your podTemplate should include a container that provides the GCC toolchain, and the build stage for the library should execute within the context of the GCC container.



The Prisma Cloud Jenkins plugin lets you scan images generated in your pipeline.

The Prisma Cloud scanner can run inside the default Jenkins jnlp slave container only. It cannot be run within the context of a different container (i.e. from within the container statement block).

Scripted Pipeline

This section provides a pipeline script that you can use as a starting point for your own script.

You cannot run the Prisma Cloud scanner inside a container. The following example snippet will NOT work.

```
stage('Prisma Cloud Scan') {
  container('jenkins-slave-twistlock') {
    // THIS DOES NOT WORK
    prismaCloudScanImage ca: '', cert: '', ...
  }
}
```

Instead, run the Prisma Cloud scanner in the normal context:

```
stage('Prisma Cloud Scan') {
  // THIS WILL WORK
  prismaCloudScanImage ca: '', cert: '', ...
}
```

Prerequisites:

- You have set up a Kubernetes cluster.
- You have installed Prisma Cloud Console. You can install Prisma Cloud inside or outside of the cluster, as long as any cluster node can reach Console over the network.

- You have installed Jenkins in your cluster. The Jenkins Helm chart is the easiest path for bringing up Jenkins in a Kubernetes cluster.
- Install the Prisma Cloud Jenkins plugin.

Pipeline template

The following template can be used as a starting point for your own scripted pipeline. This template is a fully functional pipeline that pulls the nginx:stable-alpine image from Docker Hub, and then scans it with the Prisma Cloud scanner.

While this example shows how to scan container images, you can also call *prismaCloudScanFunction* to scan your severless functions.

```
#!/usr/bin/groovy

podTemplate(label: 'prismaCloud-example-builder', // See 1
  containers: [
    containerTemplate(
      name: 'jnlp',
      image: 'jenkinsci/jnlp-slave:3.10-1-alpine',
      args: '${computer.jnlpmac} ${computer.name}'
    ),
    containerTemplate(
      name: 'alpine',
      image: 'twistian/alpine:latest',
      command: 'cat',
      ttyEnabled: true
    ),
  ],
  volumes: [ // See 2
    hostPathVolume(mountPath: '/var/run/docker.sock', hostPath: '/
var/run/docker.sock'), // See 3
  ]
)
{
  node ('prismaCloud-example-builder') {

    stage ('Pull image') { // See 4
      container('alpine') {
        sh """
        curl --unix-socket /var/run/docker.sock \ // See 5
             -X POST "http:/v1.24/images/create?
fromImage=nginx:stable-alpine"
        """
      }
    }

    stage ('Prisma Cloud scan') { // See 6
        prismaCloudScanImage ca: '',
                    cert: '',
                    dockerAddress: 'unix:///var/run/docker.sock',
                    image: 'nginx:stable-alpine',
                    resultsFile: 'prisma-cloud-scan-results.xml',
                    project: '',
                    dockerAddress: 'unix:///var/run/docker.sock',
```

```
                     ignoreImageBuildTime: true,
                     key: '',
                     logLevel: 'info',
                     podmanPath: '',
                     project: '',
                     resultsFile: 'prisma-cloud-scan-results.json',
                     ignoreImageBuildTime:true
    }

    stage ('Prisma Cloud publish') {
        prismaCloudPublish resultsFilePattern: 'prisma-cloud-scan-
results.json'
    }
  }
}
```

This template has the following characteristics:

- **1** — This *podTemplate* defines two containers: the required jnlp-slave container and a custom alpine container. The custom alpine container extends the official alpine image by adding the curl package.
- **2** — The docker socket is mounted into all containers in the pod. For more information about the *volumes* field, see Pod and container template configuration.
- **3** — By default, the docker socket lets the root user or any member of the docker group read or write to it. The default user in the jnlp container is *jenkins* The Prisma Cloud plugin functions need access to the docker socket, so you must add the jenkins user to the docker group. The following listing shows the default permissions for the docker socket:

  ```
  $ ls -l /var/run/docker.sock
  srw-rw----  1 root docker   0 May 30 07:58 docker.sock
  ```

- **4** — The first stage of the build pulls down the nginx image. We run the curl command inside the alpine container because the alpine container was specifically built to provide curl. Note that the *prismaCloudScanImage* and *prismaCloudPublish* functions cannot be run inside the *container('<NAME')* block . The must be run in the default jnlp container context.
- **5** — There is a lot of debate about docker-in-docker, especially with respect to CI/CD pipelines. In most cases, docker-in-docker is not required for build pipelines. In this example, we run docker commands using the API exposed by the docker socket. Alternatively, we could use a container with just the Docker client installed.
- **6** — The second stage runs the Prisma Cloud scanner on the nginx image in the default jnlp container.

You can run the Prisma Cloud scanner inside a container using the 'containerized' flag. Scanning from inside a container is only required for special situations.

```
stage(‘Parallel’) {
  agent {
    docker {
      image ‘ubuntu:latest’
    }
  }
  stages {
    stage(‘Prisma Cloud Scan’) {
      steps {
        prismaCloudScanImage ca: '', cert: '',
 containerized:true, ...
      }
    }
    ...
}
```

When using the containerized mode, image ID won’t be displayed in the scan results (only image name).

CI plugin policy

Edit on GitHub

Prisma Cloud lets you centrally define your CI policy in Console in **Defend > Compliance > Containers and images > CI**. These policies establish security gates at build-time. Use policies to pass or fail builds, and surface security issues early during the development process.

There are two types of policies you can use to target your CI tool: vulnerability policies and compliance policies. CI rules have the same parameters as the rules for registries and deployed components, letting you evenly enforce policy in all phases of the app lifecycle.

Prisma Cloud offers the following components for integrating with CI tools:

- A native Jenkins plugin.
- A stand-alone, statically compiled binary, called *twistcli*, that can be integrated with any CI tool.

Vulnerability policy

For more information about the parameters in vulnerability management rules, see here.

Vulnerability rules that target the build tool can allow specific vulnerabilities by creating an exception and setting the effect to 'ignore'. Block them by creating an exception and setting hte effect to 'fail'. For example, you could create a vulnerability rule that explicitly allows CVE-2018-1234 to suppress warnings in the scan results.

Rules take effect as soon as they are saved.

Compliance policy

Prisma Cloud's compliance checks are based on the Center for Internet Security (CIS) Docker Benchmarks. We also provide numerous checks from our lab. You can also implement your own checks using custom checks.

Compliance rules that target the CI tool can permit specific compliance issues by setting the action to 'ignore'.

Rules take effect as soon as they are saved.

Code repo scanning

Edit on GitHub

Both twistcli and the Jenkins plugin can evaluate package dependencies in your code repositories for vulnerabilities.

The runtimes supported are:

- Go
- Java
- Node.js
- Python
- Ruby

Integrate code scanning into CI builds

Point the Jenkins plugin to your code repo in the build directory.

Prerequisites: You've installed and configured the Prisma Cloud Jenkins plugin.

STEP 1 | In your Jenkins job configuration, click **Add build step**, and select **Scan Prisma Cloud Code Repositories**.

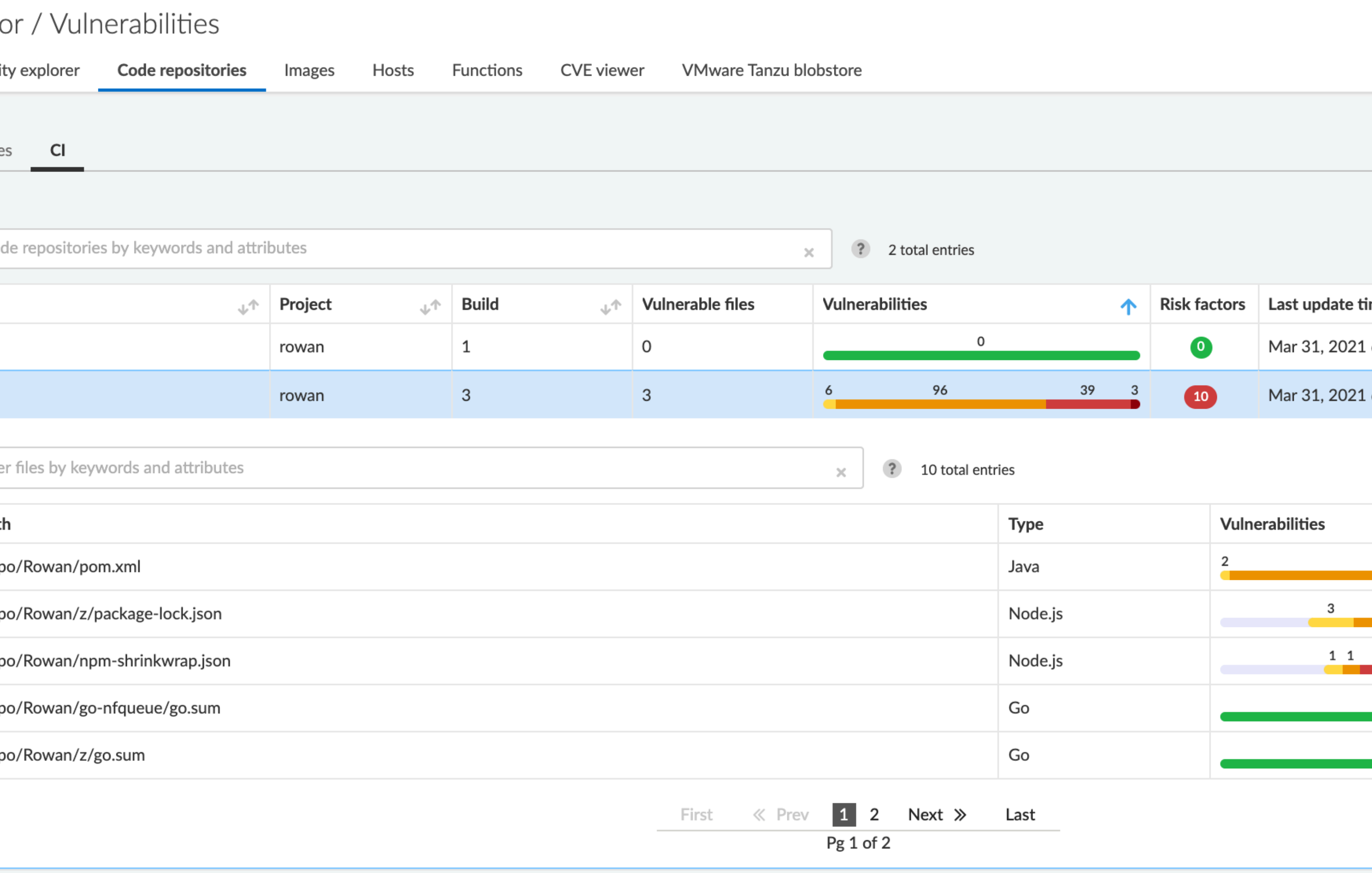
STEP 2 | Configure the repo scan.

1. In **Repository Name**, specify the name to be used when reporting the results in Console.
2. In **Repository path**, specify the path to the repo in the build directory.

   For example, it could simply be the current working directory (.) or some relative directory.

STEP 3 | Click **Save**, and then execute a build job.

To see the scan results, log into Console, and go to **Monitor > Vulnerabilities > Code repositories > CI**. Prisma Cloud evaluates the contents of the repo according to the policy you've specified in **Defend Vulnerabilities > Code repositories > CI**. Prisma Cloud ships with a single default rule that alerts on all vulnerabilities.



Use twistcli to scan repos in the CI

If you're using a CI tool other than Jenkins, Prisma Cloud ships a command line utility that can be invoked from the shell in the build pipeline.

For more information, see code repo scanning with twistcli.

WAAS

Edit on GitHub

WAAS (Web-Application and API Security, formerly known as CNAF, Cloud Native Application Firewall) is a web application firewall (WAF) designed for HTTP-based web applications deployed directly on hosts, as containers, application embedded or serverless functions. WAFs secure web applications by inspecting and filtering layer 7 traffic to and from the application.

- Web-Application and API Security (WAAS)
- Deploy WAAS
- WAAS Explorer
- App Firewall Settings
- API Protection
- DoS protection
- Bot Protection
- WAAS Access Controls
- Advanced Settings
- WAAS Analytics
- API observations
- API definition scan
- WAAS custom rules
- Detecting unprotected web apps
- WAAS Log Scrubbing

Web-Application and API Security (WAAS)

Edit on GitHub

WAAS (Web-Application and API Security, formerly known as CNAF, Cloud Native Application Firewall) is a web application firewall (WAF) designed for HTTP-based web applications deployed directly on hosts, as containers, application embedded or serverless functions. WAFs secure web applications by inspecting and filtering layer 7 traffic to and from the application.

WAAS enhances the traditional WAF protection model by deploying closer to the application, easily scaling up or down, and allowing for inspection of "internal" traffic (east-to-west) from other microservices as well as inbound traffic (north-to-south).

- For containerized web applications, WAAS binds to the application's running containers, regardless of the cloud, orchestrator, node, or IP address where it runs, and without the need to configure any complicated routing.
- For non-containerized web applications, WAAS simply binds to the host where the application runs.
- WAAS monitors the remote applications by monitoring the mirrored traffic generated from the network interfaces attached to your instances.

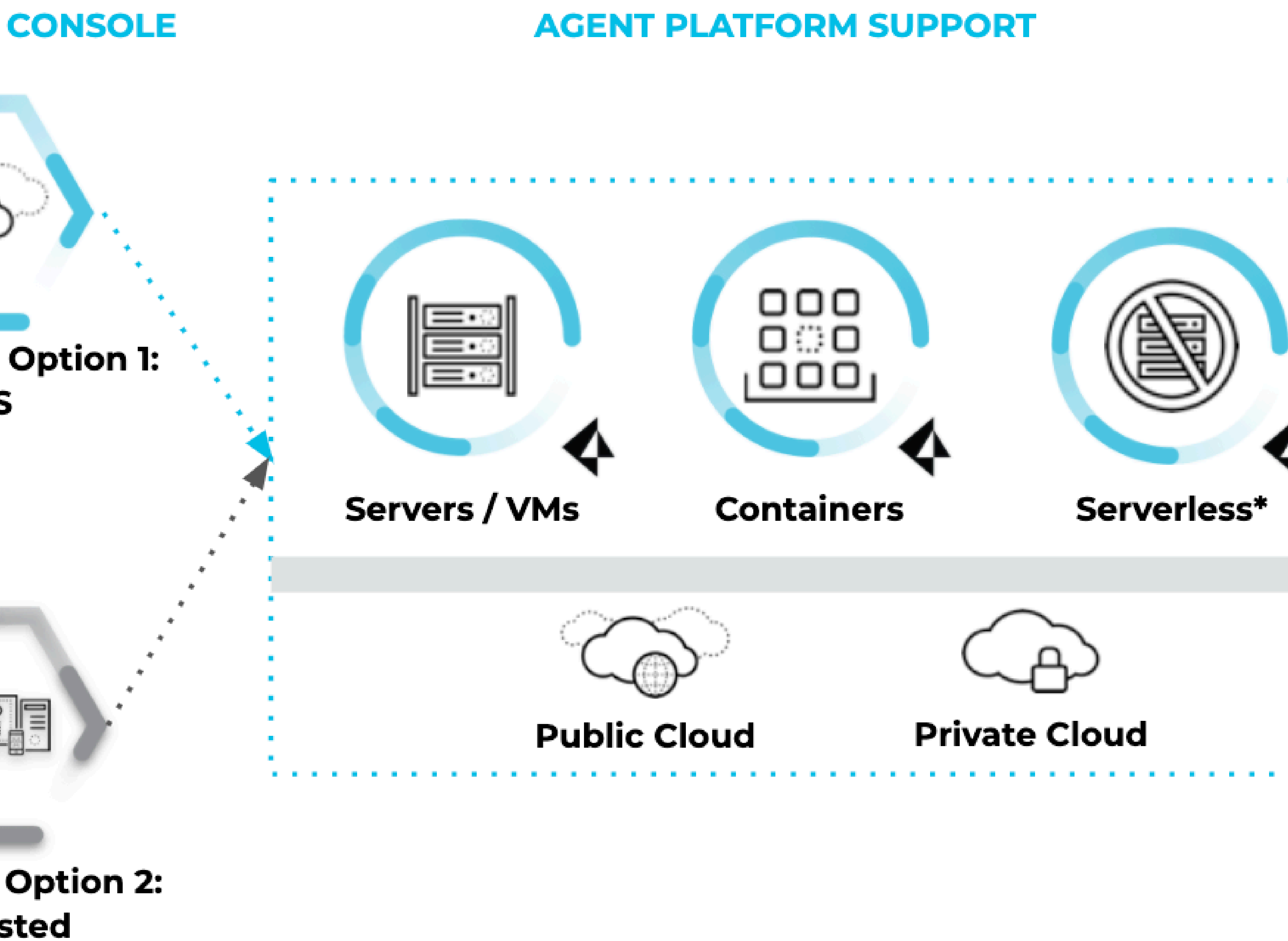
Highlights of WAAS's capabilities:

- **OWASP Top-10 Coverage** - Protection against most critical security risks to web applications, including injection flaws, broken authentication, broken access control, security misconfigurations, etc.
- **API Protection** - WAAS can enforce API traffic security based on definitions/specs provided in the form of Swagger or OpenAPI files.
- **Access Control** - WAAS controls access to protected applications using Geo-based, IP-based, or HTTP Header-based user-defined restrictions.
- **File Upload Control** - WAAS secures application file uploads by enforcing file extension rules.
- **Detection of Unprotected Web Applications** - WAAS detects unprotected web applications and flags them in the radar view.
- **Penalty Box for Attackers** - WAAS supports a 5 minutes ban of IPs triggering one of its protections to slow down vulnerability scanners and other attackers probing the application.
- **Bot Protection** - WAAS detects good-known bots and other bots, headless browsers, and automation frameworks. WAAS is also able to fend off cookie droppers and other primitive clients by mandating the use of cookies and Javascript for the client to reach the protected origin.
- **DoS Protection** - WAAS is able to enforce rate limitation on IPs or Prisma Sessions to protect against high-rate and "low and slow" layer-7 DoS attacks.

How to deploy WAAS?

WAAS is deployed with Prisma Compute Defenders. The Defenders can operate as a transparent HTTP proxy as well as monitor the traffic from an Out-of-band network for the remote applications that do not have any Defenders installed.

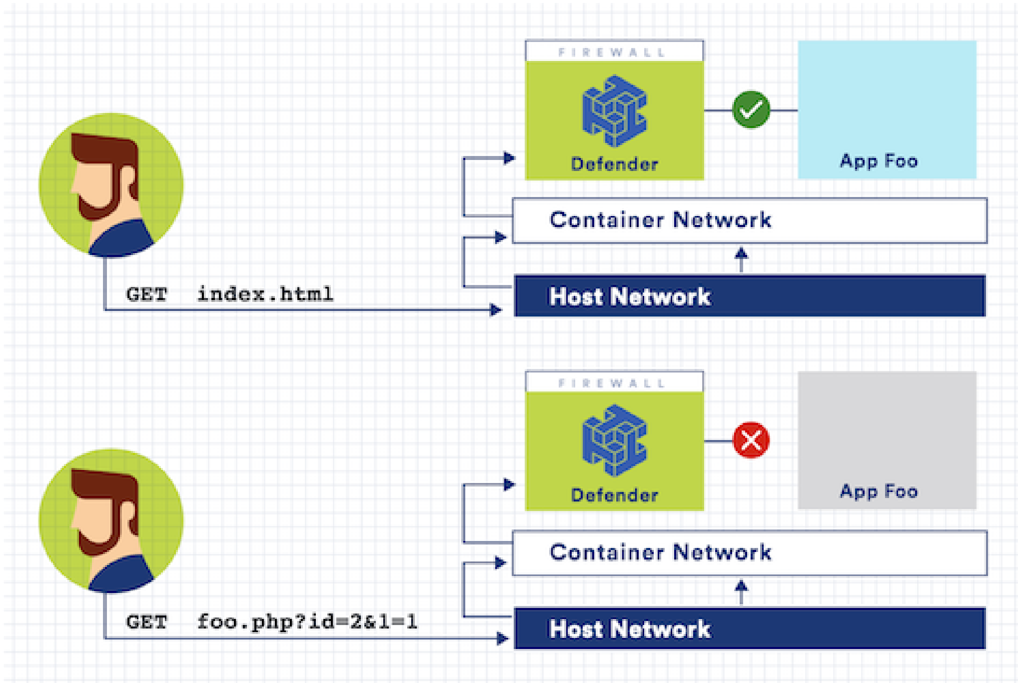
The Inline Defenders evaluate client requests against security policies before relaying the requests to your application. The Out-of-band Defenders only send out alerts from the read-only copy of the network traffic.



Defenders are deployed into the environment in which the web applications run, and you can view the data on the Prisma Cloud management console.

How does WAAS work?

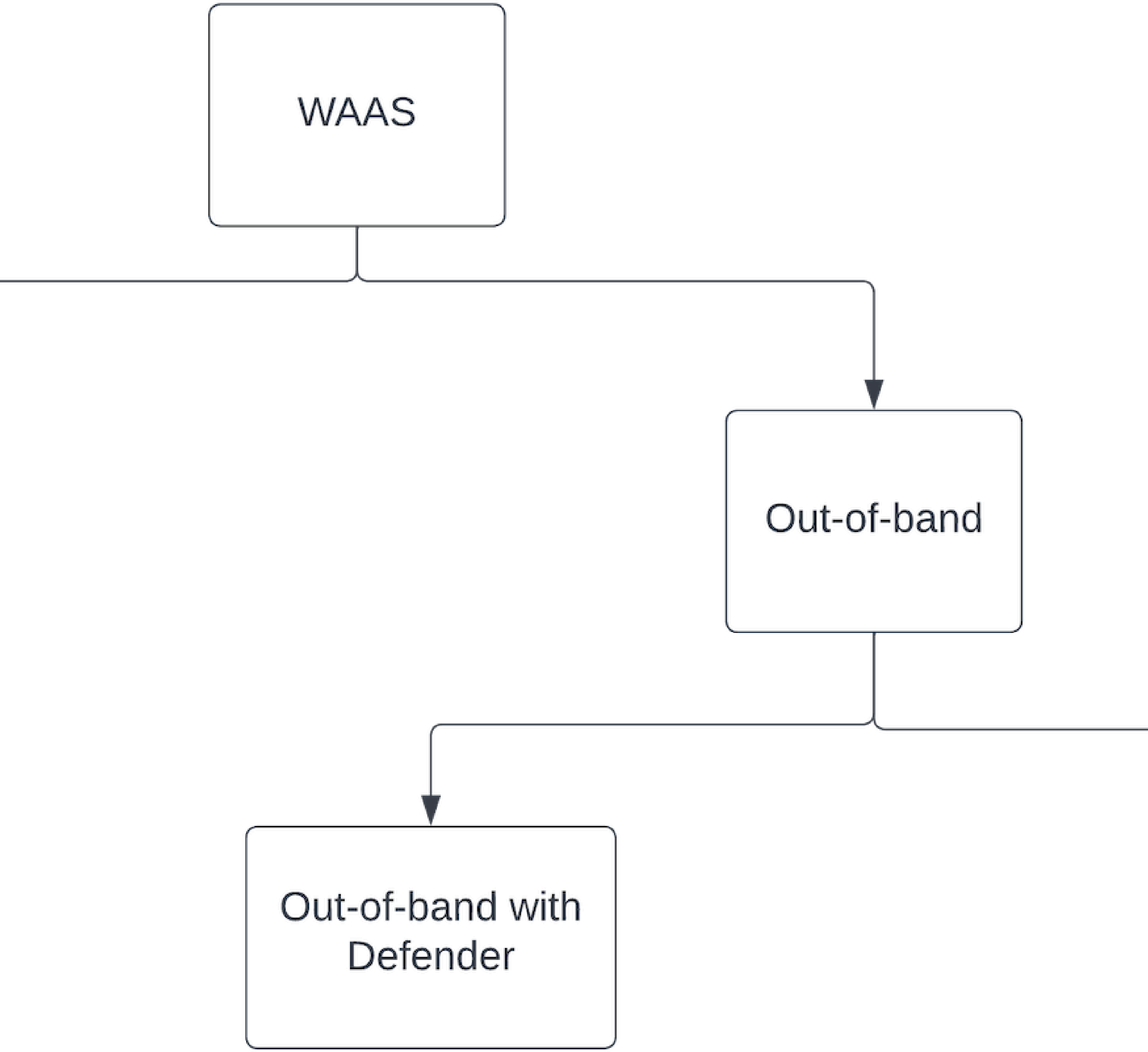
WAAS inspects the incoming and outgoing traffic to and from your application for discovery, monitoring, and protection. Once you deploy WAAS you get visibility into your attack surfaces such as the host, containers, and serverless functions. WAAS API observations list the endpoints of the APIs and the methods used by the APIs for communication. The WAAS discovery and API observations help in risk assessment and placing policies to protect your workflow.



Requests triggering one or more WAAS protections generate a WAAS "event audit" and action is taken based on the preconfigured action (see "WAAS Actions" below). WAAS's event audits can be further explored in the "Monitor" section of Prisma Compute's management console (**Monitor > Events**). In addition, event audits are registered in the Defender's syslog thus allowing for integration with third-party analytics engines or SIEM platforms of choice.

How does WAAS inspection work on Prisma Cloud?

WAAS can inspect the traffic as an Inline proxy as well as an Out-of-band network.


WAAS
Out-of-band
Out-of-band with
Defender

WAAS Inline proxy

WAAS inspects all incoming requests and forwards them to the protected application if there are no malicious activities. The response from the application is in turn inspected by WAAS and sent to the user if it's not violating any rules.

This way, the Inline proxy is more secure as it can stop the incoming and outgoing traffic flow, but it consumes high resources, and may also result in application outage. The inline proxy needs a Defender to be deployed in the environment.

WAAS Out-of-band

Out-of-band monitors both protected and unprotected workloads by inspecting the mirrored traffic. WAAS Out-of-band doesn't interfere with client-server communications, nor does it impact the application performance.

WAAS v can be deployed with Defender or with CSP traffic mirroring.

1. **WAAS Out-of-band with Defender** needs a Defender to be deployed in your workload environment to monitor the protected applications by using Out-of-band network communication.
2. **WAAS Out-of-band with VPC traffic mirroring** is used in cases where it's not possible to install Defender for each microservice. VPC traffic mirroring extends WAAS monitoring to instances regardless of whether they have Defenders deployed or not.

   This setup requires you to install a Defender on the target instance outside your workload environment, to remotely monitor the unprotected applications on your source instance by using the in-built traffic mirroring provided by CSP.

   For example, AWS VPC traffic mirroring feature copies the traffic from the source EC2 instance (with no Defender) to the target EC2 instance that has a host Defender installed within the same VPC.

WAAS Out-of-band setup has no latency cost. But as WAAS can't control the traffic, it can only send out alerts to the Prisma Console.

Where do I begin with WAAS?

WAAS is enabled by adding a new WAAS rule. Whenever new policies are created, or existing policies are updated, Prisma Cloud immediately pushes them to all the resources to which they apply.

To deploy WAAS, create a new WAAS rule, select the resources on which to apply the rule, define your web application and select the protections to enable. For containerized web applications, Prisma Cloud creates a firewall instance for each container instance. For legacy (non-containerized web applications), Prisma Cloud creates a firewall for each host specified in the configuration.

Prisma Cloud can also protect Fargate-based web containers. +See WAAS for Fargate.

WAAS Actions

Requests that trigger a WAAS protection are subject to one of the following actions:

- **Alert** - The request is passed to the protected application (where, the deployed Defender has complete visibility on your workload) or unprotected application (where, there is no Defender deployed on the workload instance but on a remote instance, for example, in v with VPC mirroring), and an audit is generated for visibility.

  Both In-line and Out-of-band WAAS deployment generate alerts to the Console.
- **Prevent** - The request is denied from reaching the protected application, an audit is generated and WAAS responds with an HTML page indicating the request was blocked.

  Supported only in WAAS Inline proxy setup.
- **Ban** - Can be applied on either IP or Prisma Session IDs. All requests originating from the same IP/Prisma Session to the protected application are denied for the configured time period (default is 5 minutes) following the last detected attack.

  Supported only in WAAS Inline proxy setup.

  WAAS implements state, which is required for banning user sessions by IP address. Because Defenders do not share state, any application replicated across multiple nodes must enable IP stickiness on the load balancer.
- **Disable** - The WAAS action is disabled.

  Supported for both WAAS Inline and WAAS Out-of-band setups.

Supported Protocols, Message Parsers, and Decoders

Supported Protocols

- HTTP 1.0, 1.1, 2.0 - full support of all HTTP methods
- TLS 1.0, 1.1, 1.2, 1.3 for WAAS In-line only (not supported for WAAS Out-of-band)
- gRPC
- WebSockets Passthrough

Supported Message Parsers, and Decoders

- GZip, deflate content encoding
- HTTP Multipart content type
- URL Query, x-www-form-urlencoded, JSON and XML parameter parsing
- URL, HTML Entity, JS, BASE64 decoding
- Overlong UTF-8

Deploy WAAS

Edit on GitHub

WAAS (Web-Application and API Security) can secure both containerized and non-containerized web applications. To deploy WAAS, create a new rule, and declare the entity to protect.

Although the deployment method varies slightly depending on the type of entity you're protecting, the steps, in general, are:

1. Define rule resource.
2. Define application scope.
3. Enable relevant protections.

Understanding WAAS rule resources and application scope

The WAAS rule engine is designed to let you tailor the best-suited protection for each part of your deployment. Each rule has two scopes:

- Rule resources.
- Application list.

Rule Resources

This scope defines, for each type of deployment, a combination of one or more elements to which WAAS should attach itself to protect the web application:

- ***For containerized applications*** - Containers, images, namespaces, cloud account IDs, hosts.
- ***For non-containerized applications*** - Host on which the application is running.
- ***For containers protected with App-Embedded Defender*** - App ID.
- ***For serverless functions*** - Function name.

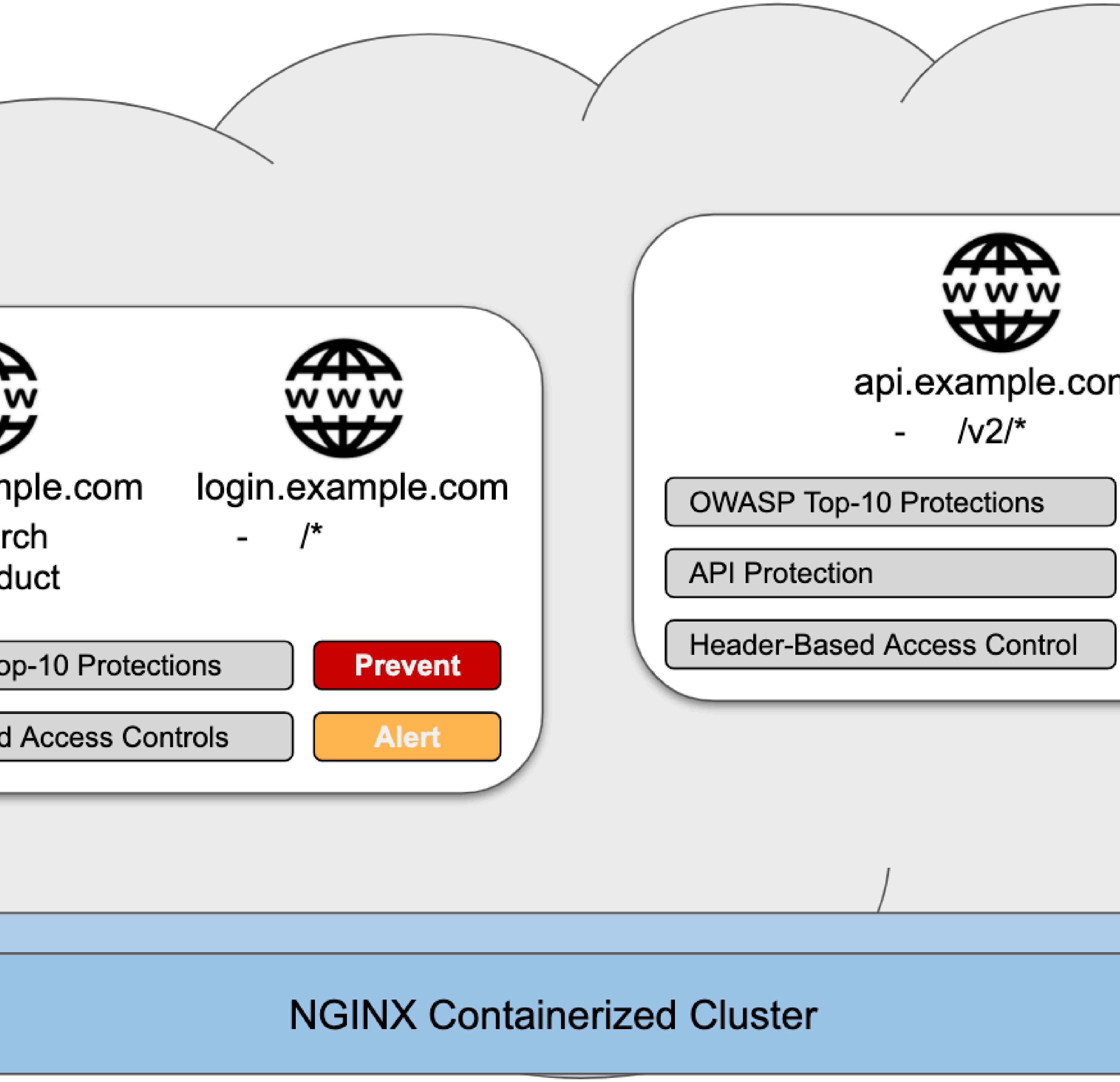
In the event of scope overlap (when multiple rules are applied to the same resource scope), the first rule by order will apply and all others will not apply. You can reorder rules via the Order column in WAAS rule tables by dragging and dropping rules.

Application List

This scope defines the protected application's endpoints within the deployment as a combination of one or more of the following:

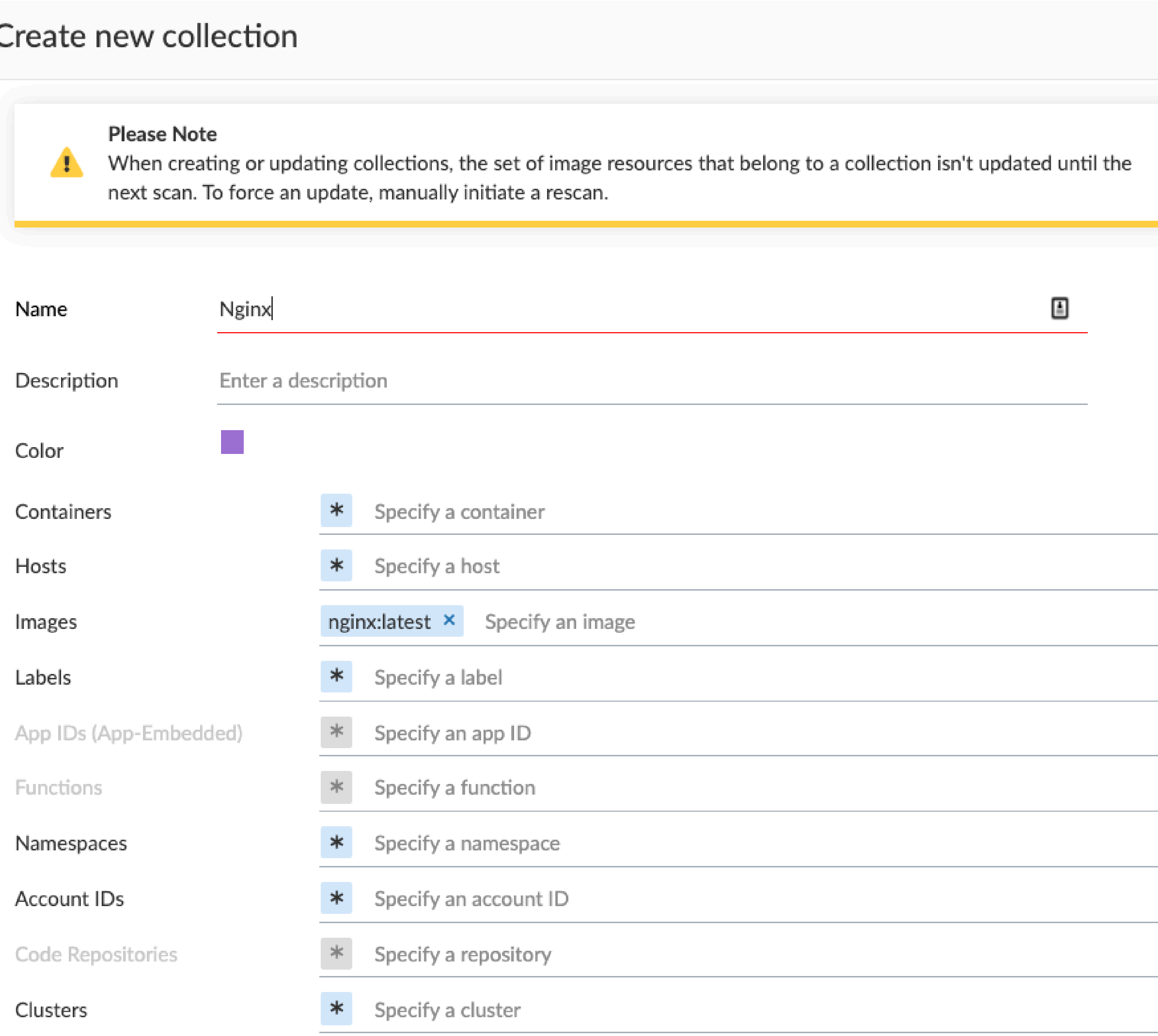
- ***Port (Required)*** - For containerized applications, the internal port on which the application is listening. For all other types, the externally facing port.
- ***HTTP hostname*** - The default setting is set to * (wildcard indicating all hostnames)
- ***Base path*** - Lets you apply protection policy on certain paths of the application (e.g. "/admin", "/admin/*", etc.)
- ***TLS*** - TLS certificate to be used when expecting encrypted inbound traffic.

To better illustrate, consider the following deployment scenario for a web application running on top of an NGINX cluster:



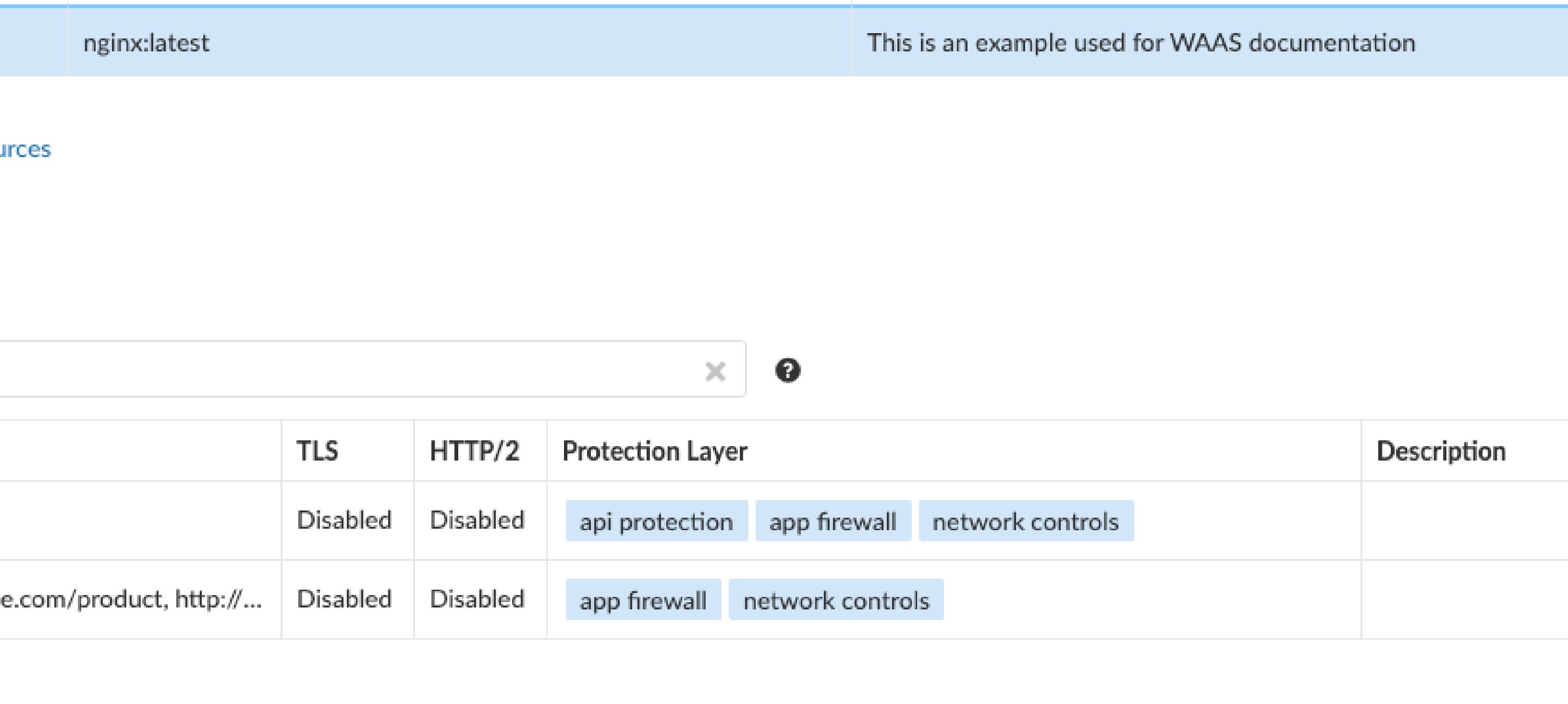
In this example, different policies apply for different parts of the application. The steps for deploying a WAAS rule to protect the above-described web application would be as follows:

1. **Define rule resources** - Specify the resource collection the rule applies to. Collections are comprised of image names and one or more elements to which WAAS should attach itself to protect the web application. In the following example, the rule will apply to all containers created by the Nginx image.



2. **Define protection policy for 'login', 'search', and 'product' endpoints** - Set OWASP Top 10 protection to "Prevent" and geo-based access control to "Alert".
3. **Define protection policy for the application's API endpoints** - Set OWASP Top 10 and API protection to "Prevent" and HTTP header-based access control to "Alert".

Once the policy is defined, the rule overview shows the following rule resource and application definitions:



- ***Rule Resources*** - Protection is applied to all NGINX images
- ***Apps List*** - We deployed two policies each covering a different endpoint in the application (defined by HTTP hostname, port, and path combinations).

Protection evaluation flow

WAAS offers a range of protection targeted at different attack vectors. Requests inspected by WAAS will be inspected in the following order of protection:

- Bot protection
- App firewall (OWASP Top-10)
- API protection
- DoS protection

WAAS Inline proxy will continue to inspect a request until "Prevent" or "Ban" actions are triggered, at which point the request will be blocked, and the evaluation flow will be halted. In the case of WAAS Out-of-band, the requests will be inspected and alerts will be sent to the Console.

For example, in the WAAS Inline proxy setup, assume all protections in bot protection are set to "Prevent". An incoming request originating from a bot and containing a SQL injection payload would be blocked by the bot protection (since it precedes the app firewall in the evaluation flow), and the SQL injection payload will not be assessed by the app firewall.

In a different scenario, suppose that all bot protections are set to "Alert" and all app firewall protections are set to "Prevent". A request originating from a bot containing a command injection payload will generate an alert event by bot protection and will be blocked by the app firewall protection.

Recommended WAAS Deployment Phases

It is recommended that WAAS is first deployed in non-production environments, and then promoted and implemented in production environments gradually. Below are the guidelines for each of the recommended phases and their prerequisites.

Prerequisites:

- A way to test the application before deploying WAAS and verify that it's working properly, e.g., a working cURL command with the expected outcome.
- A certificate (public certificate and private key files in PEM format) is required if the application employs TLS.
- If you are planning to protect API endpoints, please provide API specification files if available (Swagger or OpenAPI 3)

STEP 1 | Deploy WAAS in a test environment (preferably one that is as similar to production as possible).

All protections will be set to "Alert".

STEP 2 | Allow WAAS to inspect traffic to the test environment for a few days, then regroup to examine triggers and findings. It is recommended to generate traffic to the test environment preferably requests that simulate real user messages.

The goal here is to fine-tune protections so that they correspond with the design of the protected application.

This would also be a good way to assess the performance impact introduced by WAAS and compare it to the performance of the application before the deployment of WAAS.

STEP 3 | Following the successful completion of phases 1 and 2, deploy WAAS on a small portion of production with the same configuration that you tested in the test environment.

Next, examine the findings after a few days and make any necessary adjustments to the policies.

STEP 4 | Deploy WAAS across the entire production deployment of the application.

Deploy WAAS for Containers

Edit on GitHub

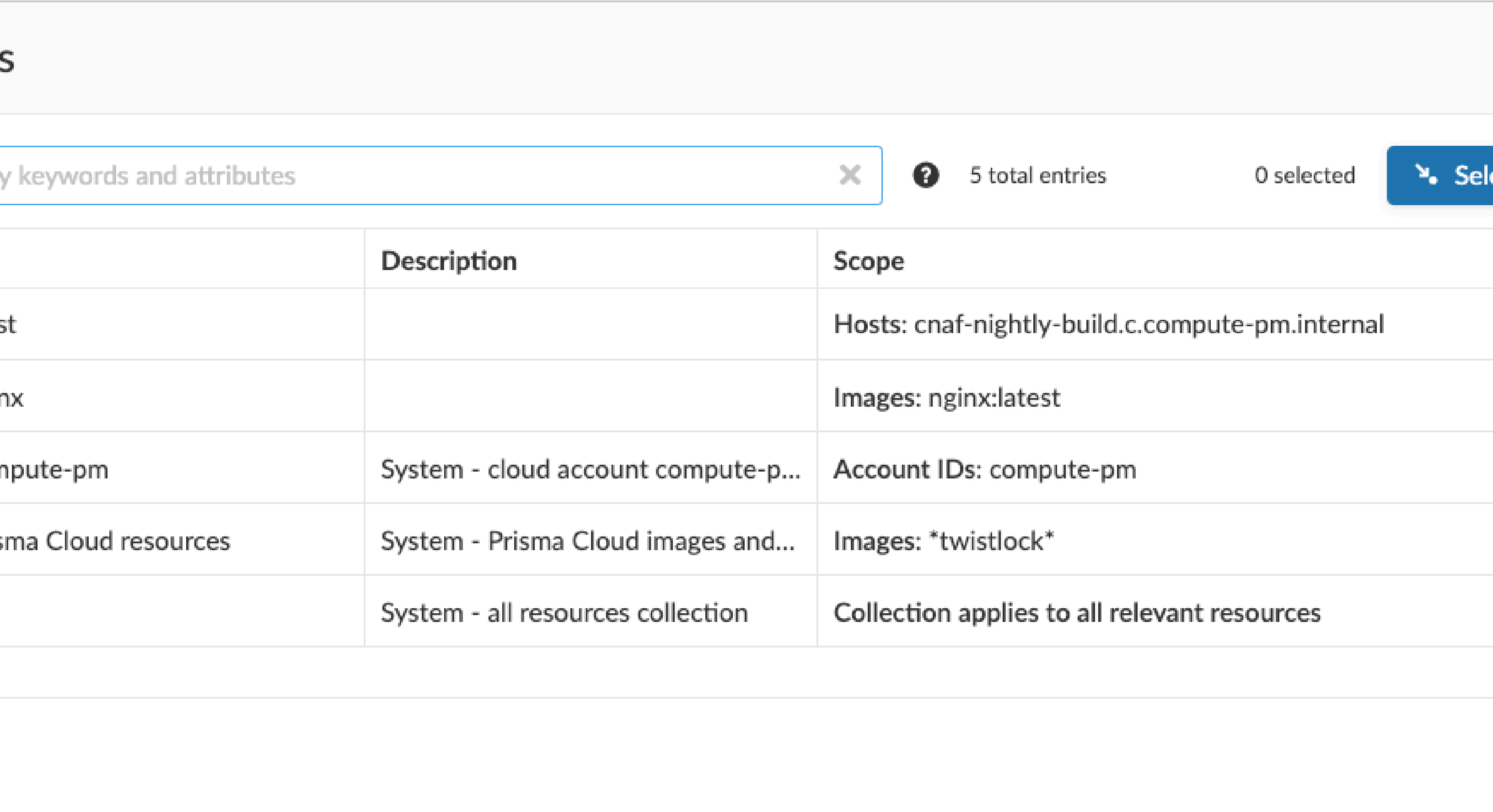
Create a WAAS rule for Containers

STEP 1 | Open Console, and go to **Defend > WAAS > *Container**.

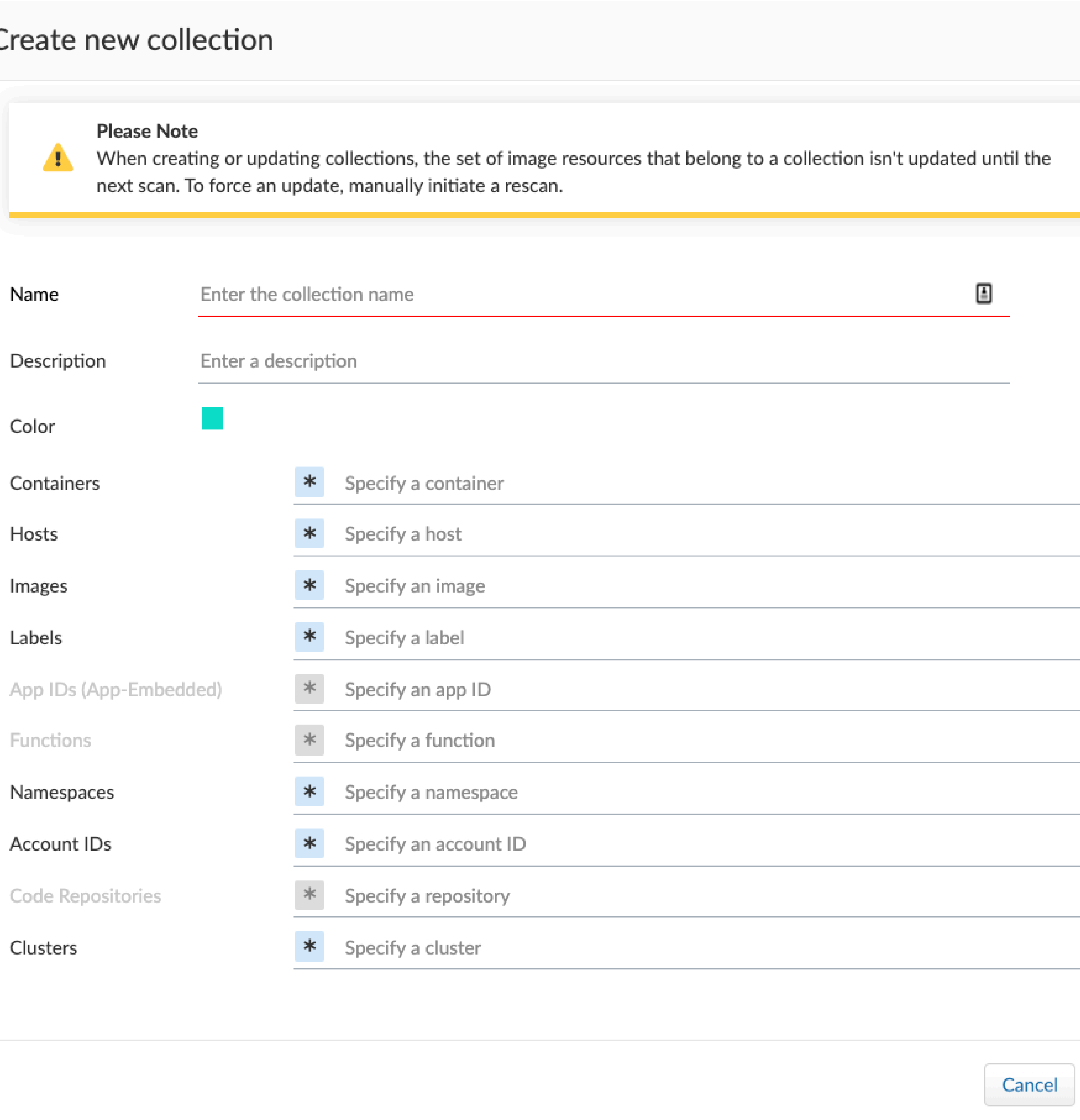
STEP 2 | Click **Add Rule**.

STEP 3 | Enter a **Rule Name** and **Notes** (Optional) for describing the rule.

STEP 4 | Choose the rule **Scope** by specifying the resource collection(s) to which it applies.



Collections define a combination of image names and one or more elements to which WAAS should attach itself to protect the web application:


Create new collection
Please Note
When creating or updating collections, the set of image resources that belong to a collection isn't updated until the next scan. To force an update, manually initiate a rescan.
Name
Enter the collection name
Description
Enter a description
Color
Containers
Specify a container
Hosts
Specify a host
Images
Specify an image
Labels
Specify a label
App IDs (App-Embedded)
Specify an app ID
Functions
Specify a function
Namespaces
Specify a namespace
Account IDs
Specify an account ID
Code Repositories
Specify a repository
Clusters
Specify a cluster
Cancel

Applying a rule to all images using a wild card (*) is invalid - instead, only specify your web application images.

STEP 5 | (Optional) Enable **Automatically detect ports** for an endpoint to protect the ports identified in the unprotected web apps report **Monitor > WAAS > Unprotected web apps** for each of the workloads in the rule scope.

As an additional measure, you can specify additional ports by specifying them in the protected HTTP endpoints within each app to also include the ports that may not have been detected automatically.

STEP 6 | (Optional) Enable **API endpoint discovery**.

When enabled, the Defender inspects the API traffic to and from the protected API. Defender reports a list of the endpoints and their resource path in Compute > Monitor > WAAS > API observations > Out-of-band observations.

*By enabling both **Automatically detect ports** and **API endpoint discovery**, you can monitor your API endpoints and ports without having to add an application and without configuring any policies.*

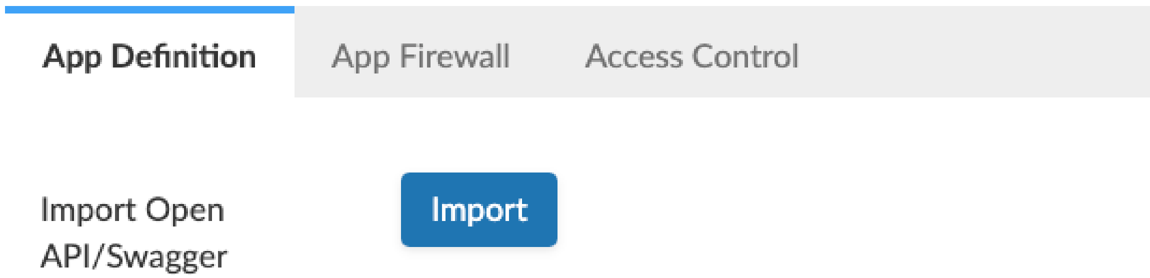
STEP 7 | **Save** the rule.

Add an App (policy) to the rule

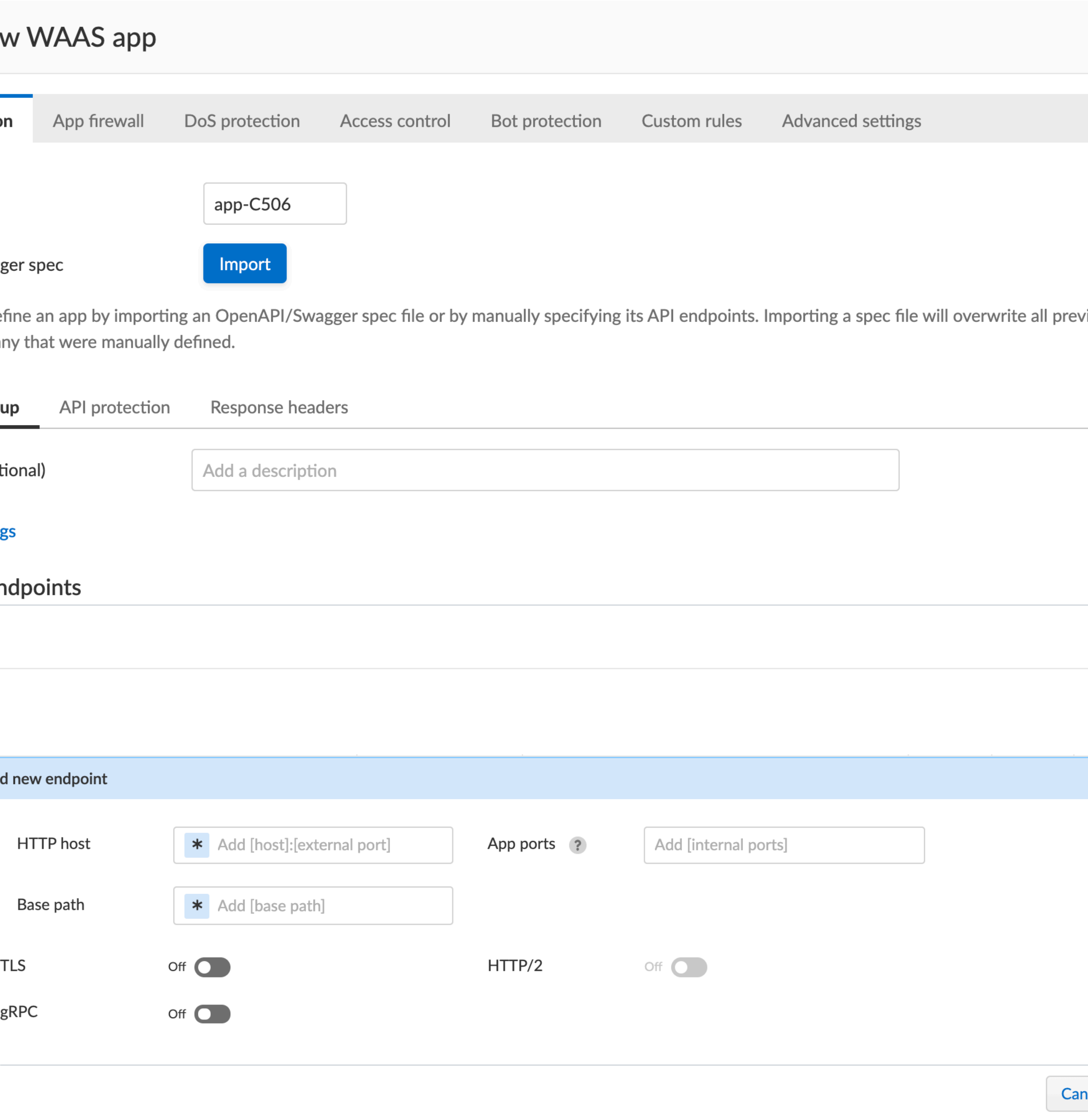
STEP 1 | Select a WAAS container rule to add an App in.

1. Click **Add app**.
2. In **App Definition**, specify the endpoints in your web application that should be protected.

   Each defined application can have multiple protected endpoints. If you have a Swagger or OpenAPI file, click **Import**, and select the file to load. Otherwise, skip to the next step to manually define your application's endpoints.

   If you do not have a Swagger or OpenAPI file, manually define each endpoint by specifying the host, port, and path.
3. In the **Endpoint setup** tab, click **Add Endpoint**.



- Enter **HTTP host** (optional, wildcards supported).

HTTP host names are specified in the form of [hostname]:[external port].

External port is defined as the TCP port on the host, listening for inbound HTTP traffic. If the the value of the external port is "80" for non-TLS endpoints or "443" for TLS endpoints it can be omitted. Examples: "*.example.site", "docs.example.site", "www.example.site:8080", etc.

- Enter **App ports** (optional, if you selected **Automatically detect ports** while creating the rule).

  When **Automatically detect ports** is selected, any ports specified in a protected endpoint definition will be appended to the list of protected ports.
- Specify the TCP port listening for inbound HTTP traffic.

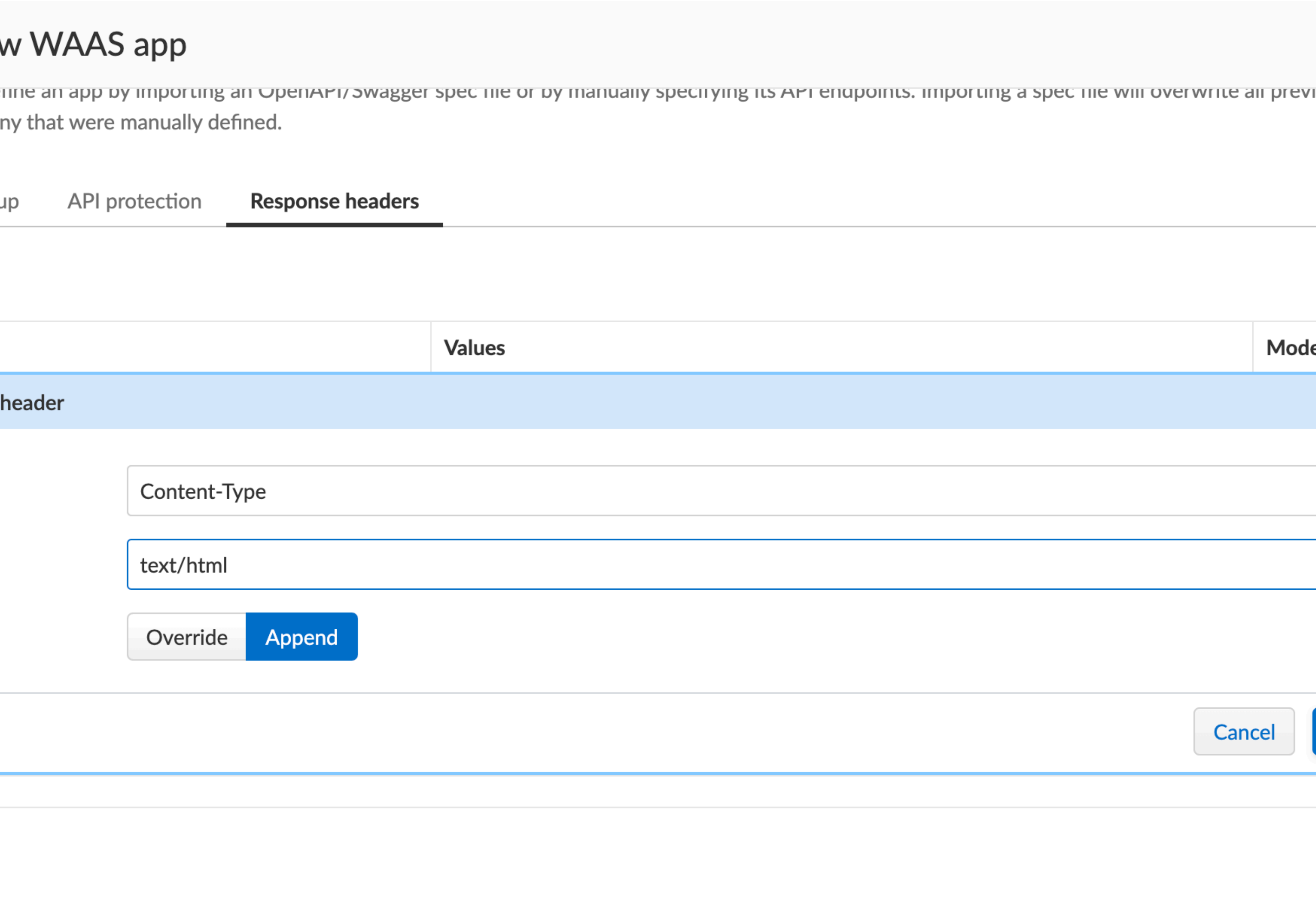
  *If your application uses **TLS** or **gRPC**, you must specify a port number.*
- Enter **Base path** (optional, wildcards supported):

  Base path for WAAS to match on, when applying protections.

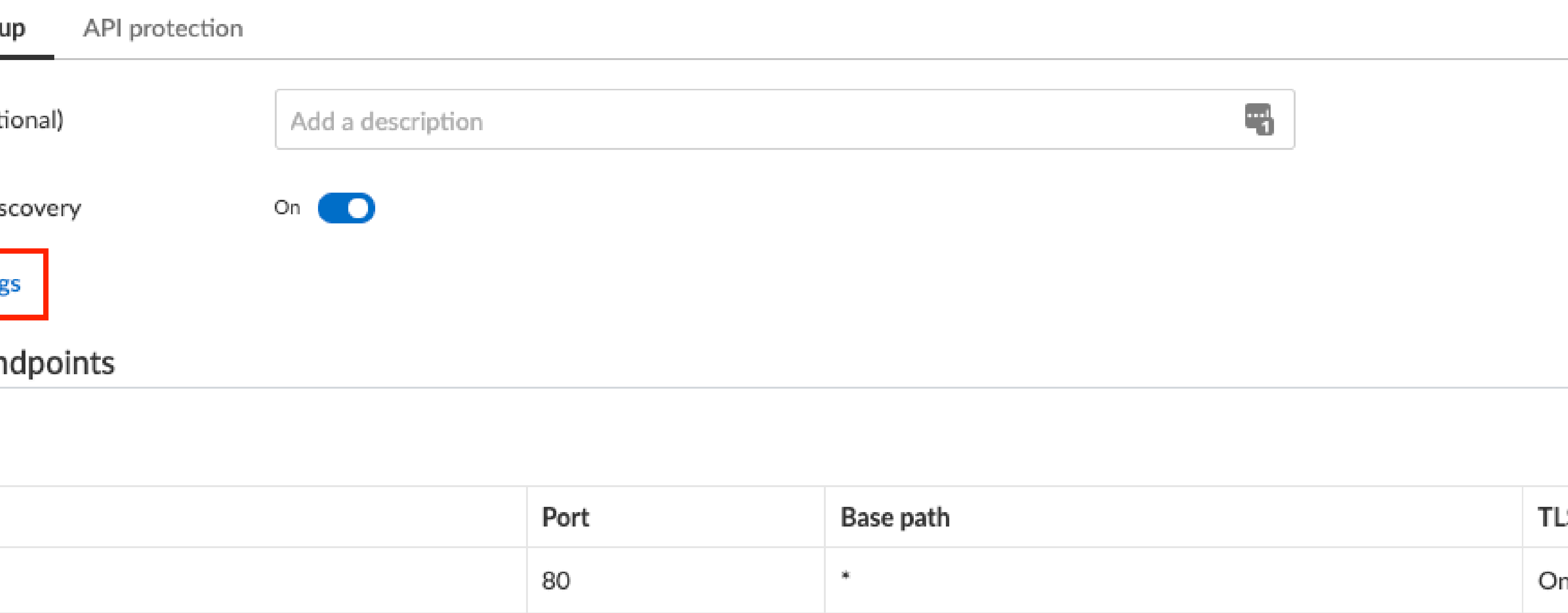
  Examples: "/admin", "/" (root path only), "/*", /v2/api", etc.
- If your application uses TLS, set **TLS** to **On**.
- If your application uses HTTP/2, set **HTTP/2** to **On**.

  WAAS must be able to decrypt and inspect HTTPS traffic to function properly.
- If your application uses gRPC, set **gRPC** to **On**.

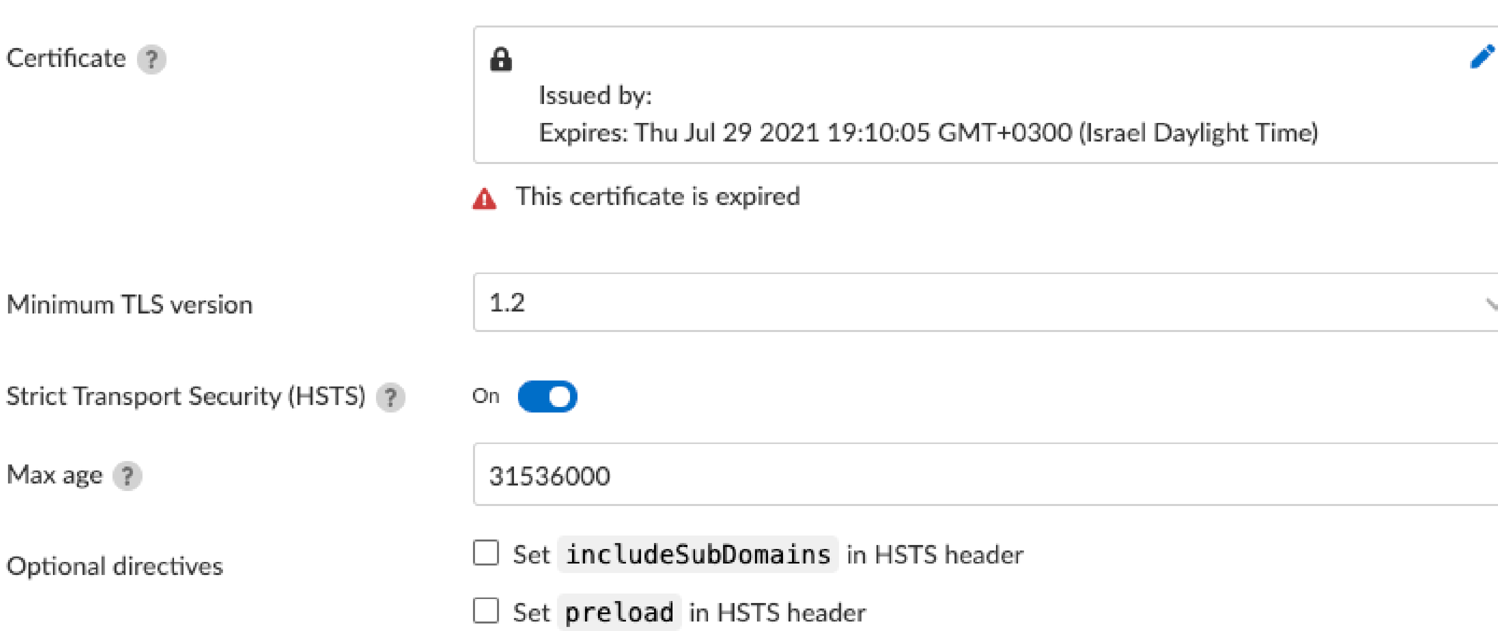
4. Click **Response headers** to add or override HTTP response headers in responses sent from the protected application.



5. Click **Create Endpoint**.
6. To facilitate inspection, after creating all endpoints, click **View TLS settings** in the endpoint setup menu.



TLS settings:



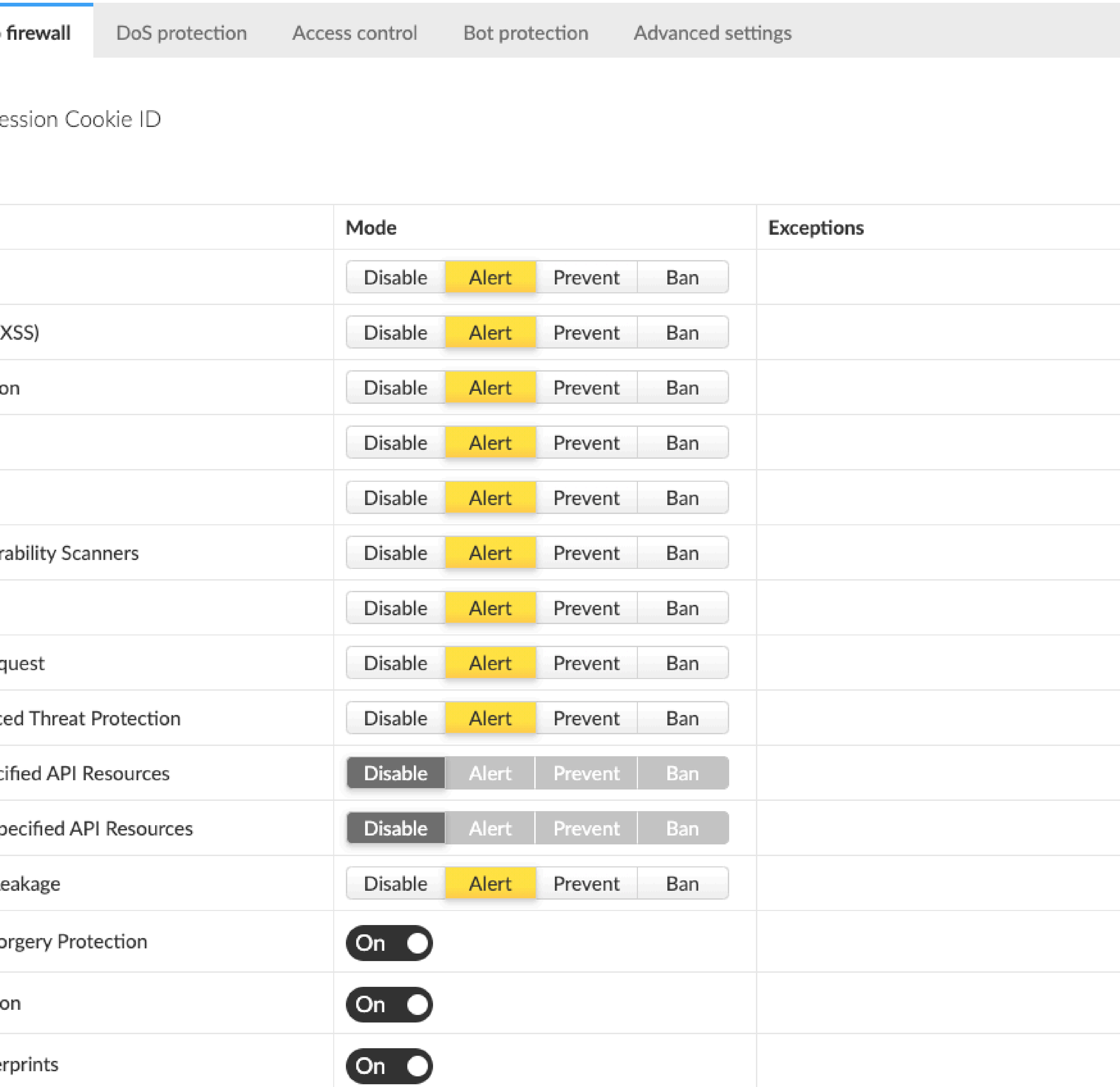
- **Certificate** - Copy and paste your server's certificate and private key into the certificate input box (e.g., *cat server-cert.pem server-key > certs.pem*).
- **Minimum TLS version** - A minimum version of TLS can be enforced by WAAS to prevent downgrading attacks (the default value is TLS 1.2).
- **HSTS** - The HTTP Strict-Transport-Security (HSTS) response header lets web servers tell browsers to use HTTPS only, not HTTP. When enabled, WAAS would add the

HSTS response header to all HTTPS server responses (if it is not already present) with the preconfigured directives - *max-age*, *includeSubDomains*, and *preload*.

1. *max-age=<expire-time>* - Time, in seconds, that the browser should remember that a site is only to be accessed using HTTPS.
2. *includeSubDomains* (optional) - If selected, HSTS protection applies to all the site's subdomains as well.
3. *preload* (optional) - For more details, see the following link.

7. If your application requires [API protection], select the **API Protection** tab and define for each path the allowed methods, parameters, types, etc. See detailed definition instructions on the [API protection] help page.

STEP 2 | Continue to **App Firewall** tab, select protections to enable and assign them with WAAS Actions.



STEP 3 | Continue to **Access Control** tab and select access controls to enable.

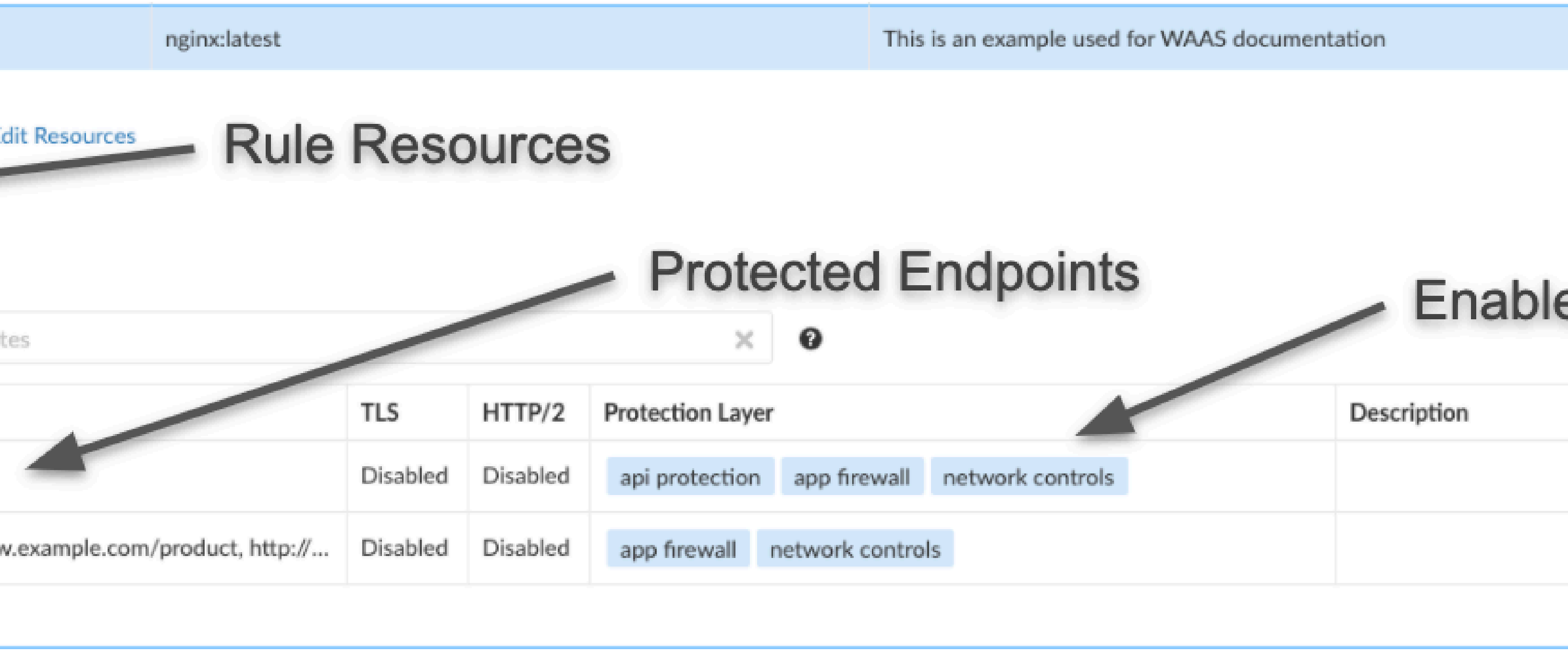
STEP 4 | Continue to **DoS protection** tab and configure DoS protection thresholds.

STEP 5 | Continue to **Bot protection** tab and select bot protections to enable.

STEP 6 | Click **Save**.

STEP 7 | You should be redirected to the **Rule Overview** page.

Select the created new rule to display **Rule Resources** and for each application a list of **protected endpoints** and **enabled protections**.



STEP 8 | Test protected endpoint using the following sanity tests.

STEP 9 | Go to **Monitor > Events**, click on **WAAS for containers** and observe events generated.

For more information please see the WAAS analytics help page.

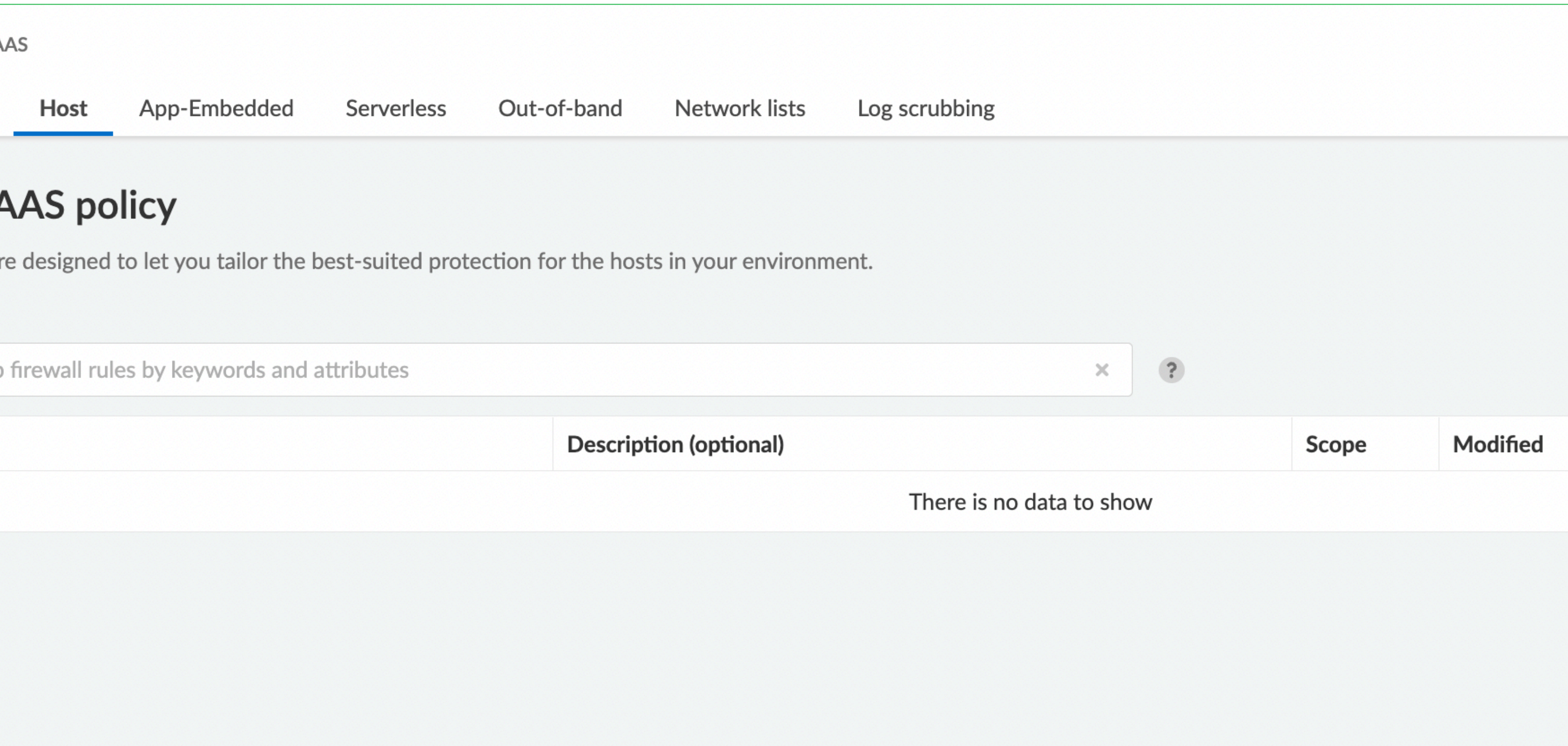
Deploy WAAS for Hosts

Edit on GitHub

To deploy WAAS to protect a host running a non-containerized web application, create a new rule, specify the host(s) where the application runs, define protected HTTP endpoints, and select protections.

Create a WAAS rule for Hosts

STEP 1 | Open Console, and go to **Defend** > **WAAS** > **Host**.

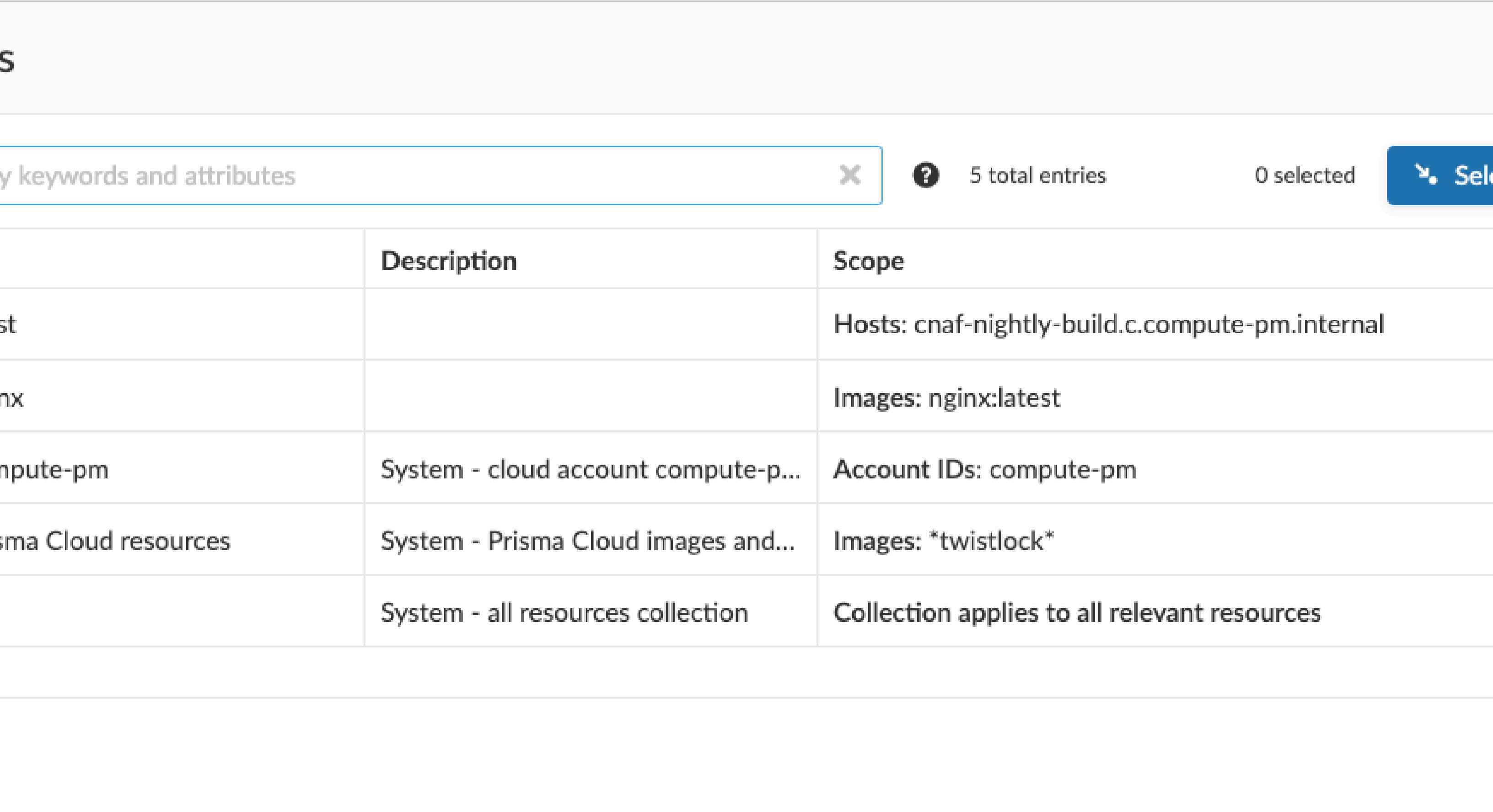


STEP 2 | Click **Add rule**.

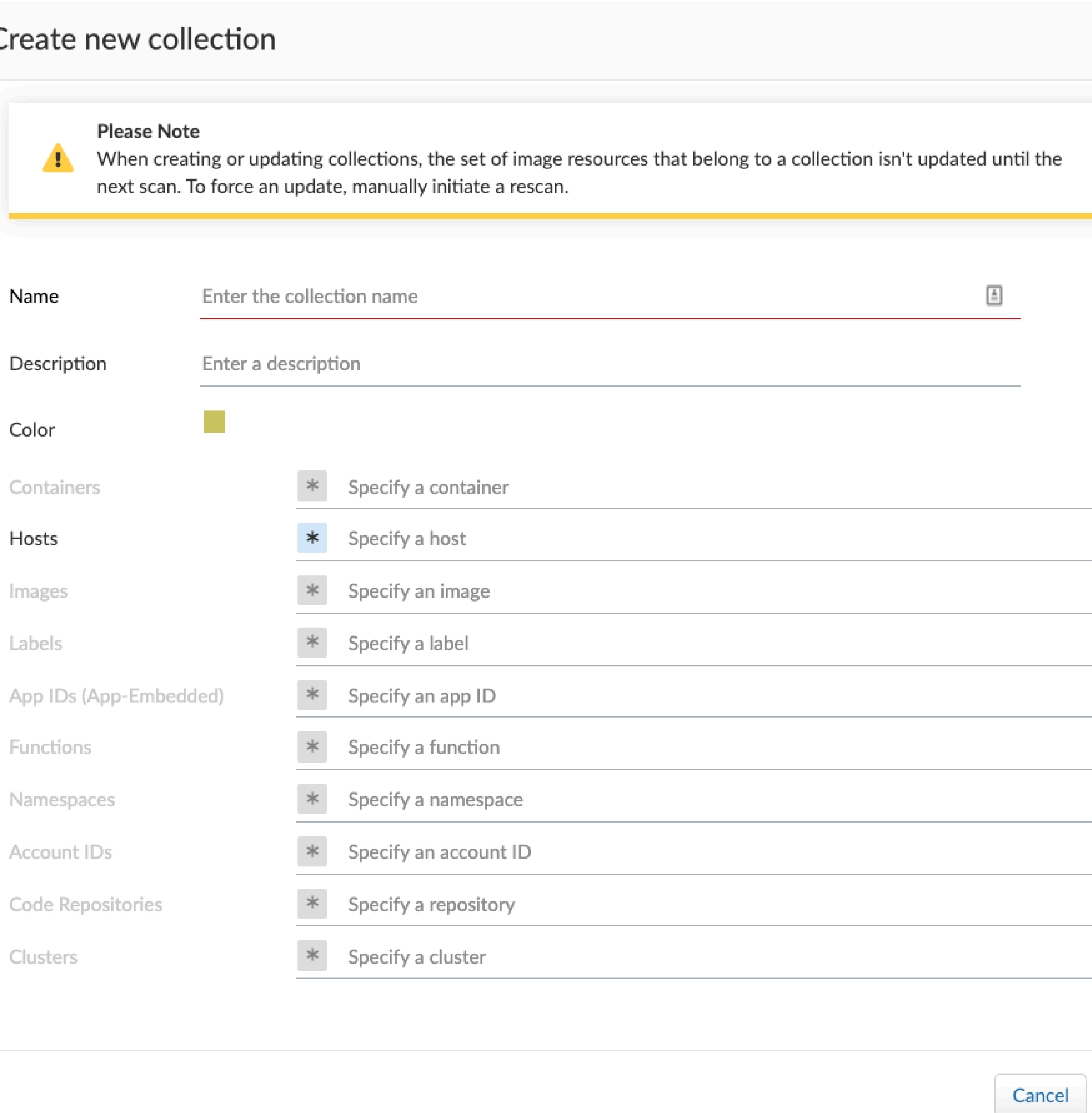
1. Enter a **Rule Name**
2. Enter **Notes** (Optional) for describing the rule.
3. Select **Operating system**
4. If necessary, adjust the **Proxy timeout**

The maximum duration in seconds for reading the entire request, including the body. A 500 error response is returned if a request is not read within the timeout period. For applications dealing with large files, adjusting the proxy timeout is necessary.

STEP 3 | Choose the rule **Scope** by specifying the resource collection(s) to which it applies.



Collections define a combination of hosts to which WAAS should attach itself to protect the web application:


Create new collection
Please Note
When creating or updating collections, the set of image resources that belong to a collection isn't updated until the next scan. To force an update, manually initiate a rescan.
Name
Enter the collection name
Description
Enter a description
Color
Containers
Specify a container
Hosts
Specify a host
Images
Specify an image
Labels
Specify a label
App IDs (App-Embedded)
Specify an app ID
Functions
Specify a function
Namespaces
Specify a namespace
Account IDs
Specify an account ID
Code Repositories
Specify a repository
Clusters
Specify a cluster
Cancel

Applying a rule to all hosts using a wild card (*) is invalid and a waste of resources. WAAS only needs to be applied to hosts that run applications that transmit and receive HTTP/HTTPS traffic.

STEP 4 | (Optional) Toggle to enable **Automatically detect ports** for an endpoint.

When you select this option, WAAS deploys its protection on ports identified in the unprotected web apps report in **Monitor > WAAS > Unprotected web apps** for each of the workloads in the rule scope. You can specify additional ports by specifying them in the protected HTTP endpoints within each app.

STEP 5 | (Optional) Toggle to enable **API endpoint discovery**.

STEP 6 | **Save** the rule.

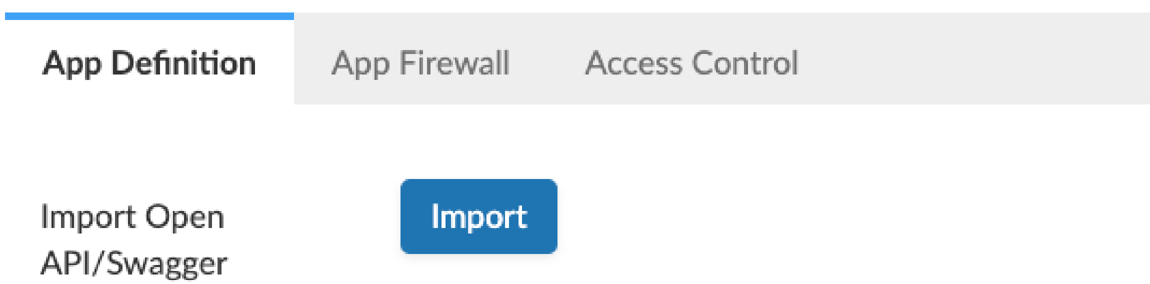
Add an App (policy) to the Host rule

STEP 1 | Select a WAAS host rule to add an App in.

STEP 2 | Click **Add app**.

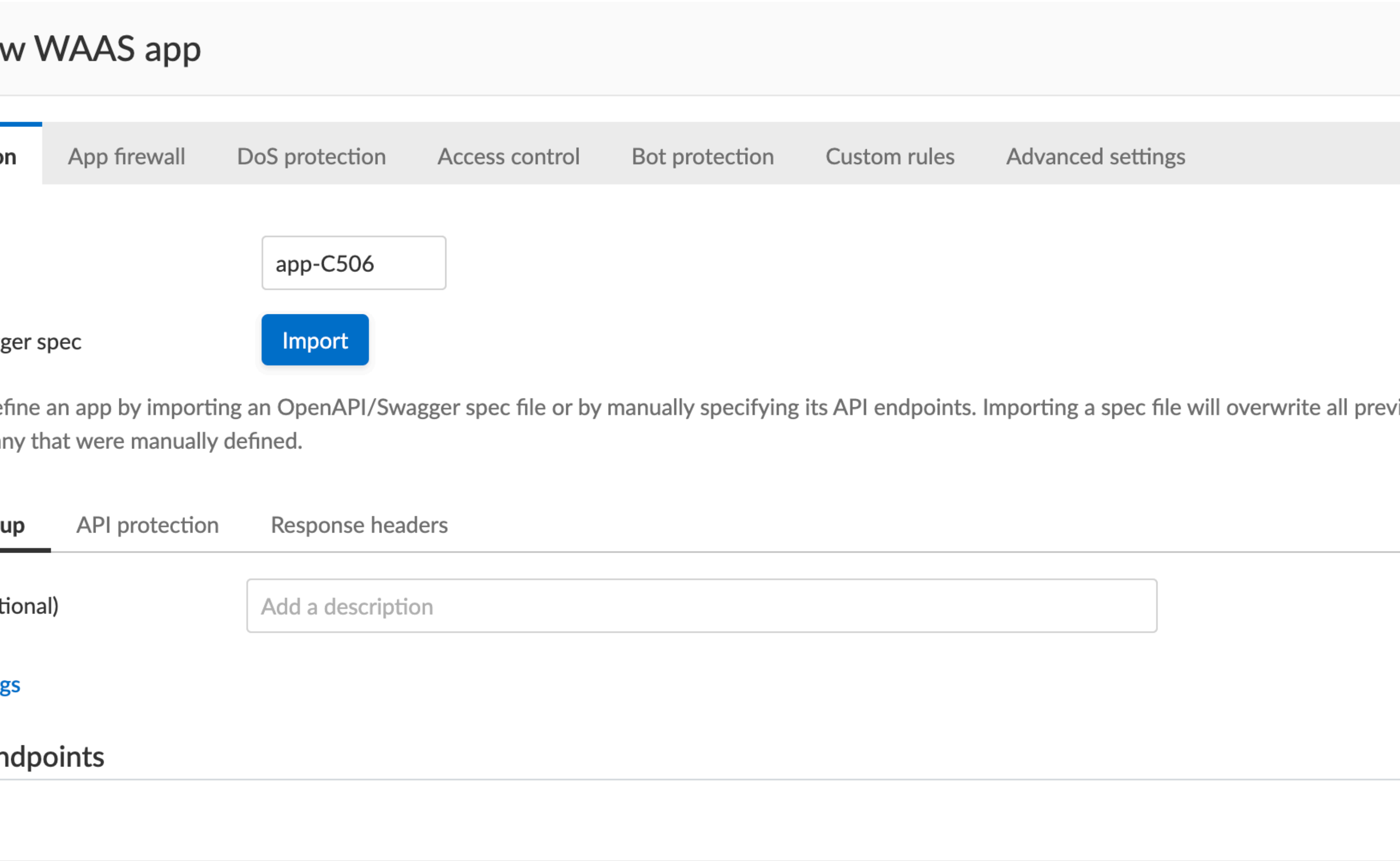
STEP 3 | In the App Definition tab, specify the endpoints in your web application that should be protected.

Each defined application can have multiple protected endpoints. If you have a Swagger or OpenAPI file, click Import, and select the file to load. Otherwise, skip to the next step to manually define your application's endpoints.

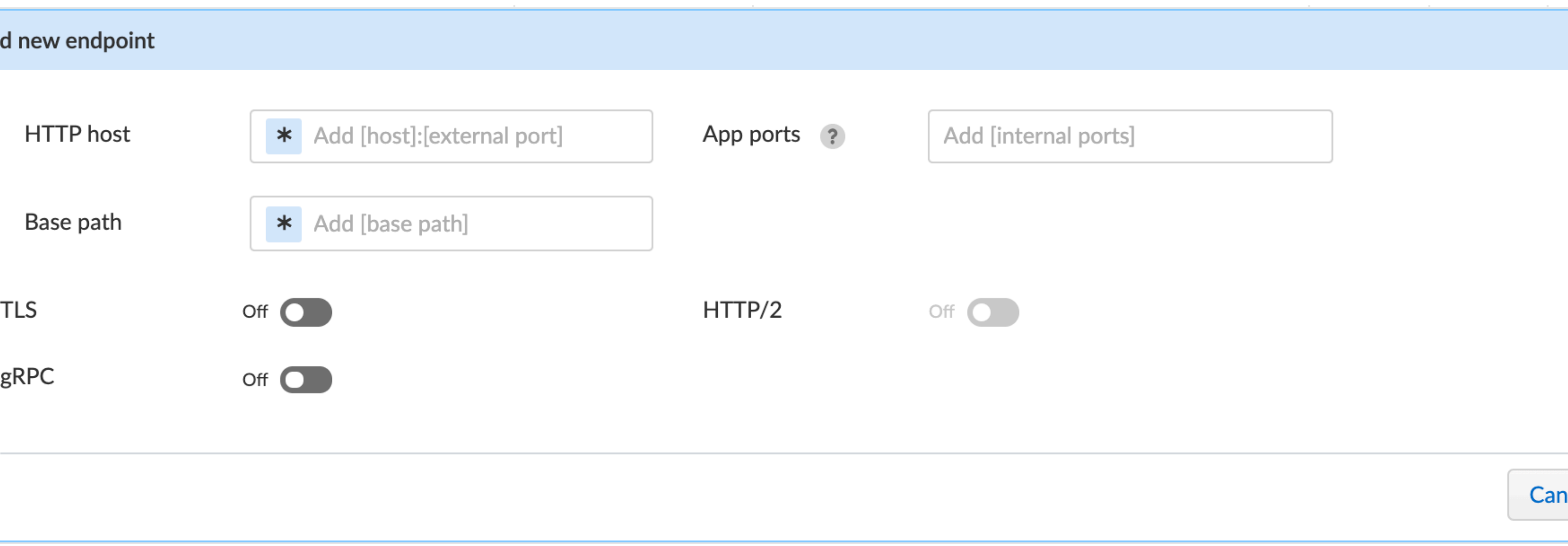


STEP 4 | If you don’t have a Swagger or OpenAPI file, manually define each endpoint by specifying the host, port, and path.

1. In the **Endpoint Setup** tab, click on **Add Endpoint**



2. Specify endpoint details:



3. Enter **Port** (optional, if you selected **Automatically detect ports** while creating the rule). When **Automatically detect ports** is selected, any ports specified in a protected endpoint definition will be appended to the list of protected ports.

   Specify the TCP port protected app listens on, WAAS sends traffic to your app over this port.

   *If your application uses **TLS** or **gRPC**, you must specify a port number.*

4. Enter **WAAS Port (only required for Windows or when using "Remote host" option)**.

   Specify the TCP port on which WAAS listens. WAAS receives traffic from your end-users on this port, processes it, and then sends it to your app on the App port.

   *Protecting Linux-based hosts does not require specifying a **WAAS port** since WAAS listens on the same port as the protected application. Because of Windows internal traffic routing mechanisms WAAS and the protected application cannot use the same **App port**. Consequently, when protecting Windows-based hosts the **WAAS port** should be set to the port end-users send requests to, and the **App port** should be set to a **different** port on which the protected application would listen on and WAAS would forward traffic to.*

5. Enter **HTTP host** (optional, wildcards supported).

   HTTP host names are specified in the form of [hostname]:[external port].

   External port is defined as the TCP port on the host, listening for inbound HTTP traffic. If the value of the external port is "80" for non-TLS endpoints or "443" for

TLS endpoints it can be omitted. Examples: "*.example.site", "docs.example.site", "www.example.site:8080", etc.

6. Enter **Base path** (optional, wildcards supported):

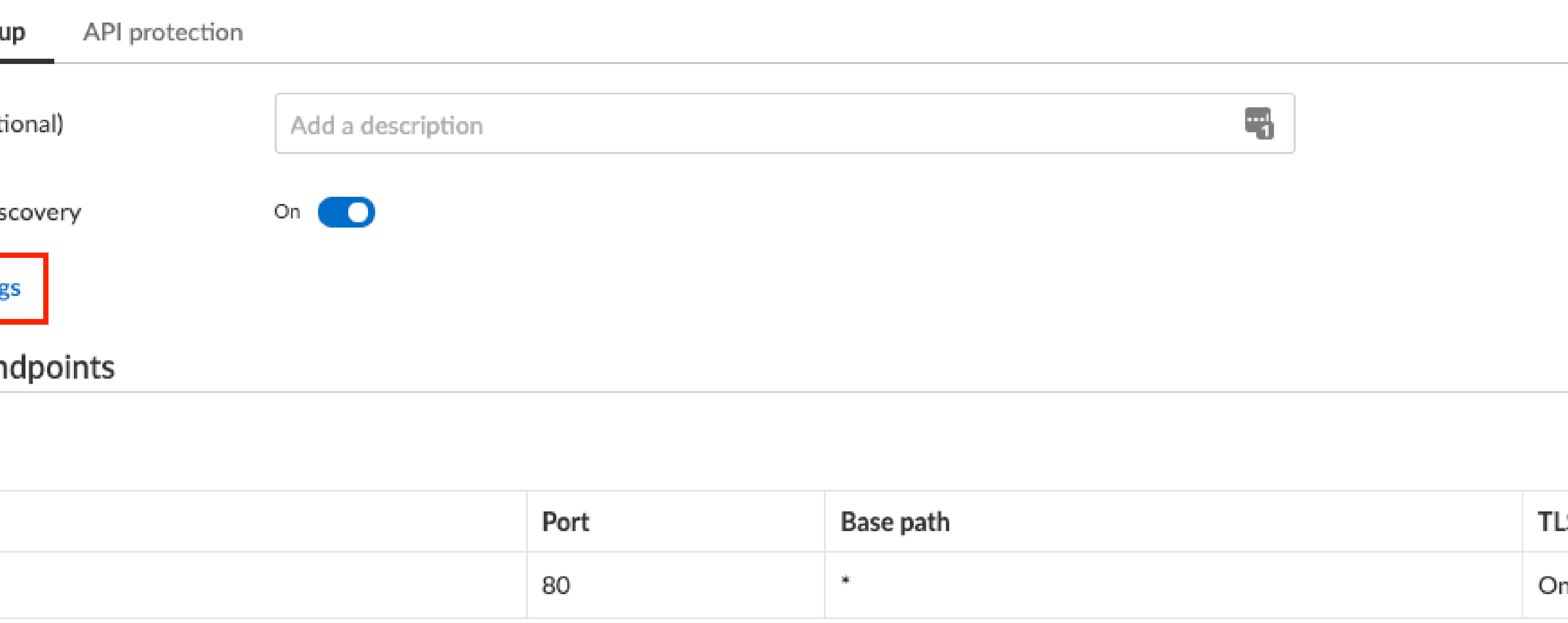
   Base path for WAAS to match on when applying protections.

   Examples: "/admin/", "/" (root path only), "/*", /v2/api/", etc.

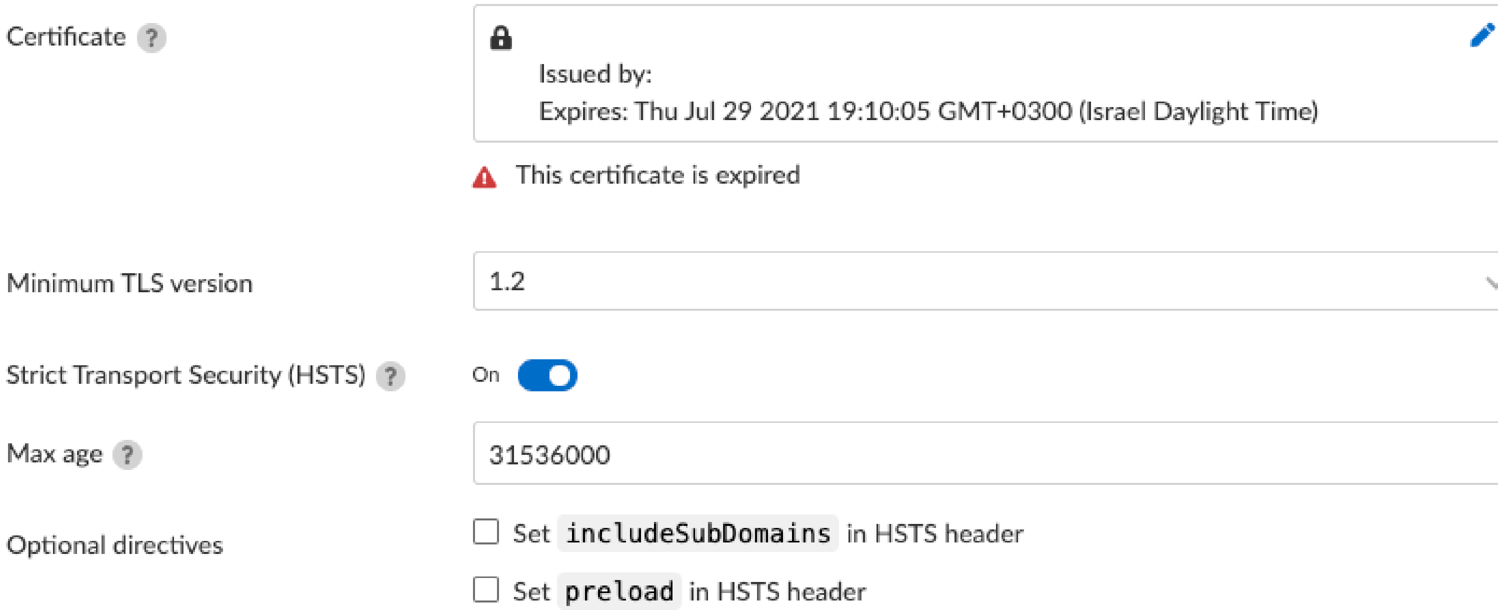
7. If your application uses TLS, set **TLS** to **On**.

   WAAS must be able to decrypt and inspect HTTPS traffic to function properly.

   To facilitate inspection, after creating all endpoints, click **View TLS settings** in the endpoint setup menu.



TLS settings:

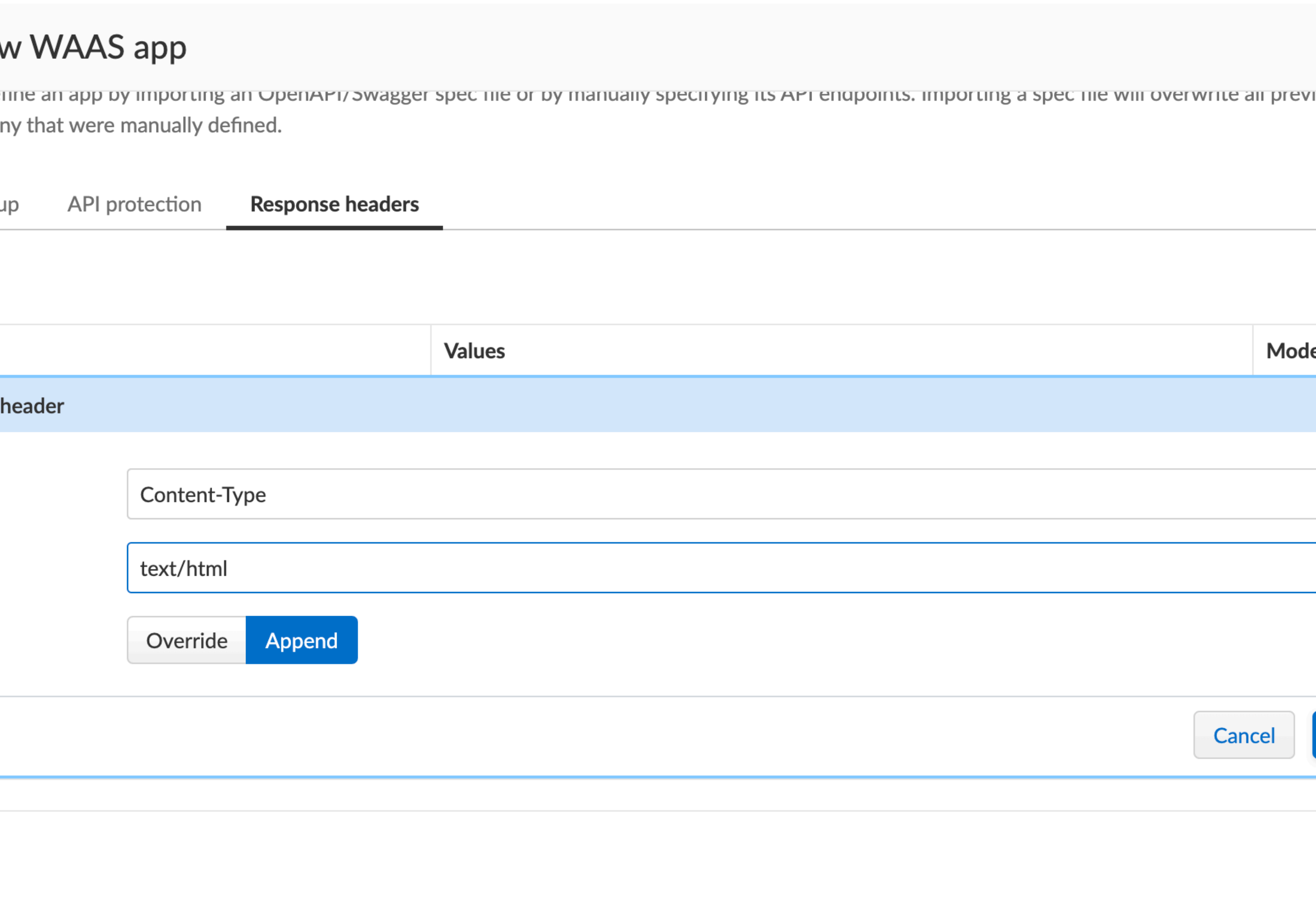


1. **Certificate** - Copy and paste your server's certificate and private key into the certificate input box (e.g. cat server-cert.pem server-key > certs.pem).
2. **Minimum TLS version** - Minimum version of TLS can be enforced by WAAS to prevent downgrading attacks (the default value is TLS 1.2).
3. **HSTS** - HTTP Strict-Transport-Security (HSTS) response header lets web servers tell browsers to use HTTPS only, not HTTP. When enabled, WAAS adds the HSTS

response header to all HTTPS server responses (if not already present) with the preconfigured directives - *max-age*, *includeSubDomains*, and *preload*.

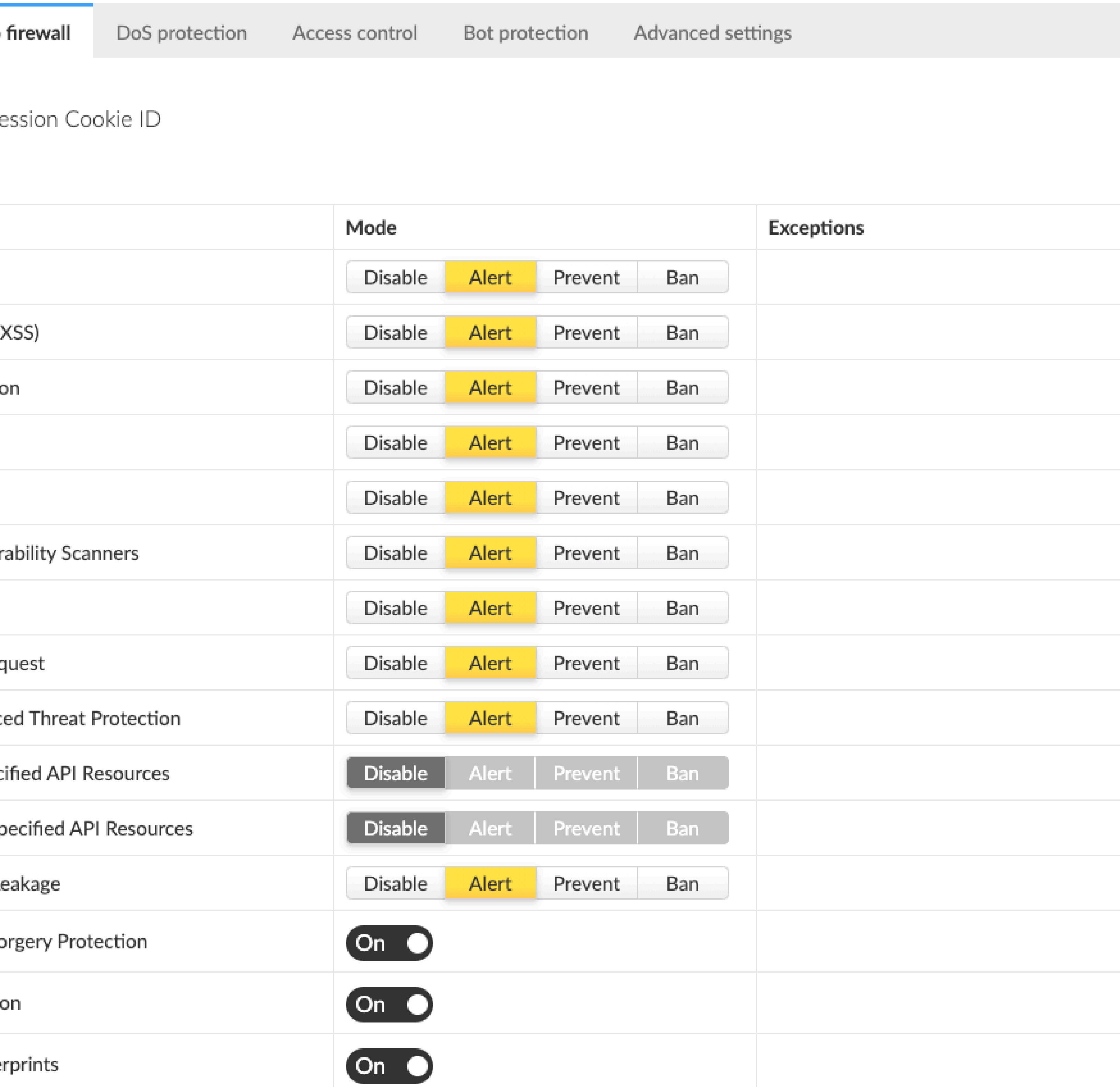
- *max-age=<expire-time>* - Time, in seconds, that the browser should remember that a site is only to be accessed using HTTPS.
- *includeSubDomains* (optional) - If selected, HSTS protection applies to all the site's subdomains as well.
- *preload* (optional) - For more details, refer to the following link.

8. If your application uses gRPC, set **gRPC** to **On**.
9. If your application uses HTTP/2, set **HTTP/2** to **On**.
10. Click on the **Response headers** tab to add or override HTTP response headers in responses sent from the protected application.



11. Click **Create Endpoint**
12. If your application requires API protection, select the "API Protection" tab and define for each path allowed methods, parameters, types, etc. See detailed definition instructions in the API protection help page.

STEP 5 | Continue to **App firewall** tab, select protections to enable and assign them with actions.



STEP 6 | Continue to **Access Control** tab and select access controls to enable.

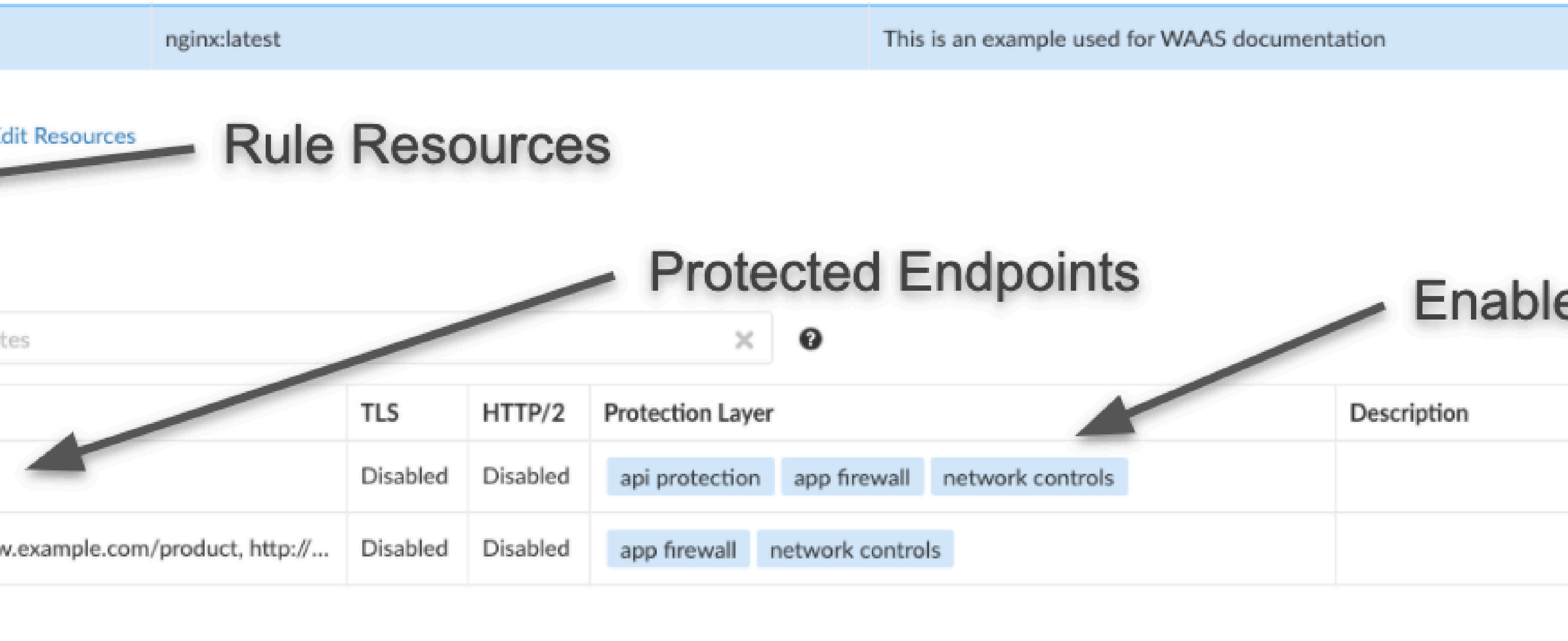
STEP 7 | Continue to **DoS protection** tab and configure DoS protection thresholds.

STEP 8 | Continue to **Bot protection** tab and select bot protections to enable.

STEP 9 | Click **Save**.

STEP 10 | You should be redirected to the **Rule Overview** page.

Select the created new rule to display **Rule Resources** and for each application a list of **protected endpoints** and **enabled protections**.



STEP 11 | Test protected endpoint using the following sanity tests.

STEP 12 | Go to **Monitor > Events**, click on **WAAS for hosts** and observe events generated.

For more information please see the WAAS analytics help page

Deploy WAAS for Containers Protected By App-Embedded Defender

Edit on GitHub

In some environments, Prisma Cloud Defender must be embedded directly inside the container it is protecting. This type of Defender is known as an App-Embedded Defender. App-Embedded Defender can secure these types of containers with all WAAS protection capabilities.

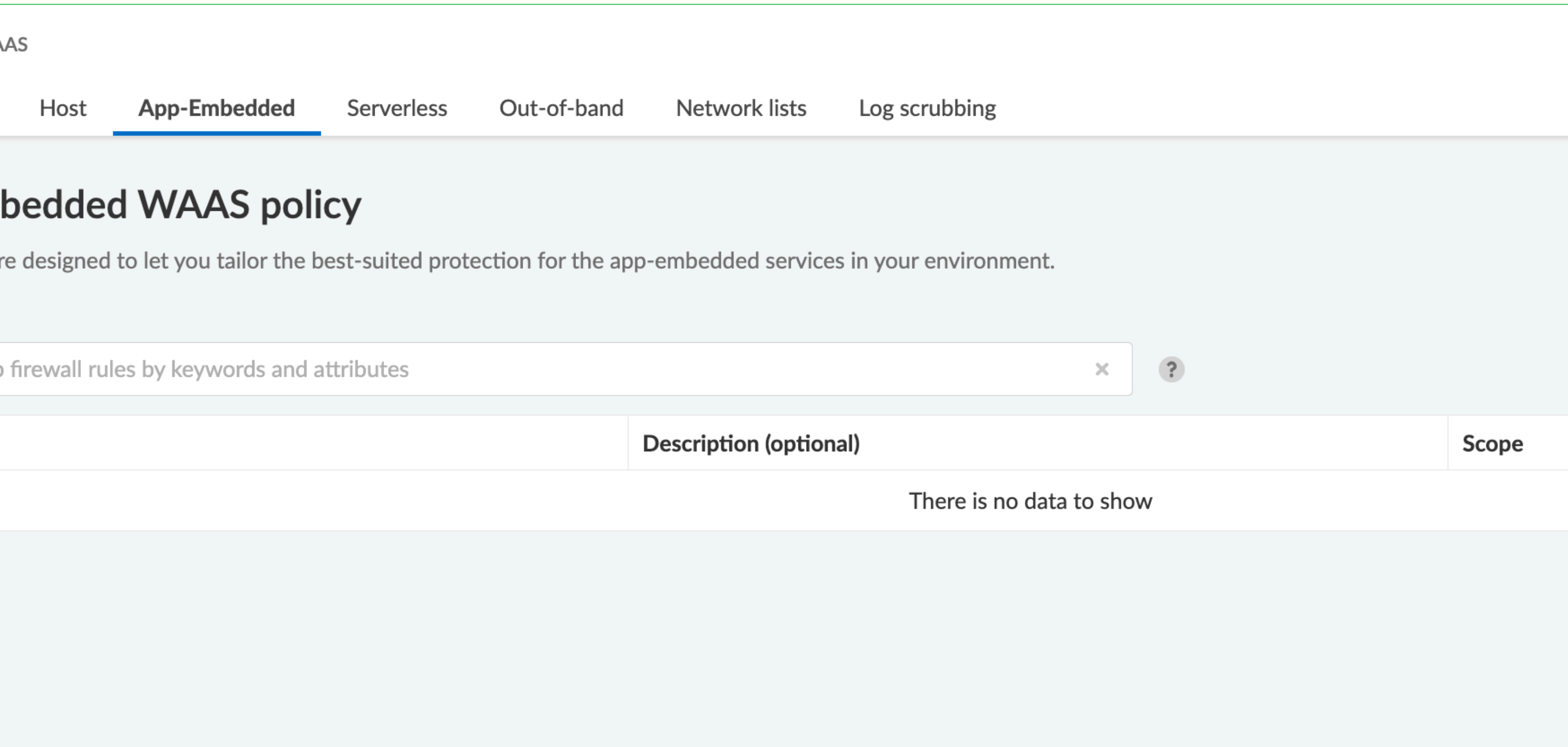
The only difference is that App-Embedded Defender runs as a reverse proxy to the container it's protecting. As such, when you set up WAAS for App-Embedded, you must specify the exposed external port where App-Embedded Defender can listen, and the port (not exposed to the Internet) where your web application listens. WAAS for App-Embedded forwards the filtered traffic to your application's port - unless an attack is detected and you set your WAAS for App-Embedded rule to **Prevent**.

When testing your Prisma Cloud-protected container, be sure you update the security group's inbound rules to permit TCP connections on the external port you entered in the WAAS rule. This is the exposed port that allows you to access your web application's container. To disable WAAS protection, disable the WAAS rule, and re-expose the application's real port by modifying the security group's inbound rule.

To embed App-Embedded WAAS into your container or Fargate task:

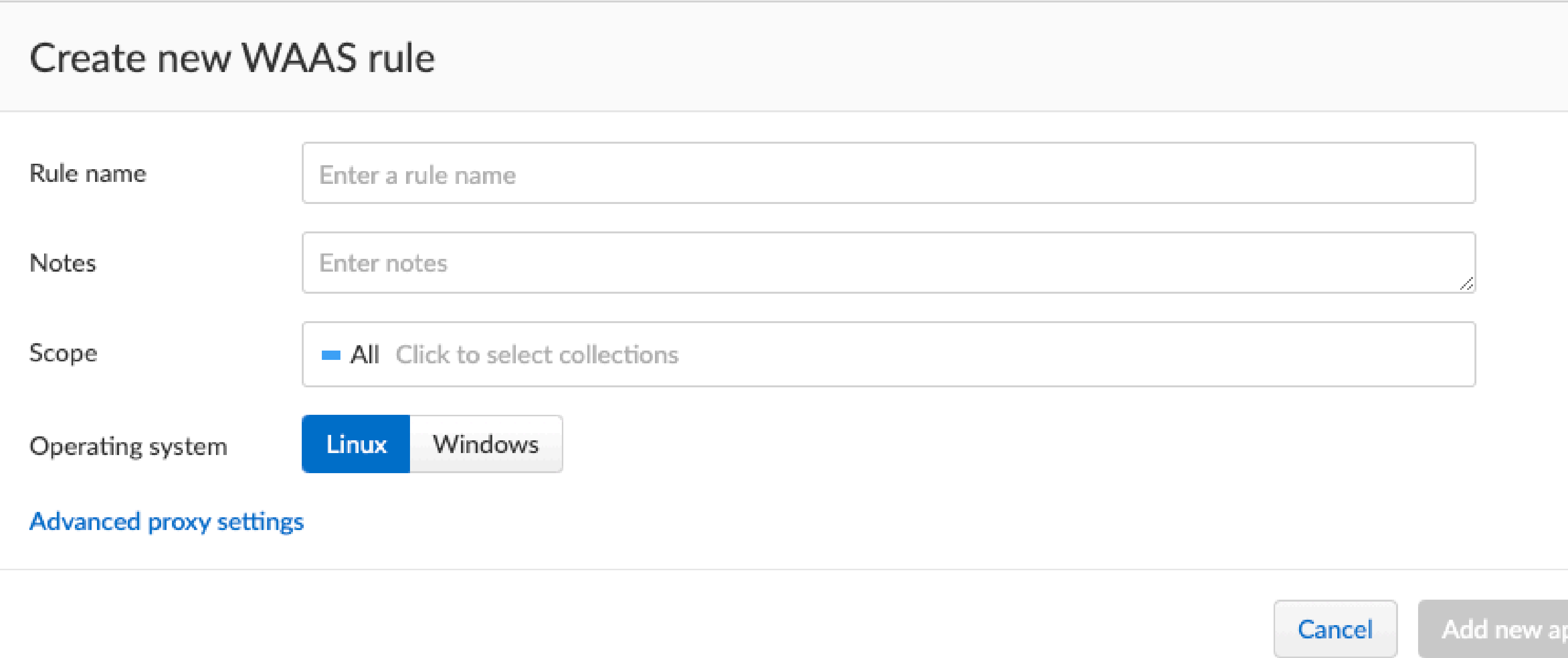
Create a rule for App-Embedded

STEP 1 | Open Console, and go to **Defend > WAAS > *App-Embedded**.

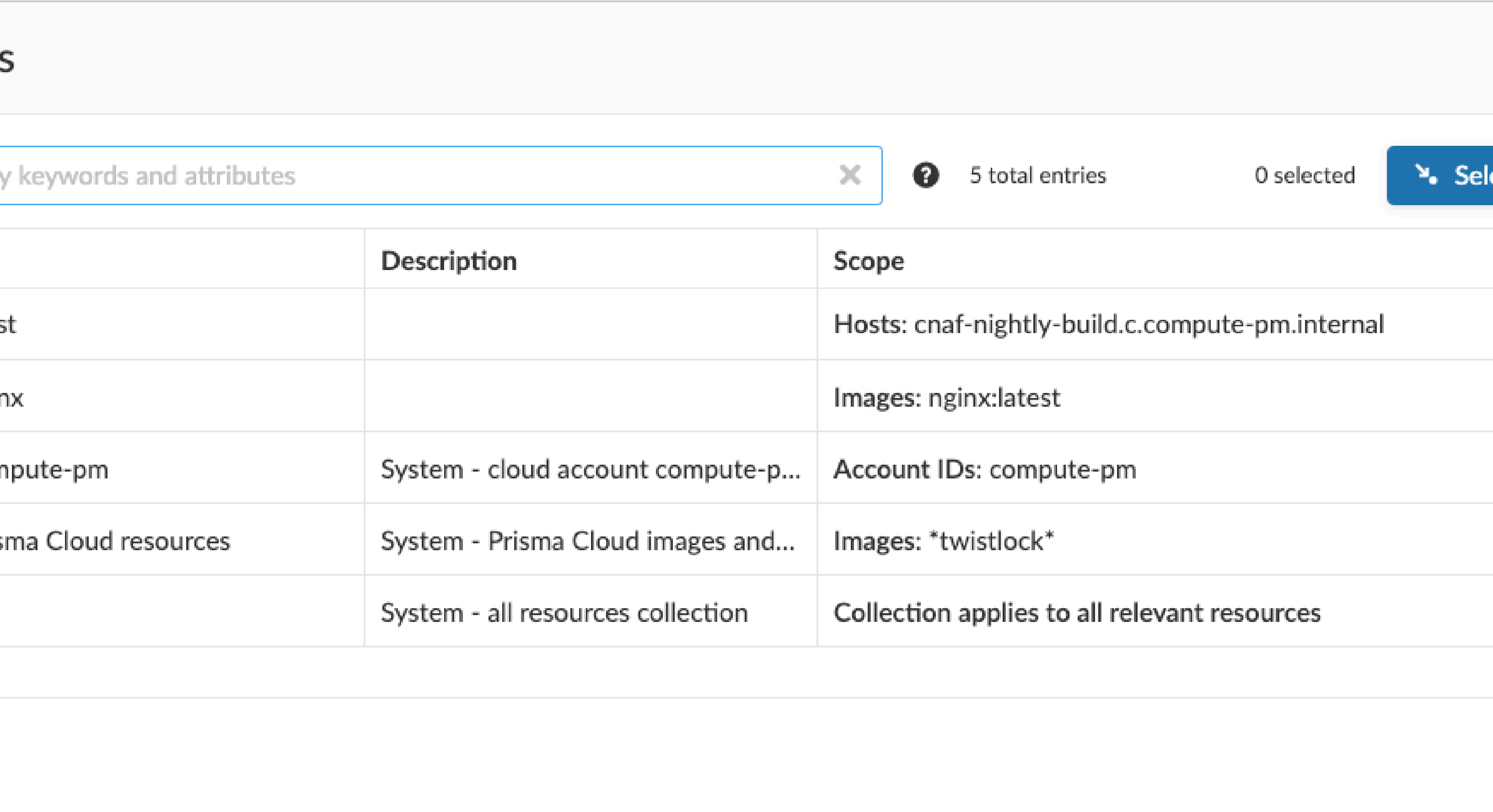


STEP 2 | Click **Add rule**.

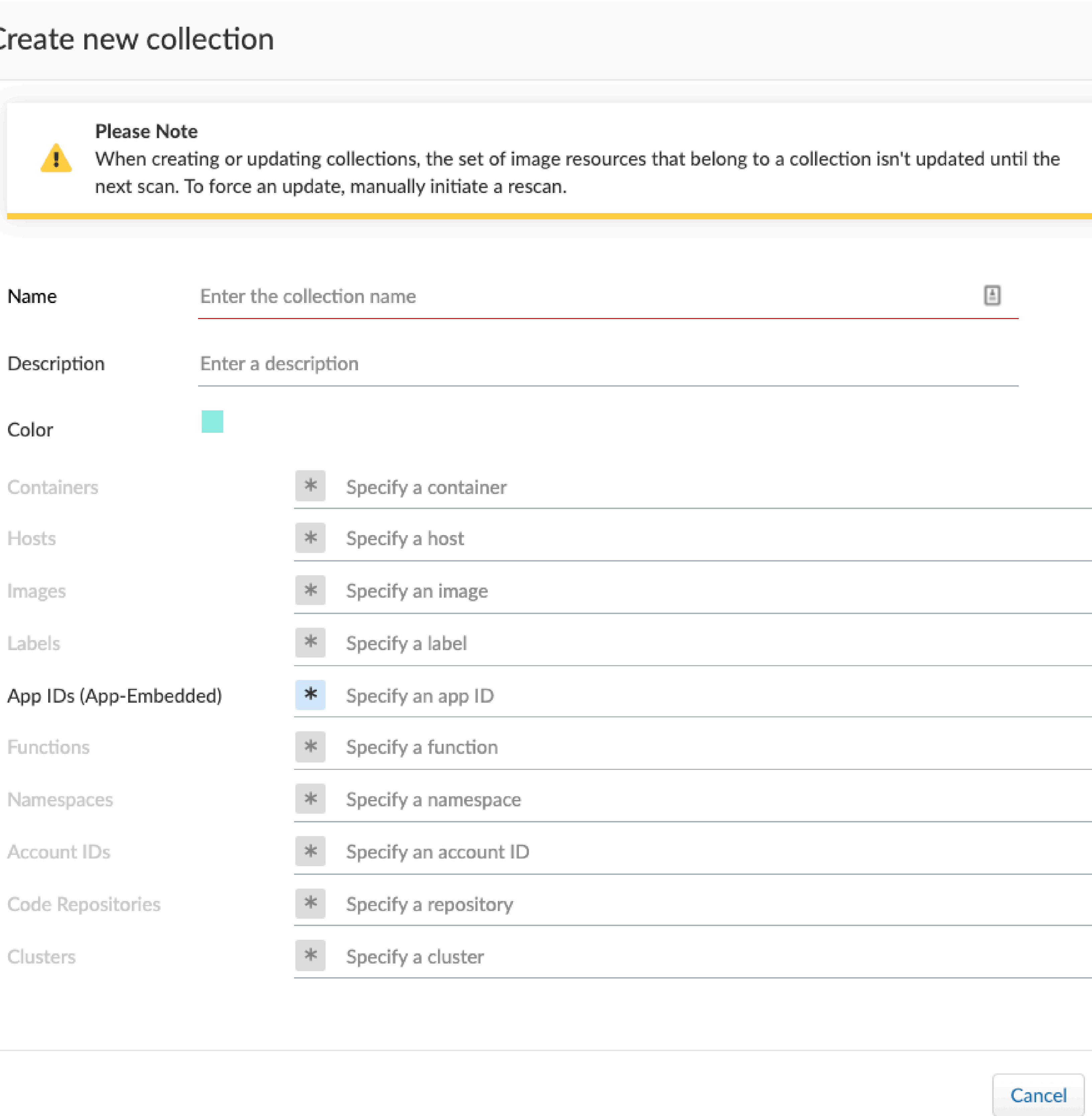
STEP 3 | Enter a **Rule Name** and **Notes** (Optional) for describing the rule.



STEP 4 | Choose the rule **Scope** by specifying the resource collection(s) to which it applies.



Collections define a combination of App IDs to which WAAS should attach itself to protect the web application:


Create new collection
Please Note
When creating or updating collections, the set of image resources that belong to a collection isn't updated until the next scan. To force an update, manually initiate a rescan.
Name
Enter the collection name
Description
Enter a description
Color
Containers
Specify a container
Hosts
Specify a host
Images
Specify an image
Labels
Specify a label
App IDs (App-Embedded)
Specify an app ID
Functions
Specify a function
Namespaces
Specify a namespace
Account IDs
Specify an account ID
Code Repositories
Specify a repository
Clusters
Specify a cluster
Cancel

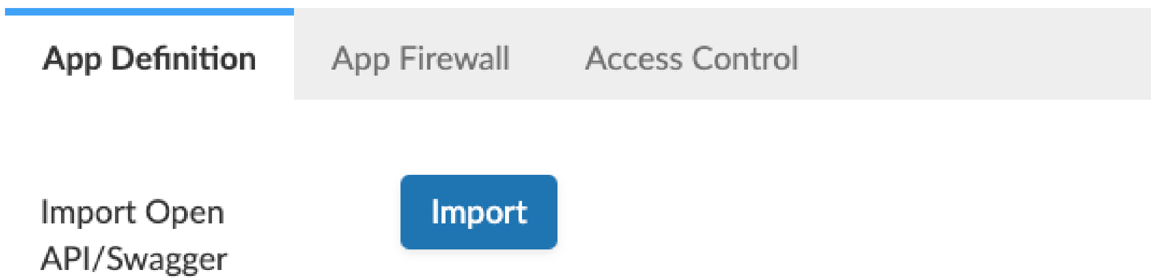
STEP 5 | **Save** the rule.

Add an App (policy) for App-Embedded

STEP 1 | Select a WAAS App-Embedded rule to add an App in.

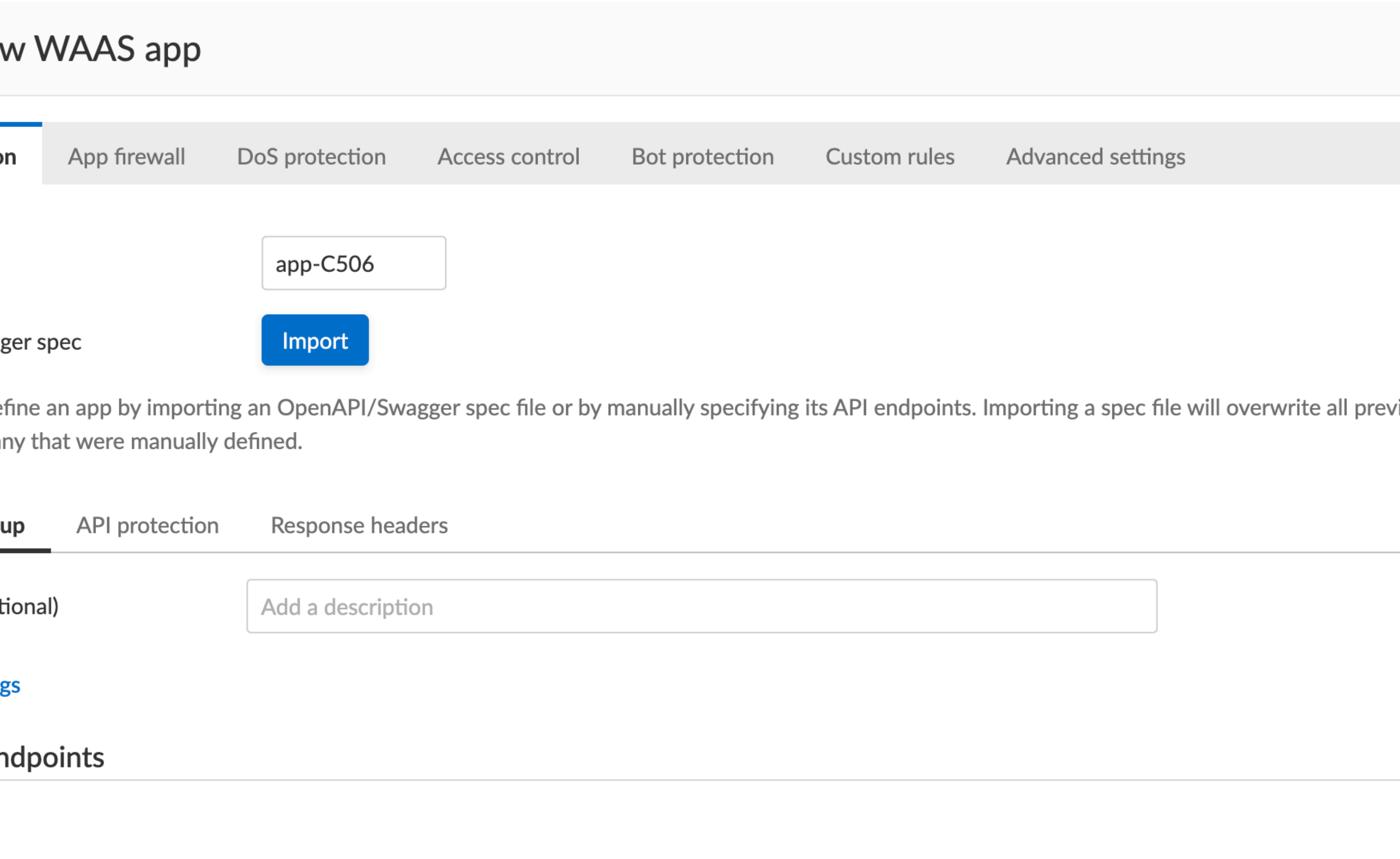
STEP 2 | Click **Add app**.

STEP 3 | In the App Definition tab, specify the endpoints in your web application that should be protected. Each defined application can have multiple protected endpoints. If you have a Swagger or OpenAPI file, click Import, and select the file to load. Otherwise, skip to the next step to manually define your app's endpoints.

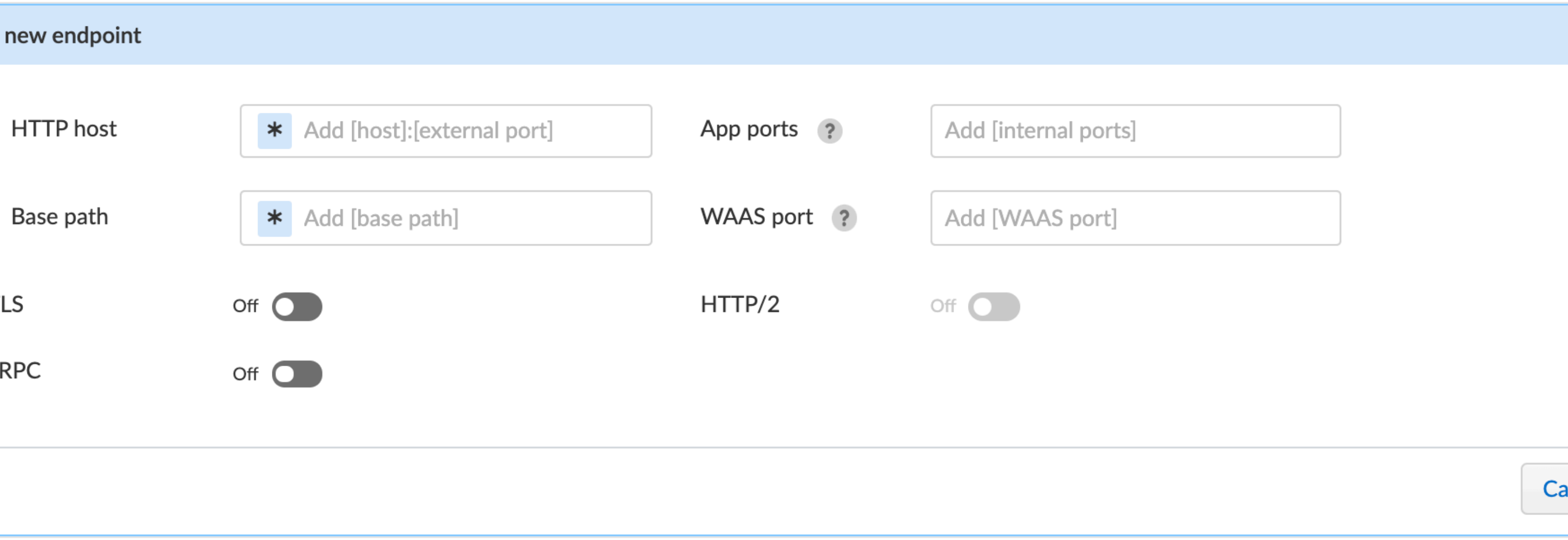


STEP 4 | If you don't have a Swagger or OpenAPI file, manually define each endpoint by specifying the host, port, and path.

1. In the **Endpoint Setup** tab, click on **Add Endpoint**.



2. Specify endpoint details:



3. Enter **App port (required)**

   Specify the TCP port protected app listens on, WAAS sends traffic to your app over this port.

4. Enter **WAAS Port (required)**.

   The external port is the TCP port for the App-Embedded Defender to listen on for inbound HTTP traffic.

5. Enter **HTTP host** (optional, wildcards supported).

   HTTP host names are specified in the form of [hostname]:[external port].

   The external port is defined as the TCP port on the host, listening for inbound HTTP traffic. If the value of the external port is "80" for non-TLS endpoints or "443" for

TLS endpoints it can be omitted. Examples: "*.example.com", "docs.example.com", "www.example.com:8080", etc.

6. Enter **Base path** (optional, wildcards supported):

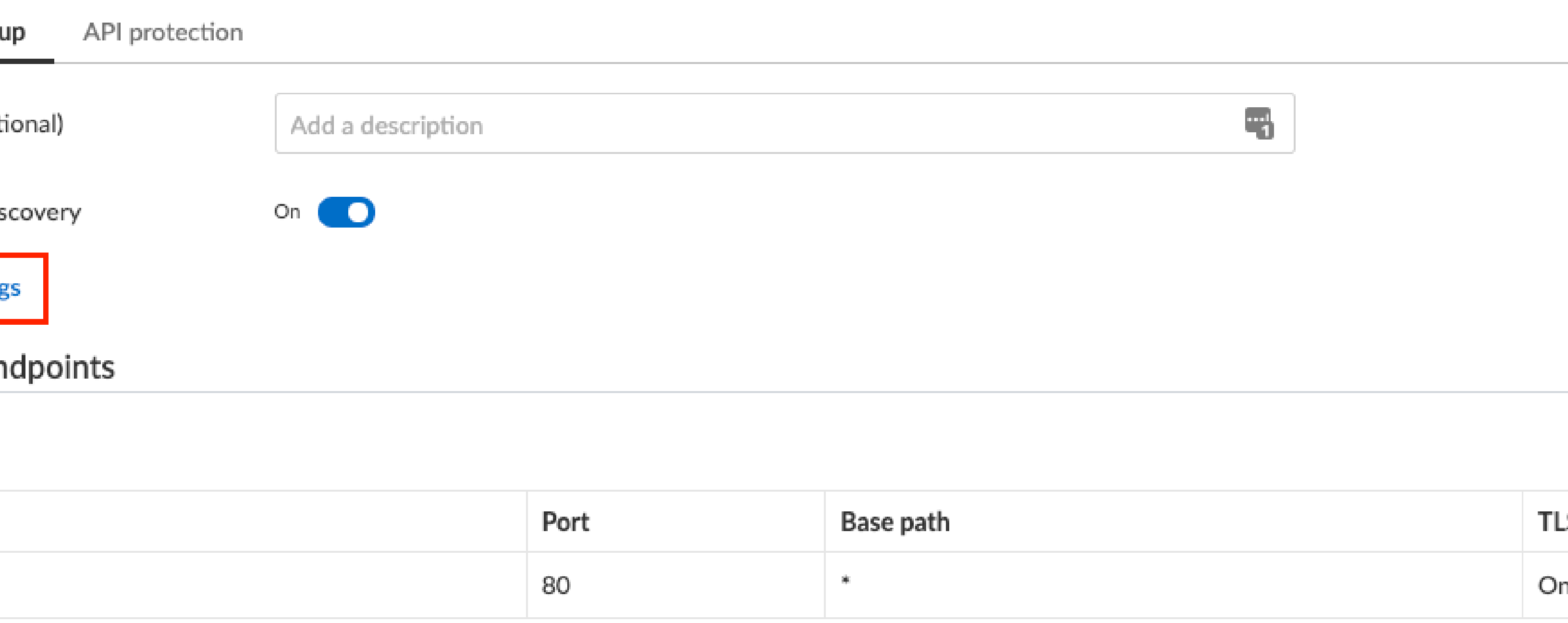
   Base path for WAAS to match on when applying protections.

   Examples: "/admin/", "/" (root path only), "/*", /v2/api/", etc.

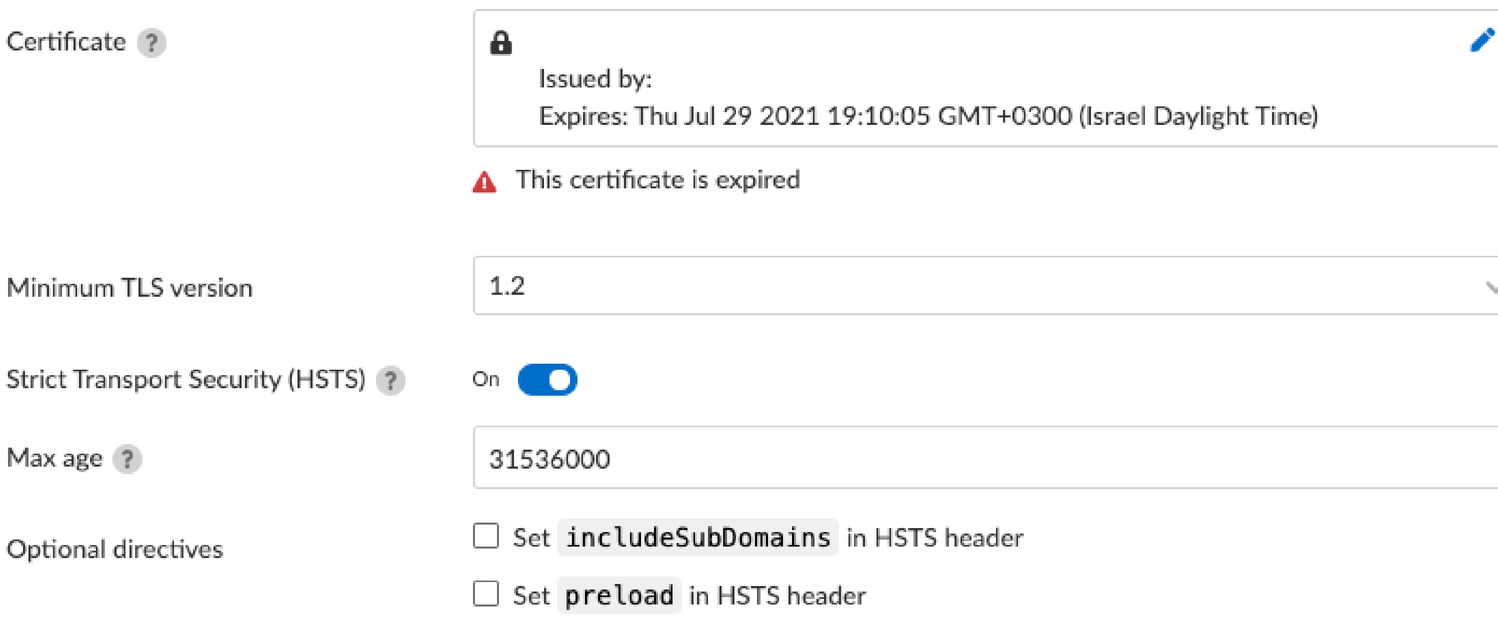
7. If your application uses TLS, set **TLS** to **On**.

   WAAS must be able to decrypt and inspect HTTPS traffic to function properly.

   To facilitate that, after creating all endpoints click on **View TLS settings** in the endpoint setup menu



TLS settings:

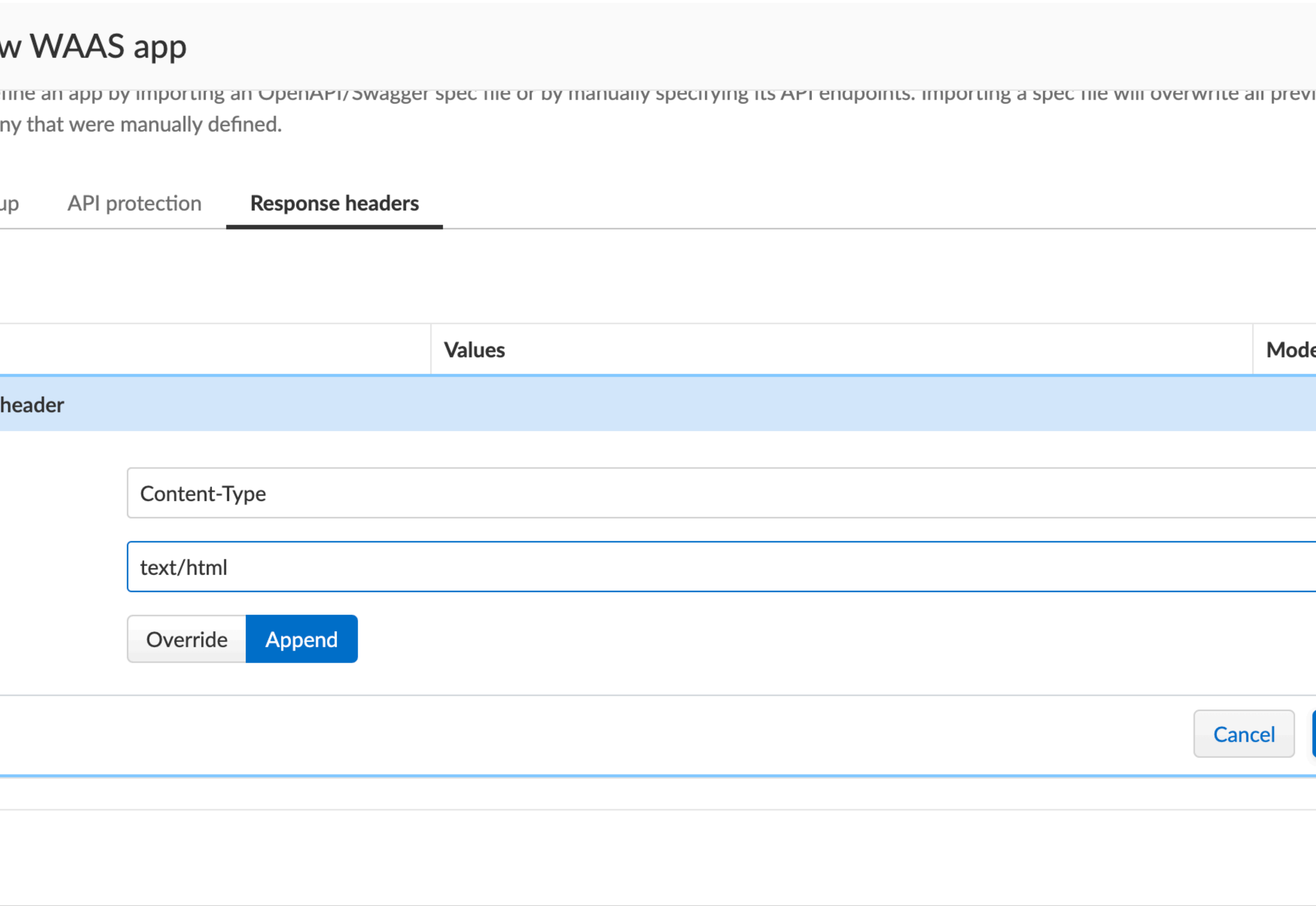


1. **Certificate** - Copy and paste your server's certificate and private key into the certificate input box (e.g. cat server-cert.pem server-key > certs.pem).
2. **Minimum TLS version** - A minimum version of TLS can be enforced by WAAS to prevent downgrading attacks (the default value is "1.2").
3. **HSTS** - The HTTP Strict-Transport-Security (HSTS) response header lets web servers tell browsers to use HTTPS only, not HTTP. When enabled, WAAS adds the HSTS

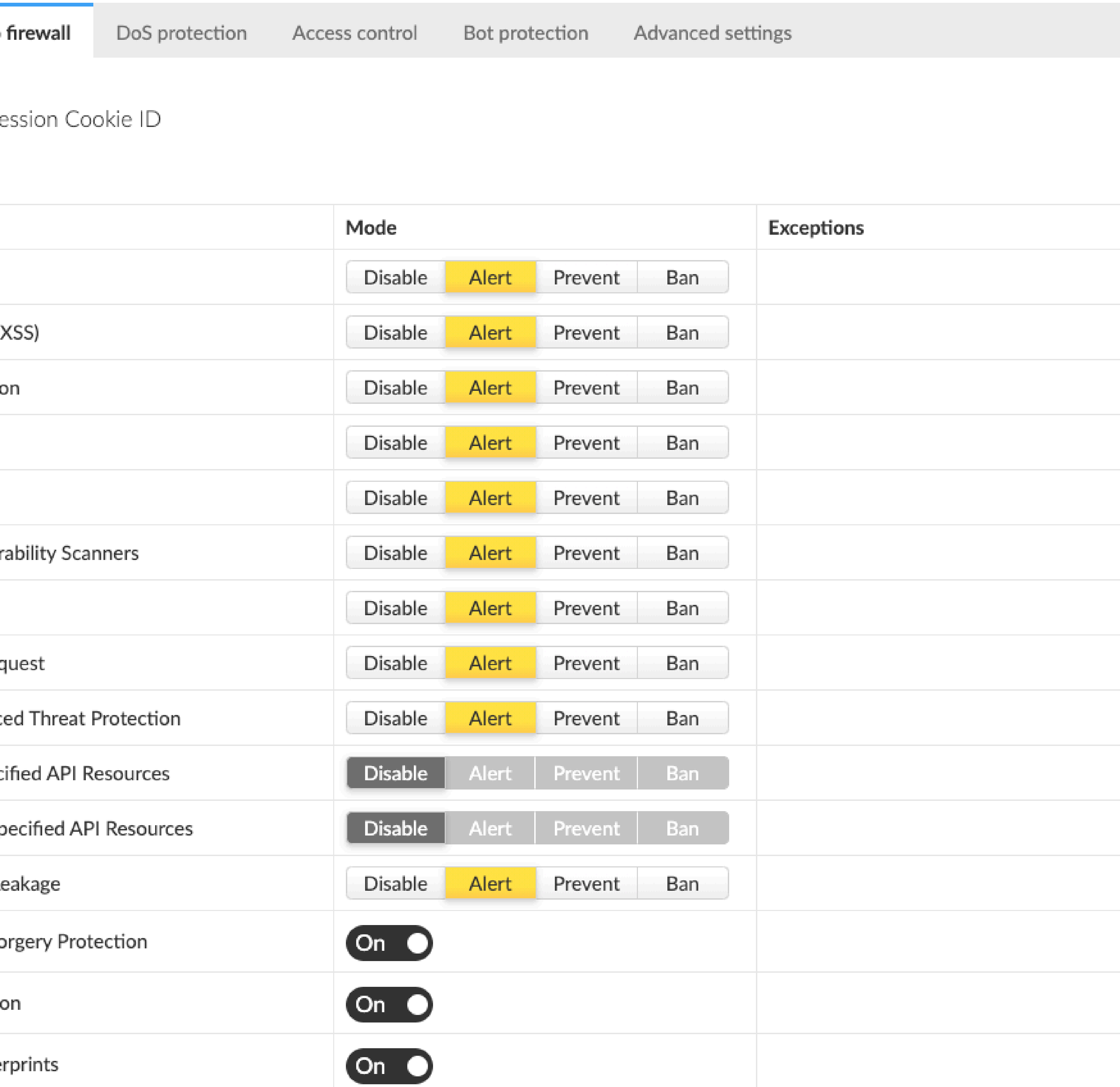
response header to all HTTPS server responses (if not already present) with the preconfigured directives - *max-age*, *includeSubDomains*, and *preload*.

- *max-age=<expire-time>* - The time, in seconds, that the browser should remember that a site is only to be accessed using HTTPS.
- *includeSubDomains* (optional) - If selected this HSTS protection applies to all of the site's subdomains as well.
- *preload* (optional) - for more details please refer to the following link.

8. If your application uses gRPC, set **gRPC** to **On**.
9. If your application uses HTTP/2, set **HTTP/2** to **On**.
10. Click **Create Endpoint**
11. If your application requires API protection, select the "API Protection" tab and define for each path allowed methods, parameters, types, etc. See detailed definition instructions in the API protection help page.
12. Click on the **Response headers** tab to add or override HTTP response headers in responses sent from the protected application.



STEP 5 | Continue to **App Firewall** tab, select protections to enable and assign them with actions.



STEP 6 | Continue to **Access Control** tab and select access controls to enable.

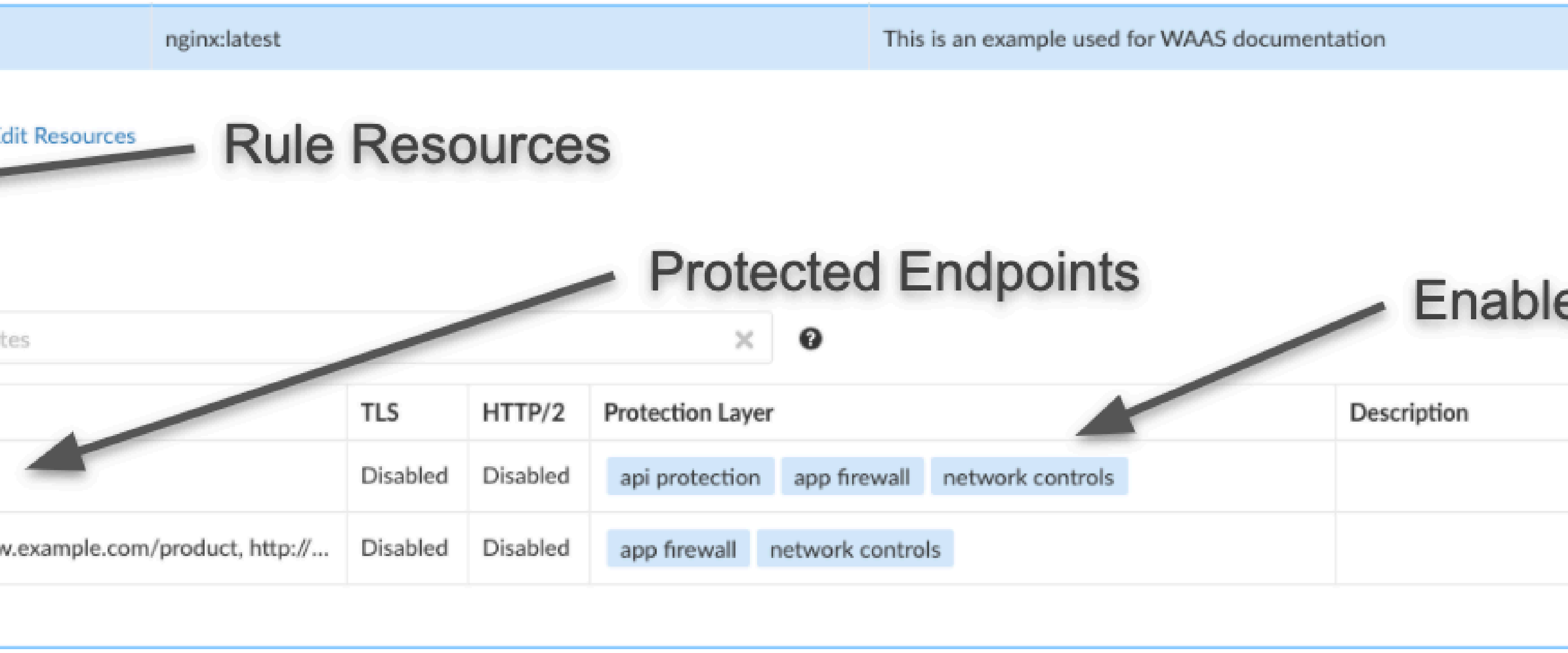
STEP 7 | Continue to **DoS protection** tab and configure DoS protection thresholds.

STEP 8 | Continue to **Bot protection** tab and select bot protections to enable.

STEP 9 | Click **Save**.

STEP 10 | You should be redirected to the **Rule Overview** page.

Select the new rule to display **Rule Resources** and for each application a list of **protected endpoints** and **enabled protections**.



STEP 11 | Test protected container using the following sanity tests.

STEP 12 | Go to **Monitor > Events**, click on **WAAS for App-Embedded** and observe the events generated.

For more information please see the WAAS analytics help page

Deploy WAAS for serverless functions

Edit on GitHub

Create a WAAS rule for serverless

When Serverless Defender is embedded in a function, it offers built-in web application firewall (WAF) capabilities, including protection against:

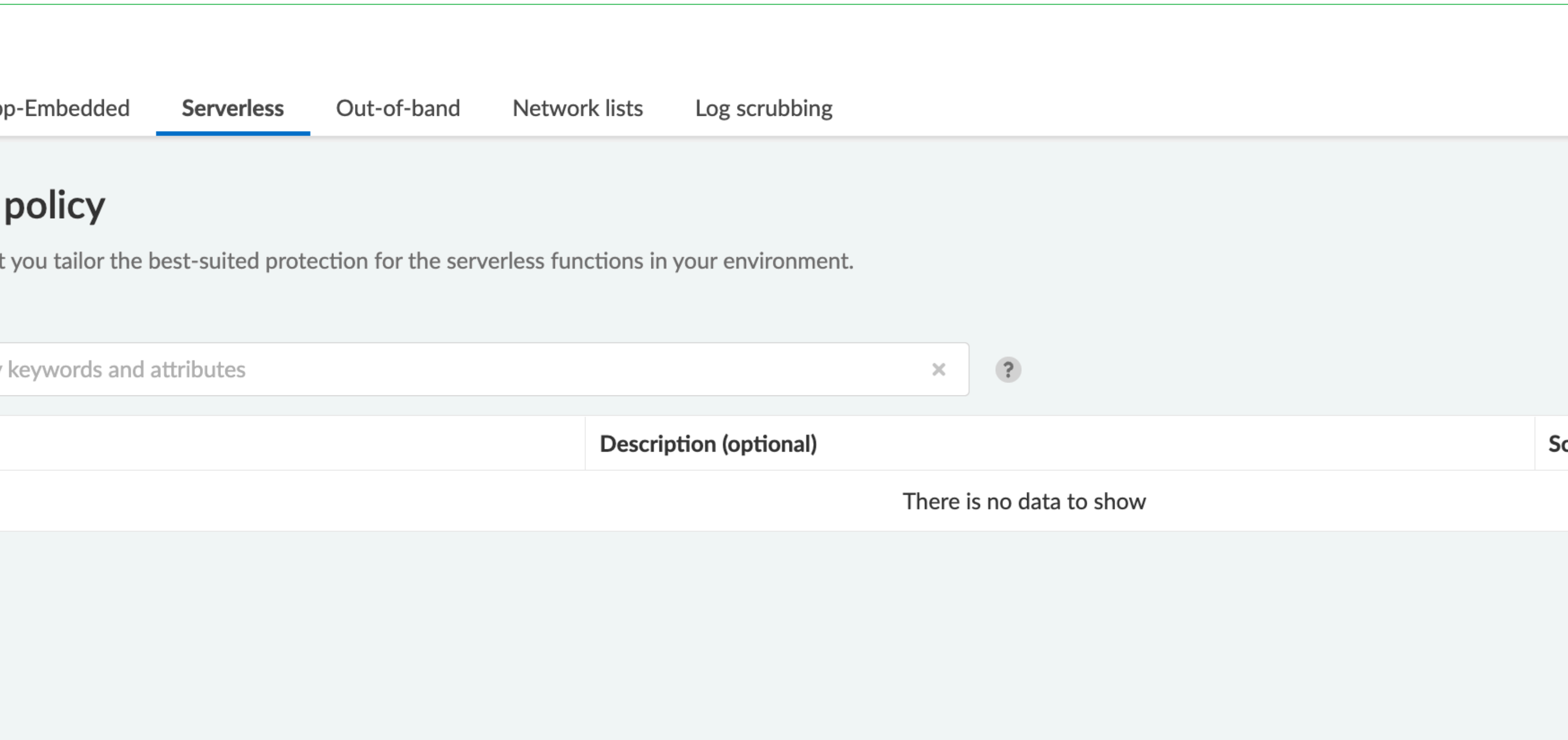
- SQL injection (SQLi) attacks
- Cross-site scripting (XSS) attacks
- Command injection (CMDi) attacks

- Local file system inclusion (LFI) attacks
- Code injection attacks

Some protections are not available for WAAS serverless deployment.

Prerequisites: You already embedded Serverless Defender into your function.

STEP 1 | Open Console and go to **Defend > WAAS > Serverless**.



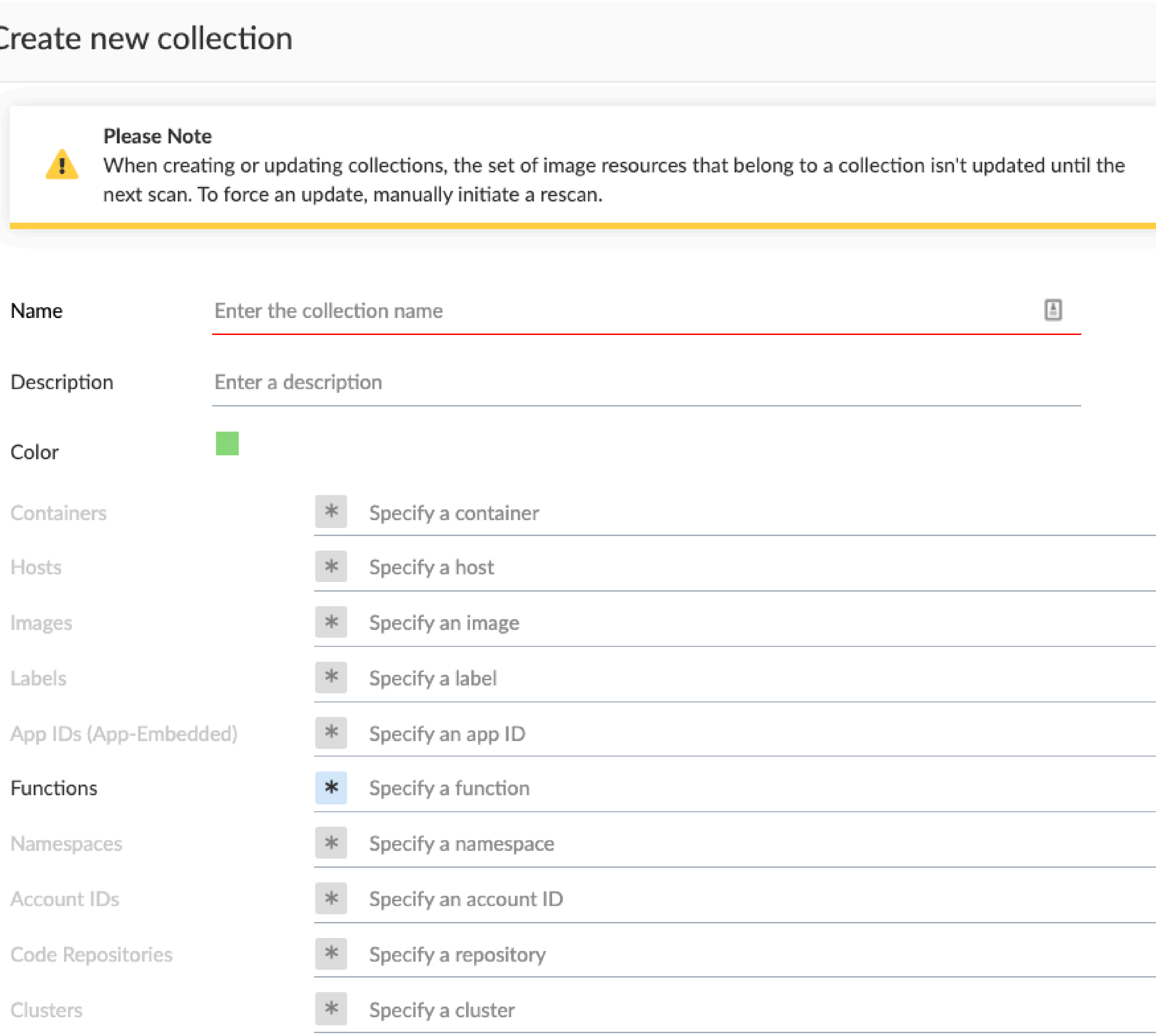
STEP 2 | Click **Add rule**.

STEP 3 | Enter a rule name.

STEP 4 | Choose the rule **Scope** by specifying the resource collection(s) to which it applies.

Collections define a combination of functions to which WAAS should attach itself to protect the web application:

Use pattern matching to precisely target your rule.



STEP 5 | Select the protections to enable.

Firewall settings

Protection	Effect
SQLi attack protection	Disable Alert Prevent
XSS attack protection	Disable Alert Prevent
CMDi attack protection	Disable Alert Prevent
LFI attack protection	Disable Alert Prevent
Code injection attack protection	Disable Alert Prevent

STEP 6 | Select **Alert** or **Prevent**.

STEP 7 | If necessary, adjust the **Proxy timeout**

The maximum duration in seconds for reading the entire request, including the body. A 500 error response is returned if a request is not read within the timeout period. For applications dealing with large files, adjusting the proxy timeout is necessary.

Deploy WAAS Out-of-band

Edit on GitHub

Out-of-band WAAS rules inspect HTTP requests and responses through a mirror of the traffic to provide WAAS detections. VPC traffic mirroring can mirror the traffic for Out-of-band inspection to Prisma Cloud Compute Defenders without additional configurations.

In Out-of-band mode, WAAS does not proxy traffic to or from the protected application and all the detections are applied on a read-only copy of the traffic. As a result, there is no risk of interfering with the application flow.

Prerequisites

- You have installed a Container Defender in your workload environment.
- The minimum version of Console and Defender is 22.06.

  Out-Of-Band WAAS is not supported on earlier versions of Console and Defender.

Create a WAAS rule for Out-of-band network traffic

To deploy WAAS for Out-of-band network traffic, create a new rule, define application endpoints, and select protections.

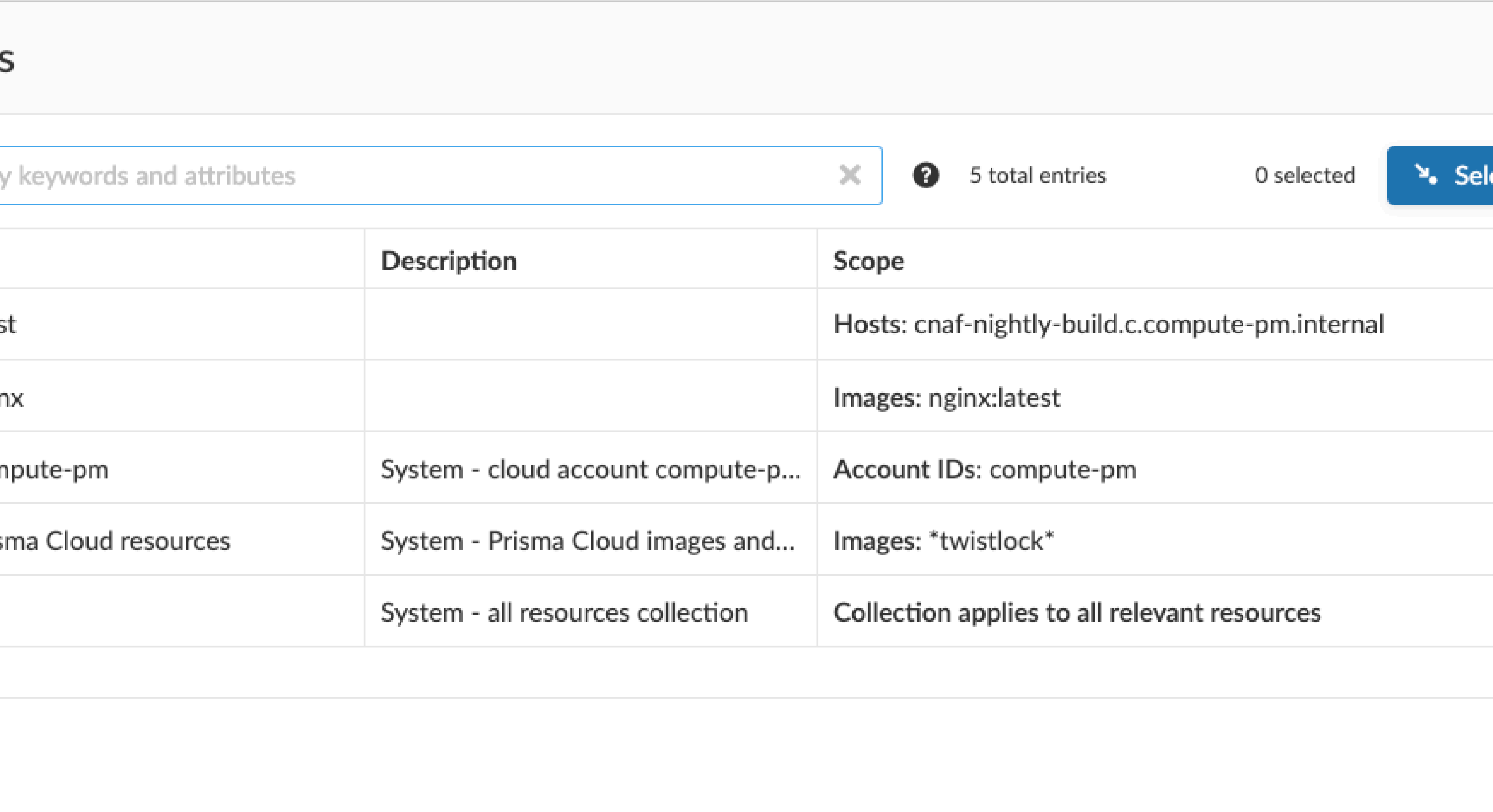
STEP 1 | Open Console, and go to **Defend > WAAS**.

STEP 2 | Select **Out-of-band**.

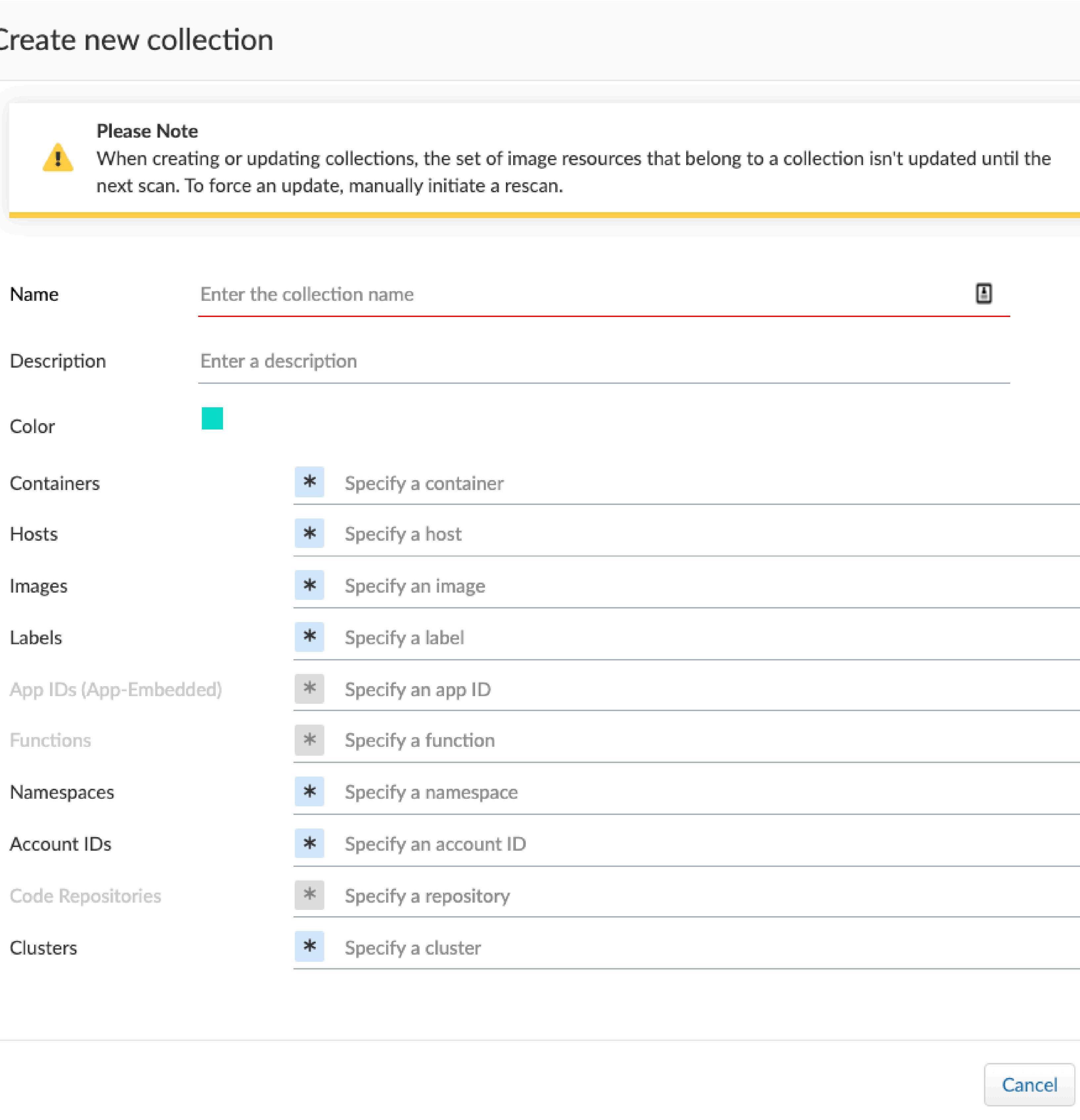
STEP 3 | Click **Add rule**.

STEP 4 | Enter a **Rule Name** and **Notes** (Optional) for describing the rule.

STEP 5 | Choose the rule **Scope** by specifying the resource collection(s) to which it applies.



Collections define a combination of image names and one or more elements to which WAAS should attach itself to protect the web application:


Create new collection
Please Note
When creating or updating collections, the set of image resources that belong to a collection isn't updated until the next scan. To force an update, manually initiate a rescan.
Name
Enter the collection name
Description
Enter a description
Color
Containers
Specify a container
Hosts
Specify a host
Images
Specify an image
Labels
Specify a label
App IDs (App-Embedded)
Specify an app ID
Functions
Specify a function
Namespaces
Specify a namespace
Account IDs
Specify an account ID
Code Repositories
Specify a repository
Clusters
Specify a cluster
Cancel

STEP 6 | (Optional) Enable **Automatically detect ports** for an endpoint to deploy the WAAS's protection on ports identified in the un-protected web apps report in **Monitor > WAAS > Unprotected web apps** for each of the workloads in the rule scope.

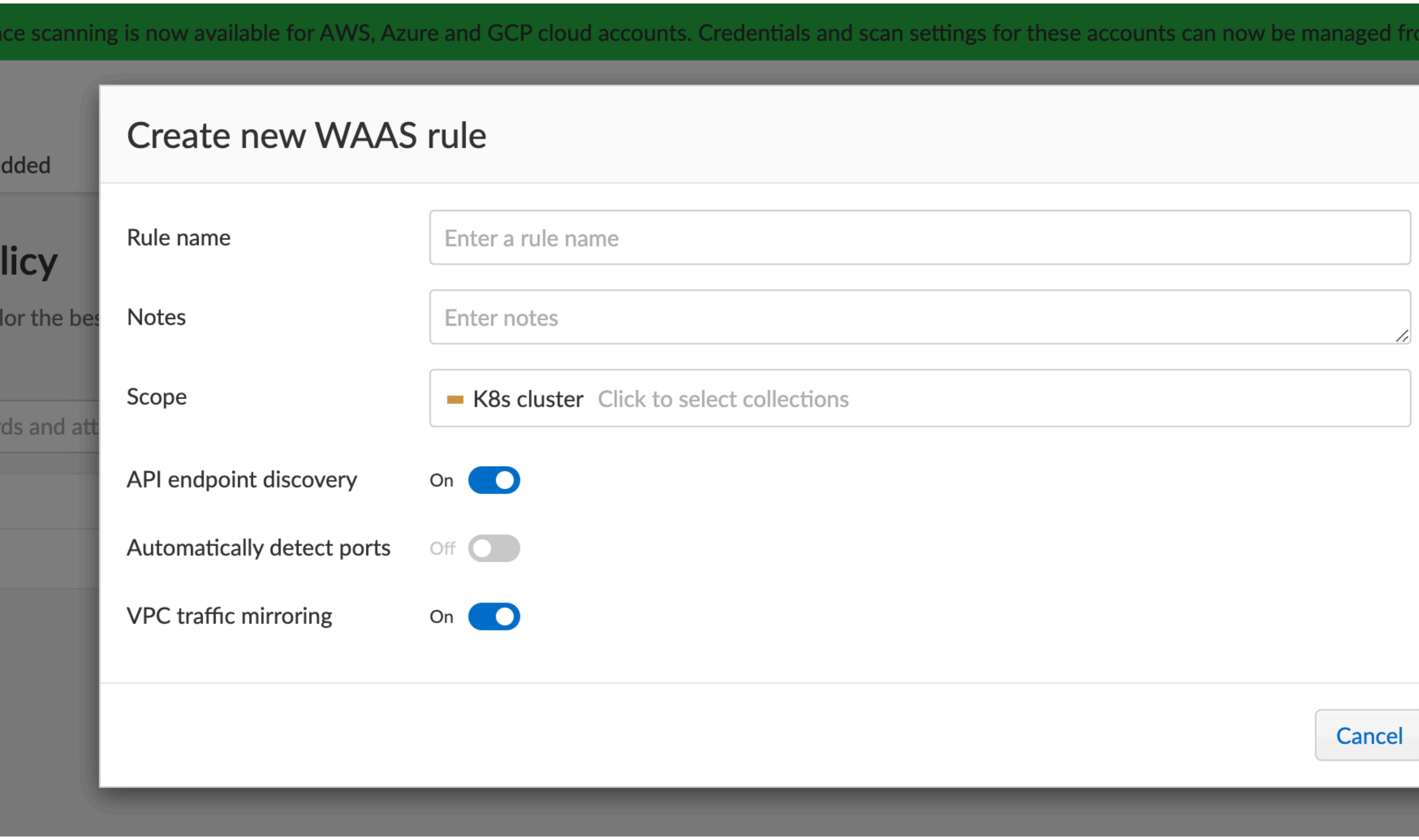
As an additional measure, you can specify additional ports by specifying them in the protected HTTP endpoints within each app to also include the ports that may not have been detected automatically.

STEP 7 | (Optional) Enable **API endpoint discovery**

When enabled, the Defender inspects the API traffic to and from the protected API. Defender reports a list of the endpoints and their resource path in **Compute > Monitor > WAAS > API observations > Out-of-band observations**.

*By enabling both **Automatically detect ports** and **API endpoint discovery**, you can monitor your API endpoints and ports without having to add an application and without configuring any policies.*

STEP 8 | (Optional) Enable **VPC traffic mirroring** when using **WAAS Out-of-band with VPC traffic mirroring** setup.



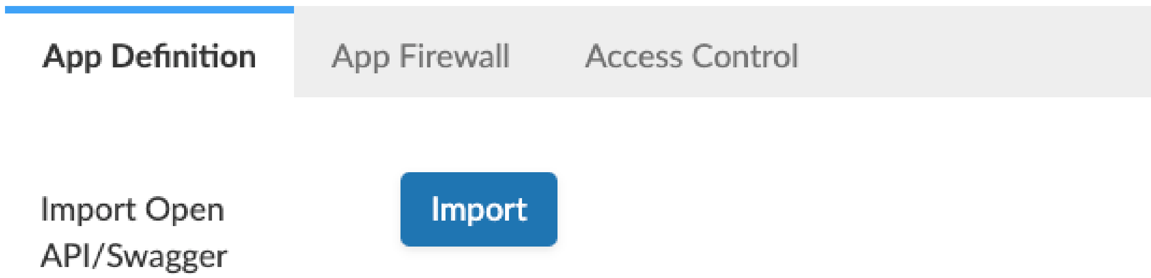
STEP 9 | **Save** the rule.

Add an App (policy) to the rule

STEP 1 | Select a WAAS rule to add an App in.

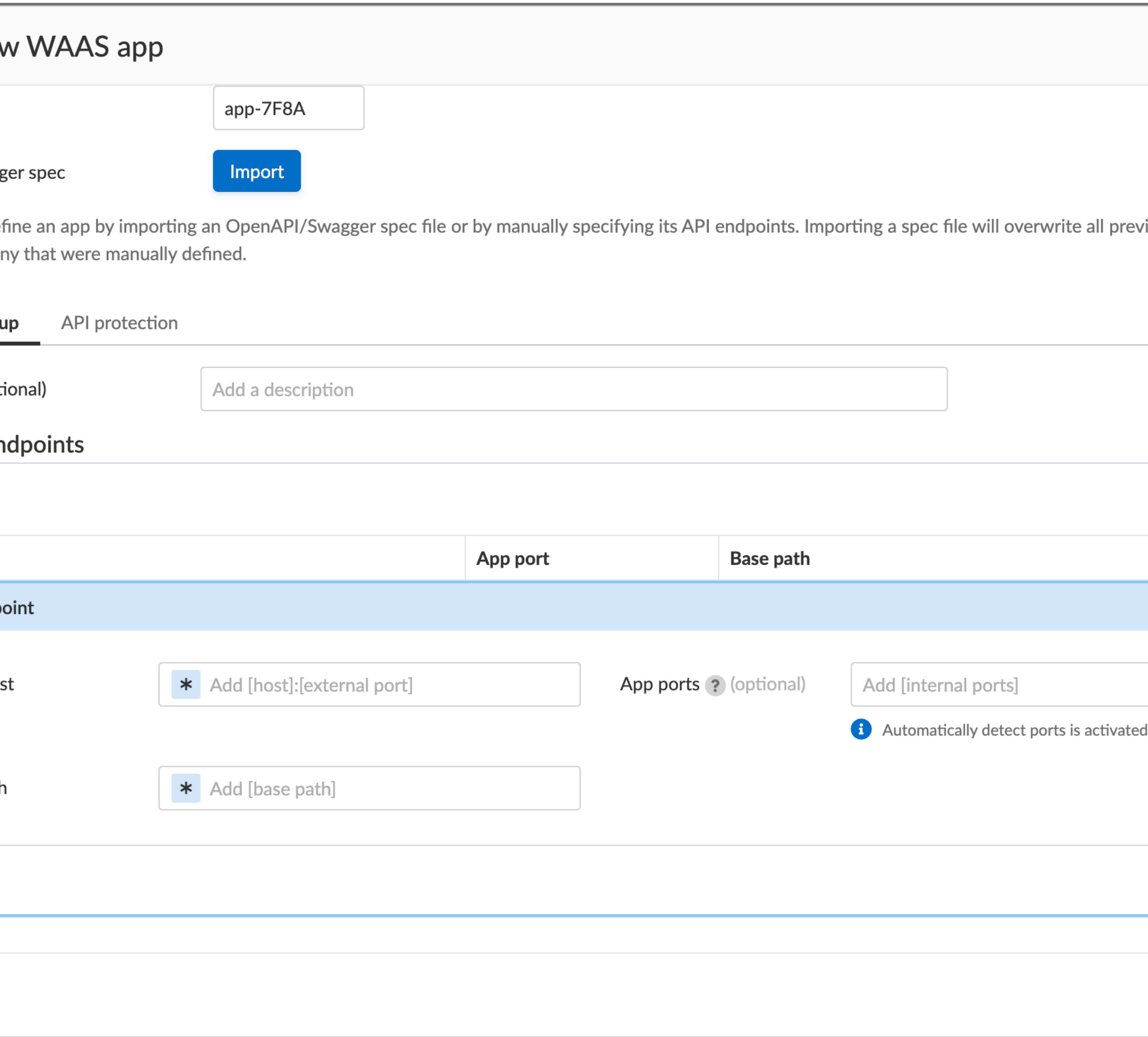
STEP 2 | Click **Add app**.

STEP 3 | In the **App Definition** tab, specify the endpoints in your web application that should be protected. Each defined application can have multiple protected endpoints. If you have a Swagger or OpenAPI file, click **Import**, and select the file to load. Otherwise, skip to the next step to manually define your application's endpoints.



STEP 4 | If you do not have a Swagger or OpenAPI file, manually define each endpoint by specifying the host, port, and path.

1. In **Endpoint Setup**, click **Add Endpoint**.
2. Specify endpoint details:



3. Enter **Port** (optional, if you selected **Automatically detect ports** while creating the rule). When **Automatically detect ports** is selected, any ports specified in a protected endpoint definition will be appended to the list of protected ports.

Specify the TCP port listening for inbound HTTP traffic.

4. Enter **HTTP host** (optional, wildcards supported).

   HTTP host names are specified in the form of [hostname]:[external port].

   External port is defined as the TCP port on the host, listening for inbound HTTP traffic.

5. Enter **Base path** (optional, wildcards supported):

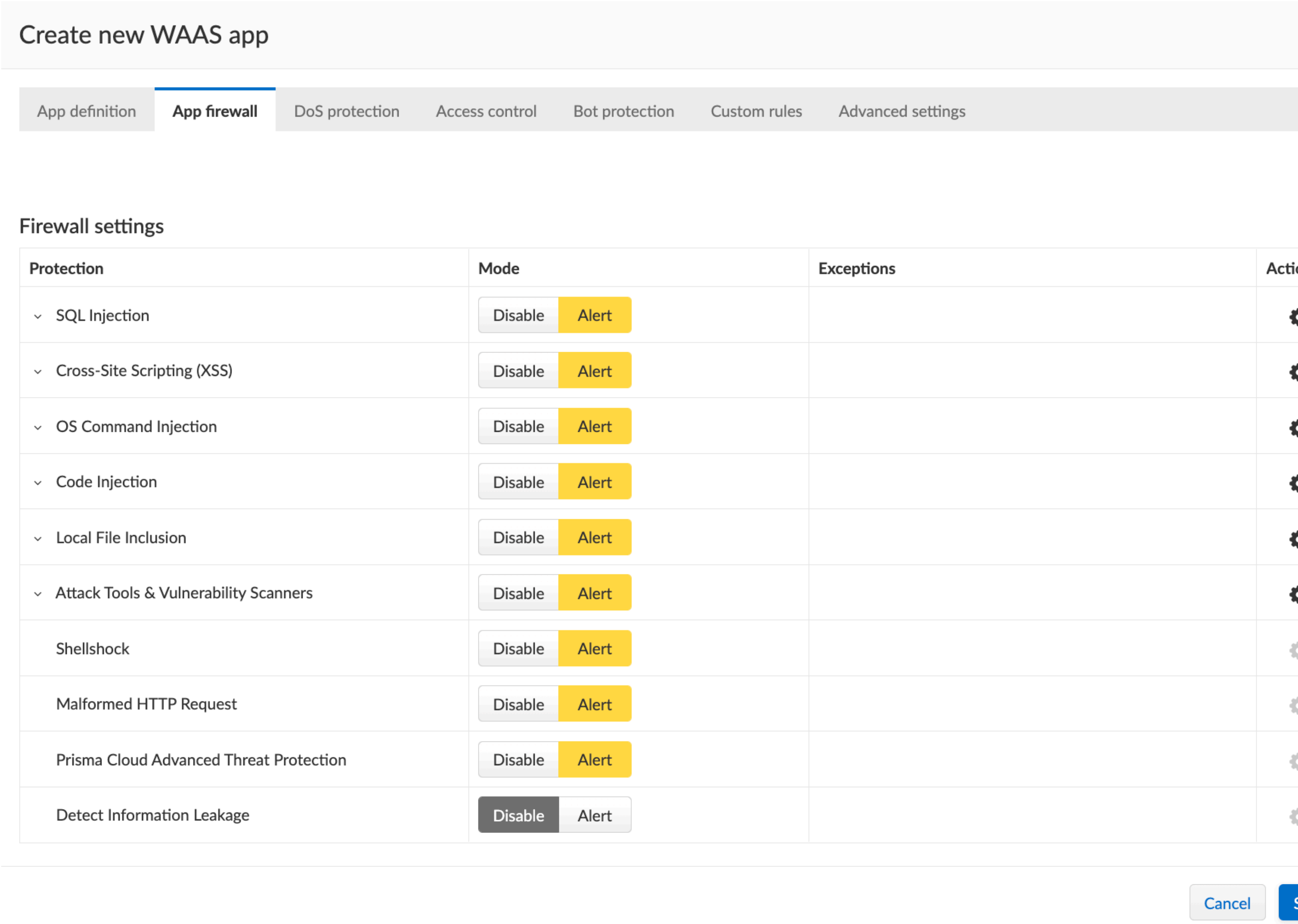
   Base path for WAAS to match on, when applying protections.

   Examples: "/admin", "/" (root path only), "/*", /v2/api", etc.

6. Click **Create**

7. If your application requires API protection, select the "API Protection" tab and define for each path the allowed methods, parameters, types, etc. See detailed definition instructions in the API protection help page.

STEP 5 | Continue to **App Firewall** tab, and select the protections as shown in the screenshot below:

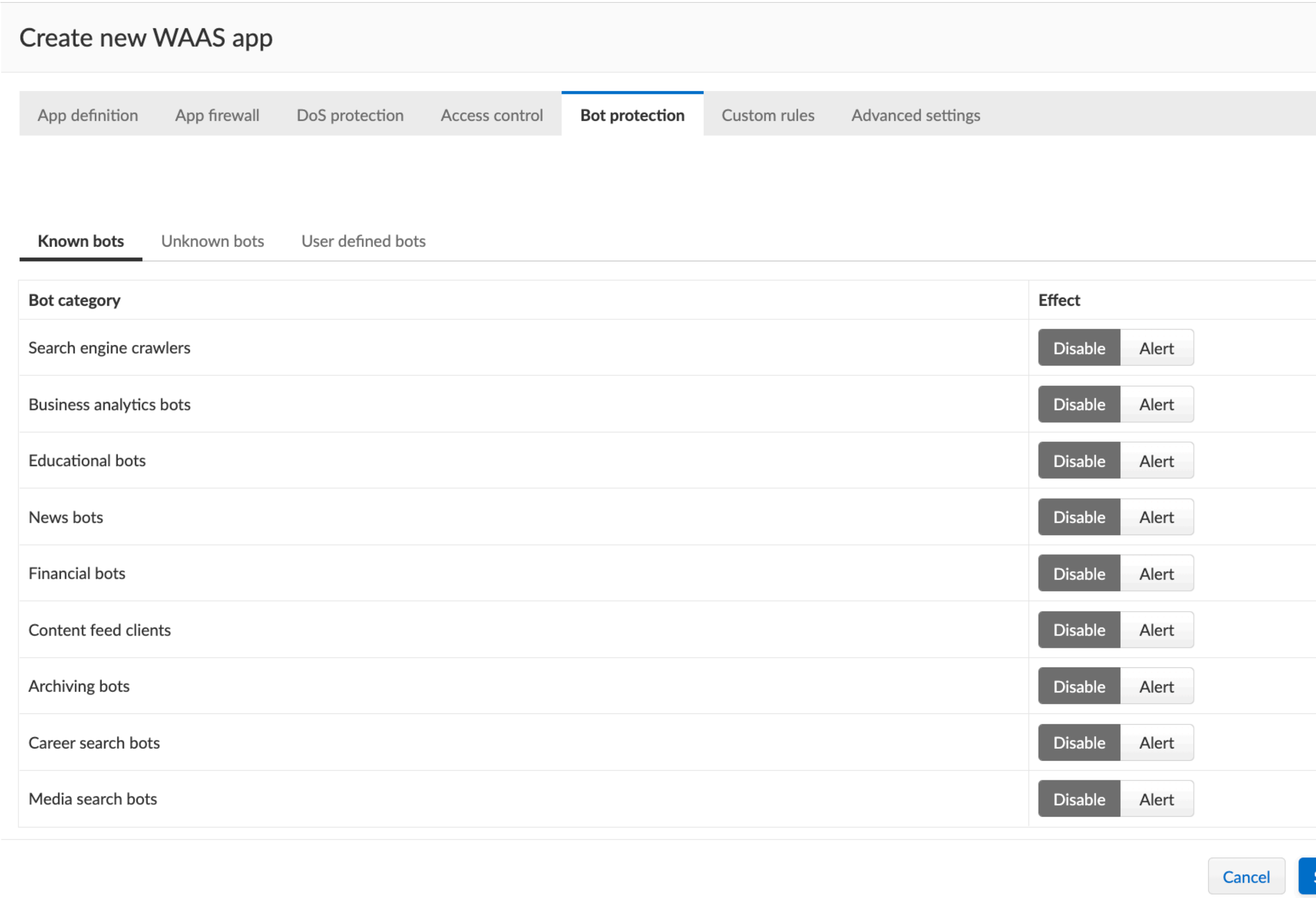


For more information, see App Firewall settings.

STEP 6 | Continue to **DoS protection** tab, and select DoS protection to enable.

STEP 7 | Continue to **Access Control** tab, and select access controls to enable.

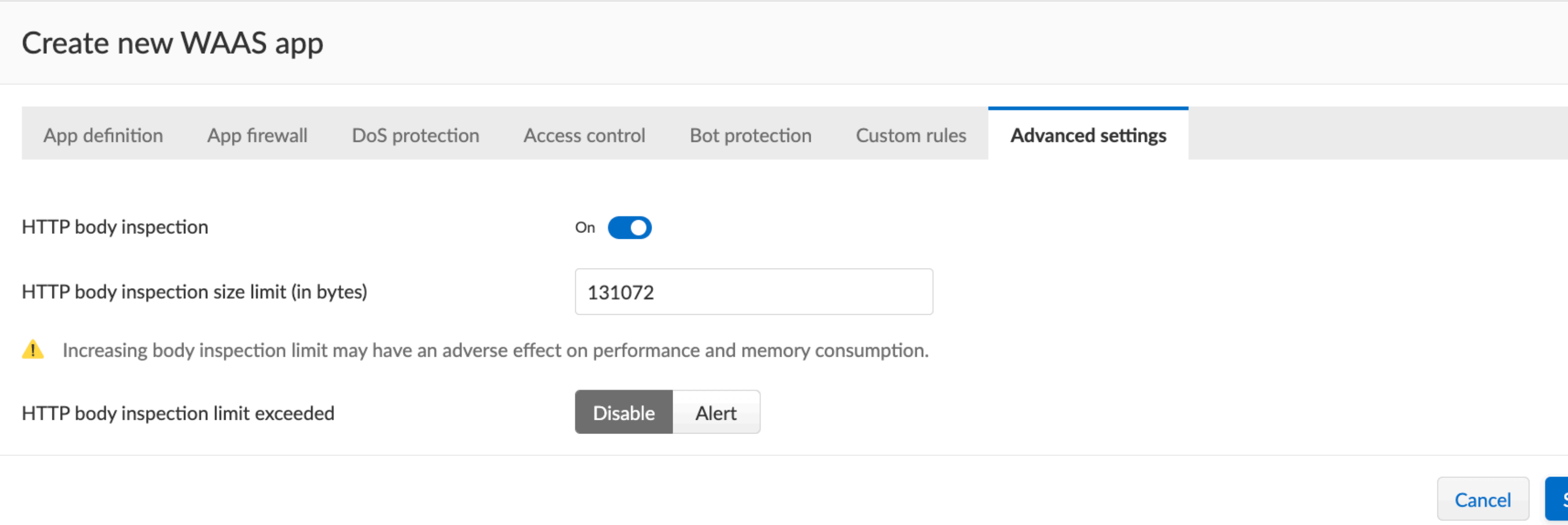
STEP 8 | Continue to **Bot protection** tab, and select the protections as shown in the screenshot below:



For more information, see Bot protections.

STEP 9 | Continue to **Custom rules** tab and select Custom rules to enable.

STEP 10 | Continue to **Advanced settings** tab, and set the options shown in the screenshot below:

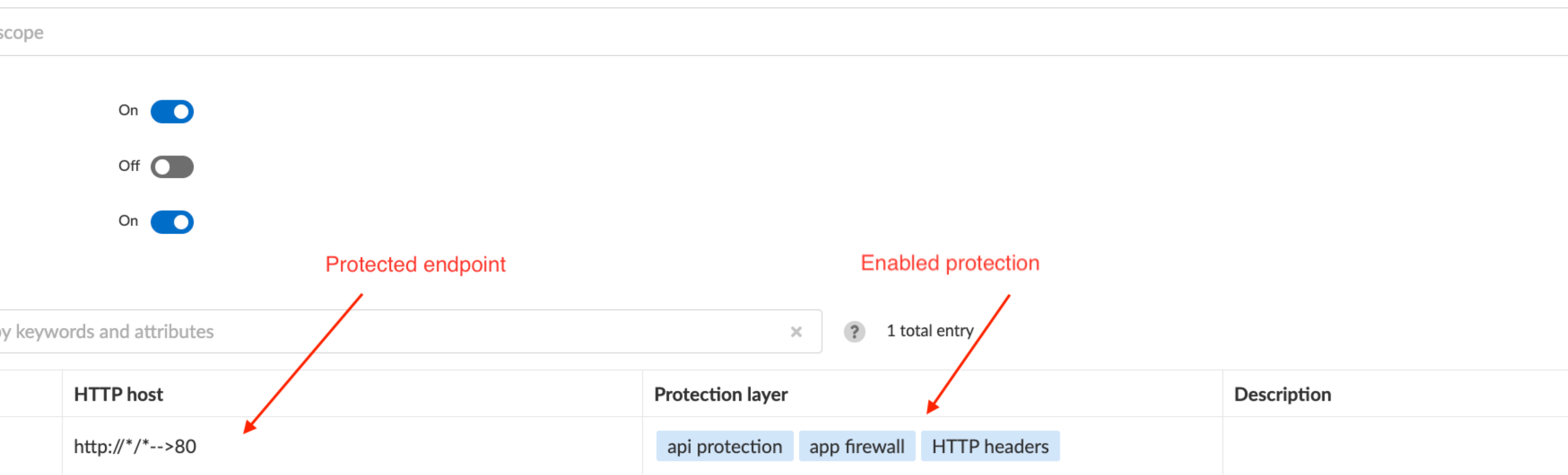


For more information, see Advanced settings.

STEP 11 | Click **Save**.

STEP 12 | You should be redirected to the **Rule Overview** page.

Select the created new rule to display **Rule Resources** and for each application a list of **protected endpoints** and **enabled protections** are displayed.



STEP 13 | Test protected endpoint using the following sanity tests.

STEP 14 | Go to **Monitor > Events**, click on **WAAS for out-of-band** and observe the events generated.

For more information, see the WAAS analytics help page

WAAS Actions for Out-of-band traffic

The following actions are applicable for the HTTP requests or responses related to the **Out-of-band traffic**:

- **Alert** - An audit is generated for visibility.
- **Disable** - The WAAS action is disabled.

Deploy WAAS Out-of-band with VPC Traffic Mirroring

Edit on GitHub

Out-of-band WAAS rules inspect HTTP requests and responses via a mirror of the traffic to provide WAAS detections. VPC traffic mirroring feature can mirror the traffic for Out-of-band inspection to Prisma Cloud Compute Defenders. In Out-of-band mode, WAAS does not proxy traffic to or from the protected application and all the detections are applied on a read-only copy of the traffic. As a result, there is no risk of interfering with the application flow.

WAAS can observe a mirror of HTTP traffic flowing to and from CSP (AWS) instances even if they are not protected by a Prisma Cloud Compute Defender.

Prerequisites

To enable Out-of-band protection using VPC traffic mirroring, deploy one or more Prisma Cloud Compute agents on the target instance on which the traffic will be mirrored.x The agents deployed for Out-of-band traffic mirror are termed Observers. The target instance is configured on a separate instance within the same VPC to receive Out-of-band traffic from the unprotected applications on the source instance. These Observers on the target instance inspect Out-of-band traffic and send audits of any events they identify to the console. For more information, see the CloudFormation traffic mirroring examples section.

NOTE:

- Deployed Observers should have connectivity to Prisma Cloud Compute console.

  Console and the Observers must be running 22.06 version or later.
- Monitoring applications Out-Of-Band via VPC traffic mirroring is subject to limitations, quotas, and checksum offloading as defined in the AWS documentation.

Deploy WAAS Out-of-band with AWS VPC Traffic Mirroring

STEP 1 | Create a CloudFormation template to deploy Prisma Cloud Observer(s) and establish VPC traffic mirroring sessions. Please see the CloudFormation traffic mirroring examples section below.

STEP 2 | Create a WAAS rule for Out-of-band network traffic and enable **VPC traffic mirroring** to allow the mirrored traffic to flow from the source instance to the Prisma Cloud Observer deployed on the target instance.

STEP 3 | Specify the instance name of the Prisma Cloud Observer created in the CloudFormation template.

Create a WAAS rule for Out-of-band network traffic

To deploy WAAS for Out-of-band network traffic, create a new rule, define application endpoints, and select protections.

STEP 1 | Open Console, and go to **Defend > WAAS**.

STEP 2 | Select **Out-of-band**.

STEP 3 | Click **Add rule**.

STEP 4 | Enter a **Rule Name** and **Notes** (Optional) for describing the rule.

STEP 5 | Choose the rule **Scope** by specifying a collection containing the instance names of the Prisma Cloud Observers created in the AWS account as part of the CloudFormation template.

STEP 6 | (Optional) Toggle to enable **API endpoint discovery**.

When enabled, the Observer inspects the mirrored traffic to and from the remote applications. The Observer reports a list of the endpoints and their resource path in **Compute > Monitor > WAAS > API observations > Out-of-band observations**.

STEP 7 | Toggle to enable **VPC traffic monitoring** to allow the mirrored traffic to flow from the source instance to the Prisma Cloud Observer, which is deployed on the target instance.

*Ports cannot be auto-detected when using **VPC traffic mirroring** because no agent is directly deployed on the source workload and the traffic is routed to the Prisma Cloud Observer through the CSP's traffic mirroring service.*

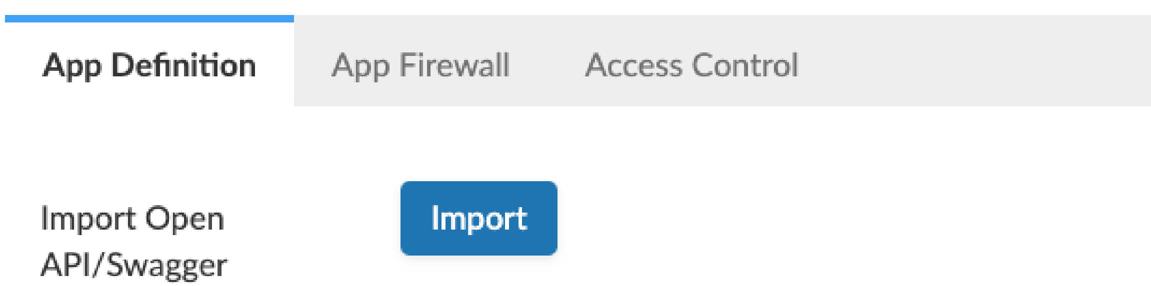
STEP 8 | **Save** the rule.

Add an App (policy) to the rule

STEP 1 | Select a WAAS rule to add an App in.

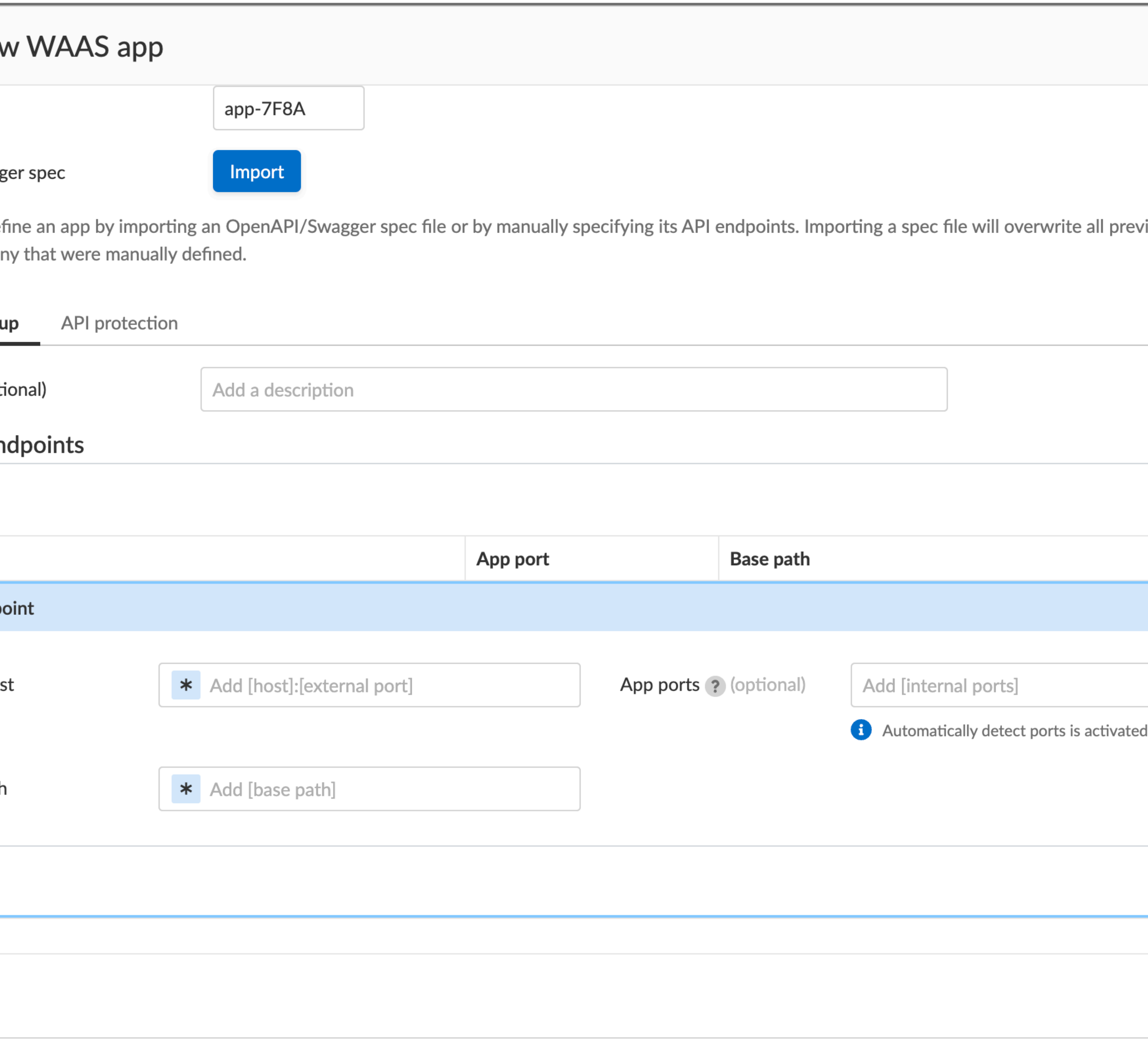
STEP 2 | Click **Add app**.

STEP 3 | In the **App Definition** tab, specify the endpoints in your web application that should be protected. Each defined application can have multiple protected endpoints. If you have a Swagger or OpenAPI file, click **Import**, and select the file to load. Otherwise, skip to the next step to manually define your application's endpoints.



STEP 4 | If you do not have a Swagger or OpenAPI file, manually define each endpoint by specifying the host, port, and path.

1. In the **Endpoint Setup** tab, click **Add Endpoint**.
2. Specify endpoint details:



3. Enter **Port**.

   Specify the TCP port listening for inbound HTTP traffic.

4. Enter **HTTP host** (optional, wildcards supported).

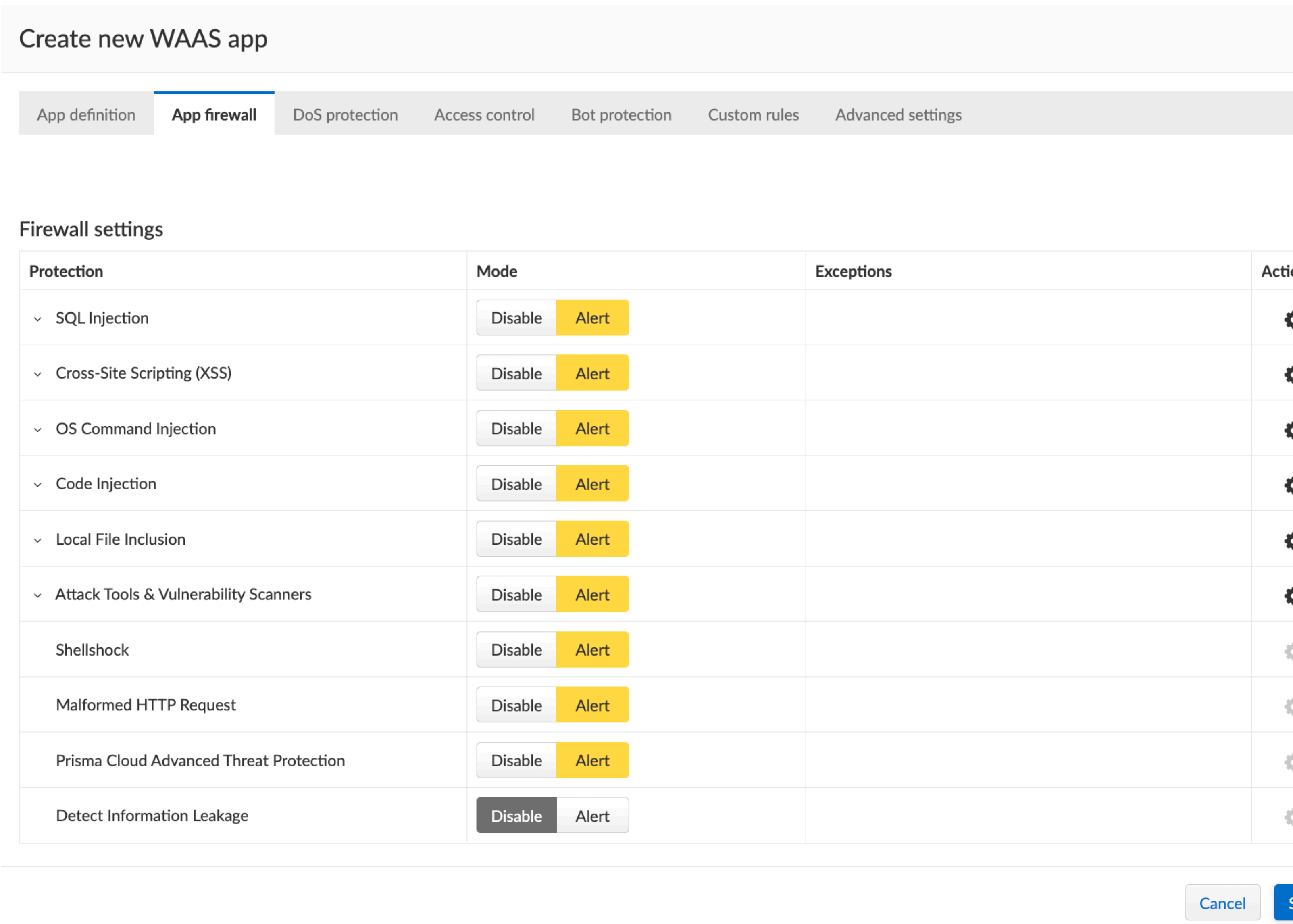
   HTTP host names are specified in the form of [hostname]:[external port].

   External port is defined as the TCP port on the host, listening for inbound HTTP traffic.
5. Enter **Base path** (optional, wildcards supported):

   Base path for WAAS to match on, when applying protections.

   Examples: "/admin", "/" (root path only), "/*", /v2/api", etc.
6. Click **Create**
7. If your application requires API protection, select the "API Protection" tab and define for each path the allowed methods, parameters, types, etc. See detailed definition instructions in the API protection help page.

STEP 5 | Continue to **App Firewall** tab, and select the protections as shown in the screenshot below:

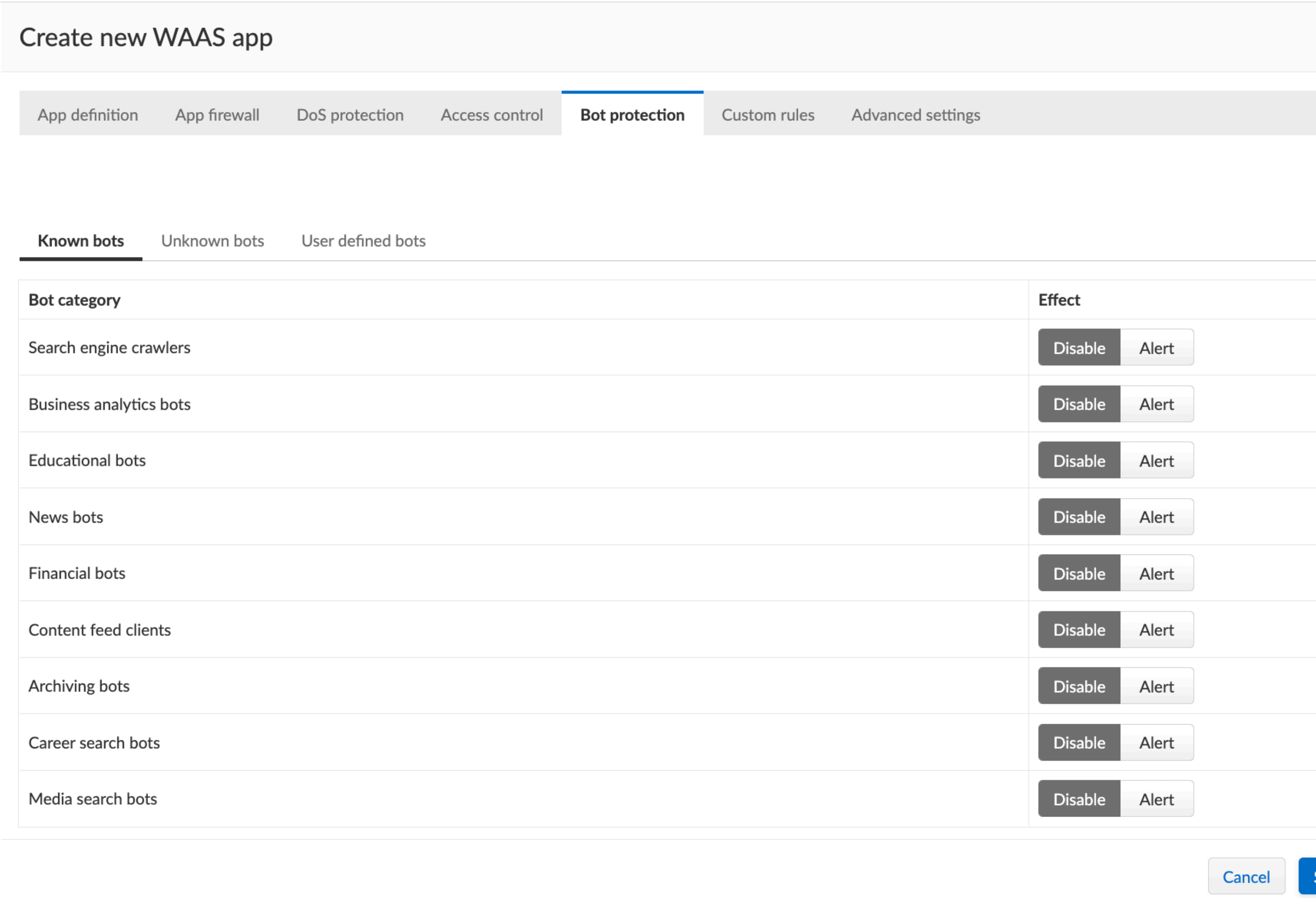


For more information, see App Firewall settings.

STEP 6 | Continue to **DoS protection** tab and select DoS protection to enable.

STEP 7 | Continue to **Access Control** tab and select access controls to enable.

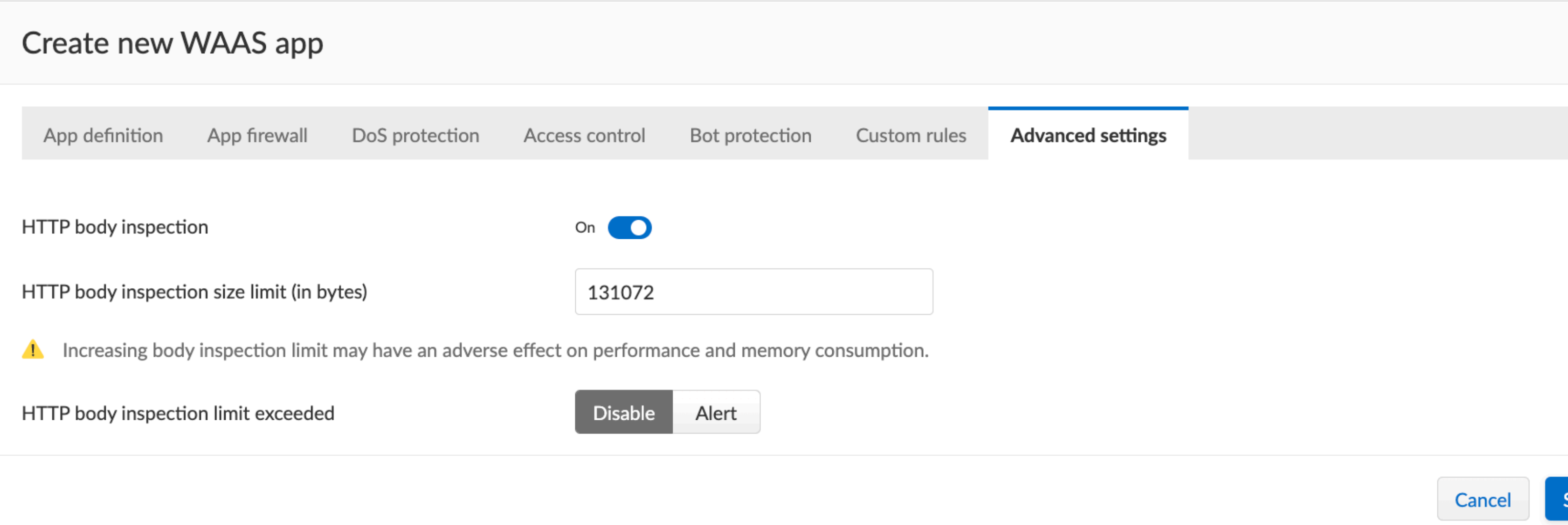
STEP 8 | Continue to **Bot protection** tab, and select the protections as shown in the screenshot below:



For more information, see Bot protections.

STEP 9 | Continue to **Custom rules** tab and select Custom rules to enable.

STEP 10 | Continue to **Advanced settings** tab, and set the options shown in the screenshot below:

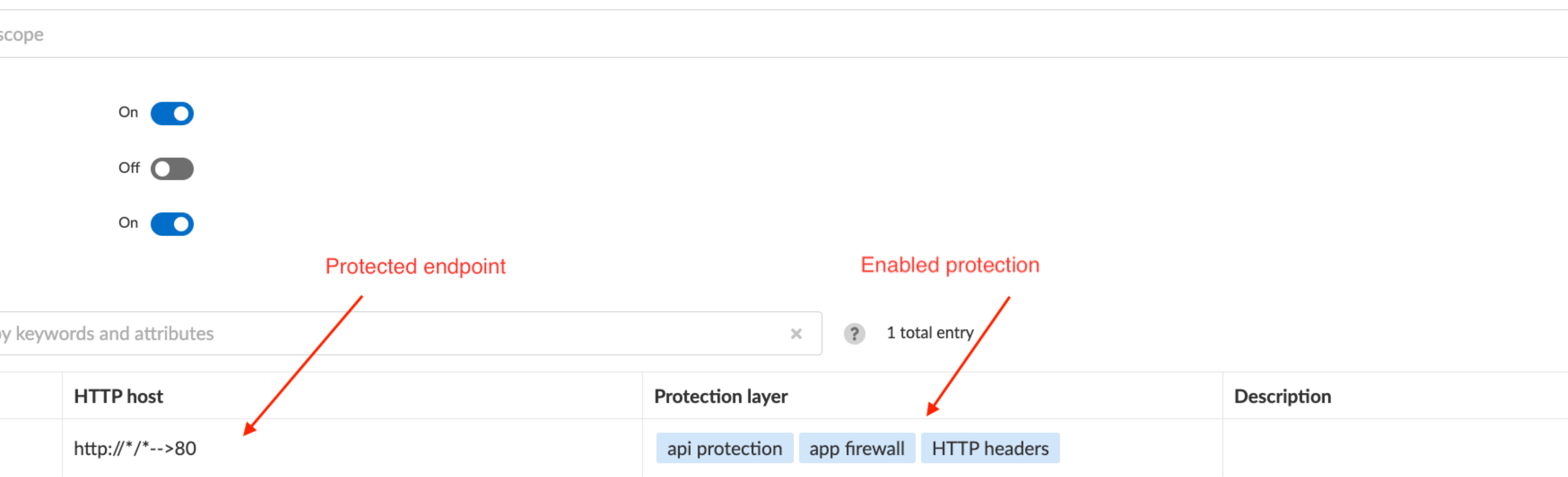


For more information, see Advanced settings.

STEP 11 | Click **Save**.

STEP 12 | You should be redirected to the **Rule Overview** page.

Select the created new rule to display **Rule Resources** and for each application a list of **protected endpoints** and **enabled protections**.



STEP 13 | Test protected endpoint using the following sanity tests.

STEP 14 | Go to **Monitor > Events**, click on **WAAS for Out-of-band** and observe the events generated.

For more information, see the WAAS analytics help page

WAAS Actions for Out-of-band traffic

The following actions are applicable for the HTTP requests or responses related to the **Out-of-band traffic**:

- **Alert** - An audit is generated for visibility.
- **Disable** - The WAAS action is disabled.

CloudFormation traffic mirroring examples

CloudFormation template for mirroring traffic between an HTTP server and a single observer

```
AWSTemplateFormatVersion: '2010-09-09'

Description: Example of CloudFormation template for mirroring traffic
 between an HTTP server and a single observer.

Parameters:
  VpcId:
    Type: AWS::EC2::VPC::Id
    Description: Specify the VPC for the environment.
    ConstraintDescription: Must be the VPC Id of an existing Virtual
 Private Cloud.
  SubnetId:
    Type: AWS::EC2::Subnet::Id
    Description: The ID of the Subnet for the environment.
    ConstraintDescription: must be the Subnet Id of an existing
 Subnet that resides in the selected Virtual Private Cloud.
  DefenderInstanceType:
    Description: EC2 instance type for the defender.
    Type: String
    Default: t3.small
    AllowedValues: [
      t3.nano, t3.micro, t3.small, t3.medium, t3.large, t3.xlarge,
 t3.2xlarge,

      m5.large, m5.xlarge, m5.2xlarge, m5.4xlarge, m5.8xlarge,
 m5.12xlarge, m5.16xlarge, m5.24xlarge,
      m5n.large, m5n.xlarge, m5n.2xlarge, m5n.4xlarge, m5n.8xlarge,
 m5n.12xlarge, m5n.16xlarge, m5n.24xlarge,
    ]
    ConstraintDescription: must be a valid EC2 instance type.
  DefenderDiskVolumeSize:
    Default: 20
    Description: Disk volume size in GB. Must be at least 20.
    ConstraintDescription: Must be a number greater or equal to 20
    MinValue: 20
    Type: Number
  DefenderDeploymentScript:
    Description: The command to run for deploying the defender
    Type: String
    AllowedPattern: 'curl.*/api/v1/scripts/defender\.sh.*'
    ConstraintDescription: must be the script to install a Defender
 on host provided by the console
  HttpServersInstanceType:
    Description: EC2 instance type for the http servers.
```

```
    Type: String
    Default: t3.small
    # t2 instance types cannot be mirrored
    AllowedValues: [
      t3.nano, t3.micro, t3.small, t3.medium, t3.large, t3.xlarge,
 t3.2xlarge,

      m5.large, m5.xlarge, m5.2xlarge, m5.4xlarge, m5.8xlarge,
 m5.12xlarge, m5.16xlarge, m5.24xlarge,
      m5n.large, m5n.xlarge, m5n.2xlarge, m5n.4xlarge, m5n.8xlarge,
 m5n.12xlarge, m5n.16xlarge, m5n.24xlarge,
    ]
    ConstraintDescription: Must be a valid EC2 instance type.
  KeyName:
    Description: The name of the EC2 Key Pair to allow SSH access to
 the EC2 instances.
    Type: 'String'
    AllowedPattern : '.+'
    ConstraintDescription: Must be the name of an existing EC2
 KeyPair.
  SSHLocation:
    Description: The IP address range that can be used to SSH to the
 EC2 instances.
    Type: String
    MinLength: '0'
    MaxLength: '18'
    AllowedPattern: '((\d{1,3})\.(\d{1,3})\.(\d{1,3})\.(\d{1,3})/
(\d{1,2}))'
    ConstraintDescription: Must be a valid IP CIDR range of the form
 x.x.x.x/x.
  HttpClientsLocation:
    Description: The IP address range of the HTTP clients making
 requests to the HTTP server.
    Type: String
    MinLength: '0'
    MaxLength: '18'
    AllowedPattern: '((\d{1,3})\.(\d{1,3})\.(\d{1,3})\.(\d{1,3})/
(\d{1,2}))'
    ConstraintDescription: Must be a valid IP CIDR range of the form
 x.x.x.x/x.
  MirroredHostsCIDR:
    Description: The IP address range of the mirrored hosts.
    Type: String
    MinLength: '9'
    MaxLength: '18'
    AllowedPattern: '(\d{1,3})\.(\d{1,3})\.(\d{1,3})\.(\d{1,3})/
(\d{1,2})'
    ConstraintDescription: Must be a valid IP CIDR range of the form
 x.x.x.x/x.
  DefenderAmiIdX86:
    Description: DO NOT change this parameter. The image to use for
 the Defender, default is latest Amazon Linux 2 AMI.
    Type: 'AWS::SSM::Parameter::Value<AWS::EC2::Image::Id>'
    Default: '/aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-
x86_64-gp2'
```

```
    ConstraintDescription: 'only use /aws/service/ami-amazon-linux-
latest/amzn2-ami-hvm-x86_64-gp2'
  HttpServersAmiIdX86:
    Description: DO NOT change this parameter. The image to use for
 the HTTP Servers, Default is Ubuntu Server 20.04 AMI.
    Type: 'AWS::SSM::Parameter::Value<AWS::EC2::Image::Id>'
    Default: '/aws/service/canonical/ubuntu/server/20.04/
stable/20211129/amd64/hvm/ebs-gp2/ami-id'
    ConstraintDescription: 'Only use Ubuntu Server images'

Metadata:
  AWS::CloudFormation::Interface:
    ParameterGroups:
      -
        Label:
          Default: "Network"
        Parameters:
          - VpcId
          - SubnetId
      -
        Label:
          default: "Instances"
        Parameters:
          - DefenderInstanceType
          - DefenderDiskVolumeSize
          - DefenderDeploymentScript
          - HttpServersInstanceType
          - KeyName
          - SSHLocation
          - HttpClientsLocation
          - MirroredHostsCIDR
      -
        Label:
          default: "Do NOT change these"
        Parameters:
          - DefenderAmiIdX86
          - HttpServersAmiIdX86

Resources:
  DefenderSecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: Defender Security Group
      SecurityGroupIngress:
        - IpProtocol: udp
          FromPort: 4789
          ToPort: 4789
          CidrIp: !Ref MirroredHostsCIDR
          Description: Mirrored traffic
        - IpProtocol: tcp
          FromPort: 4789
          ToPort: 4789
          CidrIp: !Ref MirroredHostsCIDR
          Description: Health checks
        - IpProtocol: tcp
          FromPort: 22
```

```
          ToPort: 22
          CidrIp: !Ref SSHLocation
          Description: SSH
      VpcId: !Ref VpcId
      Tags:
        - Key: "Name"
          Value: !Join [ "", [ {Ref: AWS::StackName}, "-defender-
sg" ]]

  DefenderNetworkInterface:
    Type: AWS::EC2::NetworkInterface
    Properties:
      Description: Defender network interface
      GroupSet:
        - !GetAtt DefenderSecurityGroup.GroupId
      SubnetId: !Ref SubnetId

  Defender:
    Type: AWS::EC2::Instance
    Properties:
      ImageId: !Ref DefenderAmiIdX86
      InstanceType: !Ref DefenderInstanceType
      KeyName: !Ref KeyName
      BlockDeviceMappings:
        -
          DeviceName: /dev/xvda
          Ebs:
            VolumeSize: !Ref DefenderDiskVolumeSize
            VolumeType: gp2
      NetworkInterfaces:
        - NetworkInterfaceId: !Ref DefenderNetworkInterface
          DeviceIndex: '0'
      UserData:
        Fn::Base64: !Sub |
          #!/bin/bash
          ${DefenderDeploymentScript}
      Tags:
        - Key: "Name"
          Value: !Join [ "", [ {Ref: AWS::StackName}, "-defender" ]]

  HttpServer1SecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: Http Server 1 Security Group
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: 80
          ToPort: 80
          CidrIp: !Ref HttpClientsLocation
          Description: Web traffic
        - IpProtocol: tcp
          FromPort: 22
          ToPort: 22
          CidrIp: !Ref SSHLocation
          Description: SSH
      VpcId: !Ref VpcId
```

```
      Tags:
        - Key: "Name"
          Value: !Join [ "", [ {Ref: AWS::StackName}, "-http-server1-
sg" ]]

  HttpServer1NetworkInterface:
    Type: AWS::EC2::NetworkInterface
    Properties:
      Description: HTTP server network interface
      GroupSet:
        - !GetAtt HttpServer1SecurityGroup.GroupId
      SubnetId: !Ref SubnetId

  HttpServer1:
    Type: AWS::EC2::Instance
    Properties:
      ImageId: !Ref HttpServersAmiIdX86
      InstanceType: !Ref HttpServersInstanceType
      KeyName: !Ref KeyName
      NetworkInterfaces:
        - NetworkInterfaceId: !Ref HttpServer1NetworkInterface
          DeviceIndex: '0'
      UserData:
        Fn::Base64: !Sub |
          #!/bin/bash
          apt update -y
          apt install -y nginx libnginx-mod-http-echo
          cat > /etc/nginx/sites-enabled/default <<EOF
          server {
            listen 80 default_server;
            root /var/www/html;
            index index.html index.htm index.nginx-debian.html;
            server_name _;
            location ~ /echo.* {
              default_type text/plain;
              echo_duplicate 1 \$echo_client_request_headers;
              echo "\r";
              echo_read_request_body;
              echo \$request_body;
              echo \$hostname;
            }
            location ~ /json.* {
              default_type application/json;
              echo '{ "name":"nginx" }\r';
            }
            location / {
              try_files \$uri \$uri/ =404;
            }
          }
          EOF
          systemctl enable nginx
          systemctl restart nginx
      Tags:
        - Key: "Name"
          Value: !Join [ "", [ {Ref: AWS::StackName}, "-http-
server1" ]]
```

```
  HttpServer2SecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: Http Server 2 Security Group
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: 8080
          ToPort: 8080
          CidrIp: !Ref HttpClientsLocation
          Description: Web traffic
        - IpProtocol: tcp
          FromPort: 22
          ToPort: 22
          CidrIp: !Ref SSHLocation
          Description: SSH
      VpcId: !Ref VpcId
      Tags:
        - Key: "Name"
          Value: !Join [ "", [ {Ref: AWS::StackName}, "-http-server2-
sg" ]]

  HttpServer2NetworkInterface:
    Type: AWS::EC2::NetworkInterface
    Properties:
      Description: HTTP server network interface
      GroupSet:
        - !GetAtt HttpServer2SecurityGroup.GroupId
      SubnetId: !Ref SubnetId

  HttpServer2:
    Type: AWS::EC2::Instance
    Properties:
      ImageId: !Ref HttpServersAmiIdX86
      InstanceType: !Ref HttpServersInstanceType
      KeyName: !Ref KeyName
      NetworkInterfaces:
        - NetworkInterfaceId: !Ref HttpServer2NetworkInterface
          DeviceIndex: '0'
      UserData:
        Fn::Base64: !Sub |
          #!/bin/bash
          apt update -y
          apt install -y nginx libnginx-mod-http-echo
          cat > /etc/nginx/sites-enabled/default <<EOF
          server {
            listen 8080 default_server;
            root /var/www/html;
            index index.html index.htm index.nginx-debian.html;
            server_name _;
            location ~ /echo.* {
              default_type text/plain;
              echo_duplicate 1 \$echo_client_request_headers;
              echo "\r";
              echo_read_request_body;
              echo \$request_body;
```



```
              echo \$hostname;
            }
            location ~ /json.* {
              default_type application/json;
              echo '{ "name":"nginx" }\r';
            }
            location / {
              try_files \$uri \$uri/ =404;
            }
          }
          EOF
          systemctl enable nginx
          systemctl restart nginx
      Tags:
        - Key: "Name"
          Value: !Join [ "", [ {Ref: AWS::StackName}, "-http-
server2" ]]

  TrafficMirrorTarget:
    Type: AWS::EC2::TrafficMirrorTarget
    # DefenderNetworkInterface has to be connected to Defender first
    DependsOn: Defender
    Properties:
      NetworkInterfaceId: !Ref DefenderNetworkInterface
      Tags:
        - Key: "Name"
          Value: !Join [ "", [ {Ref: AWS::StackName}, "-mirror-
target" ]]

  TrafficMirrorFilter1:
    Type: AWS::EC2::TrafficMirrorFilter
    Properties:
      Tags:
        - Key: "Name"
          Value: !Join [ "", [ {Ref: AWS::StackName}, "-mirror-
filter1" ]]

  TrafficMirrorFilter1IngressRule:
    Type: AWS::EC2::TrafficMirrorFilterRule
    Properties:
      SourceCidrBlock: 0.0.0.0/0
      DestinationCidrBlock: 0.0.0.0/0
      DestinationPortRange:
        FromPort: 80
        ToPort: 80
      Protocol: 6
      RuleAction: accept
      RuleNumber: 100
      TrafficDirection: ingress
      TrafficMirrorFilterId: !Ref TrafficMirrorFilter1

  TrafficMirrorFilter1EgressRule:
    Type: AWS::EC2::TrafficMirrorFilterRule
    Properties:
      SourceCidrBlock: 0.0.0.0/0
      DestinationCidrBlock: 0.0.0.0/0
```

```
      SourcePortRange:
        FromPort: 80
        ToPort: 80
      Protocol: 6
      RuleAction: accept
      RuleNumber: 100
      TrafficDirection: egress
      TrafficMirrorFilterId: !Ref TrafficMirrorFilter1

  TrafficMirrorSession1:
    Type: AWS::EC2::TrafficMirrorSession
    # HttpServer1NetworkInterface has to be connected to HttpServer1
 first
    DependsOn: HttpServer1
    Properties:
      NetworkInterfaceId: !Ref HttpServer1NetworkInterface
      SessionNumber: 1
      TrafficMirrorFilterId: !Ref TrafficMirrorFilter1
      TrafficMirrorTargetId: !Ref TrafficMirrorTarget
      VirtualNetworkId: 1
      Tags:
        - Key: "Name"
          Value: !Join [ "", [ {Ref: AWS::StackName}, "-mirror-
session1" ]]

  TrafficMirrorFilter2:
    Type: AWS::EC2::TrafficMirrorFilter
    Properties:
      Tags:
        - Key: "Name"
          Value: !Join [ "", [ {Ref: AWS::StackName}, "-mirror-
filter2" ]]

  TrafficMirrorFilter2IngressRule:
    Type: AWS::EC2::TrafficMirrorFilterRule
    Properties:
      SourceCidrBlock: 0.0.0.0/0
      DestinationCidrBlock: 0.0.0.0/0
      DestinationPortRange:
        FromPort: 8080
        ToPort: 8080
      Protocol: 6
      RuleAction: accept
      RuleNumber: 100
      TrafficDirection: ingress
      TrafficMirrorFilterId: !Ref TrafficMirrorFilter2

  TrafficMirrorFilter2EgressRule:
    Type: AWS::EC2::TrafficMirrorFilterRule
    Properties:
      SourceCidrBlock: 0.0.0.0/0
      DestinationCidrBlock: 0.0.0.0/0
      SourcePortRange:
        FromPort: 8080
        ToPort: 8080
      Protocol: 6
```



```
      RuleAction: accept
      RuleNumber: 100
      TrafficDirection: egress
      TrafficMirrorFilterId: !Ref TrafficMirrorFilter2

  TrafficMirrorSession2:
    Type: AWS::EC2::TrafficMirrorSession
    # HttpServer2NetworkInterface has to be connected to HttpServer2
 first
    DependsOn: HttpServer2
    Properties:
      NetworkInterfaceId: !Ref HttpServer2NetworkInterface
      SessionNumber: 2
      TrafficMirrorFilterId: !Ref TrafficMirrorFilter2
      TrafficMirrorTargetId: !Ref TrafficMirrorTarget
      VirtualNetworkId: 1
      Tags:
        - Key: "Name"
          Value: !Join [ "", [ {Ref: AWS::StackName}, "-mirror-
session2" ]]

Outputs:
  DefenderHostName:
    Description: The Defender private hostname
    Value: !GetAtt Defender.PrivateDnsName
  DefenderPublicIP:
    Description: The Defender public IP
    Value: !GetAtt Defender.PublicIp
  HttpServer1PublicIP:
    Description: The HTTP server 1 public IP
    Value: !GetAtt HttpServer1.PublicIp
  HttpServer2PublicIP:
    Description: The HTTP server 2 public IP
    Value: !GetAtt HttpServer2.PublicIp
```

CloudFormation template for mirroring traffic between an HTTP server and multiple observers behind AWS Network Load Balance

```
AWSTemplateFormatVersion: '2010-09-09'

Description: Example of CloudFormation template used to mirror
 traffic between an HTTP server and multiple Observers behind an AWS
 Network Load Balance.

Parameters:
  VpcId:
    Type: AWS::EC2::VPC::Id
    Description: Specify the VPC for the environment.
    ConstraintDescription: Must be the VPC Id of an existing Virtual
 Private Cloud.
  SubnetId:
    Type: AWS::EC2::Subnet::Id
    Description: The ID of the Subnet for the environment.
    ConstraintDescription: must be the Subnet Id of an existing
 Subnet that resides in the selected Virtual Private Cloud.
  DefenderInstanceType:
```

```
    Description: EC2 instance type for the defender.
    Type: String
    Default: t3.small
    AllowedValues: [
      t3.nano, t3.micro, t3.small, t3.medium, t3.large, t3.xlarge,
 t3.2xlarge,

      m5.large, m5.xlarge, m5.2xlarge, m5.4xlarge, m5.8xlarge,
 m5.12xlarge, m5.16xlarge, m5.24xlarge,
      m5n.large, m5n.xlarge, m5n.2xlarge, m5n.4xlarge, m5n.8xlarge,
 m5n.12xlarge, m5n.16xlarge, m5n.24xlarge,
    ]
    ConstraintDescription: must be a valid EC2 instance type.
  DefenderDiskVolumeSize:
    Default: 20
    Description: Disk volume size in GB. Must be at least 20.
    ConstraintDescription: Must be a number greater or equal to 20
    MinValue: 20
    Type: Number
  DefenderDeploymentScript:
    Description: The command to run for deploying the defender
    Type: String
    AllowedPattern: 'curl.*/api/v1/scripts/defender\.sh.*'
    ConstraintDescription: must be the script to install a Defender
 on host provided by the console
  HttpServerInstanceType:
    Description: EC2 instance type for the http server.
    Type: String
    Default: t3.small
    # t2 instance types cannot be mirrored
    AllowedValues: [
      t3.nano, t3.micro, t3.small, t3.medium, t3.large, t3.xlarge,
 t3.2xlarge,

      m5.large, m5.xlarge, m5.2xlarge, m5.4xlarge, m5.8xlarge,
 m5.12xlarge, m5.16xlarge, m5.24xlarge,
      m5n.large, m5n.xlarge, m5n.2xlarge, m5n.4xlarge, m5n.8xlarge,
 m5n.12xlarge, m5n.16xlarge, m5n.24xlarge,
    ]
    ConstraintDescription: Must be a valid EC2 instance type.
  KeyName:
    Description: The name of the EC2 Key Pair to allow SSH access to
 the EC2 instances.
    Type: 'String'
    AllowedPattern : '.+'
    ConstraintDescription: Must be the name of an existing EC2
 KeyPair.
  SSHLocation:
    Description: The IP address range that can be used to SSH to the
 EC2 instances.
    Type: String
    MinLength: '0'
    MaxLength: '18'
    AllowedPattern: '((\d{1,3})\.(\d{1,3})\.(\d{1,3})\.(\d{1,3})/
(\d{1,2}))'
```



```
    ConstraintDescription: Must be a valid IP CIDR range of the form
 x.x.x.x/x.
  HttpClientsLocation:
    Description: The IP address range of the HTTP clients making
 requests to the HTTP server.
    Type: String
    MinLength: '0'
    MaxLength: '18'
    AllowedPattern: '((\d{1,3})\.(\d{1,3})\.(\d{1,3})\.(\d{1,3})/
(\d{1,2}))'
    ConstraintDescription: Must be a valid IP CIDR range of the form
 x.x.x.x/x.
  MirroredHostsCIDR:
    Description: The IP address range of the mirrored hosts.
    Type: String
    MinLength: '9'
    MaxLength: '18'
    AllowedPattern: '(\d{1,3})\.(\d{1,3})\.(\d{1,3})\.(\d{1,3})/
(\d{1,2})'
    ConstraintDescription: Must be a valid IP CIDR range of the form
 x.x.x.x/x.
  DefenderAmiIdX86:
    Description: DO NOT change this parameter. The image to use for
 the Defender, default is latest Amazon Linux 2 AMI.
    Type: 'AWS::SSM::Parameter::Value<AWS::EC2::Image::Id>'
    Default: '/aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-
x86_64-gp2'
    ConstraintDescription: 'only use /aws/service/ami-amazon-linux-
latest/amzn2-ami-hvm-x86_64-gp2'
  HttpServerAmiIdX86:
    Description: DO NOT change this parameter. The image to use for
 the HTTP Server, Default is Ubuntu Server 20.04 AMI.
    Type: 'AWS::SSM::Parameter::Value<AWS::EC2::Image::Id>'
    Default: '/aws/service/canonical/ubuntu/server/20.04/
stable/20211129/amd64/hvm/ebs-gp2/ami-id'
    ConstraintDescription: 'Only use Ubuntu Server images'

Metadata:
  AWS::CloudFormation::Interface:
    ParameterGroups:
      -
        Label:
          Default: "Network"
        Parameters:
          - VpcId
          - SubnetId
      -
        Label:
          default: "Instances"
        Parameters:
          - DefenderInstanceType
          - DefenderDiskVolumeSize
          - DefenderDeploymentScript
          - HttpServerInstanceType
          - KeyName
          - SSHLocation
```

```
          - HttpClientsLocation
          - MirroredHostsCIDR
      -
        Label:
          default: "Do NOT change these"
        Parameters:
          - DefenderAmiIdX86
          - HttpServerAmiIdX86

Resources:
  DefenderSecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: Defender Security Group
      SecurityGroupIngress:
        - IpProtocol: udp
          FromPort: 4789
          ToPort: 4789
          CidrIp: !Ref MirroredHostsCIDR
          Description: Mirrored traffic
        - IpProtocol: tcp
          FromPort: 4789
          ToPort: 4789
          CidrIp: !Ref MirroredHostsCIDR
          Description: Health checks
        - IpProtocol: tcp
          FromPort: 22
          ToPort: 22
          CidrIp: !Ref SSHLocation
          Description: SSH
      VpcId: !Ref VpcId
      Tags:
        - Key: "Name"
          Value: !Join [ "", [ {Ref: AWS::StackName}, "-defender-
sg" ]]

  DefenderNetworkInterface:
    Type: AWS::EC2::NetworkInterface
    Properties:
      Description: Defender network interface
      GroupSet:
        - !GetAtt DefenderSecurityGroup.GroupId
      SubnetId: !Ref SubnetId

  Defender:
    Type: AWS::EC2::Instance
    Properties:
      ImageId: !Ref DefenderAmiIdX86
      InstanceType: !Ref DefenderInstanceType
      KeyName: !Ref KeyName
      BlockDeviceMappings:
        -
          DeviceName: /dev/xvda
          Ebs:
            VolumeSize: !Ref DefenderDiskVolumeSize
            VolumeType: gp2
```



```
      NetworkInterfaces:
        - NetworkInterfaceId: !Ref DefenderNetworkInterface
          DeviceIndex: '0'
      UserData:
        Fn::Base64: !Sub |
          #!/bin/bash
          ${DefenderDeploymentScript}
      Tags:
        - Key: "Name"
          Value: !Join [ "", [ {Ref: AWS::StackName}, "-defender" ]]

  HttpServerSecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: Http Server Security Group
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: 80
          ToPort: 80
          CidrIp: !Ref HttpClientsLocation
          Description: Web traffic
        - IpProtocol: tcp
          FromPort: 22
          ToPort: 22
          CidrIp: !Ref SSHLocation
          Description: SSH
      VpcId: !Ref VpcId
      Tags:
        - Key: "Name"
          Value: !Join [ "", [ {Ref: AWS::StackName}, "-http-server-
sg" ]]

  HttpServerNetworkInterface:
    Type: AWS::EC2::NetworkInterface
    Properties:
      Description: HTTP server network interface
      GroupSet:
        - !GetAtt HttpServerSecurityGroup.GroupId
      SubnetId: !Ref SubnetId

  HttpServer:
    Type: AWS::EC2::Instance
    Properties:
      ImageId: !Ref HttpServerAmiIdX86
      InstanceType: !Ref HttpServerInstanceType
      KeyName: !Ref KeyName
      NetworkInterfaces:
        - NetworkInterfaceId: !Ref HttpServerNetworkInterface
          DeviceIndex: '0'
      UserData:
        Fn::Base64: !Sub |
          #!/bin/bash
          apt update -y
          apt install -y nginx libnginx-mod-http-echo
          cat > /etc/nginx/sites-enabled/default <<EOF
          server {
```



```
              listen 80 default_server;
              root /var/www/html;
              index index.html index.htm index.nginx-debian.html;
              server_name _;
              location ~ /echo.* {
                default_type text/plain;
                echo_duplicate 1 \$echo_client_request_headers;
                echo "\r";
                echo_read_request_body;
                echo \$request_body;
                echo \$hostname;
              }
              location ~ /json.* {
                default_type application/json;
                echo '{ "name":"nginx" }\r';
              }
              location / {
                try_files \$uri \$uri/ =404;
              }
            }
            EOF
            systemctl enable nginx
            systemctl restart nginx
      Tags:
        - Key: "Name"
          Value: !Join [ "", [ {Ref: AWS::StackName}, "-http-
server" ]]

  NetworkLoadBalancerTargetGroup:
    Type: AWS::ElasticLoadBalancingV2::TargetGroup
    Properties:
      Port: 4789
      Protocol: UDP
      HealthCheckEnabled: True
      HealthCheckProtocol: TCP
      Targets:
        - Id: !Ref Defender
      VpcId: !Ref VpcId
      Name: !Join [ "", [ {Ref: AWS::StackName}, "-nlb-tg" ]]

  NetworkLoadBalancer:
    Type: AWS::ElasticLoadBalancingV2::LoadBalancer
    Properties:
      Type: network
      Scheme: internal
      Subnets:
        - !Ref SubnetId
      Name: !Join [ "", [ {Ref: AWS::StackName}, "-nlb" ]]

  NetworkLoadBalancerListener:
    Type: AWS::ElasticLoadBalancingV2::Listener
    Properties:
      LoadBalancerArn: !Ref NetworkLoadBalancer
      Port: 4789
      Protocol: UDP
      DefaultActions:
```

```
        - Type: forward
          TargetGroupArn: !Ref NetworkLoadBalancerTargetGroup

  TrafficMirrorTarget:
    Type: AWS::EC2::TrafficMirrorTarget
    DependsOn: NetworkLoadBalancerListener
    Properties:
      NetworkLoadBalancerArn: !Ref NetworkLoadBalancer
      Tags:
        - Key: "Name"
          Value: !Join [ "", [ {Ref: AWS::StackName}, "-mirror-
target" ]]

  TrafficMirrorFilter:
    Type: AWS::EC2::TrafficMirrorFilter
    Properties:
      Tags:
        - Key: "Name"
          Value: !Join [ "", [ {Ref: AWS::StackName}, "-mirror-
filter" ]]

  TrafficMirrorFilterIngressRule:
    Type: AWS::EC2::TrafficMirrorFilterRule
    Properties:
      SourceCidrBlock: 0.0.0.0/0
      DestinationCidrBlock: 0.0.0.0/0
      DestinationPortRange:
        FromPort: 80
        ToPort: 80
      Protocol: 6
      RuleAction: accept
      RuleNumber: 100
      TrafficDirection: ingress
      TrafficMirrorFilterId: !Ref TrafficMirrorFilter

  TrafficMirrorFilterEgressRule:
    Type: AWS::EC2::TrafficMirrorFilterRule
    Properties:
      SourceCidrBlock: 0.0.0.0/0
      DestinationCidrBlock: 0.0.0.0/0
      SourcePortRange:
        FromPort: 80
        ToPort: 80
      Protocol: 6
      RuleAction: accept
      RuleNumber: 100
      TrafficDirection: egress
      TrafficMirrorFilterId: !Ref TrafficMirrorFilter

  TrafficMirrorSession:
    Type: AWS::EC2::TrafficMirrorSession
    # HttpServerNetworkInterface has to be connected to HttpServer
 first
    DependsOn: HttpServer
    Properties:
      NetworkInterfaceId: !Ref HttpServerNetworkInterface
```

```
      SessionNumber: 1
      TrafficMirrorFilterId: !Ref TrafficMirrorFilter
      TrafficMirrorTargetId: !Ref TrafficMirrorTarget
      VirtualNetworkId: 1
      Tags:
        - Key: "Name"
          Value: !Join [ "", [ {Ref: AWS::StackName}, "-mirror-
session" ]]

Outputs:
  DefenderHostName:
    Description: The Defender private hostname
    Value: !GetAtt Defender.PrivateDnsName
  DefenderPublicIP:
    Description: The Defender public IP
    Value: !GetAtt Defender.PublicIp
  HttpServerPublicIP:
    Description: The HTTP server public IP
    Value: !GetAtt HttpServer.PublicIp
```

CloudFormation template for deploying a Prisma Cloud Compute console

```
AWSTemplateFormatVersion: '2010-09-09'

Description: Example of CloudFormation template used to deploy a
 Prisma Cloud Compute console.

Parameters:
  VpcId:
    Type: AWS::EC2::VPC::Id
    Description: Specify the VPC for the environment.
    ConstraintDescription: Must be the VPC Id of an existing Virtual
 Private Cloud.
  SubnetId:
    Type: AWS::EC2::Subnet::Id
    Description: The ID of the Subnet for the environment.
    ConstraintDescription: must be the Subnet Id of an existing
 Subnet that resides in the selected Virtual Private Cloud.
  ConsoleInstanceType:
    Description: EC2 instance type for the console.
    Type: String
    Default: t3.small
    AllowedValues: [
      t3.nano, t3.micro, t3.small, t3.medium, t3.large, t3.xlarge,
 t3.2xlarge,

      m5.large, m5.xlarge, m5.2xlarge, m5.4xlarge, m5.8xlarge,
 m5.12xlarge, m5.16xlarge, m5.24xlarge,
      m5n.large, m5n.xlarge, m5n.2xlarge, m5n.4xlarge, m5n.8xlarge,
 m5n.12xlarge, m5n.16xlarge, m5n.24xlarge,
    ]
    ConstraintDescription: Must be a valid EC2 instance type.
  ConsoleDiskVolumeSize:
    Default: 24
    Description: Disk volume size in GB. Must be at least 24 since
 console requires 20 GB free.
```



```
    ConstraintDescription: Must be a number greater or equal to 24
    MinValue: 24
    Type: Number
  KeyName:
    Description: The name of the EC2 Key Pair to allow SSH access to
 the EC2 instances.
    Type: 'String'
    AllowedPattern : '.+'
    ConstraintDescription: Must be the name of an existing EC2
 KeyPair.
  SSHLocation:
    Description: The IP address range that can be used to SSH to the
 EC2 instances.
    Type: String
    MinLength: '0'
    MaxLength: '18'
    AllowedPattern: '((\d{1,3})\.(\d{1,3})\.(\d{1,3})\.(\d{1,3})/
(\d{1,2}))'
    ConstraintDescription: Must be a valid IP CIDR range of the form
 x.x.x.x/x.
  ConsoleClientsLocation:
    Description: The IP address range of the clients connecting to
 the console web interface.
    Type: String
    MinLength: '0'
    MaxLength: '18'
    AllowedPattern: '((\d{1,3})\.(\d{1,3})\.(\d{1,3})\.(\d{1,3})/
(\d{1,2}))'
    ConstraintDescription: Must be a valid IP CIDR range of the form
 x.x.x.x/x.
  DefendersLocation:
    Description: The IP address range of the defenders connecting to
 the console.
    Type: String
    MinLength: '9'
    MaxLength: '18'
    AllowedPattern: '(\d{1,3})\.(\d{1,3})\.(\d{1,3})\.(\d{1,3})/
(\d{1,2})'
    ConstraintDescription: Must be a valid IP CIDR range of the form
 x.x.x.x/x.
  ConsoleAmiIdX86:
    Description: DO NOT change this parameter. The image to use for
 the Console, default is latest Amazon Linux 2 AMI.
    Type: 'AWS::SSM::Parameter::Value<AWS::EC2::Image::Id>'
    Default: '/aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-
x86_64-gp2'
    ConstraintDescription: 'only use /aws/service/ami-amazon-linux-
latest/amzn2-ami-hvm-x86_64-gp2'

Metadata:
  AWS::CloudFormation::Interface:
    ParameterGroups:
      -
        Label:
          Default: "Network"
        Parameters:
```



```
            - VpcId
            - SubnetId
        -
          Label:
            default: "Instances"
          Parameters:
            - ConsoleInstanceType
            - ConsoleDiskVolumeSize
            - KeyName
            - SSHLocation
            - ConsoleClientsLocation
            - DefendersLocation
        -
          Label:
            default: "Do NOT change these"
          Parameters:
            - ConsoleAmiIdX86

Resources:
  ConsoleSecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: Console Security Group
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: 8083
          ToPort: 8083
          CidrIp: !Ref ConsoleClientsLocation
          Description: Prisma Cloud Console UI and API
        - IpProtocol: tcp
          FromPort: 8083
          ToPort: 8083
          CidrIp: !Ref DefendersLocation
          Description: Prisma Cloud Console UI and API access from
 defender
        - IpProtocol: tcp
          FromPort: 8084
          ToPort: 8084
          CidrIp: !Ref DefendersLocation
          Description: Prisma Cloud secure websocket for Console-
Defender communication
        - IpProtocol: tcp
          FromPort: 22
          ToPort: 22
          CidrIp: !Ref SSHLocation
          Description: SSH
      VpcId: !Ref VpcId
      Tags:
        - Key: "Name"
          Value: !Join [ "", [ {Ref: AWS::StackName}, "-console-
sg" ]]

  Console:
    Type: AWS::EC2::Instance
    Properties:
      ImageId: !Ref ConsoleAmiIdX86
```



```
      InstanceType: !Ref ConsoleInstanceType
      KeyName: !Ref KeyName
      BlockDeviceMappings:
        -
          DeviceName: /dev/xvda
          Ebs:
            VolumeSize: !Ref ConsoleDiskVolumeSize
            VolumeType: gp2
      NetworkInterfaces:
        - DeviceIndex: '0'
          DeleteOnTermination: true
          GroupSet:
            - !GetAtt ConsoleSecurityGroup.GroupId
          SubnetId: !Ref SubnetId
      UserData:
        Fn::Base64: !Sub |
          #!/bin/bash
          amazon-linux-extras install -y docker
          usermod -a -G docker ec2-user
          systemctl enable docker
          systemctl restart docker
      Tags:
        - Key: "Name"
          Value: !Join [ "", [ {Ref: AWS::StackName}, "-console" ]]

Outputs:
  ConsolePublicIP:
    Description: The Console public IP
    Value: !GetAtt Console.PublicIp
```

WAAS Troubleshooting

Edit on GitHub

Troubleshooting Container or Host Rules

Follow these steps to troubleshoot WAAS issues using the table below:

STEP 1 | Ensure the protected container or host is protected by WAAS - a green firewall icon should appear next to the workload's radar entity and a "WAAS" tab should appear when clicked.

STEP 2 | Click on the workload in the radar and open *WAAS connectivity monitor* by clicking on the WAAS tab.

STEP 3 | Click on *Reset* to reset all counters.

STEP 4 | Send one or more HTTP requests to the protected application

STEP 5 | Click on *Refresh* and match changes in the request counters to the *Connectivity Monitor Indications* column in the table below

STEP 6 | If the *WAAS errors* counter has been incremented, click on *View recent errors* to view errors.

STEP 7 | A section of *troubleshooting potential outcomes* is provided below, along with possible causes and solutions.

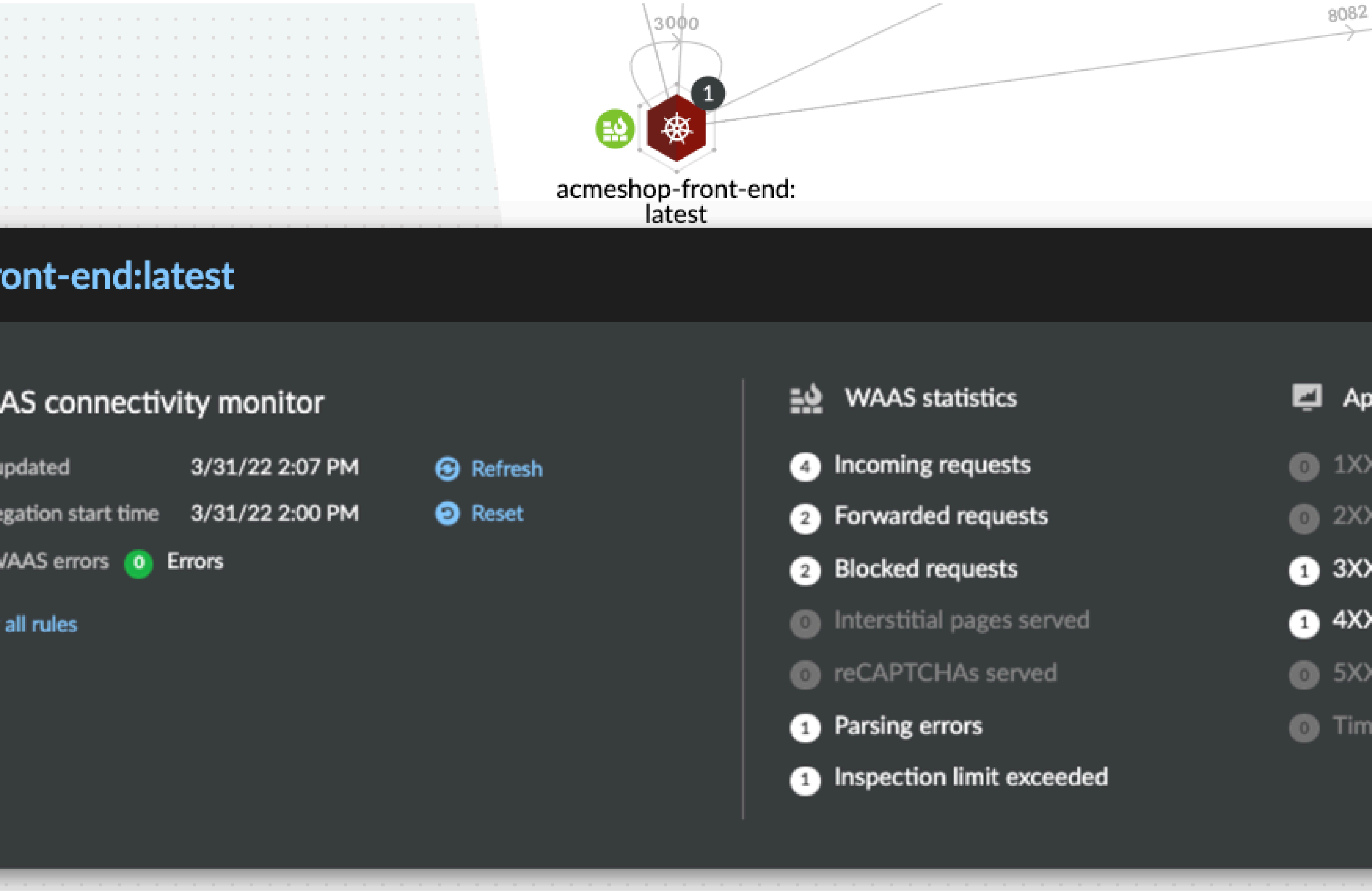
WAAS connectivity monitor

WAAS connectivity monitor monitors the connection between WAAS and the protected application.

WAAS connectivity monitor aggregates data on pages served by WAAS and the application responses.

In addition, it provides easy access to WAAS related errors registered in the Defender logs (Defenders sends logs to the Console every hour).

The monitor tab becomes available when you click on an image or host protected by WAAS.



- **Last updated** - Most recent time when WAAS monitoring data was sent from the Defenders to the Console (Defender logs are sent to the Console on an hourly basis). By clicking on the **refresh** button users can initiate sending of newer data.
- **Aggregation start time** - Time when data aggregation began. By clicking on the **reset** button users can reset all counters.
- **WAAS errors** - To view recent errors related to a monitored image or host, click the **View recent errors** link.

- **WAAS statistics:**
  - *Incoming requests* - Count of HTTP requests inspected by WAAS since the start of aggregation.
  - *Forwarded requests* - Count of HTTP requests forwarded by WAAS to the protected application.
  - *Interstitial pages served* - Count of interstitial pages served by WAAS (interstitial pages are served once Prisma Sessions Cookies are enabled).
  - *reCAPTCHAs served* - Count of reCAPTCHA challenges served by WAAS (when enabled as part of bot protection).
- **Application statistics**
  - Count of server responses returned from the protected application to WAAS grouped by HTTP response code prefix
  - Count of timeouts (a timeout is counted when a request is forwarded by WAAS to the protected application with no response received within the set timeout period).

Existing WAAS and application statistics counts will be lost once users reset the aggregation start time. ***Reset*** *will* ***not*** *affect WAAS errors and will not cause recent errors to be lost.*

For further details on WAAS deployment, monitoring and troubleshooting please refer to the WAAS deployment page.

Troubleshooting Potential Outcomes

Application is not responding

Possible reasons	Connectivity Monitor Indications	Solution
A problem with the protected application	- *Timeouts* is incremented.	Disable WAAS rule and check if the problem persists.
	- *WAAS Errors* counter incremented.	
Prisma Session Cookies is enabled and the client accessing the application does not support both cookies and Javascript.	- None of the Application Statistics counters is incremented.	Please see *Prisma Session Cookies* section for more details.

Possible reasons	Connectivity Monitor Indications	Solution
reCAPTCHA is enabled and clients and preventing clients from reaching the protected application.	- None of the Application Statistics counters is incremented.	Please see *reCAPTCHA* section for more details.

Application is responding as expected yet WAAS protections do not trigger

Possible reasons	Connectivity Monitor Indications	Solution
Minimum version requirements of Defenders for a protection or a feature are not met.	- For new features added to existing deployment methods, WAAS operations will continue as usual while new features will not function	Verify that all Defenders meet the minimum requirement stated in the feature documentation before enabling it.
WAAS port is not properly configured.	*Incoming requests* is not incremented	The *App port* should be set to the port on which the protected application is listening. For containers the app port should be set to the exposed port on the container (not necessarily the same as the publicly exposed port).
Workload is not included in rule scope.	The workload radar entity does not have a firewall icon next to it, and the WAAS tab is not available when clicked.	Verify the workload is not in scope and adjust scope to include it.
Workload is included in the scope of two WAAS rules (only first by order will match).	The workload radar entity does not have a firewall icon next to it, and the WAAS tab is not available when clicked.	Ensure that the desired rule matches first by altering rule scope collections or reordering rules.
HTTP hostname is included in the scope of two or more apps under the same WAAS rules (only first	- *Application statistics* counters are incremented.	Whenever multiple apps are defined in the same rule only the first app by order will match.

Possible reasons	Connectivity Monitor Indications	Solution
app by order will match).		
Request URL is not included in the list of protected endpoints.	None of the counters is getting incremented	- Verify base path ends with an * to include all subpaths - Verify HTTP hostname in the request matches the listed HTTP hostnames - Verify scheme in the request matches the scheme in the protected endpoints list (TLS is enabled/disabled accordingly)

Application is responding with HTTP errors (3XX, 4XX, 5XX)

Possible reasons	Connectivity Monitor Indications	Solution
Errors are generated by WAAS (requests are not forwarded to the protected application)	- *WAAS Errors* counter incremented.	
Errors are generated by the protected application	- *Application statistics* 3XX, 4XX or 5XX counters are incremented.	Check the protected application logs for errors.

WAAS is blocking legitimate requests

Possible reasons	Connectivity Monitor Indications	Solution
False positive	- *Application statistics* counters are incremented.	Add exceptions to protections causing false triggers.

WAAS events all have the same attacker IP (private IP)

Possible reasons	Connectivity Monitor Indications	Solution
Ingress controller is not set as a transparent proxy	- *Application statistics* counters are incremented.	Configure ingress controller as transparent proxy (enable “X-

Possible reasons	Connectivity Monitor Indications	Solution
		Forwarded-For" and "X-Forwarded-Host" HTTP headers).

WAAS Sanity Tests

Edit on GitHub

Below are curl-based tests that can be used to verify endpoints have been properly defined. Make sure all changes are saved before running these tests. The method for verifying test results differs according to the selected action:

- **Alert** - Go to **Monitor > Events** to see alerts logged by Prisma Cloud relating to this policy violation.
- **Prevent** - Commands return output similar to the following:

```
HTTP/1.1 403 Forbidden
Date: Wed, 15 Jul 2020 12:51:50 GMT
Content-Type: text/html; charset=utf-8
```

cURL Test Commands

In the following examples, replace *<http_hostname>* with your endpoint's hostname and *<external_port>* with the web-facing port of your application. For testing HTTP header access control, also replace *<http_header_name>* with the header name set in the rule and *<http_header_value>* with set values.

SQL injection:

```
curl -I http://<http_hostname>:<external_port>/\?id\=%27%20OR%20%271
```

Cross-site scripting:

```
curl -I http://<http_hostname>:<external_port>/\?id\=\<script\>alert
\(\1\)\</script\>
```

OS command injection:

```
curl -I http://<http_hostname>:<external_port>/\?id\=%3B+%2Fsbin
%2Fshutdown
```

Code injection:

```
curl -I http://<http_hostname>:<external_port>/\?id\=phpinfo\(\)
```

Local file inclusion:

```
curl -I http://<http_hostname>:<external_port>/\?id\=../etc/passwd
```


Attack tools and vulnerability scanners:

```
curl -I -H 'User-Agent: sqlmap' http://
<http_hostname>:<external_port>/
```

Shellshock protection:

```
curl -I -H "User-Agent: () { :; }; /bin/eject" http://
<http_hostname>:<external_port>/
```

Malformed HTTP request:

```
curl -s -i -X GET -o /dev/null -D - -d '{"test":"test"}' http://
<http_hostname>:<external_port>/
```

HTTP header access controls:

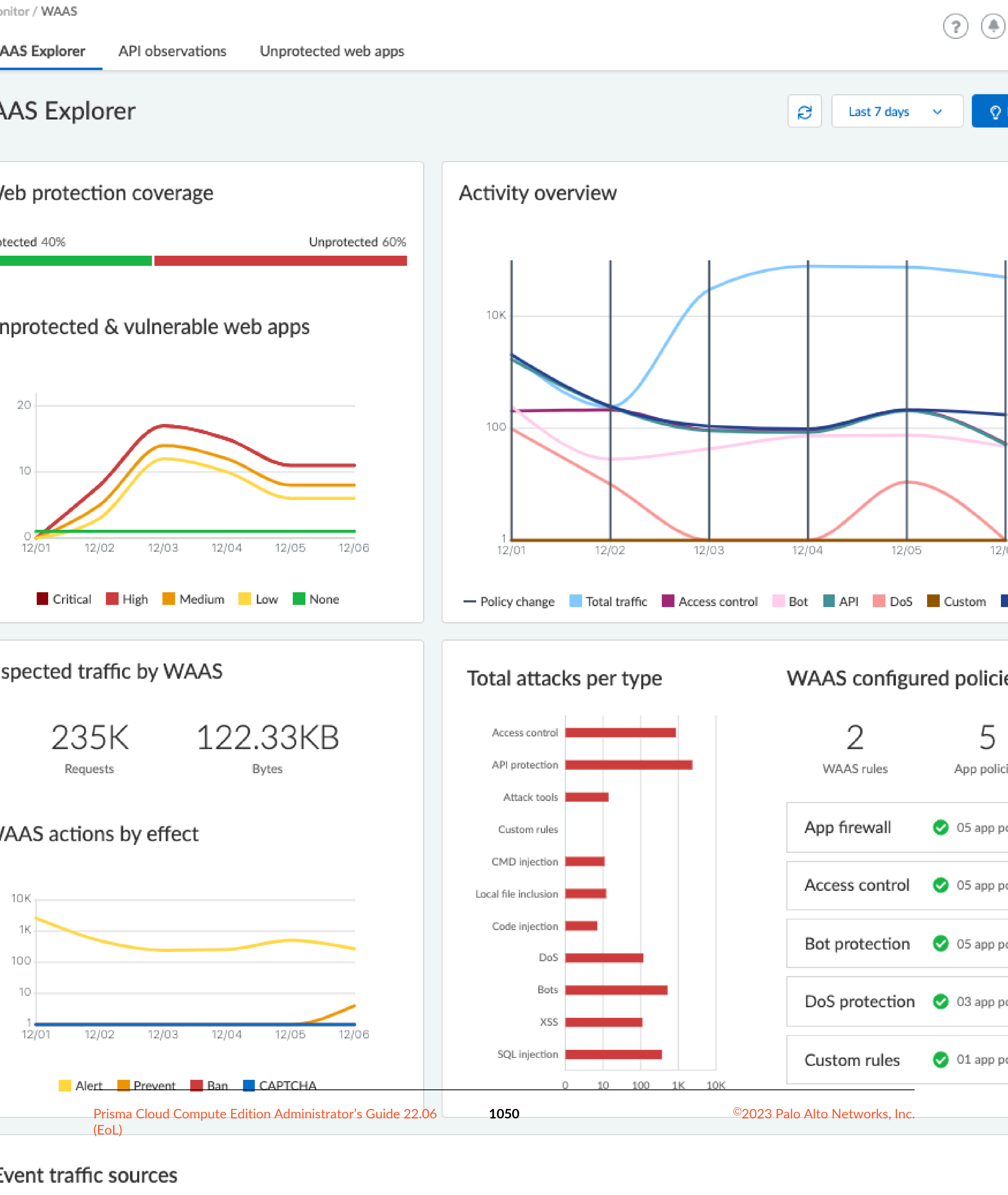
```
curl -H '<header_Name>: <header_value>' http://
<http_hostname>:<external_port>/
```

WAAS Explorer

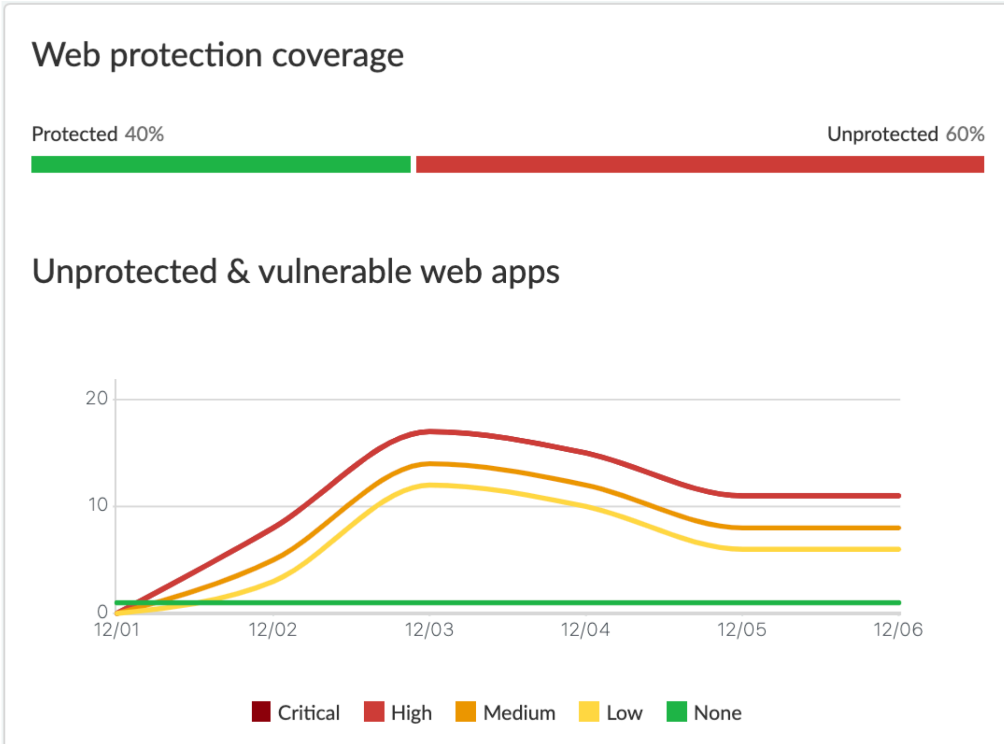
Edit on GitHub

WAAS explorer provides an overview of web application's security posture, protection coverage, usage stats and insights. This dashboard is not intended to replace WAAS built in analytics for investigating incidents and request details.

To use the WAAS Explorer, your Defenders must be running version 22.01 or later. With earlier versions of Defender, the WAAS Explorer dashboard may have errors due to incomplete or missing data.


onitor / WAAS
AAS Explorer
API observations
Unprotected web apps
AAS Explorer
Last 7 days
Veb protection coverage
otected 40%
Unprotected 60%
nprotected & vulnerable web apps
Critical
High
Medium
Low
None
Activity overview
Policy change
Total traffic
Access control
Bot
API
DoS
Custom
spected traffic by WAAS
235K
Requests
122.33KB
Bytes
VAAS actions by effect
Alert
Prevent
Ban
CAPTCHA
Total attacks per type
Access control
API protection
Attack tools
Custom rules
CMD injection
Local file inclusion
Code injection
DoS
Bots
XSS
SQL injection
WAAS configured policie
2
WAAS rules
5
App polici
App firewall
05 app po
Access control
05 app po
Bot protection
05 app po
DoS protection
03 app po
Custom rules
01 app po
Event traffic sources

Web protection coverage

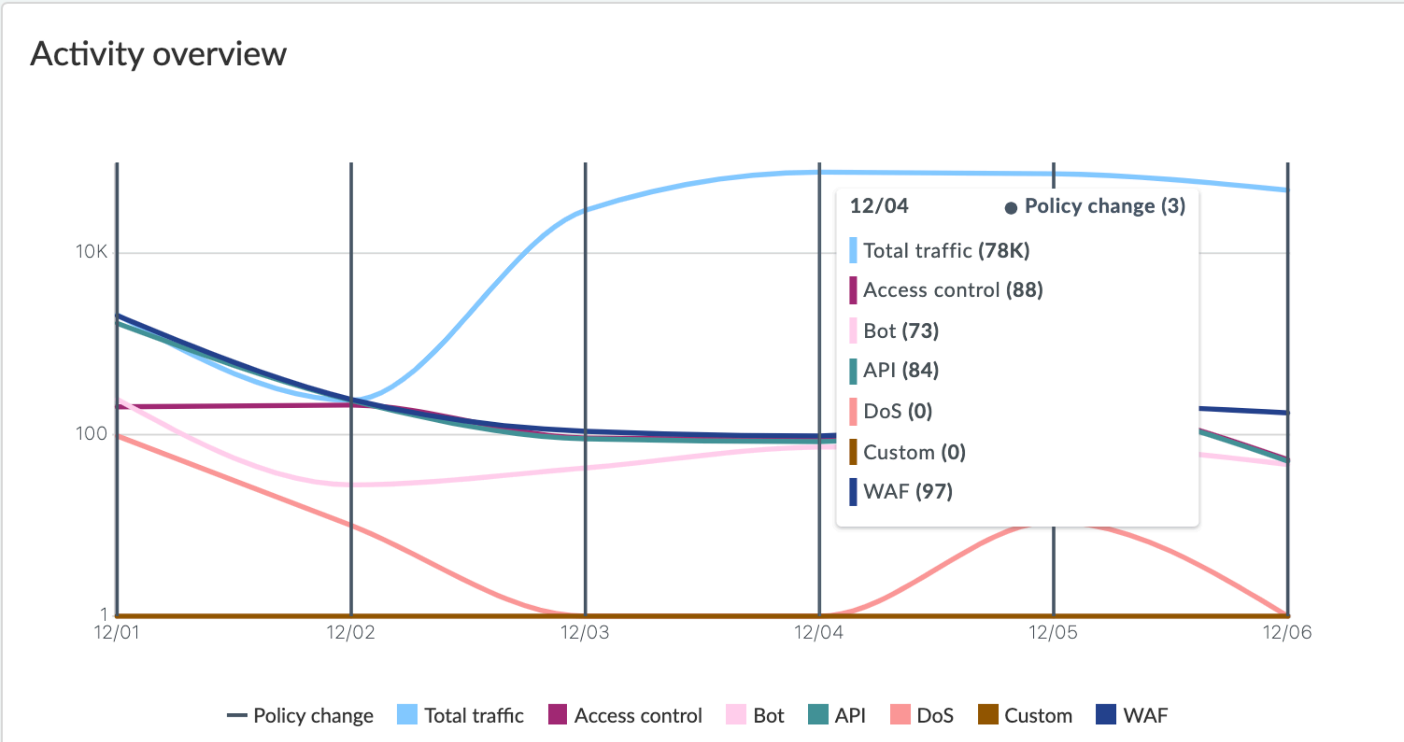


Web protection coverage provides an overview of the web applicaition and API currenlty running in the deployment with the following breakdowns:

- Protection coverage
- Vulnerabilities in unprotected web apps

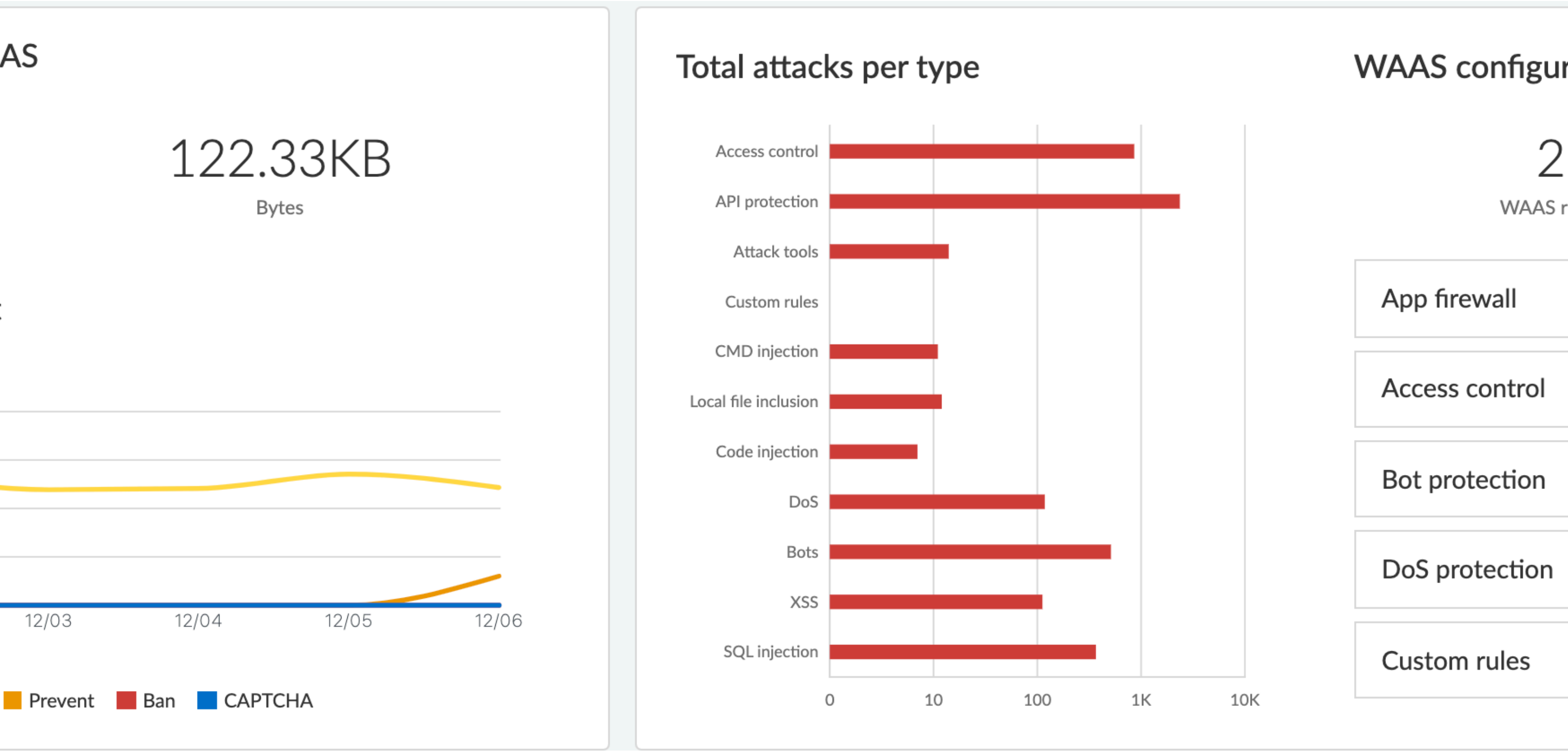
Enable WAAS protection on vulnerable web apps to detect threats and mitigate exploitation attempts.

Activity overview



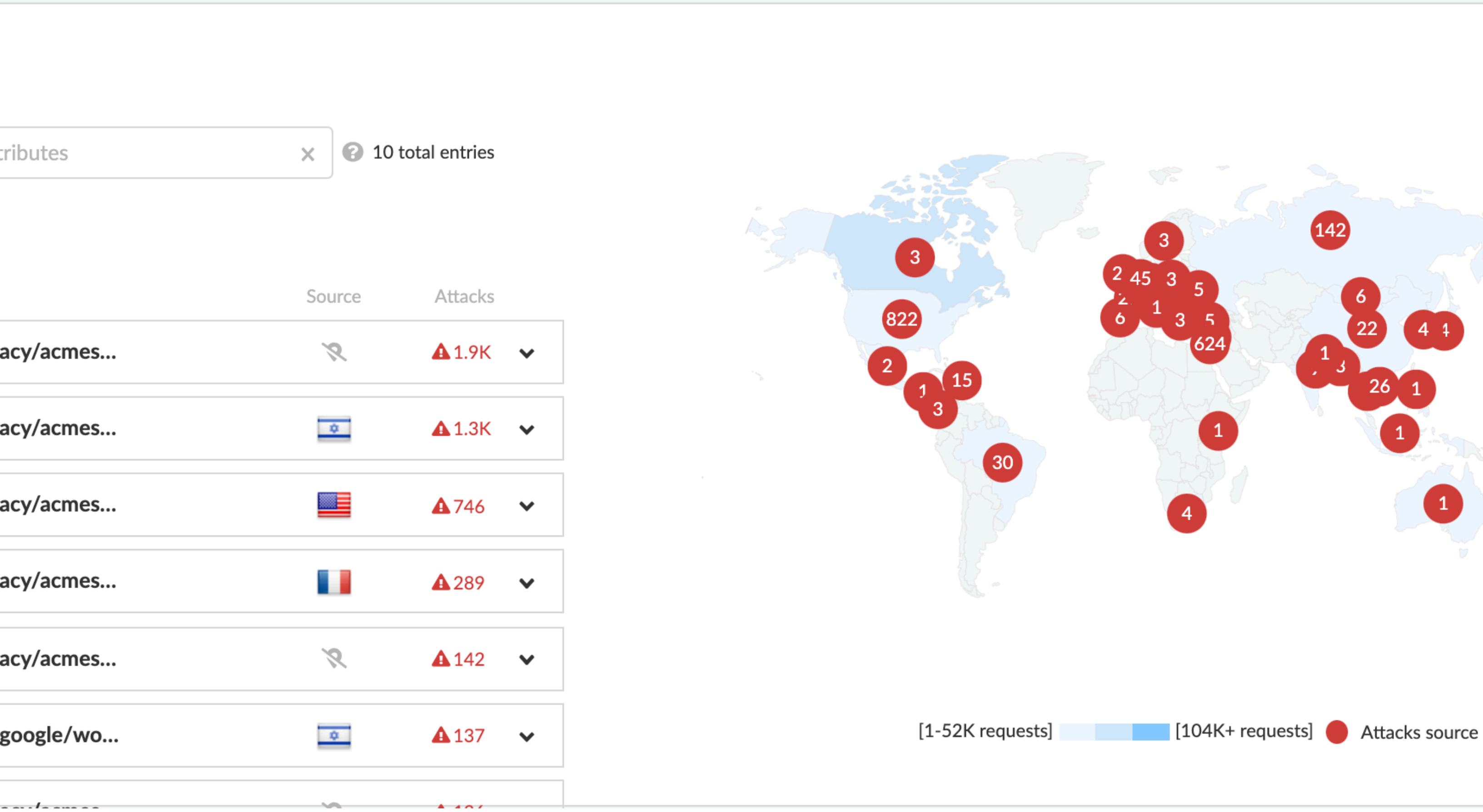
The Activity overview shows daily counts of requests and protection triggers. Policy changes to WAAS are also noted on the date they occurred.

WAAS overview



WAAS overview provides more information about the value WAAS provided - the total amount of traffic inspected, the protections currently in use, and the overall count of triggers, according to type and effect.

Event traffic sources



Using this section, users are able to easily identify attacked images and hosts in their deployment as well as where legitimate traffic and attacks originate from. Users can filter the results based on countries or image names, to obtain a comprehensive overview of attacked images - WAAS events, identified vulnerabilities, and runtime forensics.

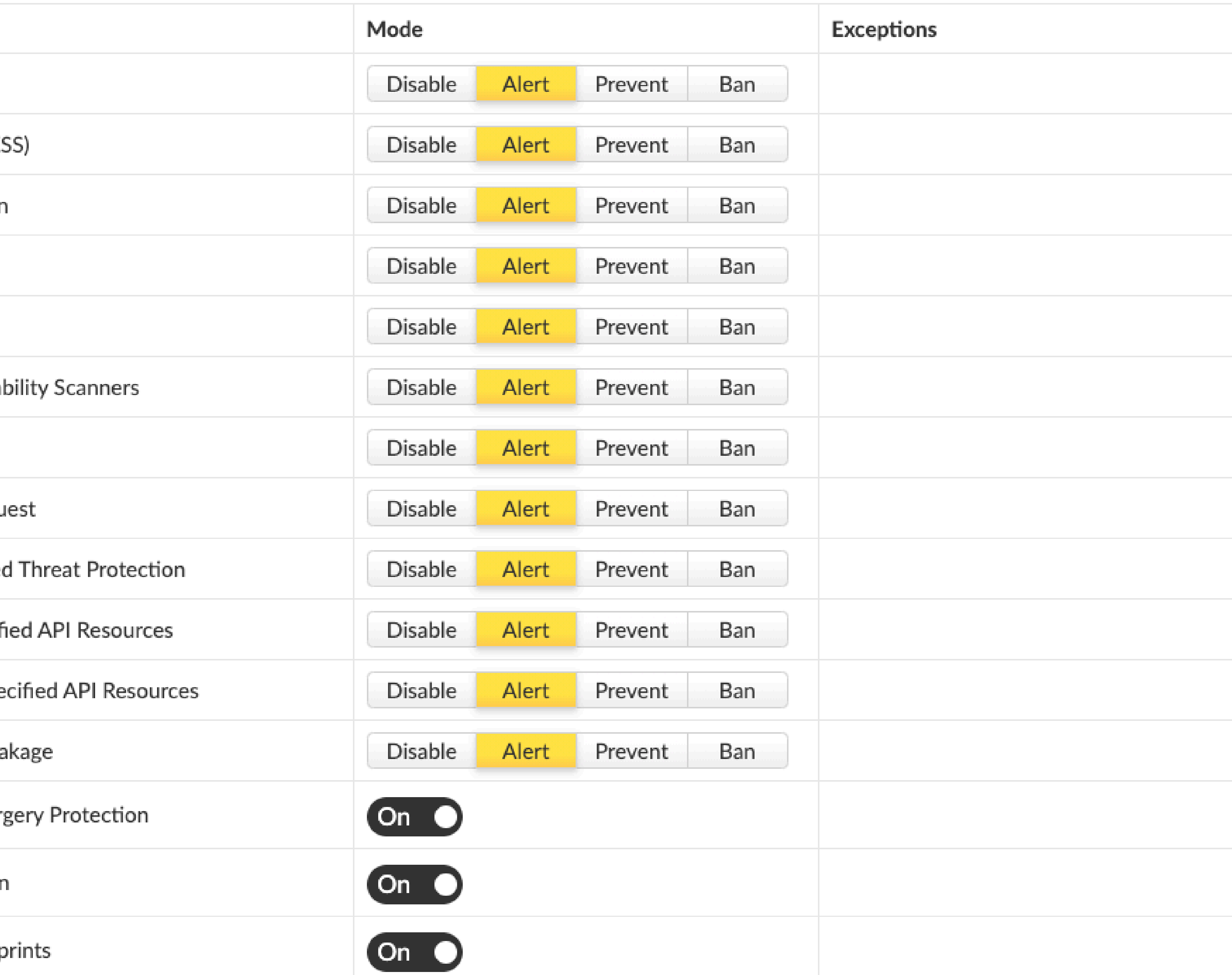
Insights

Waas Explorer insights reveal security posture gaps that need to be addressed.

App Firewall Settings

Edit on GitHub

WAAS Firewall settings control the application firewall's protections, actions and exceptions.



The following protections are available for Container, Host and App-Embedded rules. Serverless rules have a limited set of protections focusing mostly on OWASP Top-10 attacks.

OWASP Top 10 Protection

WAAS offers protection for the critical security risks described in the OWASP Top Ten list.

SQL injection

An SQL injection (SQLi) attack occurs when an attacker inserts an SQL query into the input fields of a web application. A successful attack can read sensitive data from the database, modify data in the database, or run arbitrary commands.

WAAS parses and tokenizes input streams (request data) and then detects malicious attempts to inject unauthorized SQL queries.

Cross Site Scripting

Cross-Site Scripting (XSS) is a type of injection attack, in which malicious JavaScript snippets are injected into otherwise benign and trusted web sites. Attackers try to trick the browser into switching to a Javascript context, and executing arbitrary code.

WAAS parses and tokenizes input streams (request data) and then searches for matching fingerprints of known malicious attack patterns.

Command & Code Injection

Command injection is a form of attack in which attackers attempt to run arbitrary commands on the web application's host. Code injection is a form of attack in which code is injected and interpreted by the application or other runtimes. Command and code payloads are either injected as part of HTTP requests or included from local or remote files (also known as File Inclusion).

WAAS inspects all HTTP requests sent to the application and protects against all types of injection attacks as well as local file inclusions.

Prisma Cloud architecture facilitates defense in-depth via multiple protection layers. Enabling Runtime Protection *in addition to WAAS would allow profiling of the application and identifying any anomalies resulting from command or code injections (e.g. unexpected new processes or DNS queries)*

Local File Inclusion

Local File Inclusion is a form of attack in which attackers attempt to gain unauthorized access to locally stored sensitive files on the web application host. Such access attempts are often made using directory traversal attacks or exploiting file inclusion vulnerabilities in the application.

WAAS inspects all HTTP requests sent to the application for local file inclusion attacks aiming at sensitive system files as well as other various traversal attempts.

Attack Tool & Vulnerability Scanners

Vulnerability scanners are automated tools that scan web applications for known security vulnerabilities and misconfiguration.

Web crawlers are automated tools designed to systematically access and enumerate the content of web applications. Crawling can lead to data breaches by exposing resources that should not be publicly available, or revealing opportunities for hacking by exposing software versions, environment data, and so on.

WAAS is continuously updated with new signatures of widely used web attack arsenal, crawlers and penetration testing tools.

API Protection

WAAS is able to enforce API security based on specifications provided in the form of Swagger or OpenAPI files. WAAS also allows for manual API definition. E.g. paths, allowed HTTP methods, parameter names, input types, value ranges, etc. Once defined, users can choose WAAS actions to apply for requests which do not comply with the API's expected behavior.

For further detail on configuring API protection please refer to the API Protection help page.

Security Misconfigurations

Shellshock

Shellshock is a unique privilege escalation vulnerability that permits remote code execution. In unpatched versions of the bash shell interpreter, the Shellshock vulnerability lets attackers create environment variables with specially crafted values that contain code. As soon as the shell is invoked, the attacker's code is executed.

WAAS checks for requests that are crafted to exploit the Shellshock vulnerability.

For more information about Shellshock, see CVE-2014-6271.

Malformed Request Protection

WAAS validates the structure of HTTP requests, automatically blocking those that are malformed.

Examples of malformed requests include:

- HTTP GET requests with a body.
- HTTP POST requests without a *Content-Length* header.

Cross-site Request Forgery

Cross-site request forgery (CSRF) attacks trick the victim's browser into executing unwanted actions on a web application in which the victim is currently authenticated. WAAS mitigates CSRF attacks by intercepting responses and setting the 'SameSite' cookie attribute value to 'strict'. The 'SameSite' attribute prevents browsers from sending the cookie along with cross-site requests. It only permits the cookie to be sent along with same-site requests.

There are several techniques for mitigating CSRF, including synchronizer (anti-CSRF) tokens, which developers must implement as part of your web application. The synchronizer token pattern generates random challenge tokens associated with a user's session. These tokens are inserted into forms as a hidden field, to be submitted along with your forms. If the server cannot validate the token, the server rejects the requested action.

The SameSite cookie attribute works as a complementary defense against CSRF, and helps mitigate against things such as faulty implementation of the synchronizer token pattern.

- When the SameSite attribute is not set, the cookie is always sent.
- With SameSite attribute set to strict, the cookie is never sent in cross-site requests.

- With SameSite attribute set to lax, the cookie is only sent on same-site requests or top-level navigation with a safe HTTP method, such as GET.

It is not sent with cross-domain POST requests or when loading the site in a cross-origin frame. It is sent when you navigate to a site by clicking on a <a href=...> link that changes the URL in your browser's address bar.

Currently, the following browsers support the SameSite attribute:

- Chrome 61 or later.
- Firefox 58 or later.

For more information about the SameSite attribute, see https://tools.ietf.org/html/draft-west-first-party-cookies-07

Clickjacking

Web applications that permit their content to be embedded in a frame are at risk of clickjacking attacks. Attackers can exploit permissive settings to invisibly load the target website into their own site and trick users into clicking on links which they never intended to click.

WAAS modifies all response headers, setting the *X-Frame-Options* response header value to *SAMEORIGIN*. The *SAMEORIGIN* directive only permits a page to be displayed in a frame on the same origin as the page itself.

Intelligence Gathering

Error messages give attackers insight into the inner workings of your application. It is therefore important to prevent information leakage.

The following controls limit the exposure of sensitive information.

Remove Server Fingerprints

By gathering information about the software type and version used by the web application, attackers may learn about potentially known weaknesses and bugs and exploit them.

Eliminating unnecessary headers makes it more difficult for attackers to identify the frameworks that underpin your application.

Response headers that advertise your application's web server and other server details should be scrubbed. WAAS automatically removes unnecessary headers, such as *X-Powered-By*, *Server*, *X-AspNet-Version*, and *X-AspNetMvc-Version*.

Detect Information Leakage

WAAS detects situations where the contents of critical files, such as */etc/shadow*, */etc/passwd*, and private keys, are contained in responses. WAAS will also detect when responses contain directory listings, output from php_info() function calls, and other similar data leakage cases of potentially risky information.

Prisma Cloud Advanced Threat Protection

Prisma Cloud Advanced Threat Protection (ATP) is a collection of malware signatures and IP reputation lists aggregated from commercial threat feeds, open source threat feeds, and Prisma

Cloud Labs. It is delivered to your installation via the Prisma Cloud Intelligence Stream. The data in ATP is used by WAAS to detect suspicious communication with attacker controlled clients such as a botnet herders or C2 servers. For more details please click here.

Prisma Cloud Advanced Threat Protection is not available when protecting Windows-based hosts.

Firewall Actions

Requests that trigger a WAAS protection are subject to one of the following actions:

- **Alert** - The request is passed to the protected application and an audit is generated for visibility.
- **Prevent** - The request is denied from reaching the protected application, an audit is generated and WAAS responds with an HTML page indicating the request was blocked.
- **Ban** - Can be applied on either IP or Prisma Session IDs. All requests originating from the same IP/Prisma Session to the protected application are denied for the configured time period (default is 5 minutes) following the last detected attack.

A message at the top of the page indicates the entity by which the ban will be applied (IP or Prisma Session ID). When the X-Forwarded-For HTTP header is included in the request headers, ban will apply based on the first IP listed in the header value (true client IP).

To enable ban by Prisma Session ID, Prisma Session Cookies *has to be enabled in the Advanced Settings tab. for more information please refer to the* Advanced Settings *help page.*

WAAS implements state, which is required for banning user sessions by IP address or Prisma Sessions. Because Defenders do not share state, any application that is replicated across multiple nodes must enable IP stickiness on the load balancer.

Firewall Exceptions

WAAS allows for fine-tuning to reduce false positive and tailor its protection to the application needs. Firewall exception will instruct WAAS to ignore a the value of a parameter or HTTP Header when inspecting an HTTP request e.g. WAAS can ignore a query parameter named *comments* when inspecting a request for SQL injection attacks.

WAAS supports the following locations:

- **path** - requests sent to the specified path will be excluded from inspection by the protection.
- **query** - specify the name of a query parameter to be excluded in the form of a regular expression (re2), e.g. ^*id$*.
- **query values** - specify a payload pattern to be excluded in the form of a regular expression (re2), e.g. ^.**test[1-9]{1,6}$*.
- **form/multipart** - specify the name of a body parameter (of type application/x-www-form-urlencoded or sent via a multipart HTTP request) to be excluded in the form of a regular expression (re2), e.g. ^*comment$*

- **header** - specify the name of an HTTP header to be excluded in the form of a regular expression (re2), e.g. ^*X-API-.{3,5}$* or ^*Host$*.
- **user-Agent** - specify the User-Agent HTTP header value to be excluded in the form of a regular expression (re2), e.g. ^*X-API-.{3,5}$* or ^*Host$*.
- **cookie** - specify the name of cookie to be excluded in the form of a regular expression (re2), e.g. ^*sessionID$*.
- **XML (body)** - specify an XML element to be excluded. Object can be of any data type. Path to the object should be specified in a custom path format - define an absolute path to the element, notation supports word characters (a-z, A-Z, 0-9, _, -) separated by / character. e.g: /*root/nested*, */root/nested/id*. Excluding all objects by specifying only / is not supported.
- **JSON (body)** - specify an object path to be excluded. Object can be of any data type. Path to the object should be specified in a custom path format - define an absolute path to the element, notation supports word characters (a-z, A-Z, 0-9, _, -) separated by / character. e.g: /*root/nested*, */root/nested/id*. Excluding all objects by specifying only / is not supported.
- **body** - specify a payload pattern to be excluded in the form of a regular expression (re2), e.g. ^.**test[1-9]{1,6}$*.

__Body__ exception type will match the provided pattern on the raw inspected body (based on the inspection size limit) even when not parsed. Other firewall exceptions are based on parameter names and will only be applied on requests that WAAS was able to parse correctly.

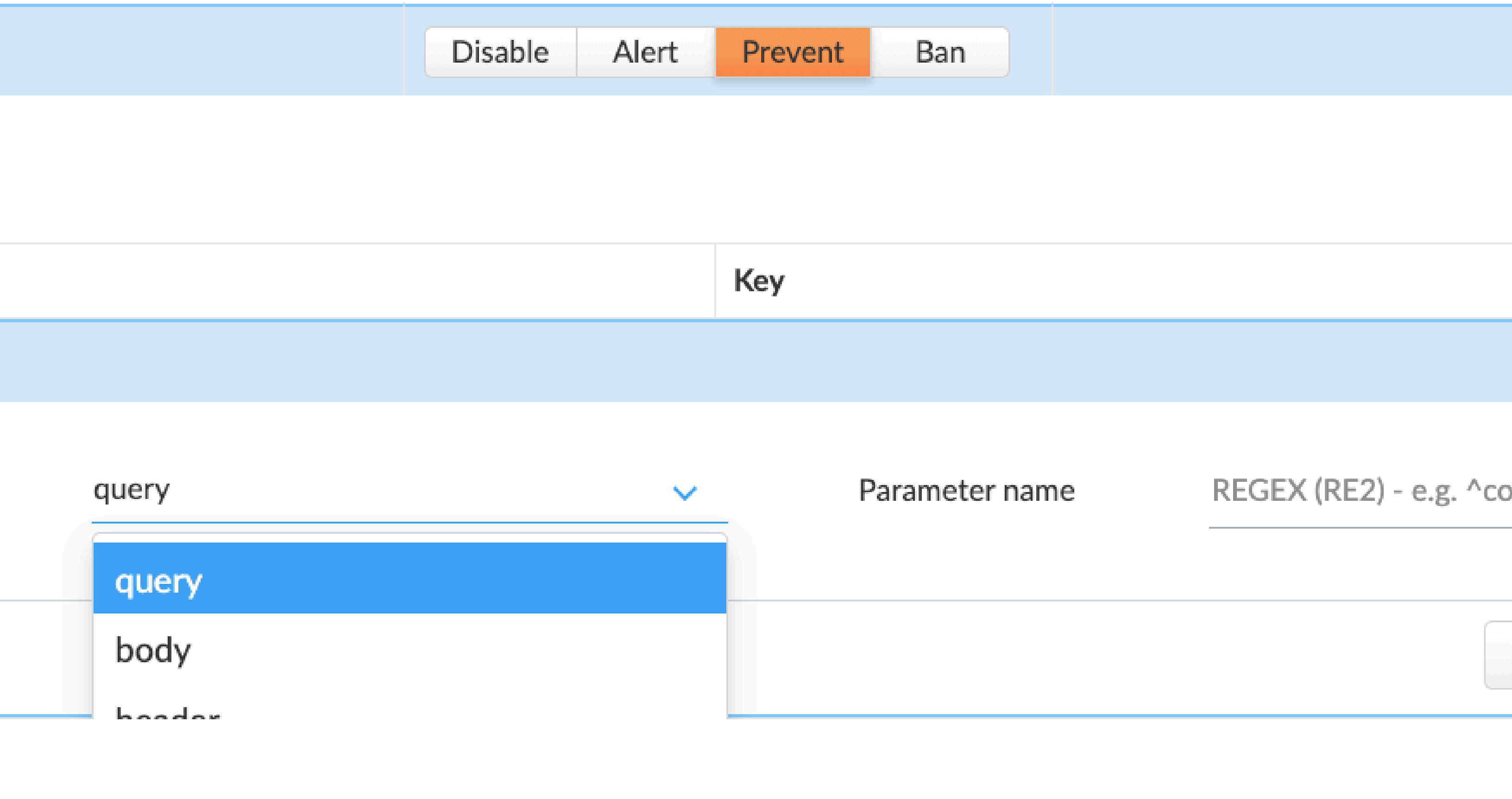
Every protection will have different locations available for exclusion based on the nature of threats.

Adding a new exception

STEP 1 | In the **App firewall** menu click on the icon for one of the OWASP Top-10 protection.

STEP 2 | Click on the + Add exception button

STEP 3 | Select the location and name of the parameter / HTTP header to be excluded



STEP 4 | Select the location and name of the parameter / HTTP header to be excluded.

> *Every protection will have different locations available for exclusion based on the nature of threats.*

STEP 5 | Click on **Save Exception**.

Managing exceptions

STEP 1 | In the **App firewall** menu click on the ⚙ icon for one of the OWASP Top-10 protection.

STEP 2 | In the table, click on the exception you'd like to edit.

STEP 3 | Edit the location and name of the parameter / HTTP header to be excluded.

> *Every protection will have different locations available for exclusion based on the nature of threats.*

STEP 4 | Click on **Done Editing**.

cURL Test Commands

Below are curl-based tests that can be used to verify endpoints have been properly defined. Make sure all changes are saved prior to running these tests. The method for verifying test results differs according to the selected action:

- **Alert** - Go to **Monitor > Events** to see alerts logged by Prisma Cloud relating to this policy violation.
- **Prevent** - Commands return output similar to the following:

```
HTTP/1.1 403 Forbidden
Date: Wed, 15 Jul 2020 12:51:50 GMT
Content-Type: text/html; charset=utf-8
```

In the following examples, replace *<http_hostname>* with your endpoint's hostname and *<external_port>* with the web facing port of your application. For testing HTTP header access control, also replace *<http_header_name>* with the header name set in the rule and *<http_header_value>* with set values.

SQL injection:

```
curl -I http://<http_hostname>:<external_port>/\?id\=%27%20OR%20%271
```

Cross-site scripting:

```
curl -I http://<http_hostname>:<external_port>/\?id\=\<script\>alert
\(\1\)\</script\>
```

OS command injection:

```
curl -I http://<http_hostname>:<external_port>/\?id\=%3B+%2Fsbin
%2Fshutdown
```

Code injection:

```
curl -I http://<http_hostname>:<external_port>/\?id\=phpinfo\(\)
```

Local file inclusion:

```
curl -I http://<http_hostname>:<external_port>/\?id\=../etc/passwd
```

Attack tools and vulnerability scanners:

```
curl -I -H 'User-Agent: sqlmap' http://
<http_hostname>:<external_port>/
```

Shellshock protection:

```
curl -I -H "User-Agent: () { :; }; /bin/eject" http://
<http_hostname>:<external_port>/
```


Malformed HTTP request:

```
curl -s -i -X GET -o /dev/null -D - -d '{"test":"test"}' http://
<http_hostname>:<external_port>/
```

HTTP header access controls:

```
curl -H '<header_Name>: <header_value>' http://
<http_hostname>:<external_port>/
```


API Protection

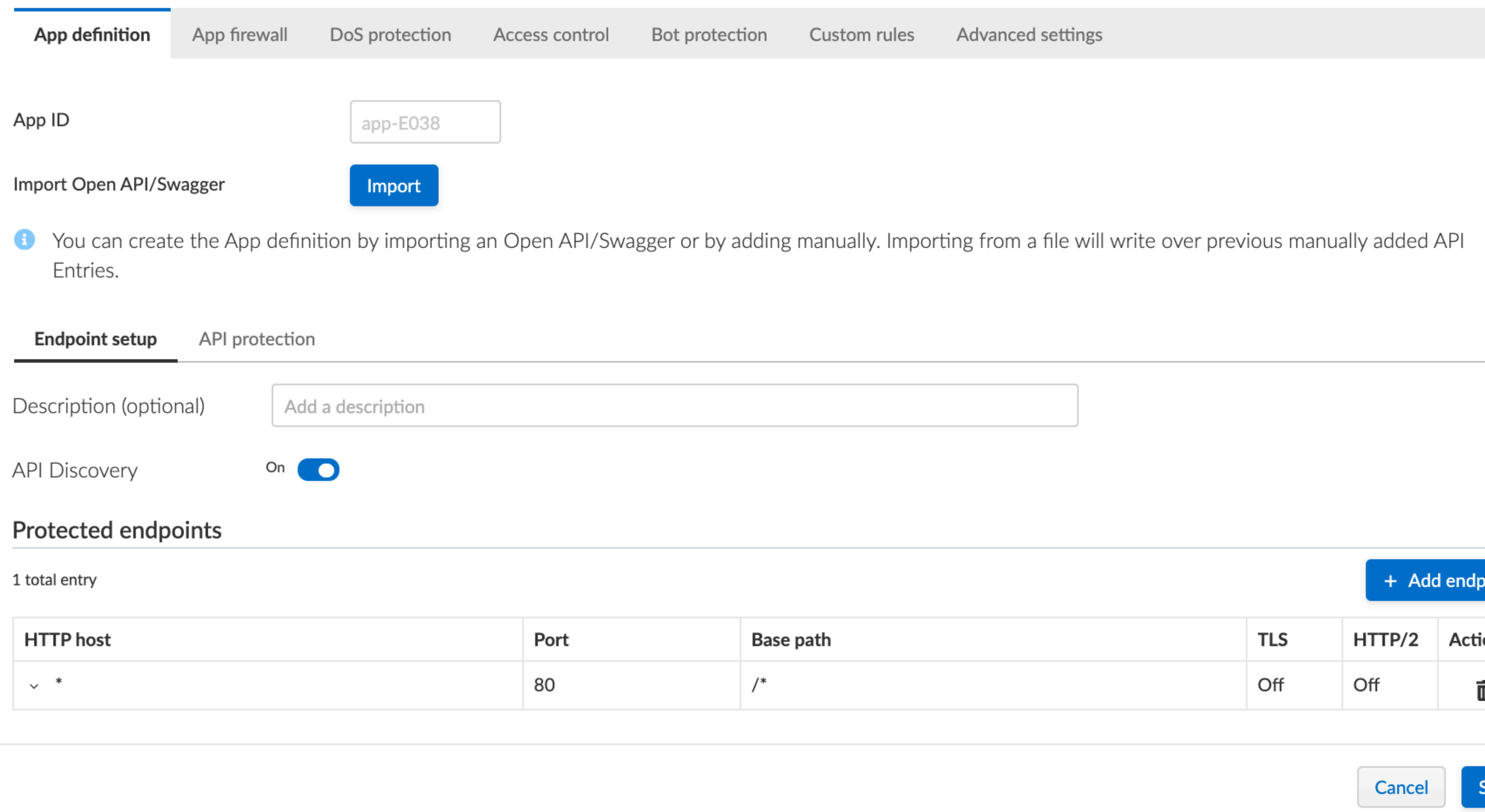
Edit on GitHub

WAAS can enforce API security based on specifications provided in the form of Swagger or OpenAPI files. Alternatively, you can manually define your API (e.g., paths, allowed HTTP methods, parameter names, input types, value ranges, and so on). Once defined, you can configure the actions WAAS applies to requests that do not comply with the API's expected behavior.

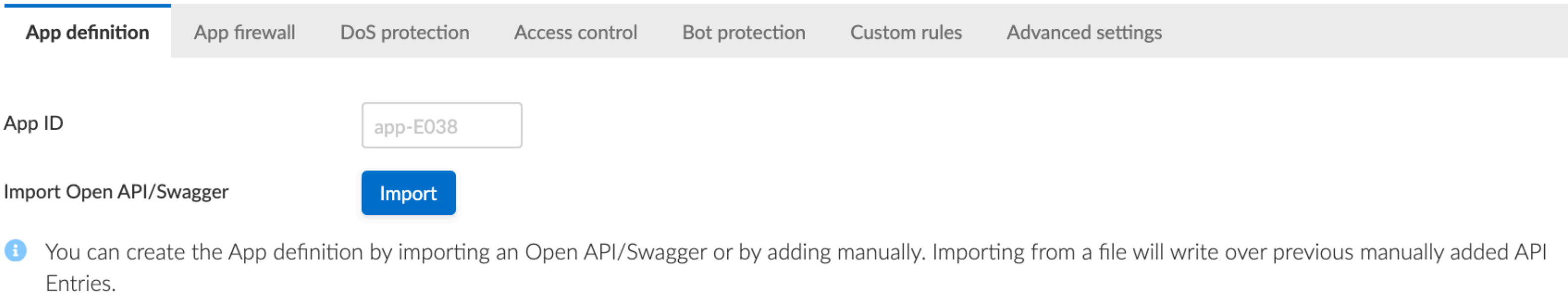
Users should be careful when enabling Prisma Session Cookies along with API protection. Prisma Session Cookies mandates client's support of cookies and javascript in order for them to reach the protected application. As APIs are often accessed by "primitive" automation clients, avoid enabling Prisma Session Cookies unless you are certain all clients accessing the protected API support BOTH cookies AND Javascript.

Import API definition from Swagger or OpenAPI files

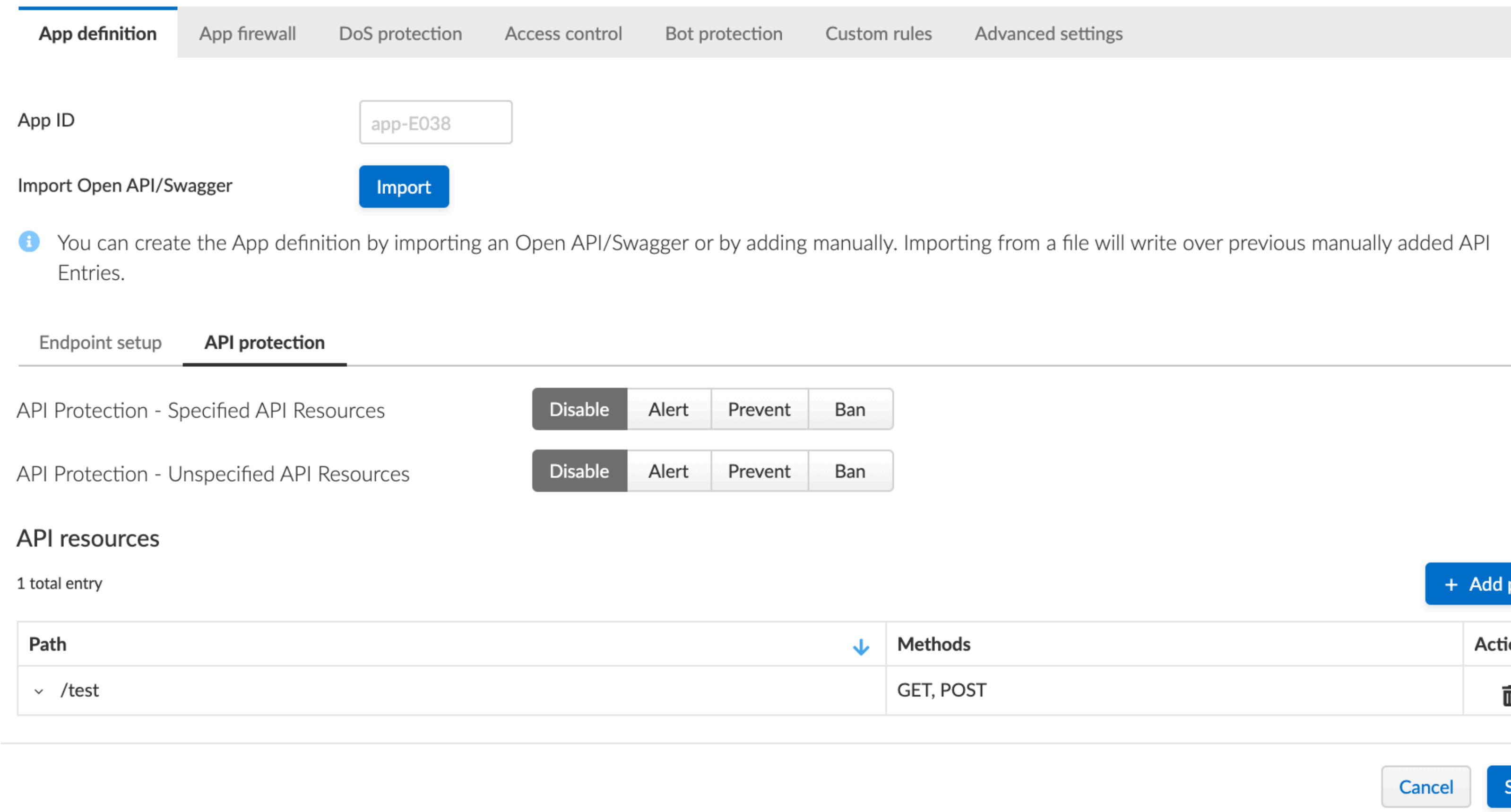
1. Click the **App definiton** tab.



2. Click **Import**.

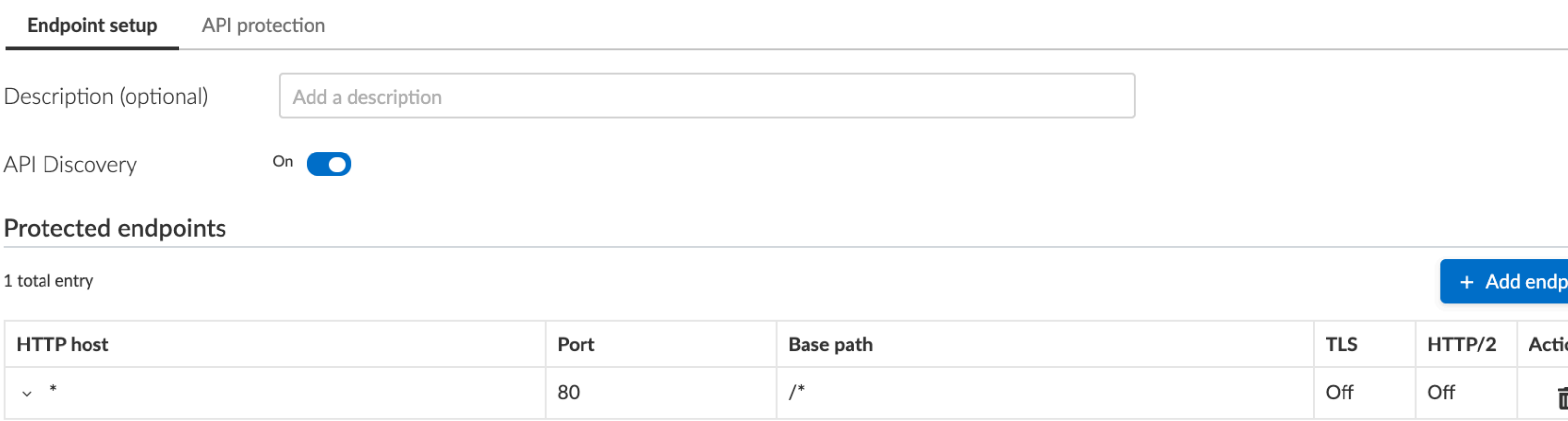


3. Select a file to load.
4. Click the **API protection** tab.



5. Review path and parameter definitions listed under **API Resources**.

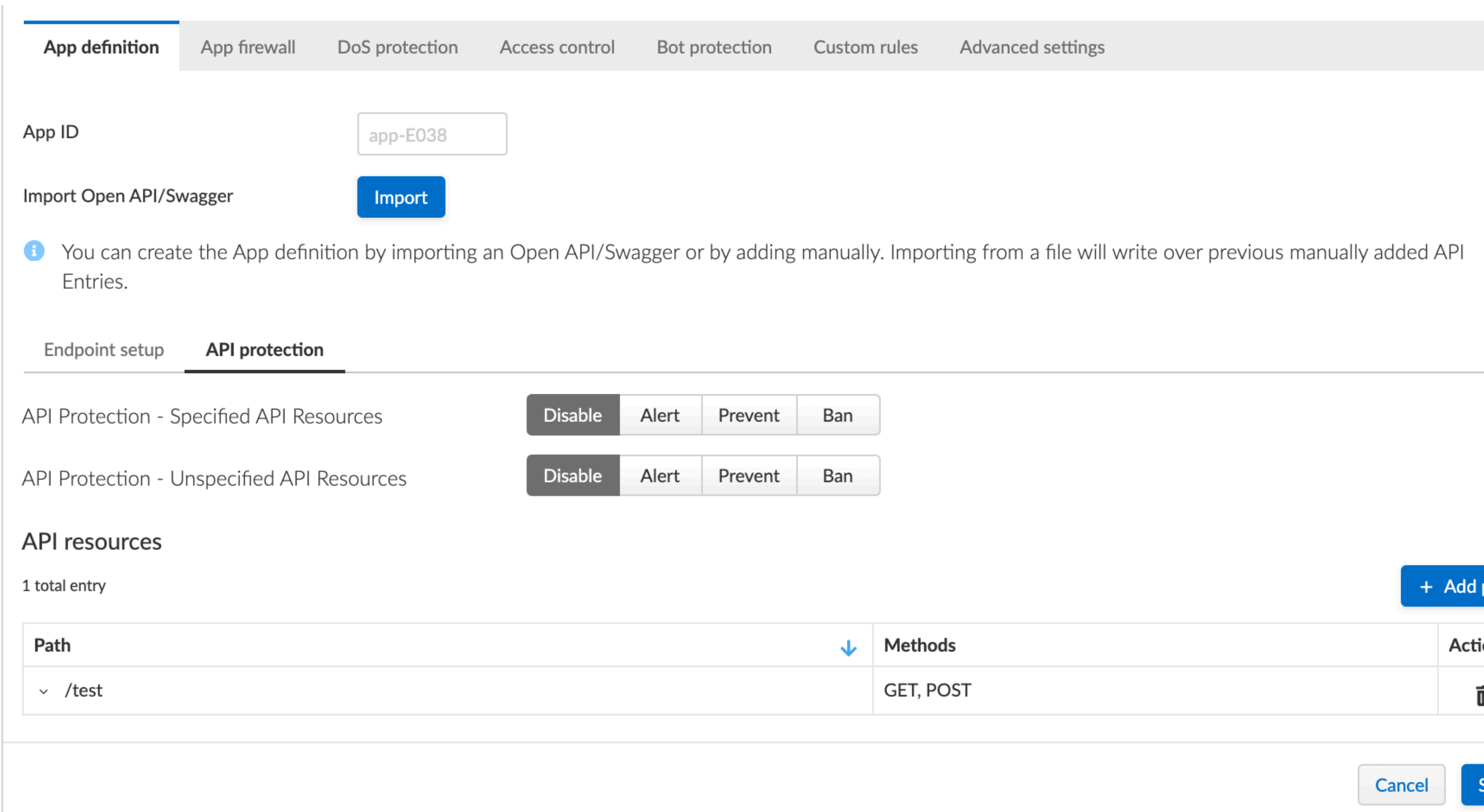
6. Click the **Endpoint setup** tab.



7. Review protected endpoints listed under **Protected Endpoints** and verify configured base paths all end with a trailing *.

*Base path in the endpoint definition should always end with a * e.g. "/*", "/api/v2/*". If not configured that way, API protection will not apply to sub-paths defined in the API protection tab.*

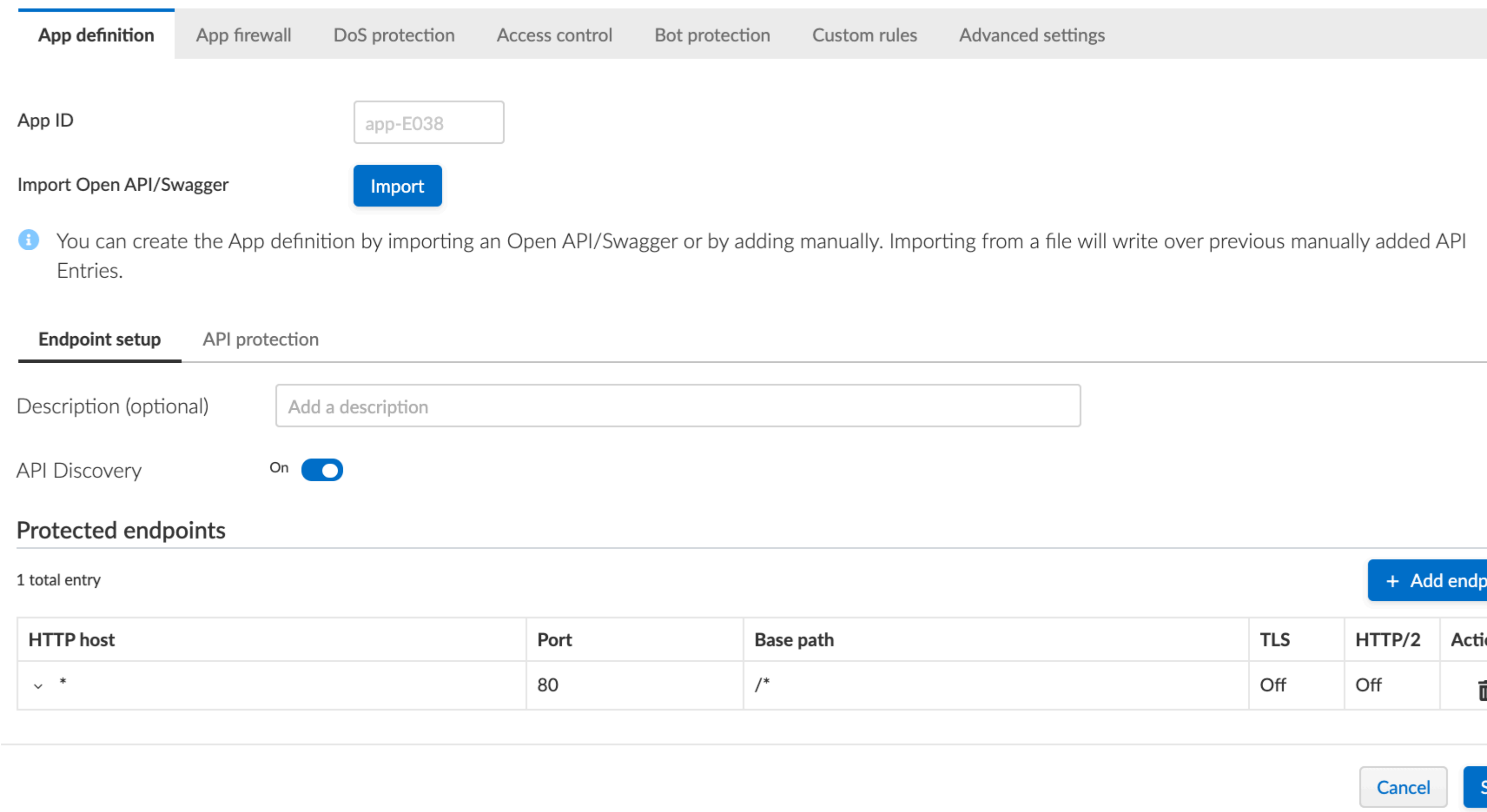
8. Go back to the **API protection** tab.



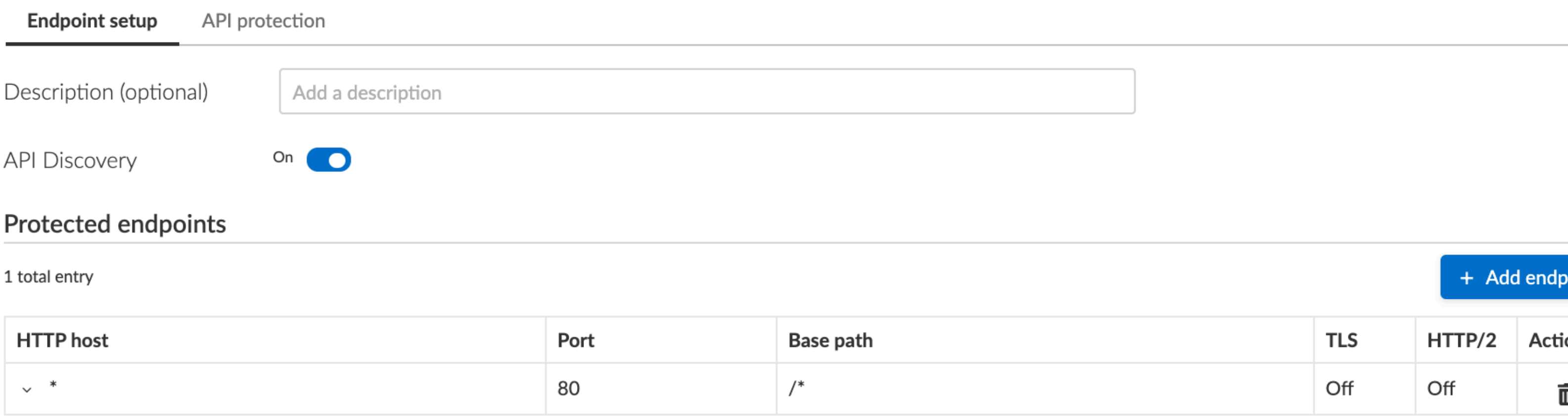
9. Configure an **API protection** action for the resources defined under **API resources**, and an action for all other resources.

Define an API manually

1. Click the **App definiton** tab.



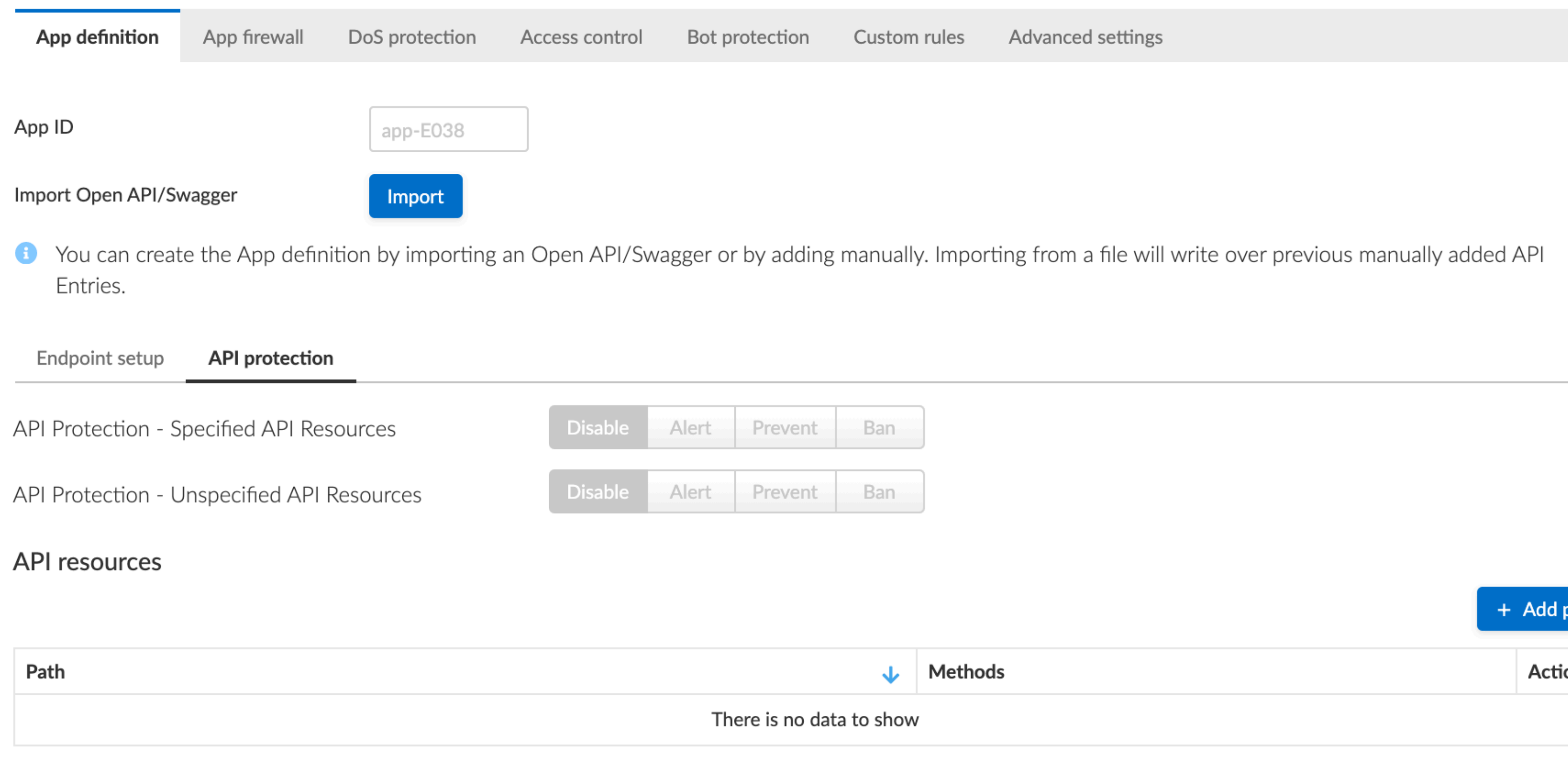
2. Click the **Endpoint setup** tab.



3. Add protected endpoints under **Protected endpoints** and verify configured base paths all end with a trailing *.

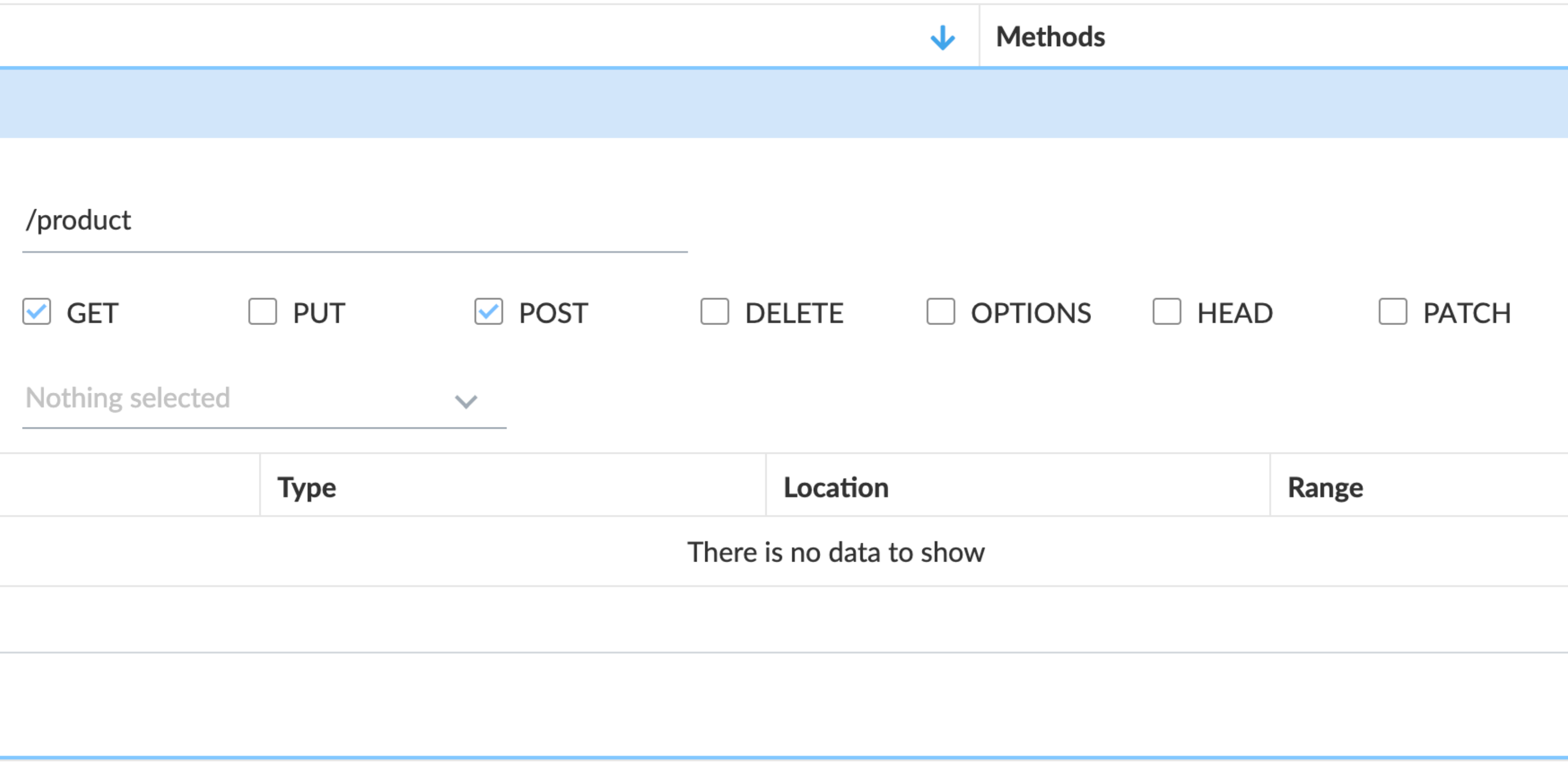
> *Base path in the endpoint definition should always end with a * e.g. "/*", "/api/v2/*". If not configured that way, API protection will not apply to sub-paths defined in the API protection tab.*

4. Click the **API protection** tab.



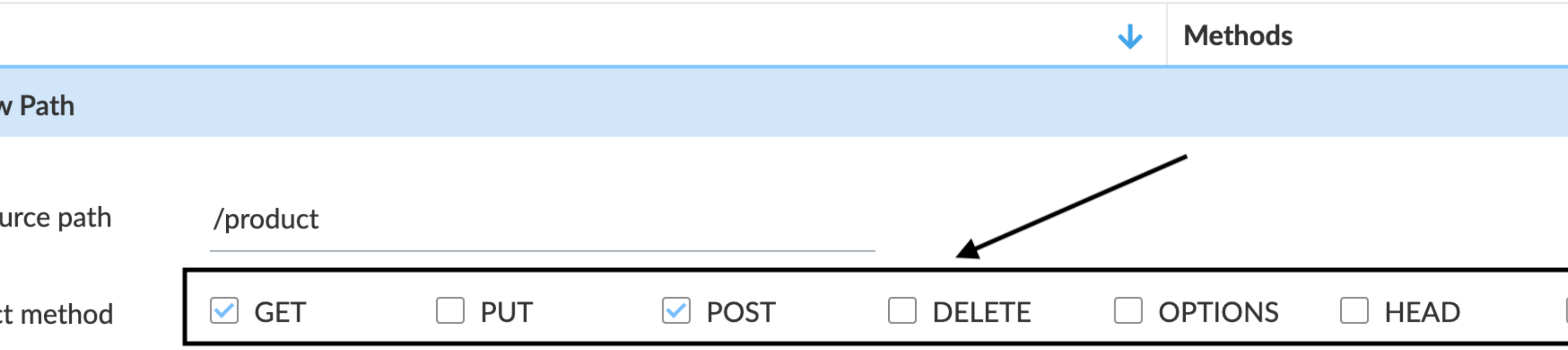
5. Click **Add path**

6. Enter **Resource path** (e.g. */product* - resource paths should not end with a trailing "/").

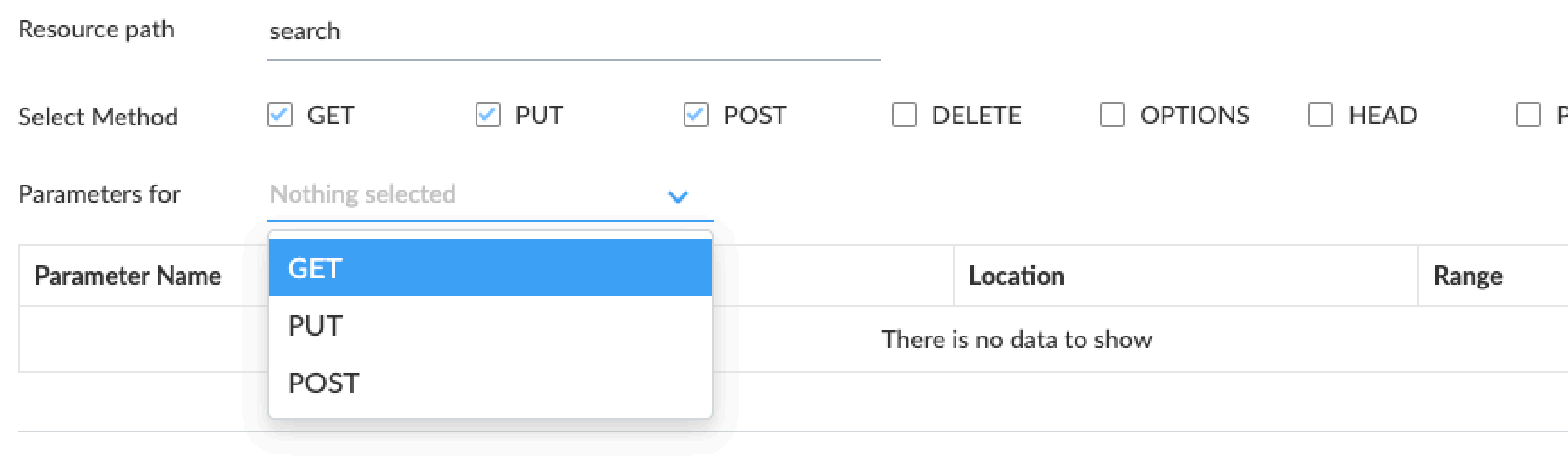


Paths entered in this section are additional subpaths to the base path defined in the previous endpoint section. For example, if in the endpoint definition hostname was set to "*www.example.com*", base path set to "*/api/v2/**" and in the **API Protection** tab resource path set to "*/product*" - full protected resource would be *www.example.com/api/v2/product*.

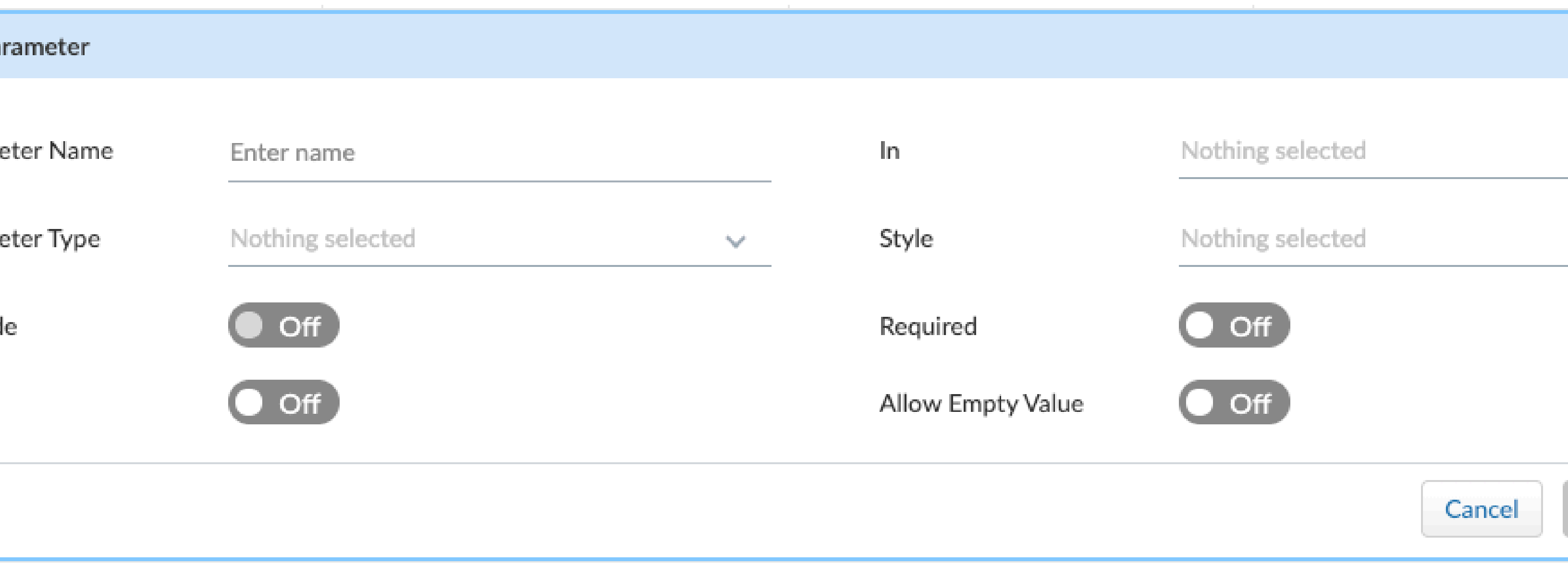
7. Select allowed methods.



8. For each allowed HTTP method, define parameters by selecting the method from **Parameters for** drop-down list.



1. Select an HTTP method from drop-down list.
2. Click **Add parameter**.
3. Enter parameter definition.



9. Configure an **API protection** action for the resources defined under **API resources**, and an action for all other resources.

- **Parameter violation**—Action to be taken when a request sent to one of the specified paths in the API resource list does not comply with the parameter provided definitions.
- **Unspecified path(s)/method(s)**—Action to be taken in one of the following cases:
  - Request sent to a resource path that is not specified in the API resources list.
  - Reqeust sent using an unsupported HTTP method for a resource path in the API list.

API Actions

HTTP requests that trigger API protections are subject to one of the following actions:

- **Alert** - Request is passed to the protected application and an audit is generated for visibility.
- **Prevent** - Request is denied from reaching the protected application, an audit is generated, and WAAS responds with an HTML banner indicating the request was blocked.
- **Ban** - Can be applied on either IP addresses or Prisma Session IDs. All requests originating from the same IP/Prisma Session to the protected application are denied for the configured time period (default is 5 minutes) following the last detected attack.

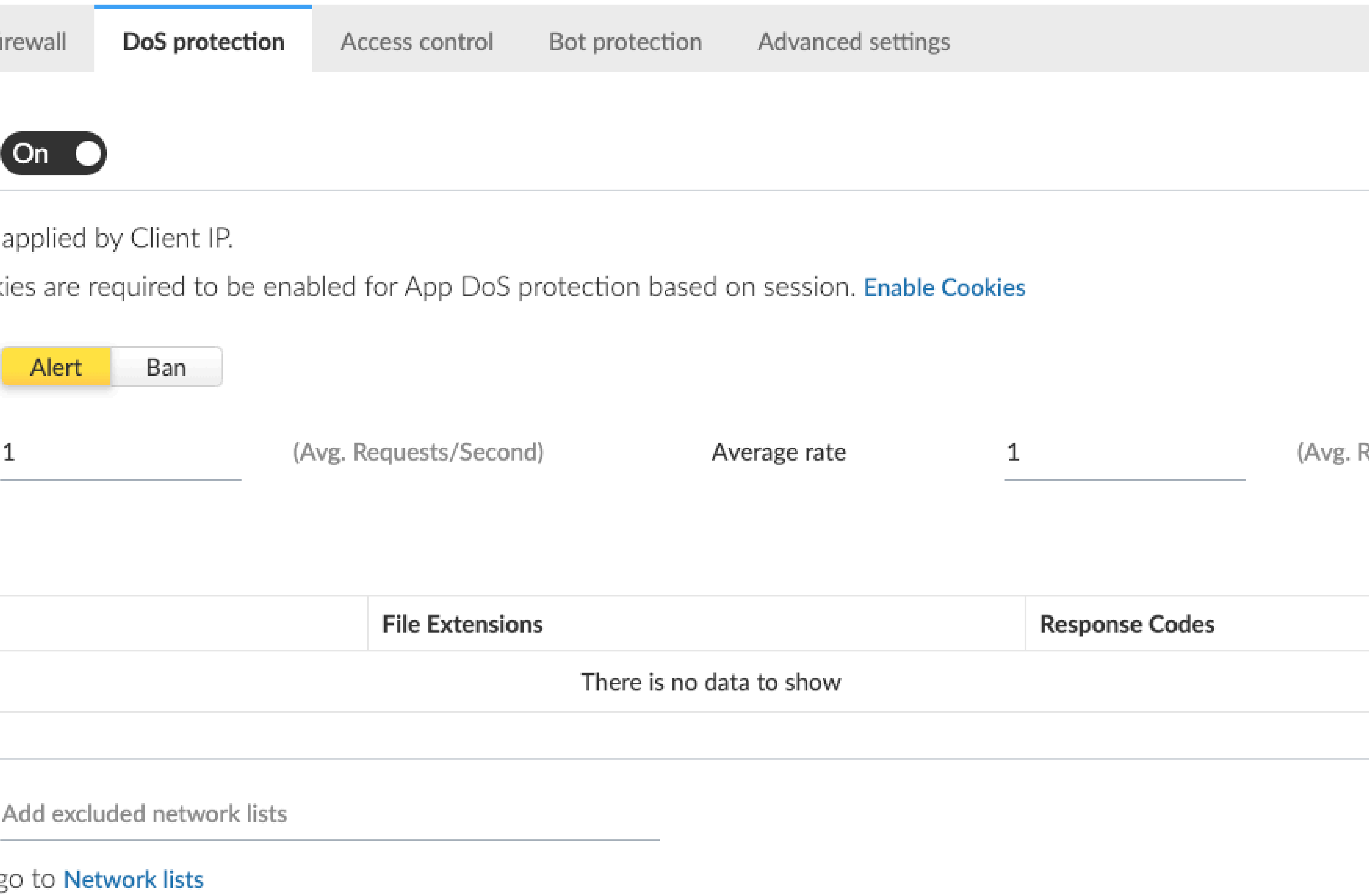
  To enable ban by Prisma Session ID, you must enable Prisma Session Cookies. For more information on enabling Prisma Sessions and configuring ban definitions, see Advanced Settings.

- *When the X-Forwarded-For HTTP header is included in the request headers, ban will apply based on the first IP listed in the header value (true client IP).*
- *WAAS implements state, which is required for banning user sessions by IP address. Because Defenders do not share state, any application that is replicated across multiple nodes must enable IP address stickiness on the load balancer.*

DoS protection

Edit on GitHub

WAAS is able to enforce rate limit on IPs or sessions to protect against high-rate and "low and slow" application layer DoS attacks.



DoS protection Overview

WAAS is able to limit the rate of requests to the protected endpoints within each app based on two configurable request rates:

- **Burst Rate** - Average rate of requests per second calculated over a 5 seconds period
- **Avarage Rate** - Average rate of requests per second calculated over a 120 seconds period

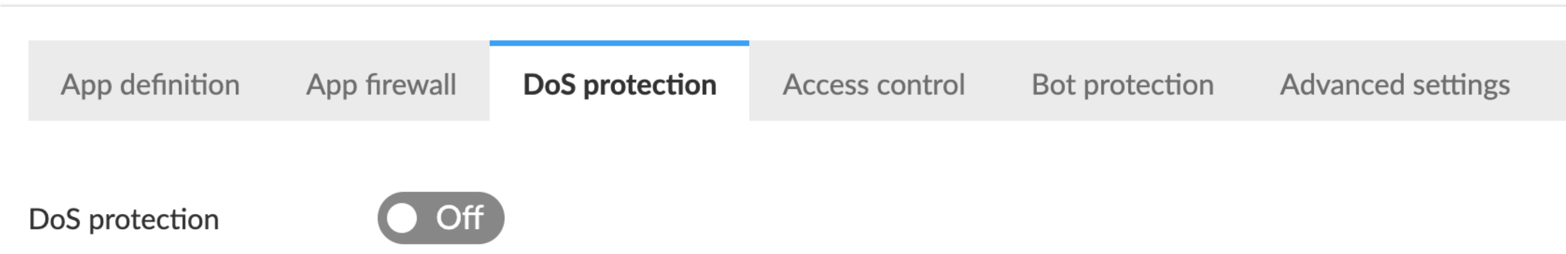
Users are able to specify match conditions for qualifying requests to be included in the count. Match conditions are based on HTTP methods, File Extensions and HTTP response codes.

Users are also able to specify Network lists to be excluded from the DoS protection rate accounting.

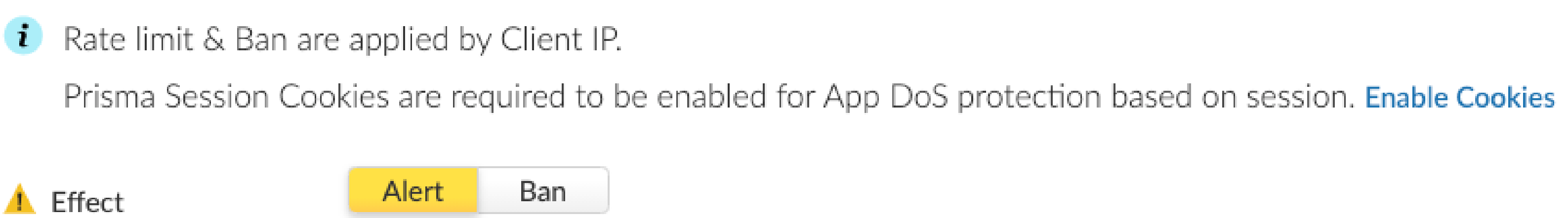
If no match conditions are specified - all requests to the protected endpoints would be included in the rate accounting.

Enabling DoS protection

STEP 1 | Enter **DoS Protection** tab and set the *DoS Protection* toggle to *On*



STEP 2 | Set the effect with the action to apply once a threshold is reached.

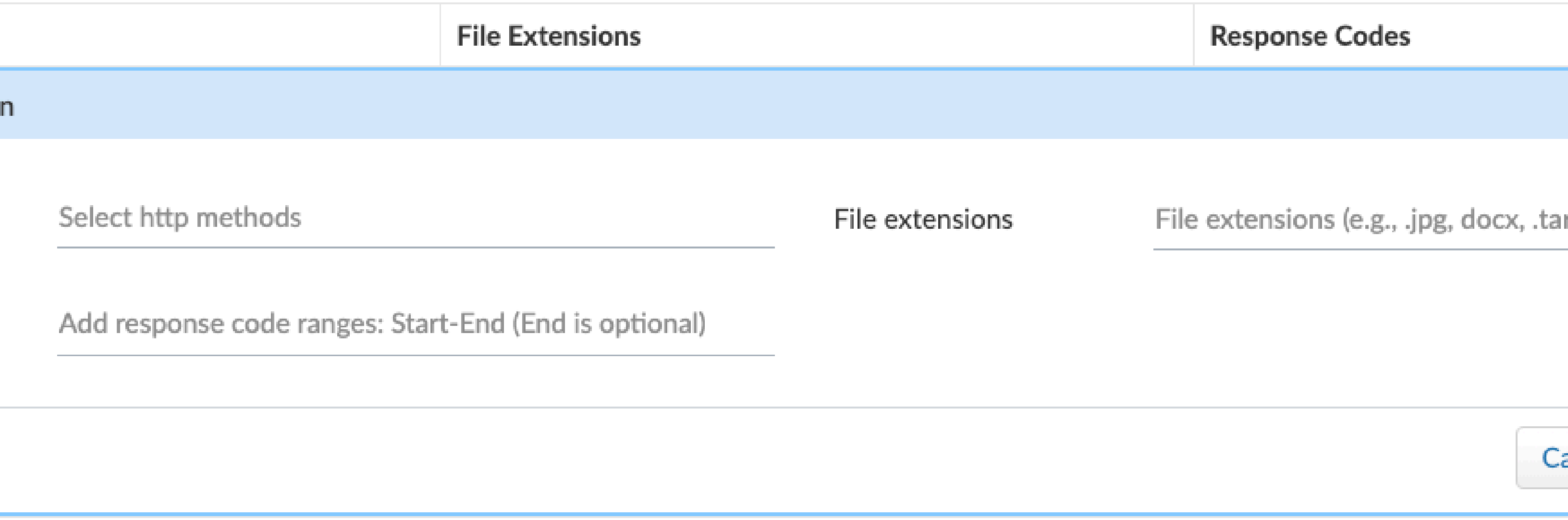


A message at the top of the page indicates the entity by which the ban will be applied (IP or Prisma Session ID).

To enable ban by Prisma Session ID, Prisma Session Cookies *has to be enabled in the Advanced Settings tab. for more information please refer to the* Advanced Settings *help page.*

STEP 3 | Apply rate limitation thresholds (requests per second) for *Burst rate* (calculated over 5 seconds) and for *Average rate* (calculated over 120 seconds)

STEP 4 | To apply the rate limitation on a subset of requests click on + Add Match Condition button.



Conditions can be specified as a combination (**AND**) of the following:

- **HTTP Methods**
- **File Extensions** - multiple extensions are allowed (e.g. *.jpg*, *.jpeg*, *.png*).
- **HTTP Response Codes** - specify either a single response code, a range or a combination of them (e.g. *302*, *400-410*, *500-599*).

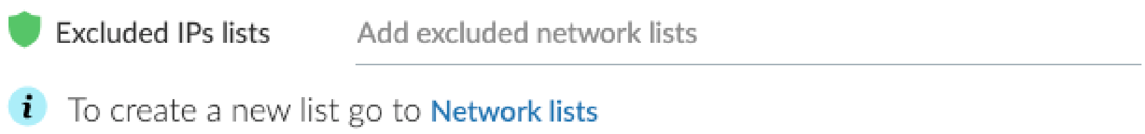
STEP 5 | Multiple match conditions are allowed (**OR** relation between them).

	File Extensions	Response Codes
	.tar.gz	
	.jpg, .jpeg, .png	302, 400-410, 500-599

In the above example the following request would be counted against the rate limitation thresholds:

- *HEAD* HTTP requests
- *POST* HTTP requests with file extension of *.tar.gz*
- *GET* or *PUT* HTTP requests with file extension of *.jpg*, *.jpeg*, *.png* to which the origin responded with and HTTP response code of *302* or in the range of *400-410* or in the range of *500-599*

STEP 6 | Specify Network lists of IP addresses to be excluded from the rate accounting.



DoS actions

Requests that exceed the rate limitation thresholds are subject to one of the following actions:

- **Alert** - The request is passed to the protected application and an audit is generated for visibility.
- **Ban** - Can be applied on either IP or Prisma Session. All requests originating from the same IP/ Prisma Session to the protected application are denied for the configured time period (default is 5 minutes) following the last detected attack.

A message at the top of the page indicates the entity by which the ban will be applied (IP or Prisma Session ID). When the X-Forwarded-For HTTP header is included in the request headers, ban will apply based on the first IP listed in the header value (true client IP).

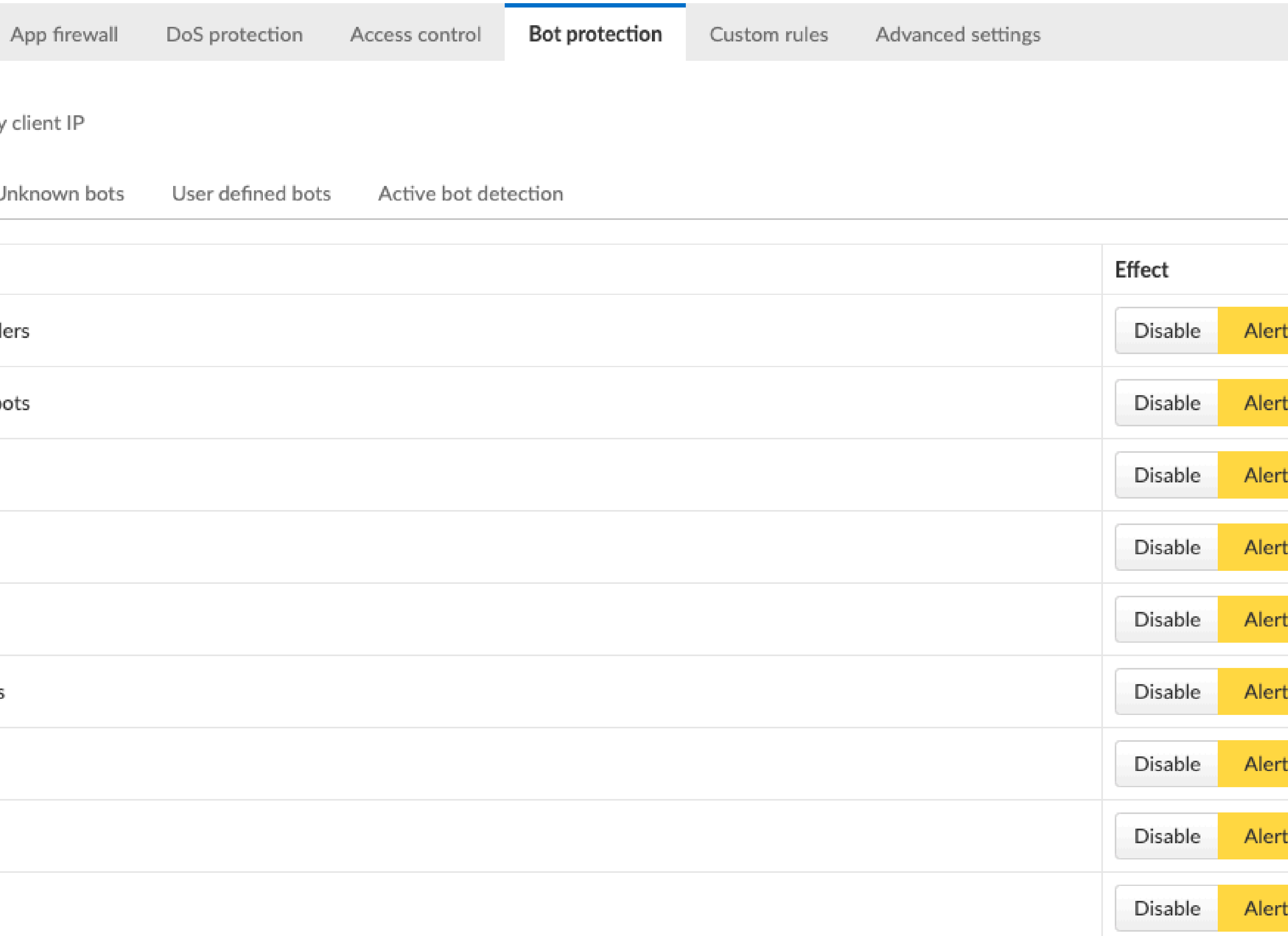
For more information on enabling Prisma Sessions and configuring ban definitions please refer to the Advanced Settings help page.

WAAS implements state, which is required for banning user sessions by IP address or Prisma Sessions. Because Defenders do not share state, any application that is replicated across multiple nodes must enable IP stickiness on the load balancer.

Bot Protection

Edit on GitHub

WAAS bot protection provides visibility into bots and other automation frameworks accessing protected web applications and APIs.



Bot Categories

WAAS detects known good bots as well as other bots, headless browsers and automation frameworks. WAAS is also able to fend off cookie-dropping clients and other primitive clients by mandating the use of cookies and javascript in order for the client to reach the protected origin.

Bots are categorized into the following Categories:

- **Search Engine Crawlers** - Bots systematically crawling and indexing the world wide web to index pages for online searching. Also known as spider bots or web crawlers.
- **Business Analytics Bots** - Bots that crawl, extract and index business related information.
- **Educational Bots** - Bots that crawl, extract and index information for educational purposes, such as academic search engines.
- **News Bots** - Bots that crawl, extract and index the latest news articles, usually for news aggregation services.
- **Financial Bots** - Bots that crawl, extract and index financial related data.
- **Content Feed Clients** - Automated tools, services or end-user clients that fetch web contents for feed readers.
- **Archiving Bots** - Bots that crawl, extract and archive web site information.
- **Career Search Bots** - Automated tools or online services that extract and index job related postings.
- **Media Search Bots** - Bots that crawl, extract and index media contents for search engine purposes.

This category contains various bots and other automation frameworks which cannot be classified by their activity or origin

- **Generic Bots** - Clients with attributes that indicate an automated bot.
- **Web Automation Tools** - Scriptable headless web browsers and similar web automation tools.
- **Web Scrapers** - Automated tools or services that scrape web site contents.
- **API Libraries** - Software code libraries for Web API communications.
- **HTTP Libraries** - Software code libraries for HTTP transactions.
- **Request Anomalies** - HTTP requests with anomalies that are not expected from common web browsers.
- **Bot Impersonators** - Bots and automation tools impersonating as known good bots to evade rate limitation and other restrictions.
- **Browser Impersonators** - Automated tools or services that impersonate common web browser software.

Users can create custom signatures to be used based on HTTP headers and source IPs. User-defined signatures are useful for tracking customer specific bots, self-developed automation clients and traffic that appears suspicious.

Detection methods

WAAS uses static and active methods for detecting bots.

Static detection examines each incoming HTTP request and analyzes it to determine whether it was sent by a bot.

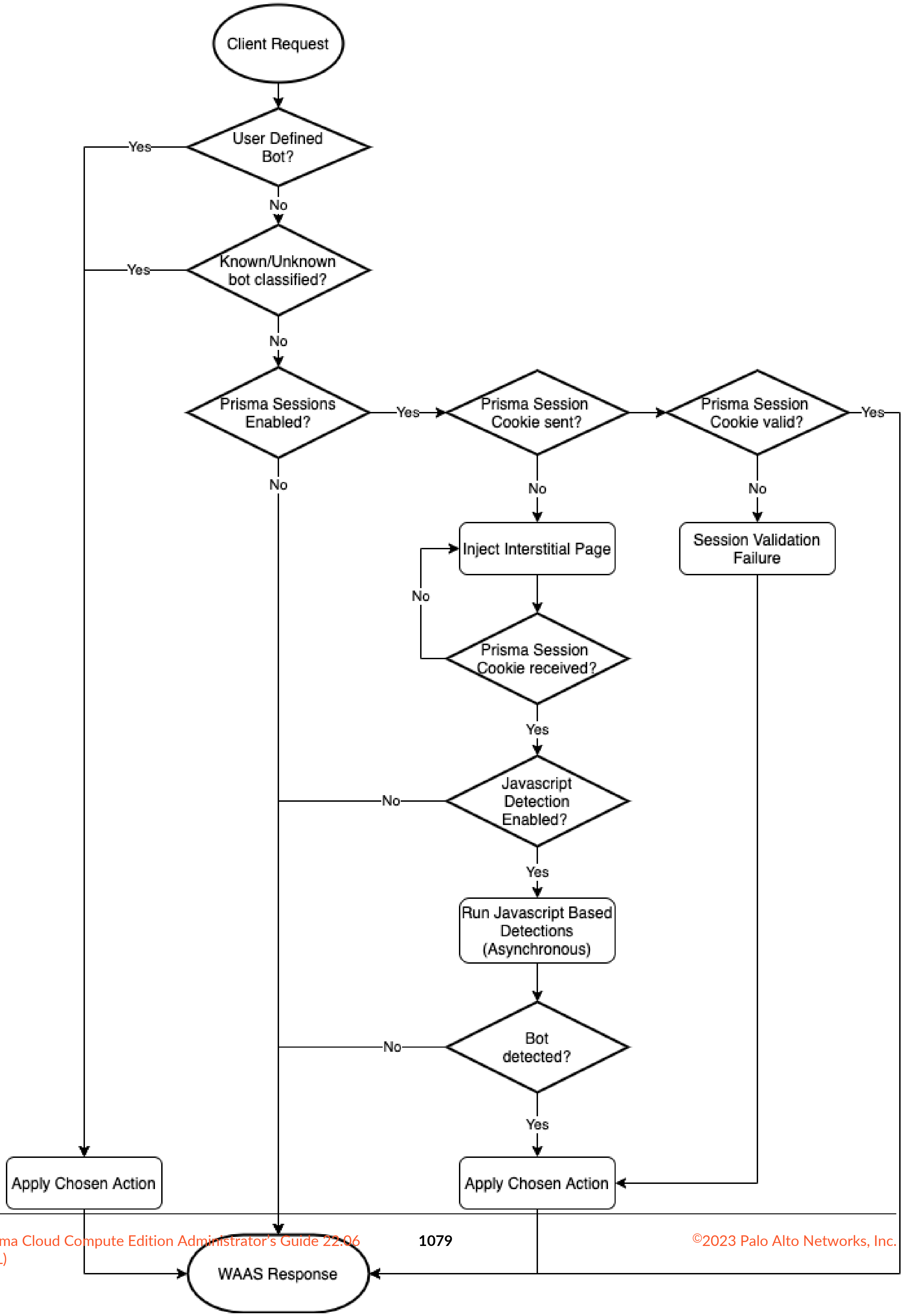
Active detections make use of javascript and Prisma Sessions Cookies to detect and classify bots.

Prisma Session Cookies set by WAAS are encrypted and signed to prevent cookie tampering. In addition, cookies include advanced protections against cookie replay attacks where cookies are harvested and re-used in other clients.

Prisma sessions are intended to address the problem of "Cookie Droppers" by validating clients support of cookies and Javascript before allowing them to reach the origin server. Once enabled, WAAS serves an interstitial page for any request that does not include a valid Prisma Session Cookie. The interstitial page sets a cookie and redirects the client to the requested page using Javascript. A client that doesn't support cookies and Javascript will keep receiving the interstitial page. Browsers can easily proceed to the requested page, and once they possess a valid cookie they will not encounter the interstitial page.

When enabled, javascript will be injected periodically in server responses to collect browser attributes and flag anomalies typical to various bot frameworks. Javascript fingerprint results are received and processed asynchronously and are used to classify session for future requests.

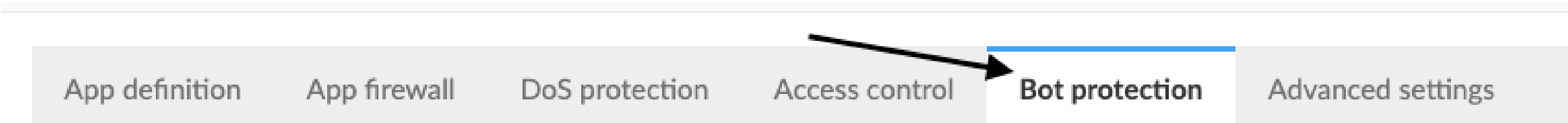
Detection workflow



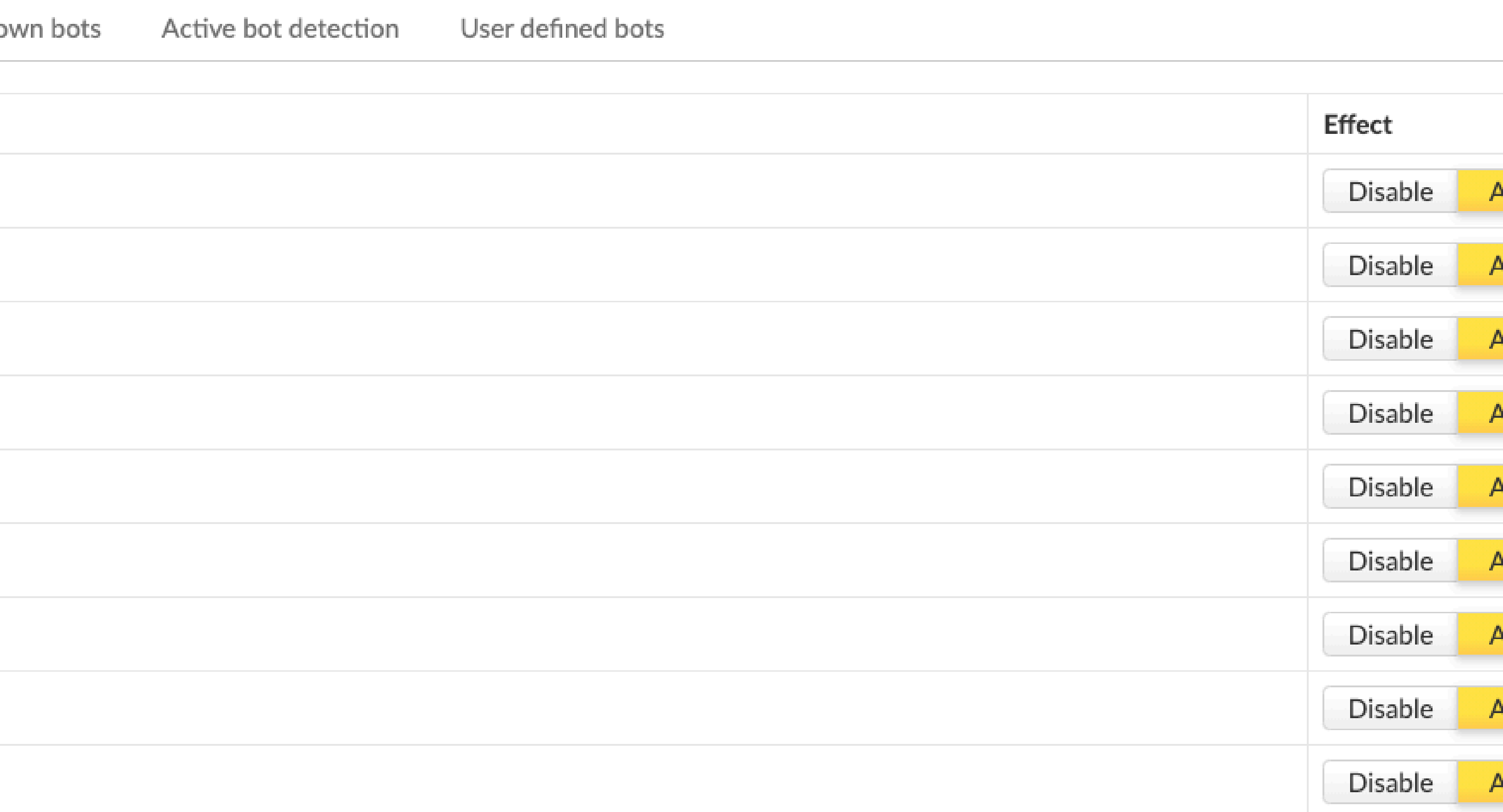
Deploying Bot Protection

Known bots

1. Click on *Bot protection* tab.



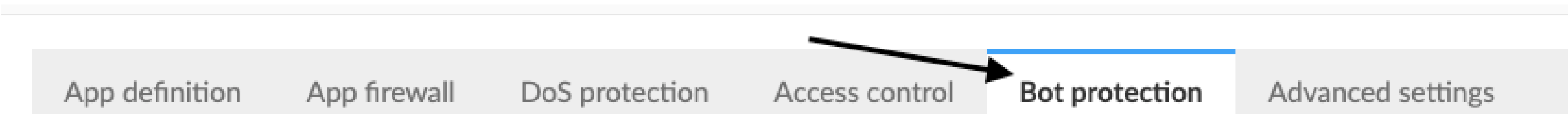
2. Click on *Known Bots*.



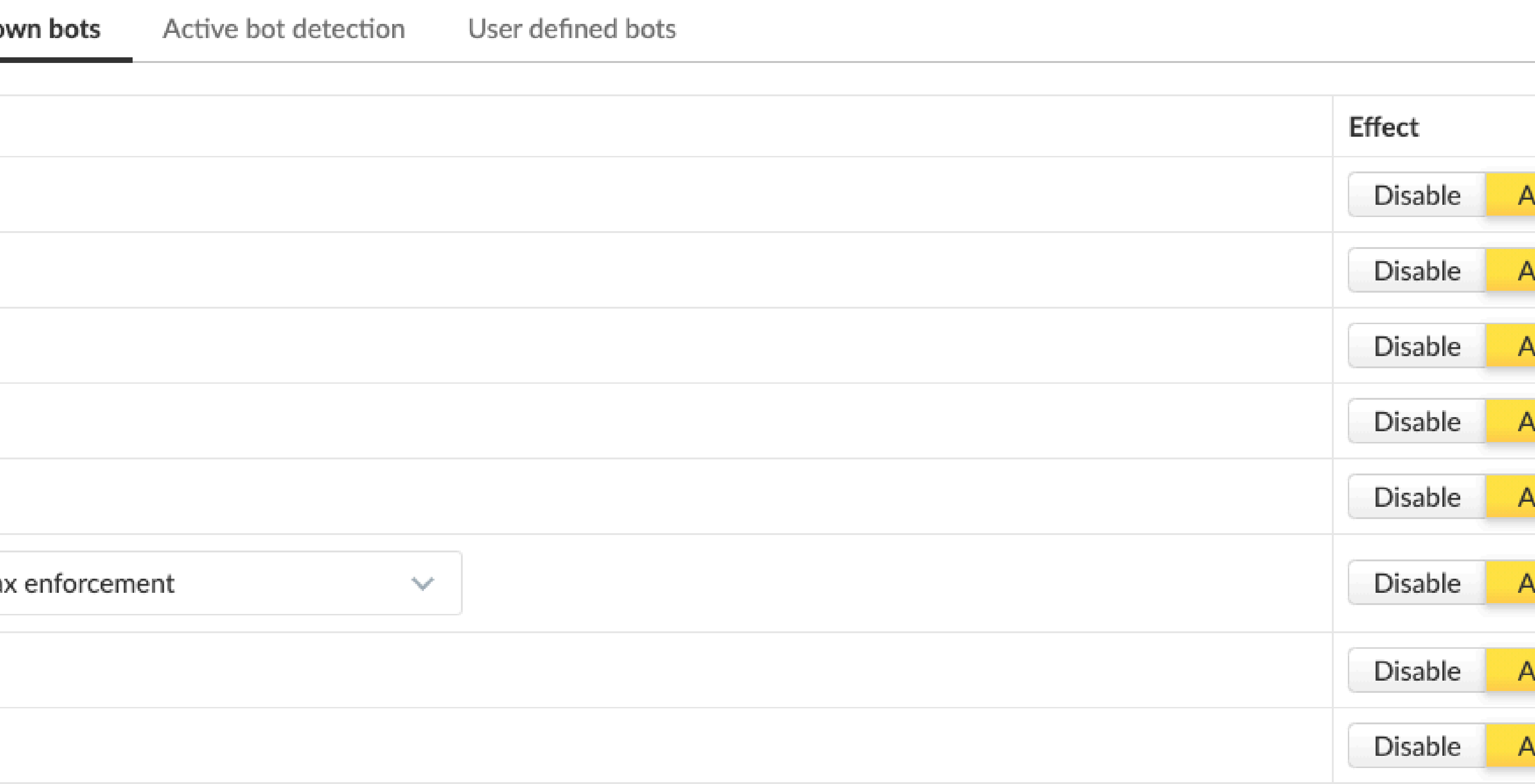
3. Choose actions for each bot category.

Unknown bots

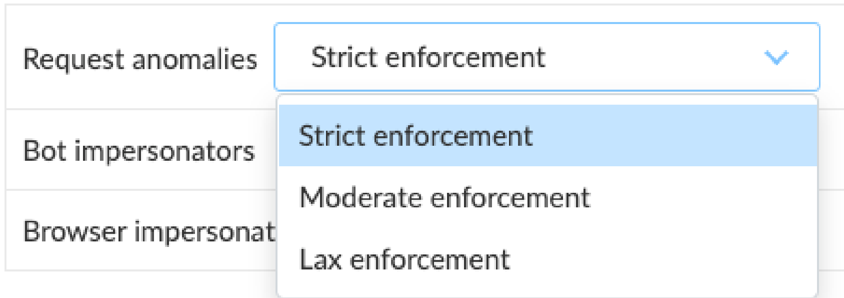
1. Click on *Bot protection* tab.



2. Click on *Unknown Bots*.



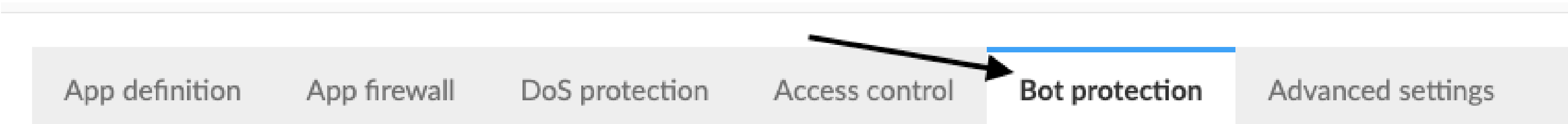
3. Choose actions for each bot category.
    1. If *Request anomalies* are enabled, choose sensitivity threshold



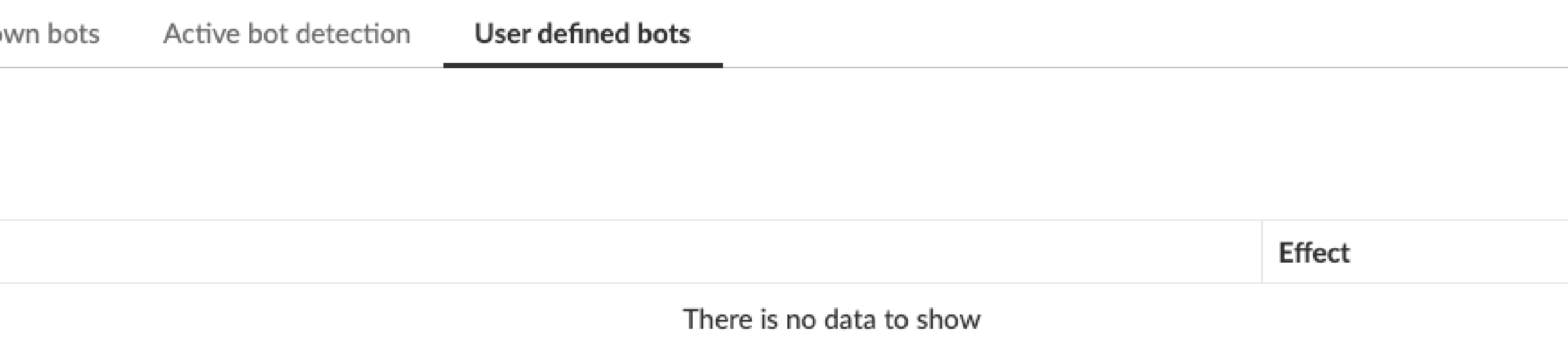
        1. **Strict enforcement** - high sensitivity (a few anomalies suffice for classifying as bot).
        2. **Moderate enforcement** - medium sensitivity.
        3. **Lax enforcement** - low sensitivity.

User-defined bots

1. Click on *Bot protection* tab.



2. Click on *User-defined bots*.

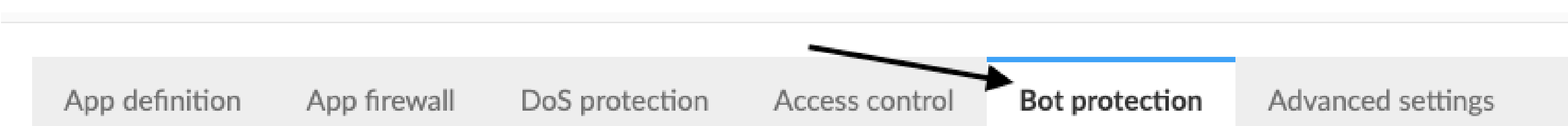


3. Click on *Define new bot* button.
4. Create bot signature by using a combination of the following fields:

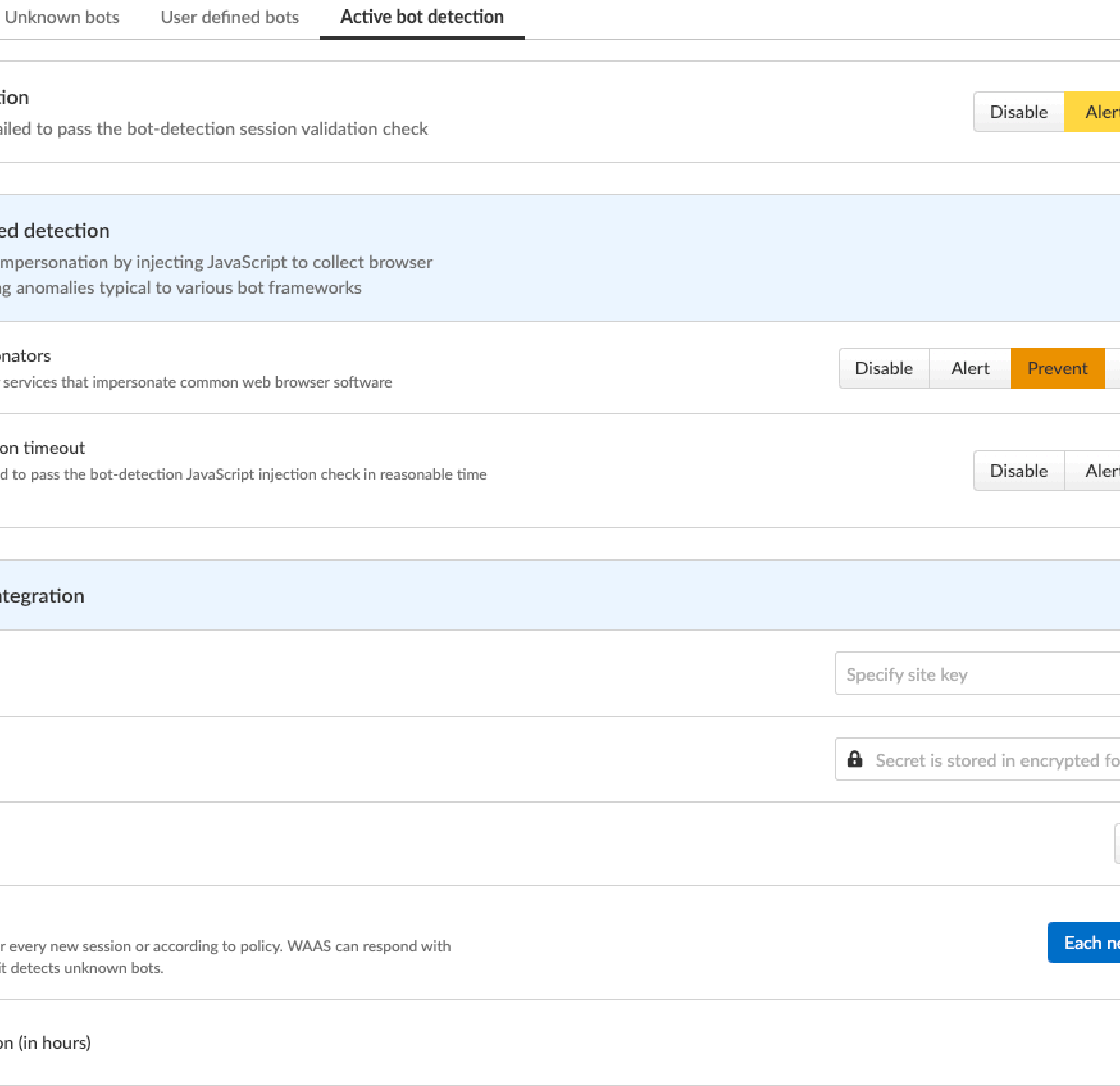
   1. **HTTP Header name** - specify HTTP header name to include in the signature
   2. **Header Values** - comma separated list of values to be matched on in the HTTP header. Wildcard is allowed.
   3. **Inbound IP sources** - specify Network list of IP addresses from which the bot originates.
5. Choose an action to apply.

Enabling active detections

1. Click on *Bot protection* tab.



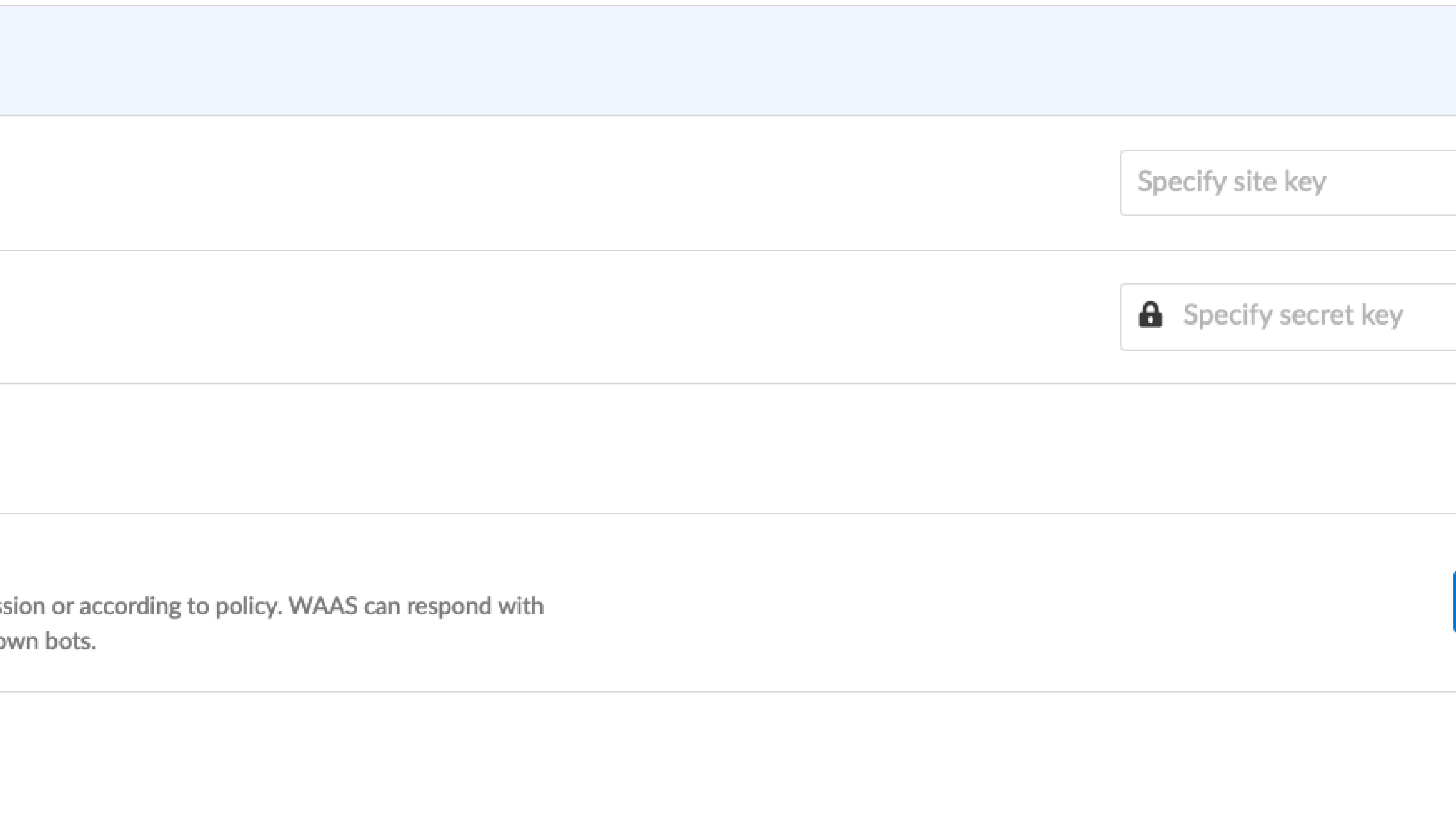
2. Click on *Active bot detections*.



Active Bot detection requires Prisma Sessions Cookies to be enabled in the advanced settings tab.

1. Choose actions to apply.
    1. **Session Validation** - action to apply when WAAS is unable to validate the session, either due to cookie tampering or cookie replay.
    2. **Javascript-based detection** - enable periodic injection of javascript to collect browser attributes and flag anomalies typical to various bot frameworks.
    3. **Javascript injection timeout** - once javascript is enabled, choose action to apply when the browser does not send a response to the javascript injection in a timely manner.
    4. **reCAPTCHA v2 integration** - enable Google's reCAPTCHA v2 integration by specifying the site key, secret key and challenge type. For more details please refer to the elaborated section on reCAPTCHA below.

reCAPTCHA v2 integration



WAAS Users can enable Google's reCAPTCHA v2 integration by specifying the site key, secret key and challenge type. According to the user's preference and settings, WAAS will serve a reCAPTCHA challenge at the beginning of each new session, or when a request is suspected of being sent by an unknown bot.

reCAPTCHA v3 is NOT supported (v2 only).

Deploy reCAPTCHA

Deploy reCAPTCHA.

STEP 1 | Enter the **Site key** provided during the site registration

STEP 2 | Enter the **Secret key** provided during the site registration

STEP 3 | Select the **Challenge type** specified during the site registration

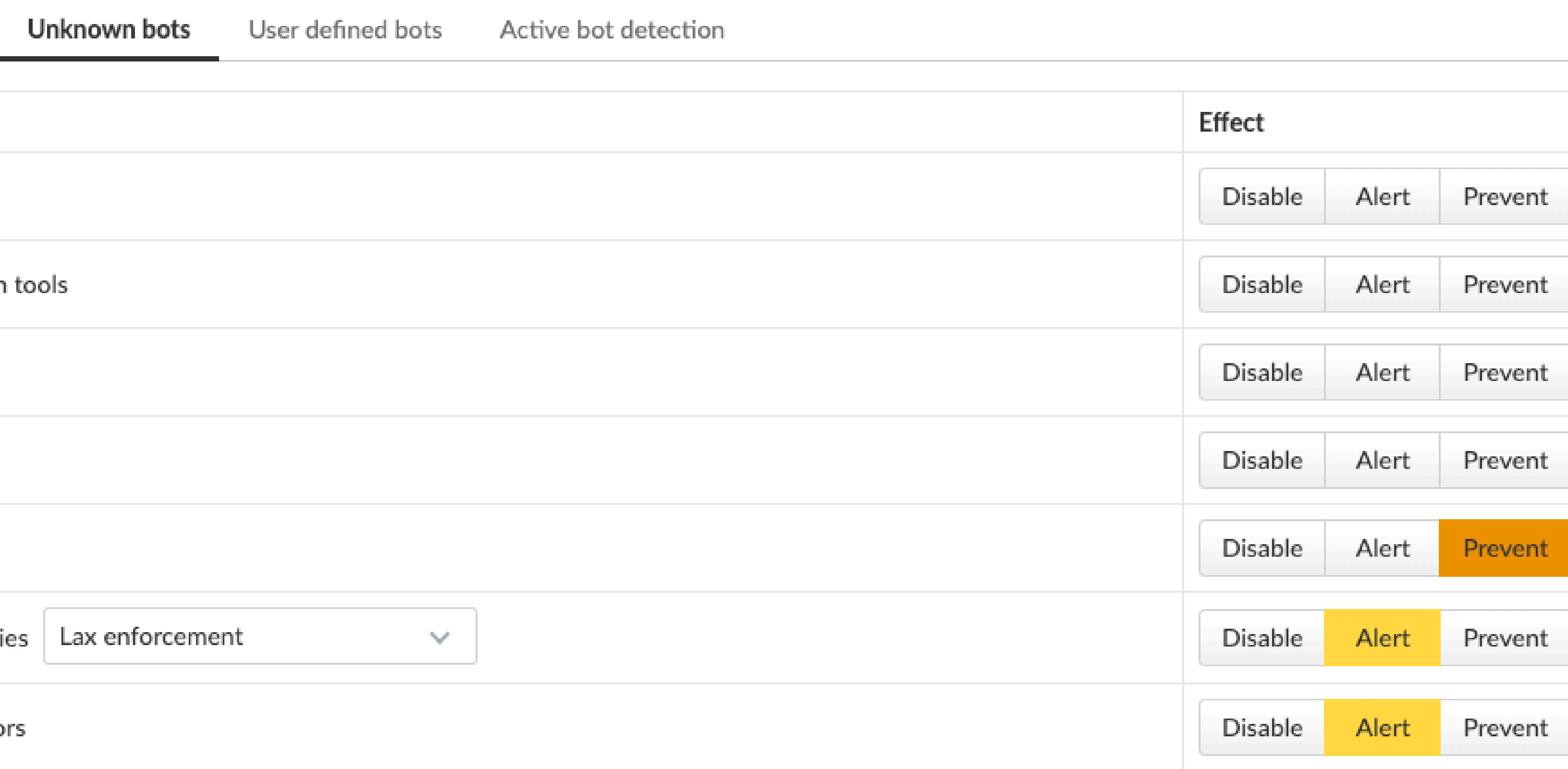
Challenge type MUST match the challenge type selected on the reCAPTCHA site registration form (invisible or checkbox) in order for the reCAPTCHA integration to function properly.

WAAS reCAPTCHA v2 integration does NOT support "reCAPTCHA Android" type

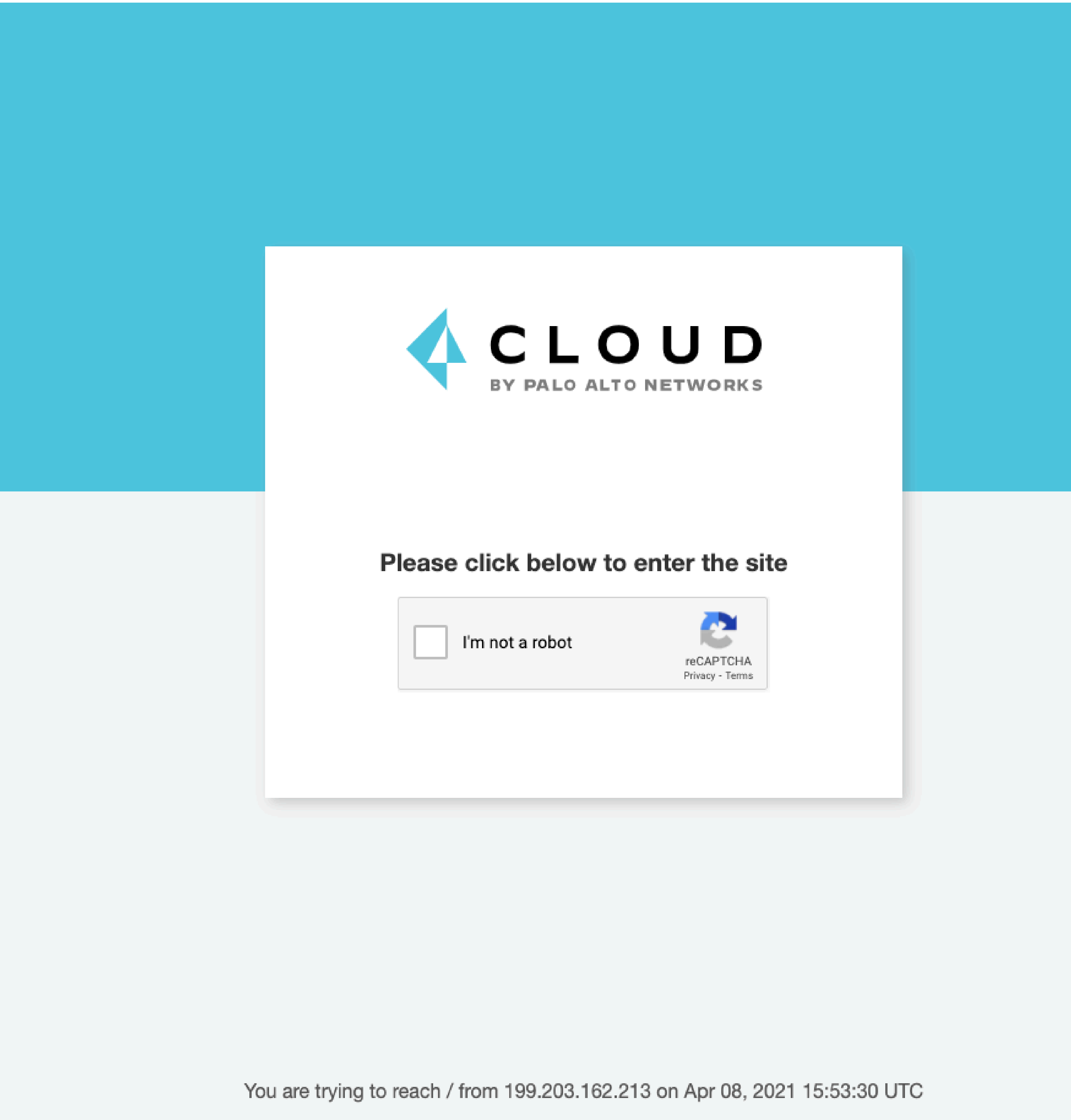
STEP 4 | Choose a preferred **friction**.

> *reCAPTCHA will ONLY be served for GET HTTP requests. WAAS will block requests sent using other methods until a reCAPTCHA challenge is solved and the success result is encoded into the Prisma Session Cookie.*

- **By policy (reCAPTCHA as an action)** - when selected, a new effect will be available in the *Unknown bot* category



When the reCAPTCHA is selected, WAAS will serve an interstitial page with a reCAPTCHA challenge whenever the protection is triggered.


CLOUD
BY PALO ALTO NETWORKS
Please click below to enter the site
I'm not a robot
reCAPTCHA
Privacy - Terms
You are trying to reach / from 199.203.162.213 on Apr 08, 2021 15:53:30 UTC

If the end-user successfully passes the challenge, it will be recorded in the Prisma Session Cookie for the duration of the *Success Expiration* setting (default is 24 hours).

- **Each new session** - Every new session will start with an interstitial page containing the reCAPTCHA challenge. If the end-user successfully passes the challenge, it will be recorded in the Prisma Session Cookie for the duration of the *Success Expiration* setting (default is 24 hours).

STEP 5 | Set the **Success Expiration** (in hours).

This field determines how long a successful solution to a reCAPTCHA challenge will remain valid. Once the expiration date has passed, a new reCAPTCHA challenge will be presented (based on the selected friction settings).

Bot protection events

- **Known bots** - if a known bot is detected, the event message and attack type will provide details regarding the bot classification.
- **Unknown bots** - if an unknown bot is detected, the event message and attack type would provide details regarding the bot classification
- **User-defined bots** - a used-defined bot has been detected.
- **Session Validation** - The web client failed to pass the bot-detection session validation checks (e.g. prisma session cookie has been tampered with etc.).
- **Javascript Injection Timeout** - the web client failed to pass the bot-detection JavaScript injection check in reasonable time (timeout).
- **Missing Cookie** - client made a non GET request without a cookie.
- **Failed reCAPTCHA verification: <error returned from Google verification API>** - Google's reCAPTCHA verification API has responded with an error message.
- **<HTTP method> request when a reCAPTCHA page is required** - As mentioned in the reCAPTCHA v2 integration section, reCAPTCHA will ONLY be served for GET HTTP requests. WAAS will block requests sent using other methods until a reCAPTCHA challenge is solved and an event carrying this message will be registered.

Bot Protection Actions

Requests that trigger a WAAS bot protection are subject to one of the following actions:

- **Alert** - The request is passed to the protected application and an audit is generated for visibility.
- **Prevent** - The request is denied from reaching the protected application, an audit is generated and WAAS responds with an HTML page indicating the request was blocked.
- **Ban** - Can be applied on either IP or Prisma Session IDs. All requests originating from the same IP/Prisma Session to the protected application are denied for the configured time period (default is 5 minutes) following the last detected attack.
- **Allow (available for user-defined bots)** - request is forwarded to the protected application and no audit is generated.

- **reCAPTCHA** - a page will be served with a reCAPTCHA v2 challenge to be solved before proceeding to the website.

A message at the top of the page indicates the entity by which the ban will be applied (IP or Prisma Session ID). When the X-Forwarded-For HTTP header is included in the request headers, ban will apply based on the first IP listed in the header value (true client IP).

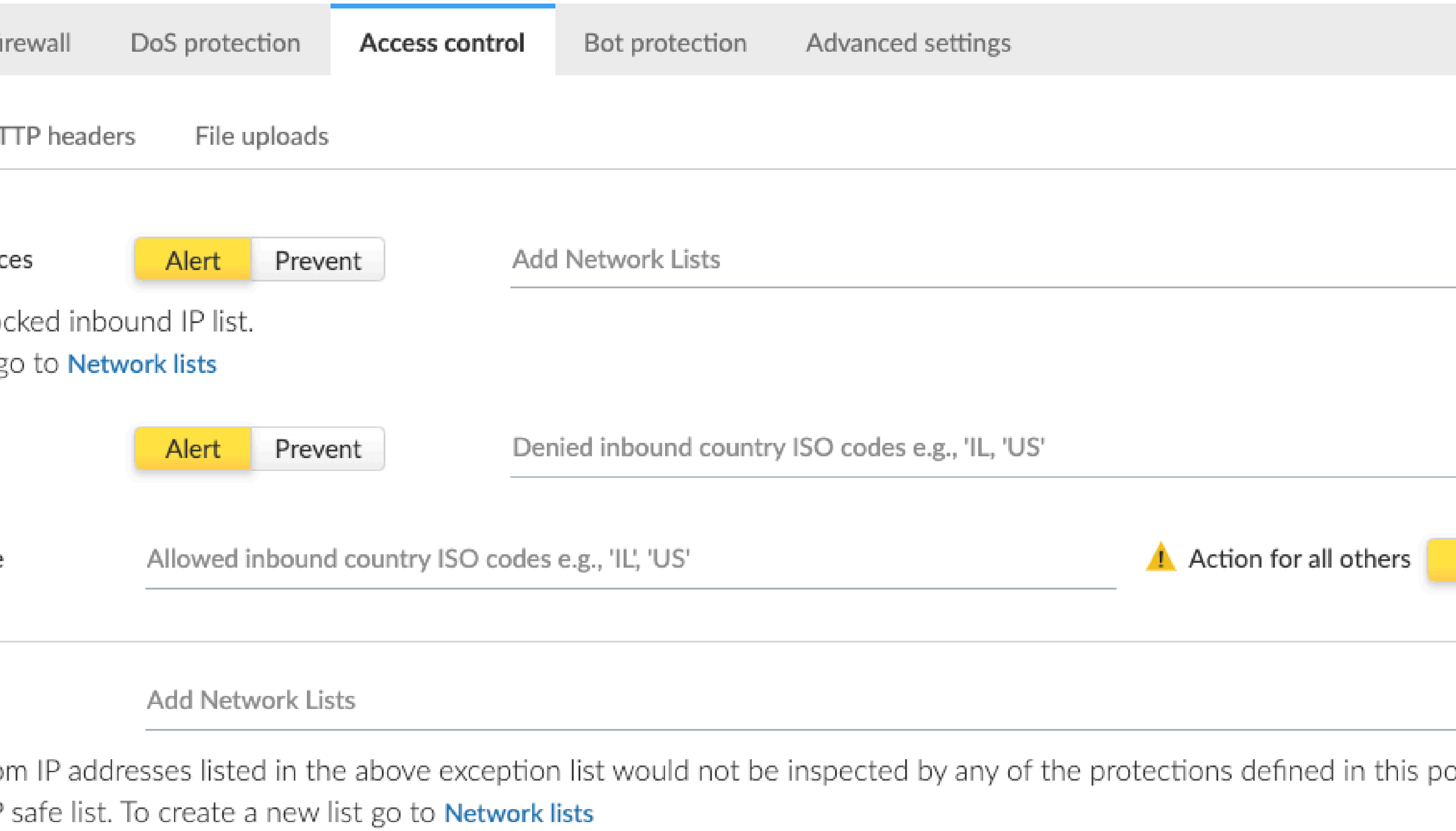
To enable ban by Prisma Session ID, Prisma Session Cookies *has to be enabled in the Advanced Settings tab. for more information please refer to the* Advanced Settings *help page.*

WAAS implements state, which is required for banning user sessions by IP address or Prisma Sessions. Because Defenders do not share state, any application that is replicated across multiple nodes must enable IP stickiness on the load balancer.

WAAS Access Controls

Edit on GitHub

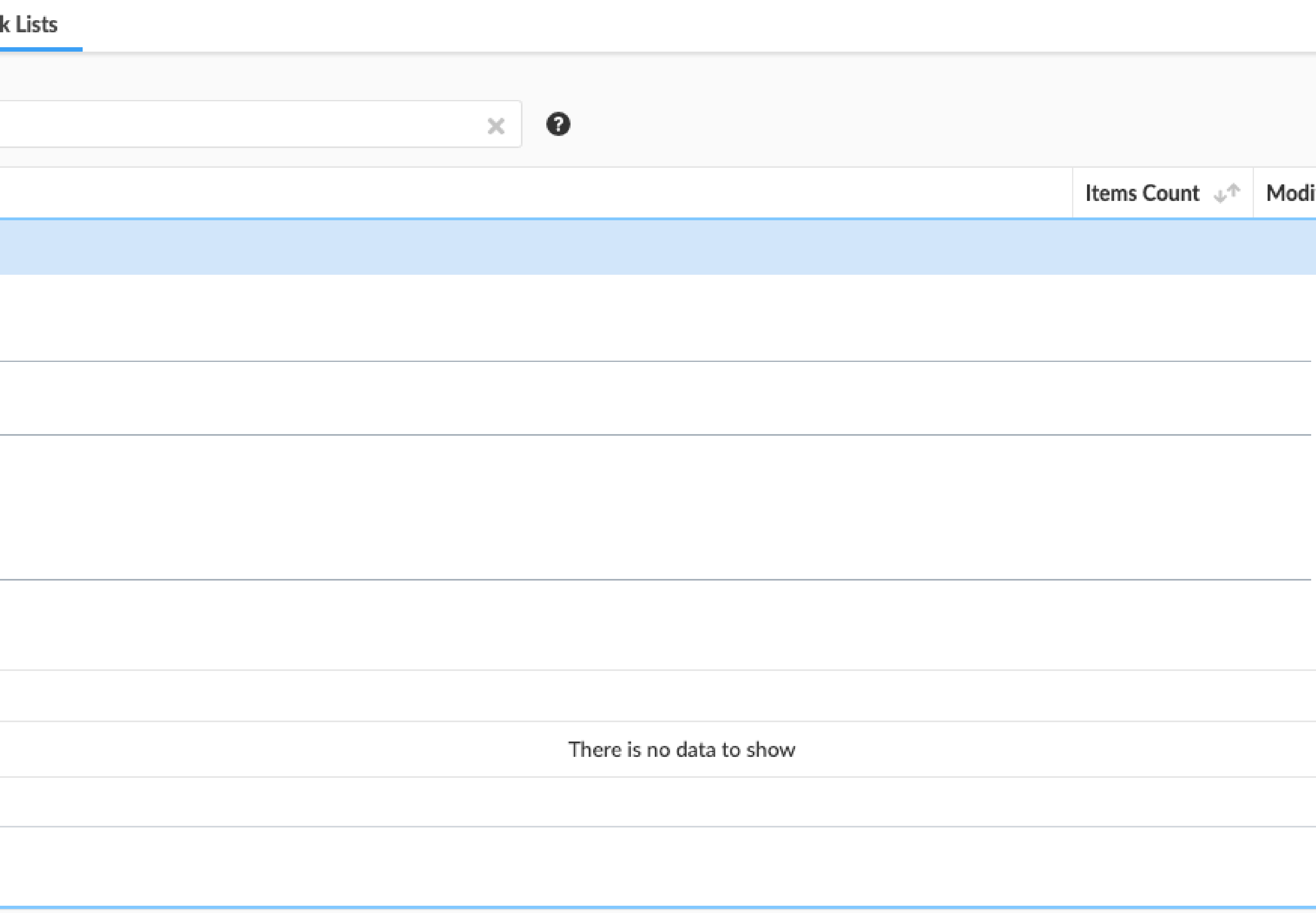
WAAS allows for control over how applications and end-users communicate with the protected web application.



Network Lists

Network Lists allow administrators to create and maintain named IP address lists e.g. "Office Branches", "Tor and VPN Exit Nodes", "Business Partners", etc. List entries are composed of IPv4 addresses or IP CIDR blocks.

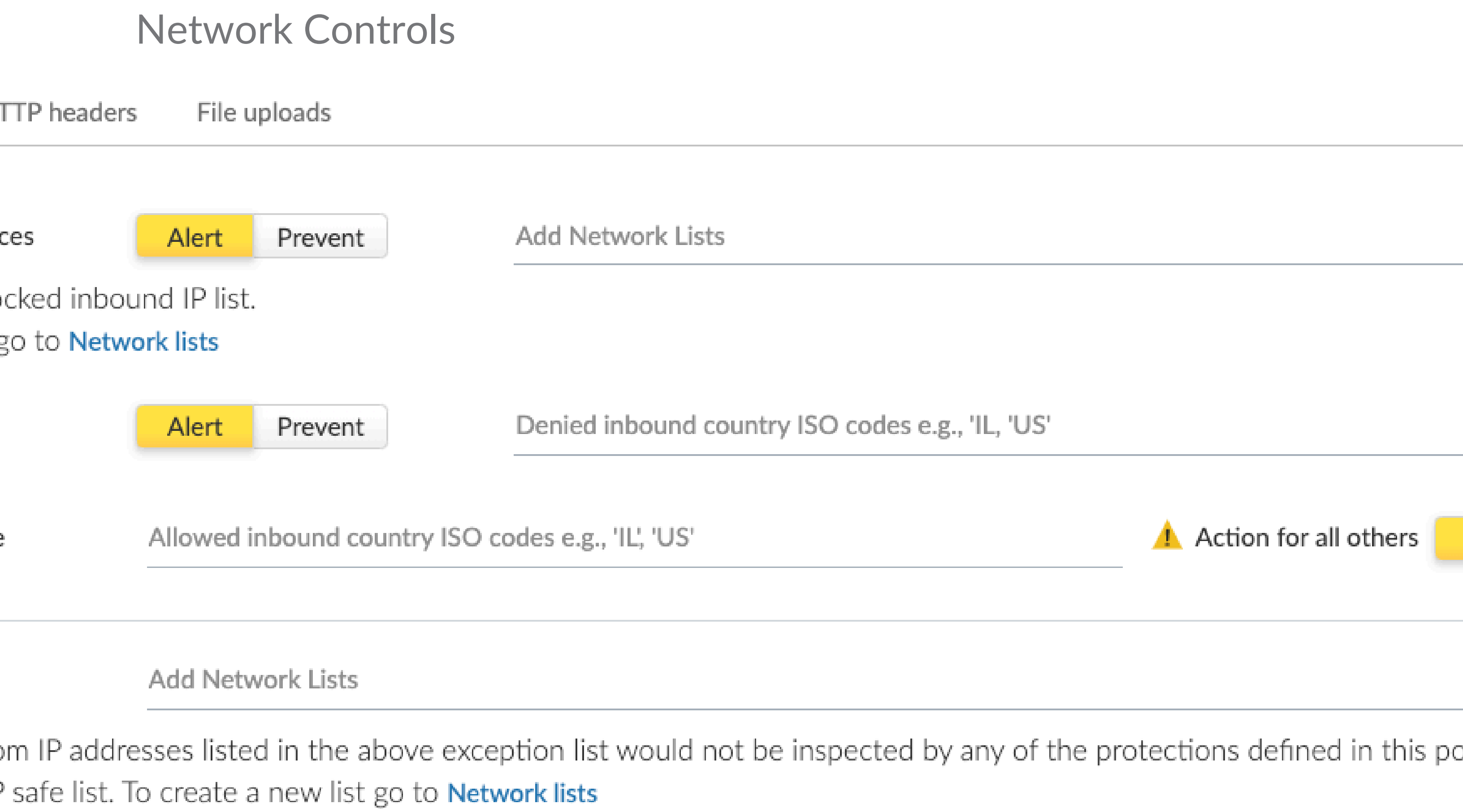
To access **Network Lists**, open Console, go to **Defend > WAAS** and select the **Network List** tab.



Lists can be updated manually or via batch importing of entries from a CSV file. Once defined, **Network Lists** can be referenced and used in IP-based access control, user-defined bots and DoS protection.

To export lists in CSV format, click **export CSV**.

- *When importing IP addresses or IP CIDR blocks from a CSV file, first record value should be set to "ip" (case sensitive).*
- *IPv6 addresses are currently not supported.*



IP-based access control

Network lists can be specified in:

- ***Denied inbound IP Sources*** - WAAS applies selected action (Alert or Prevent) for IP addresses in network lists.
- ***IP Exception List*** - Traffic originating from IP addresses listed in this category will not be inspected by any of the protections defined in this policy.

> - *When the X-Forwarded-For HTTP header is included in the request headers, actions will apply based on the first IP listed in the header value (true client IP).*
> - *Practice caution when adding network lists to the IP Exception List because protections will not be applied for traffic originating from these IP addresses.*

Country-Based Access Control

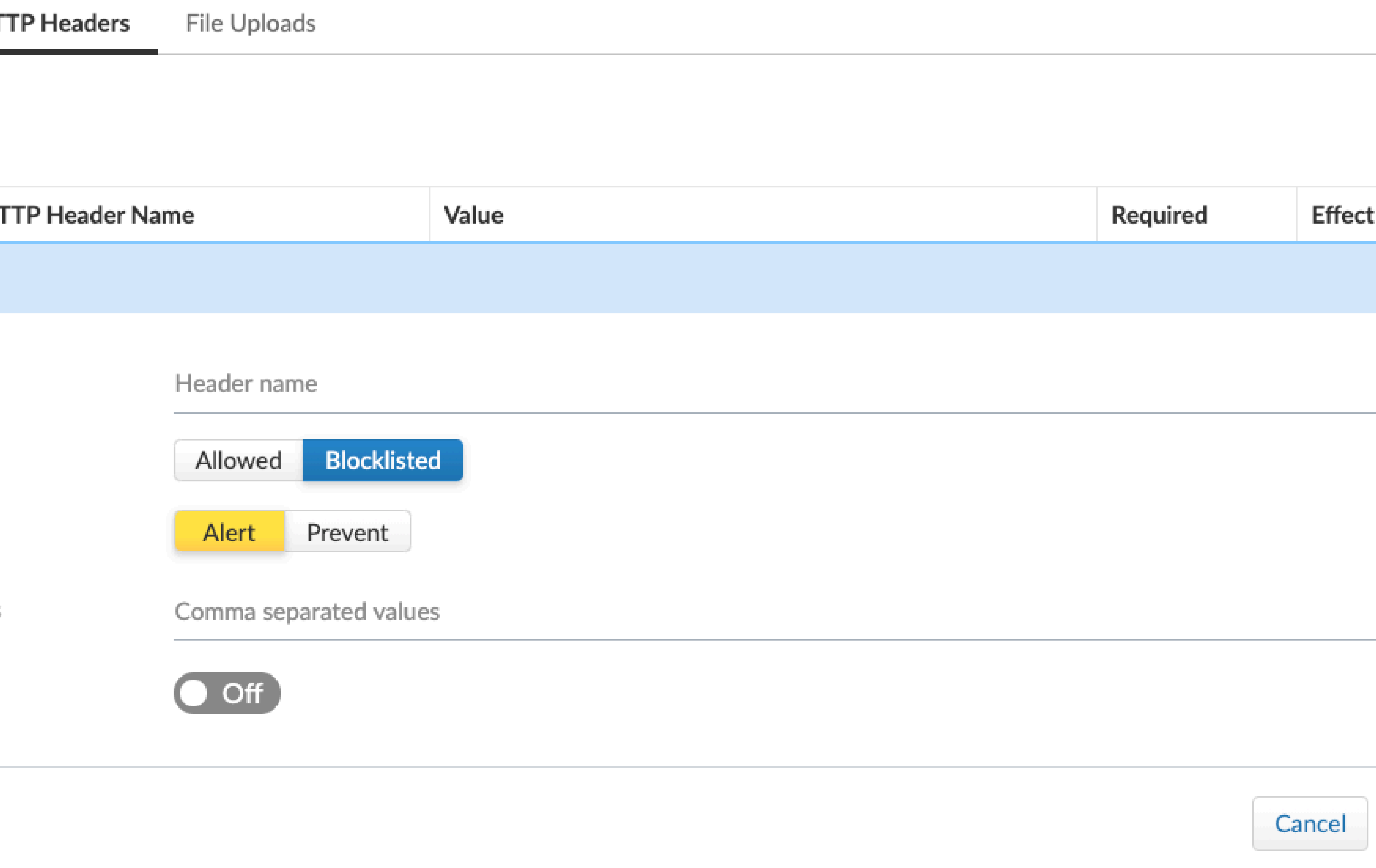
Specify country codes, ISO 3166-1 alpha-2 format, in one of the following categories (mutually exclusive):

- ***Denied Inbound Source Countries*** - WAAS applies selected action (Alert or Prevent) for requests originating from the specified countries.

- ***Allowed Inbound Source Countries*** - Requests originating from specified countries will be forwarded to the application (pending inspection). WAAS will apply action of choice (Alert or Prevent) on all other requests not originating from the specified countries.

Country of origin is determined by the IP address associated with the request. When the X-Forwarded-For HTTP header is included in the request headers, Country of origin is determined based on the first IP address listed in the header value (true client IP).

HTTP Header Controls



WAAS lets you block or allow requests which contain specific strings in HTTP headers by specifying a header name and a value to match. The value can be a full or partial string match. Standard pattern matching is supported.

If the **Required** toggle is set to **On** WAAS will apply the defined action on HTTP requests in which the specified HTTP header is missing. When the **Required** toggle is set to **Off** no action will be applied for HTTP requests missing the specified HTTP header.

HTTP Header fields consist of a name, followed by a colon, and then the field value. When decoding field values, WAAS treats all commas as delimiters. For example, the *Accept-Encoding* request header advertises which compression algorithm the client supports.

```
Accept-Encoding: gzip, deflate, br
```

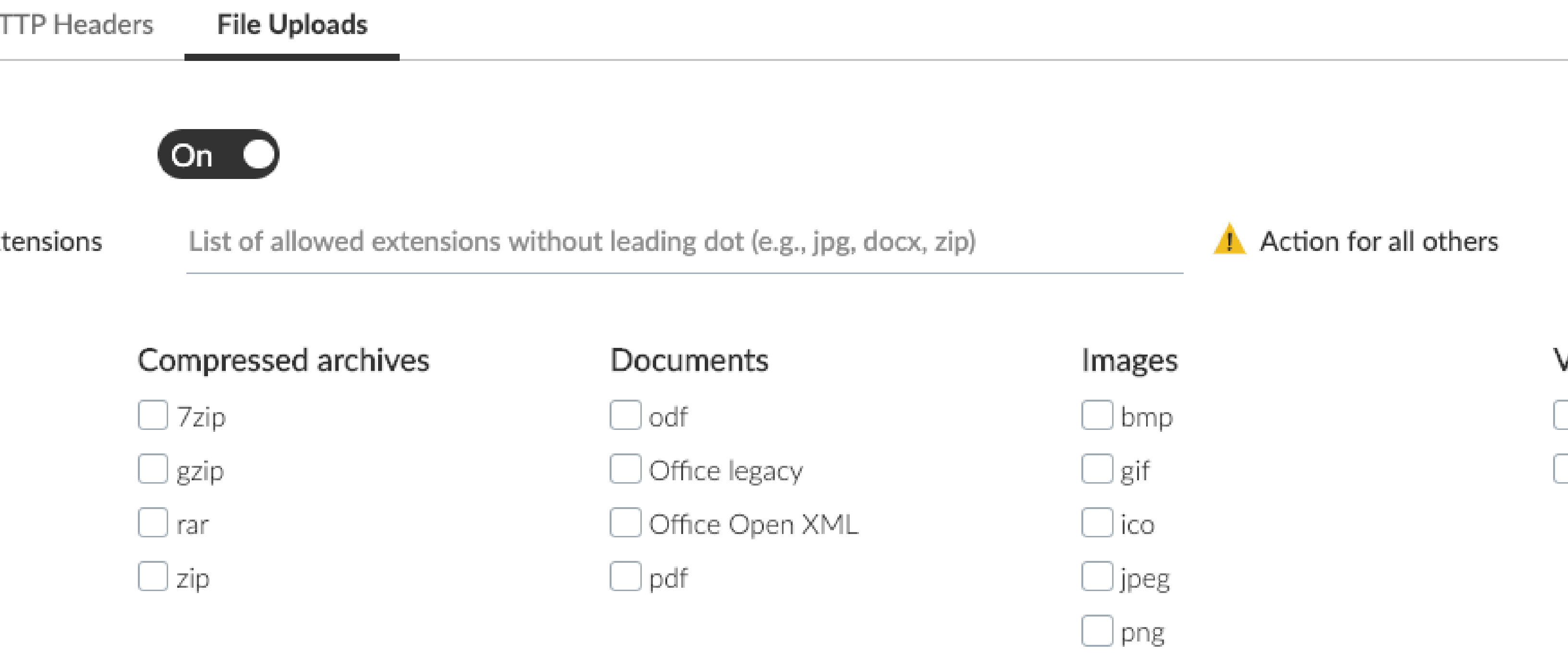
WAAS rules do not support exact matching when the value in a multi-value string contains a comma because WAAS treats all commas as delimiters. To match this type of value, use wildcards. For example, consider the following header:

```
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36
 (KHTML, like Gecko) Chrome/74.0.3729.108 Safari/537.36
```

To match it, specify the following wildcard expression in your WAAS rule:

```
Mozilla/5.0*
```

File Upload Controls



Attackers may try to upload malicious files (e.g. malware) to your systems. WAAS protects your applications against malware dropping by restricting uploads to just the files that match any allowed content types. All other files will be blocked.

Files are validated both by their extension and their magic numbers. Built-in support is provided for the following file types:

- Audio: aac, mp3, wav.
- Compressed archives: 7zip, gzip, rar, zip.
- Documents: odf, pdf, Microsoft Office (legacy, Ooxml).

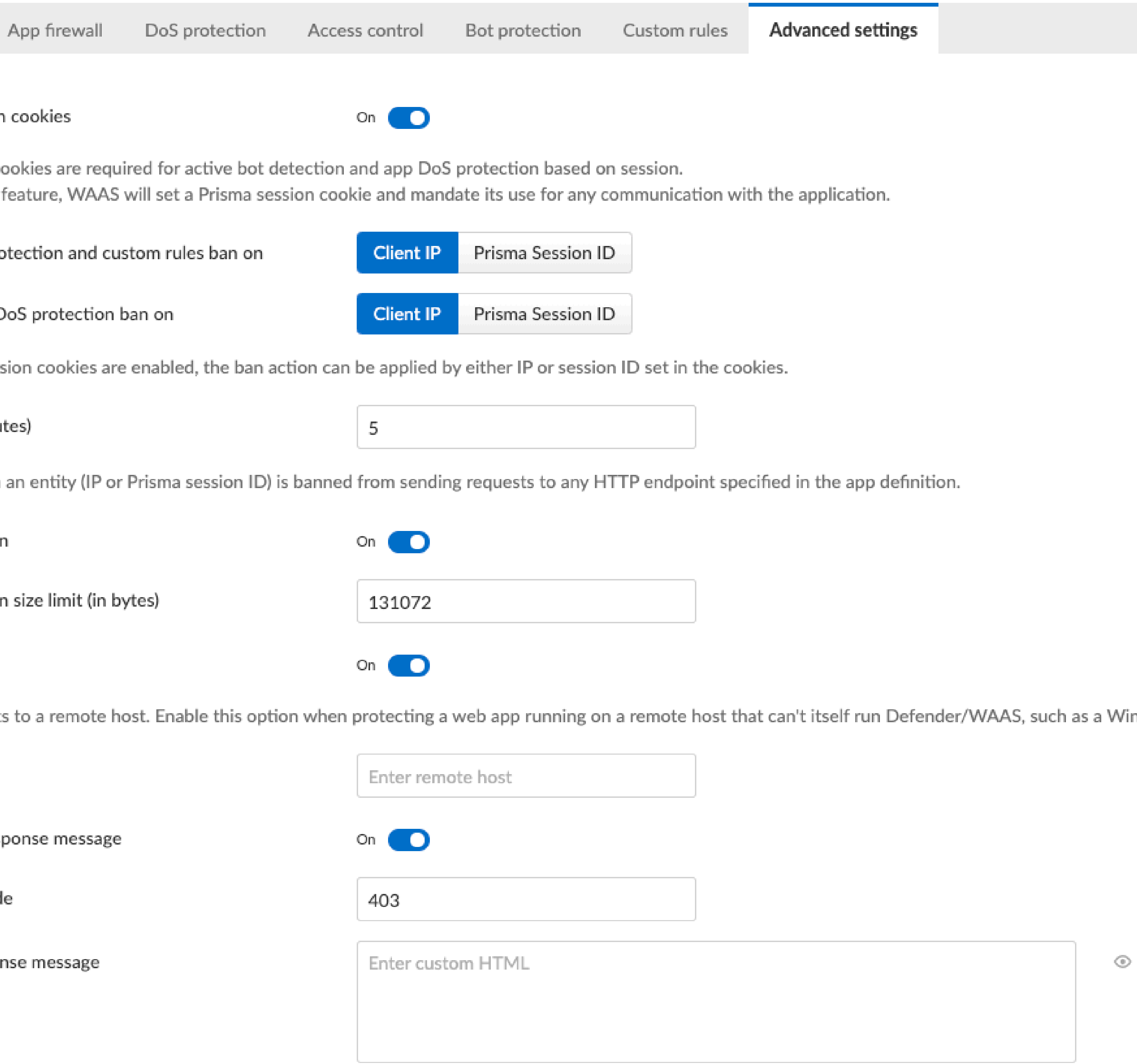
- Images: bmp, gif, ico, jpeg, png.
- Video: avi, mp4.

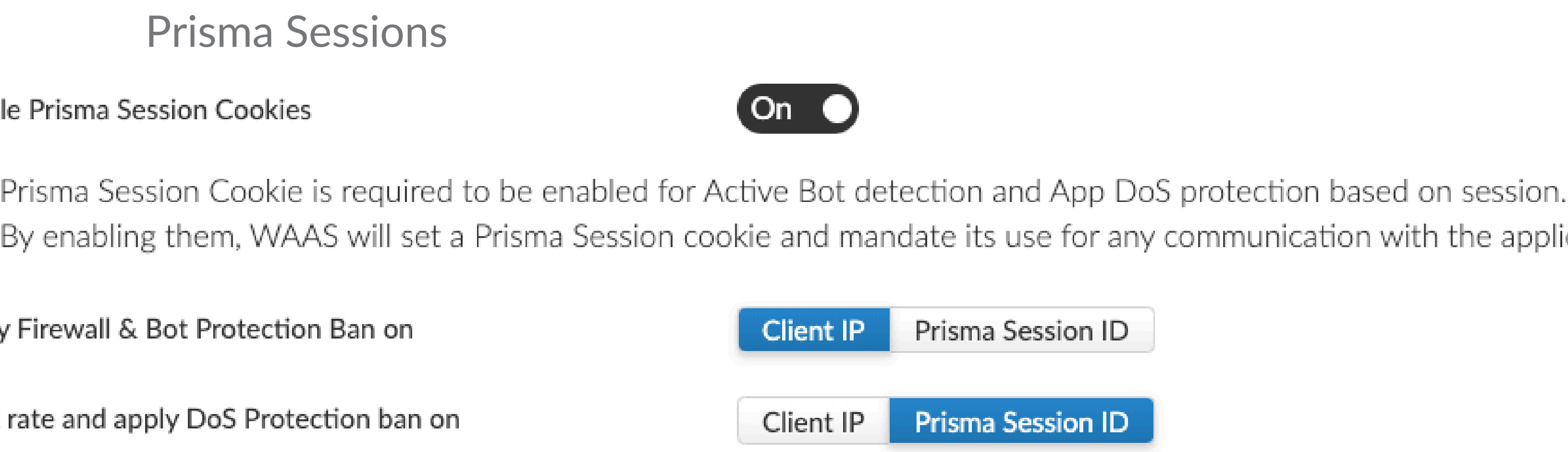
WAAS rules let you explicitly allow additional file extensions. These lists provide a mechanism to extend support to file types with no built-in support, and as a fallback in case Prisma Cloud's built-in inspectors fail to correctly identify a file of a given type. Any file with an allowed extension is automatically permitted through the firewall, regardless of its 'magic number'.

Advanced Settings

Edit on GitHub

Advanced settings control various aspects of WAAS features.


App firewall
DoS protection
Access control
Bot protection
Custom rules
Advanced settings
n cookies
On
ookies are required for active bot detection and app DoS protection based on session.
feature, WAAS will set a Prisma session cookie and mandate its use for any communication with the application.
otection and custom rules ban on
Client IP
Prisma Session ID
DoS protection ban on
Client IP
Prisma Session ID
sion cookies are enabled, the ban action can be applied by either IP or session ID set in the cookies.
tes)
5
an entity (IP or Prisma session ID) is banned from sending requests to any HTTP endpoint specified in the app definition.
n
On
n size limit (in bytes)
131072
On
s to a remote host. Enable this option when protecting a web app running on a remote host that can't itself run Defender/WAAS, such as a Wir
Enter remote host
ponse message
On
le
403
nse message
Enter custom HTML



Prisma sessions are intended to address the problem of "Cookie Droppers" by validating clients support of cookies and Javascript before allowing them to reach the origin server.

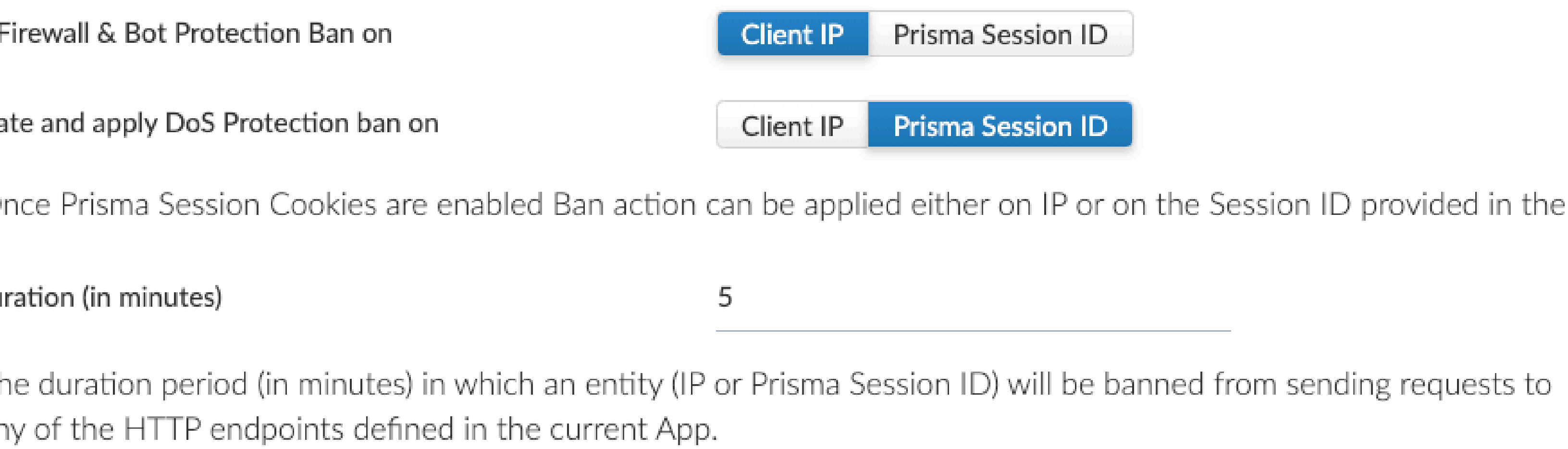
Once enabled, WAAS serves an interstitial page for any request that does not include a valid Prisma Session Cookie. The interstitial page sets a cookie and redirects the client to the requested page using Javascript.

A client that doesn’t support cookies and Javascript will keep receiving the interstitial page. Browsers can easily proceed to the requested page, and once they possess a valid cookie they will not encounter the interstitial page.

Users should take caution when enabling Prisma Session Cookies along with API protection as APIs are often accessed by "primitive" automation clients. Avoid enabling Prisma Session Cookies on such endpoints unless you are certain all clients accessing the protected API endpoints support BOTH cookies AND Javascript. Users are able to allow "primitive" clients by adding them as user-defined bots and setting the bot action to Allow. Allowed user-defined bots will not be served with an interstitial page and their requests will be forwarded to the protected application.

Prisma Session Cookies set by WAAS are encrypted and signed to prevent cookie tampering. In addition, cookies include advanced protections against cookie replay attacks where cookies are harvested and re-used in other clients.

Ban

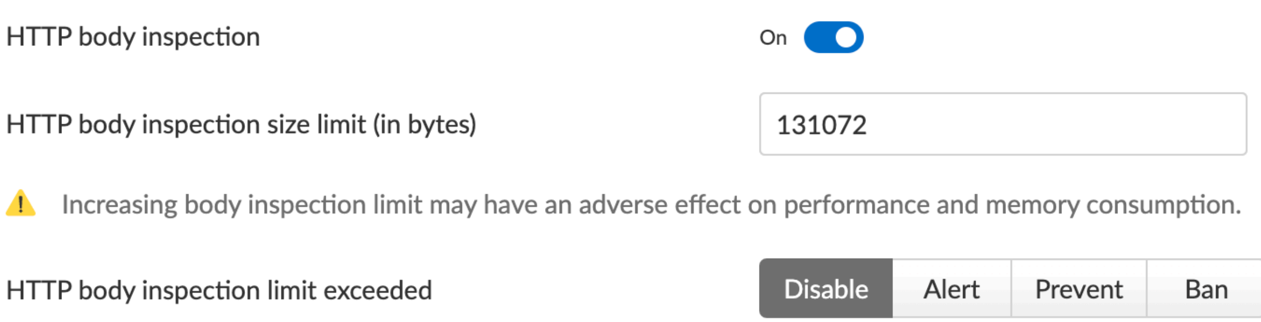


Ban action is available in the *App firewall*, *DoS protection* and *Bot protection* tabs. If triggered this action would prevent access to the protected endpoints of the app for a time period set by users (default is set to 5 minutes.)

If Prisma Session Cookies are enabled, users are able to apply ban by either *Prisma Session Id* or by IP.

When the X-Forwarded-For HTTP header is included in the request headers, actions will apply based on the first IP listed in the header value (true client IP).

Body Inspection



Body inspection can be disabled or limited up to a configurable size (in Bytes).

WAAS body inspection limit is 131,072 Bytes (128Kb). WAAS protection is subject to one of the following actions when the body inspection limit exceeds:

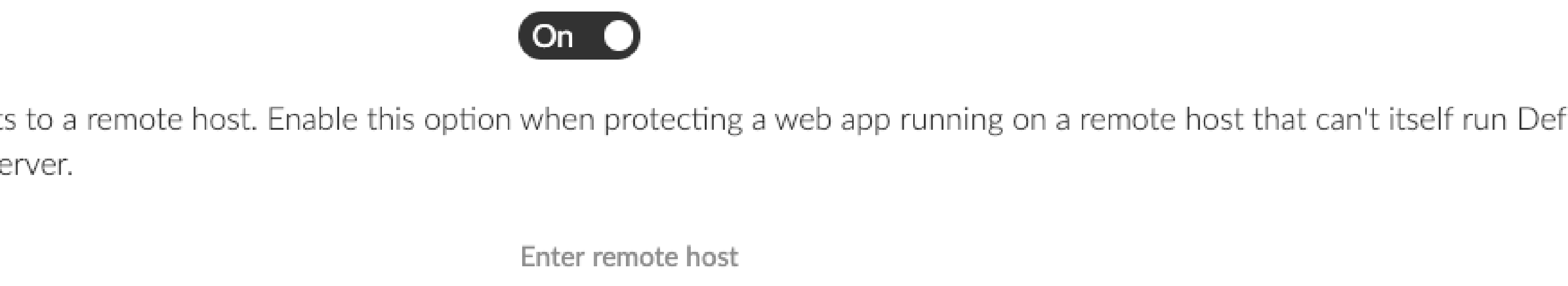
- **Disable** - The request is passed to the protected application.
- **Alert** - The request is passed to the protected application and an audit is generated for visibility.
- **Prevent** - The request is denied from reaching the protected application, an audit is generated and WAAS responds with an HTML page indicating the request was blocked.

- **Ban** - Can be applied on either IP or Prisma Session IDs. All requests originating from the same IP/Prisma Session to the protected application are denied for the configured time period (default is 5 minutes) following the last detected attack.

A minimum Defender version of 22.01 (Joule) is required to enforce body inspection limitations using the above described actions.

To enable ban by Prisma Session ID, Prisma Session Cookies has to be enabled in the Advanced Settings tab. for more information please refer to the Advanced Settings help page.

Remote Host



This option is intended to defend web applications running on remote hosts which can not be protected directly by WAAS (e.g. Windows Servers).

Remote host option is only available for WAAS host rules.

Use-case scenario:

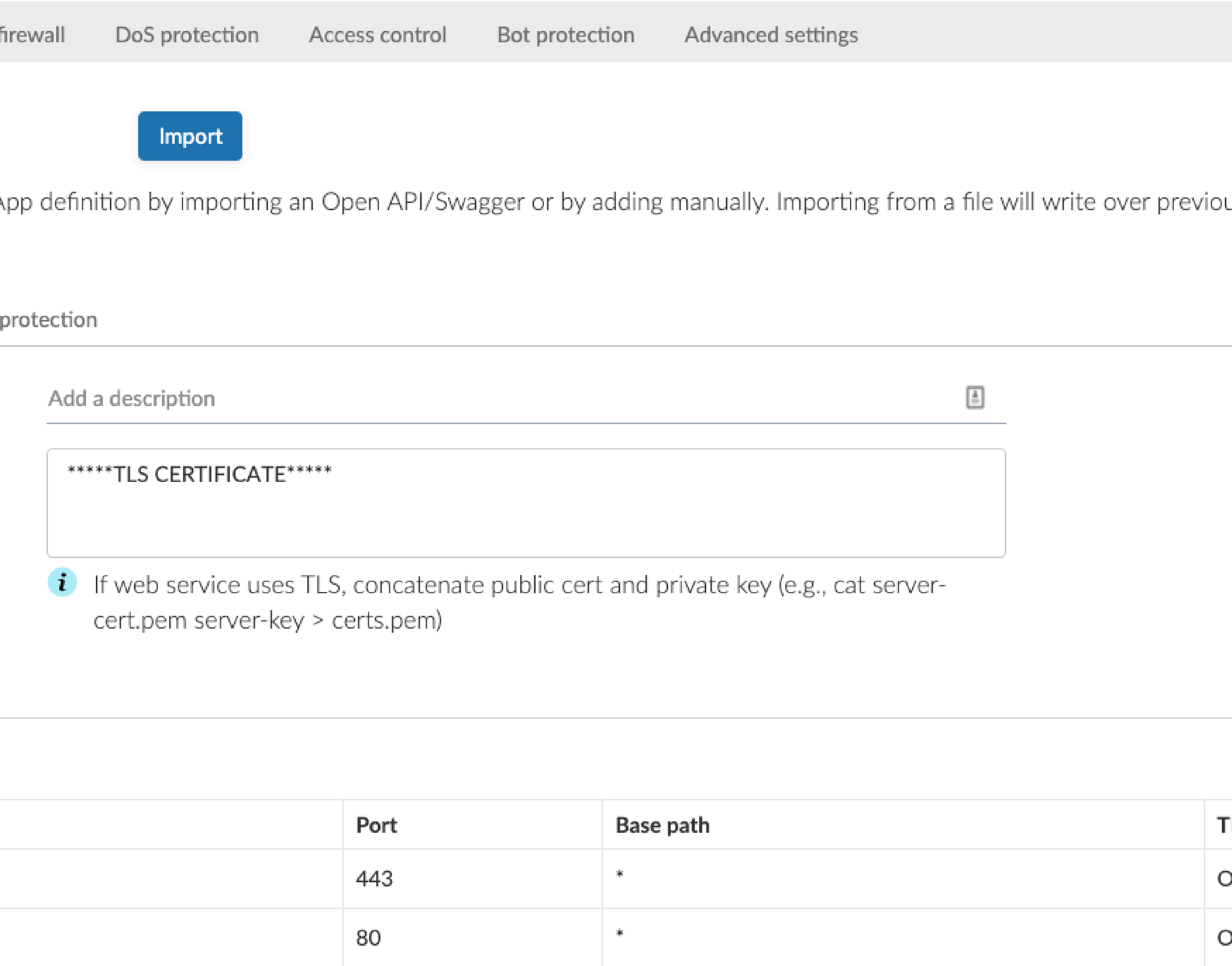
1. A "middle-box" host instance with WAAS supported OS should be set up.
2. Traffic to the web application should be directed to the "middle-box" host.
3. Ports on the "middle-box" host to which traffic is directed to should be unused (WAAS will listen on these ports for incoming requests).
4. WAAS host rule with *Remote host* settings should be deployed to protect the "middle-box" host.
5. Incoming traffic to the "middle-box" host will be forwarded to the specified address (resolvable hostname or IP address) by WAAS.

WAAS sets the original Host HTTP header value in the X-Forwarded-Host HTTP header of the forwarded request. The Host header is set to the hostname or IP mentioned in the WAAS settings.

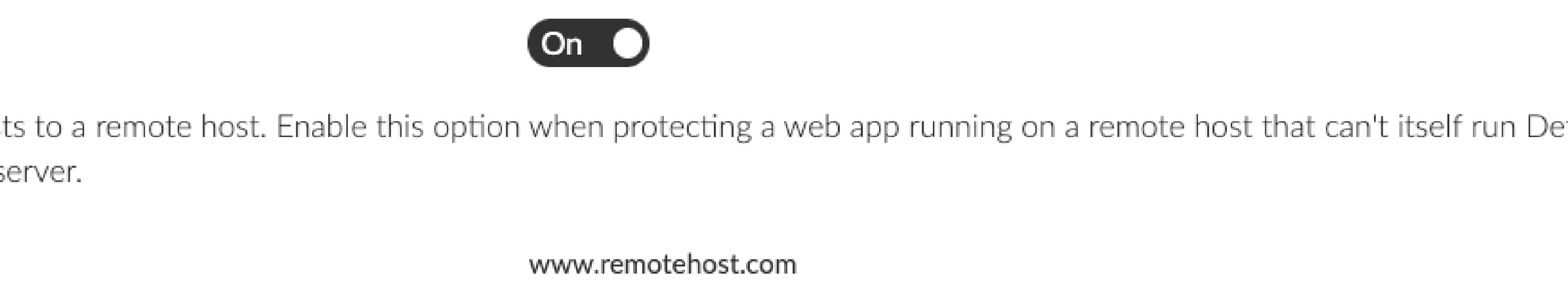
Use of TLS and destination port is determined by the endpoint configuration in the *App definition* tab.

Example:

The following protected endpoints are defined in the *App definition* tab:



Remote host has been configured as follows:

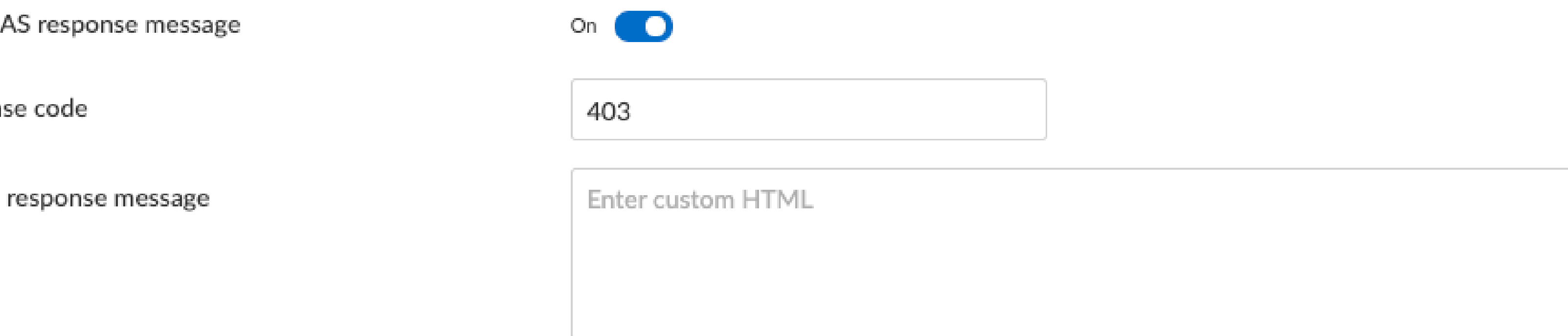


Expected result would be as follows:

- HTTPS traffic to www.example1.com on port 443 would be forwarded via HTTPS to www.remotehost.com
- HTTP traffic to www.example1.com on port 80 would be forwarded via HTTP to www.remotehost.com

Protected endpoints with TLS enabled will not forward non-TLS HTTP requests.

Customize WAAS response message



Users can customize the response HTML and HTTP status code that are returned by WAAS when a ***Prevent*** or ***Ban*** effect occurs:

- **Prevent response code** - HTTP response code
- **Custom WAAS response message** - HTML code to be served. Click on Preview html for a preview of the rendered HTML code.

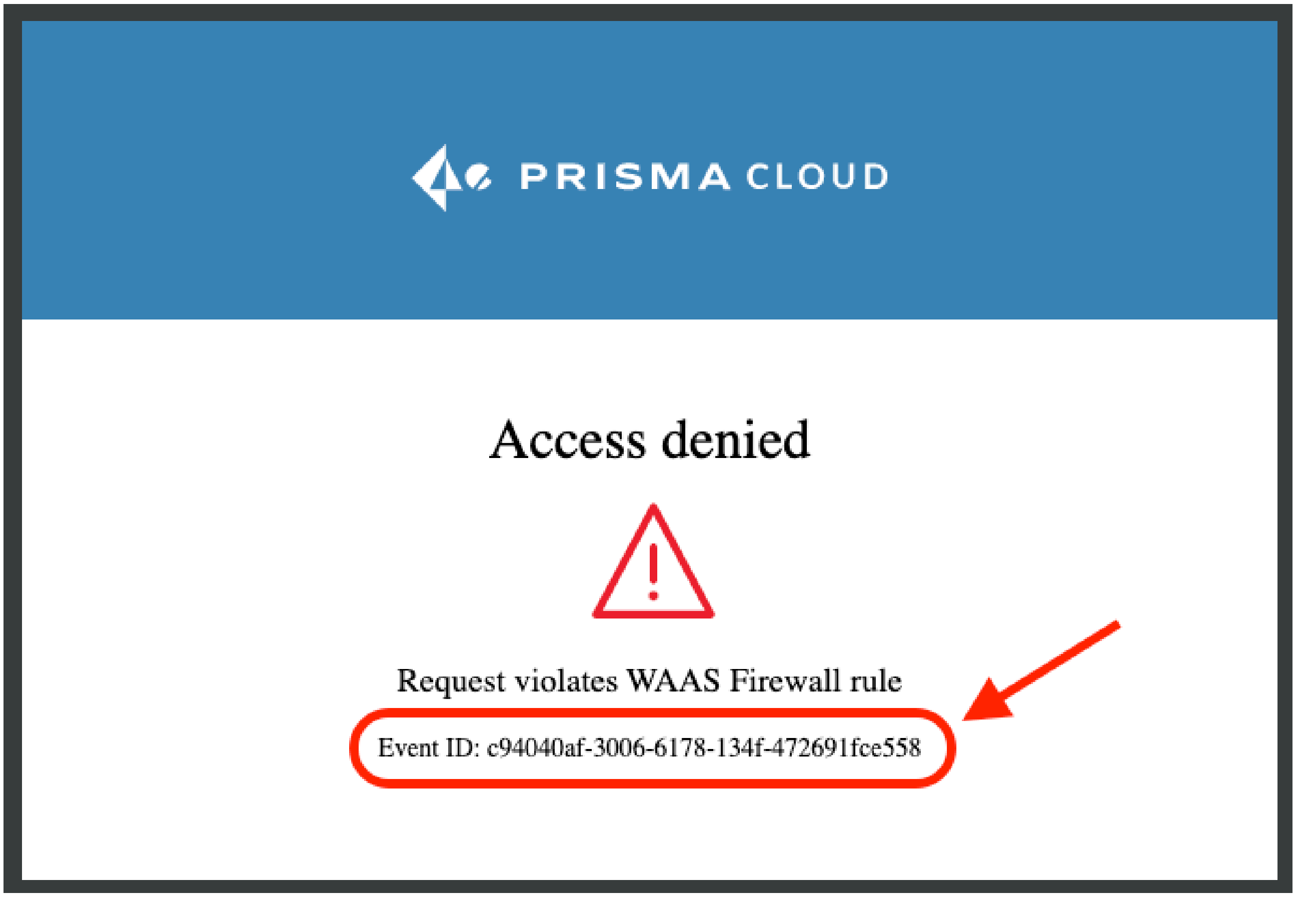
Users can include Prisma Event IDs as part of customized responses by adding the following placeholder in user-provided HTML: *#eventID*.

User-provided HTML must start and end with HTML tags.

Javascript code will not be rended in the preview window.

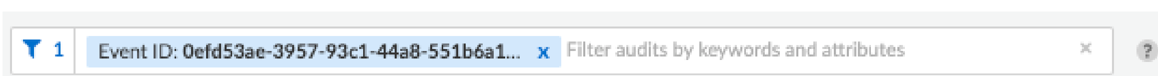
Prisma Event IDs

An event ID is included in the response header **X-Prisma-Event-Id** and is also included in the default WAAS block message:



Users can include Prisma Event IDs as part of customized responses by adding the following placeholder in user-provided HTML: *#eventID*.

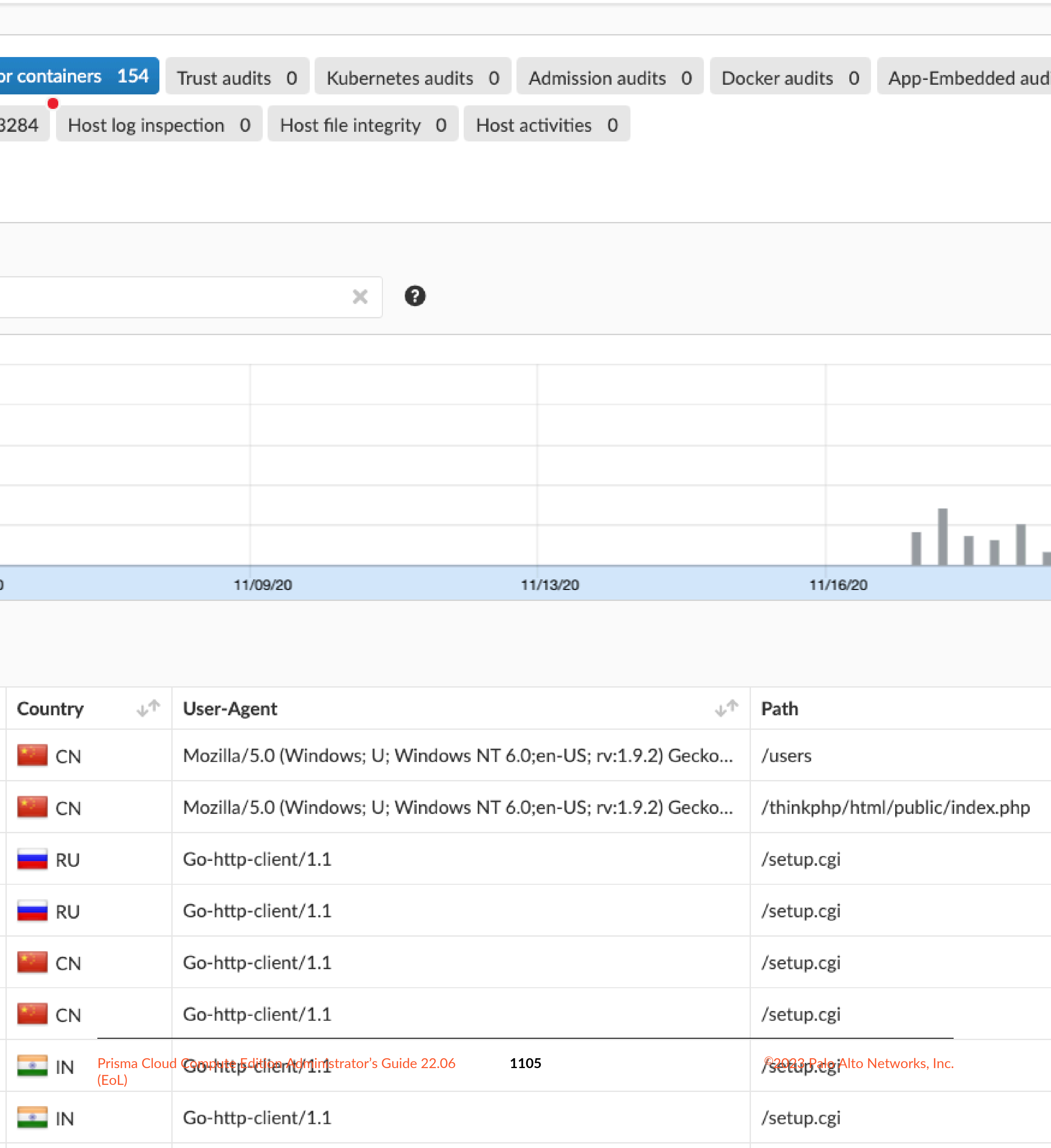
Prisma Event IDs can be referenced in WAAS Event Analytics using the *Event ID* filter:



WAAS Analytics

Edit on GitHub

WAAS analytics provide users a way to investigate events and rule triggers.



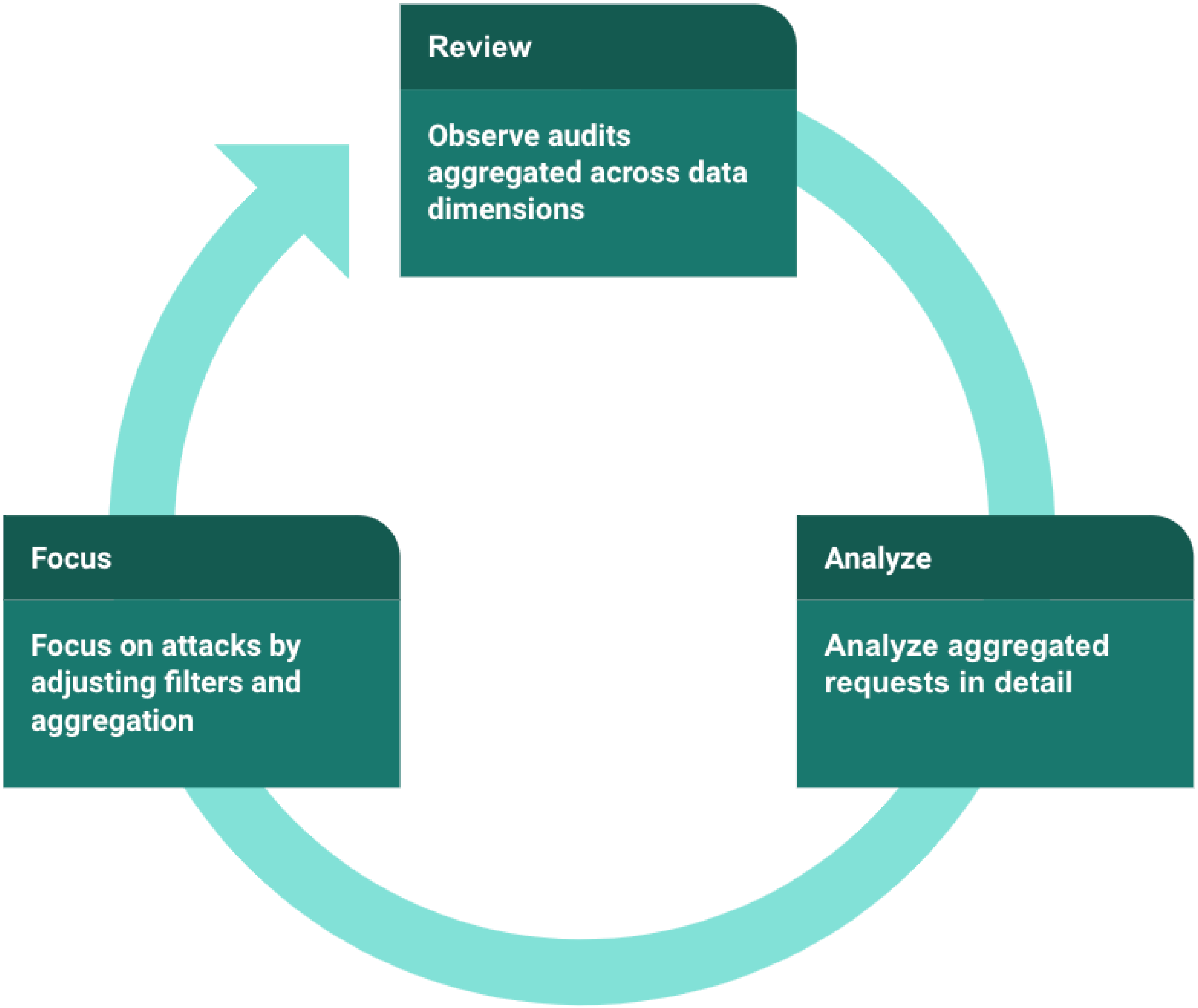
r containers 154
Trust audits 0
Kubernetes audits 0
Admission audits 0
Docker audits 0
App-Embedded aud
3284
Host log inspection 0
Host file integrity 0
Host activities 0
11/09/20
11/13/20
11/16/20
Country
User-Agent
Path
CN
Mozilla/5.0 (Windows; U; Windows NT 6.0;en-US; rv:1.9.2) Gecko...
/users
CN
Mozilla/5.0 (Windows; U; Windows NT 6.0;en-US; rv:1.9.2) Gecko...
/thinkphp/html/public/index.php
RU
Go-http-client/1.1
/setup.cgi
RU
Go-http-client/1.1
/setup.cgi
CN
Go-http-client/1.1
/setup.cgi
CN
Go-http-client/1.1
/setup.cgi
IN
Go-http-client/1.1
/setup.cgi
IN
Go-http-client/1.1
/setup.cgi

- For container WAAS events go to **Monitor > Events > WAAS for containers**
- For host WAAS events go to **Monitor > Events > WAAS for hosts**
- For App-Embedded WAAS events go to **Monitor > Events > WAAS for App-Embedded**
- For serverless WAAS events go to **Monitor > Events > WAAS for Serverless**

WAAS retains up to 200,000 events for each type (container, hosts, app-embedded and serverless) or or a total of 200MB in log size. Once the limit is reached, oldest events will get over-written by new ones.

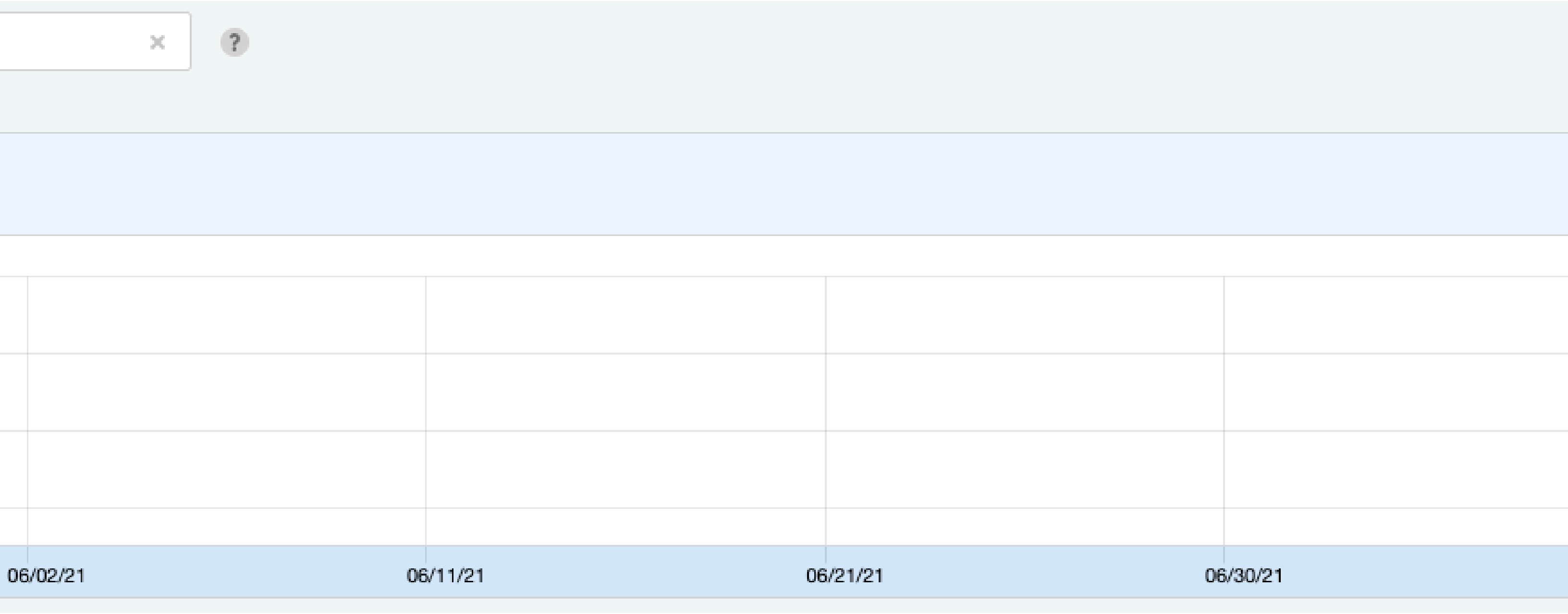
Similar audits are aggregated and grouped into a single event when received in close succession (less than 5 minutes apart). Audits are aggregated by a combination of IP, HTTP hostname, path, HTTP method, User-Agent and attack type.

Analytics workflow



WAAS analytics allows for the review of incidents by analyzing events across various dimensions, inspecting individual requests, and applying filtering to focus on common characteristics or trends.

Event graph

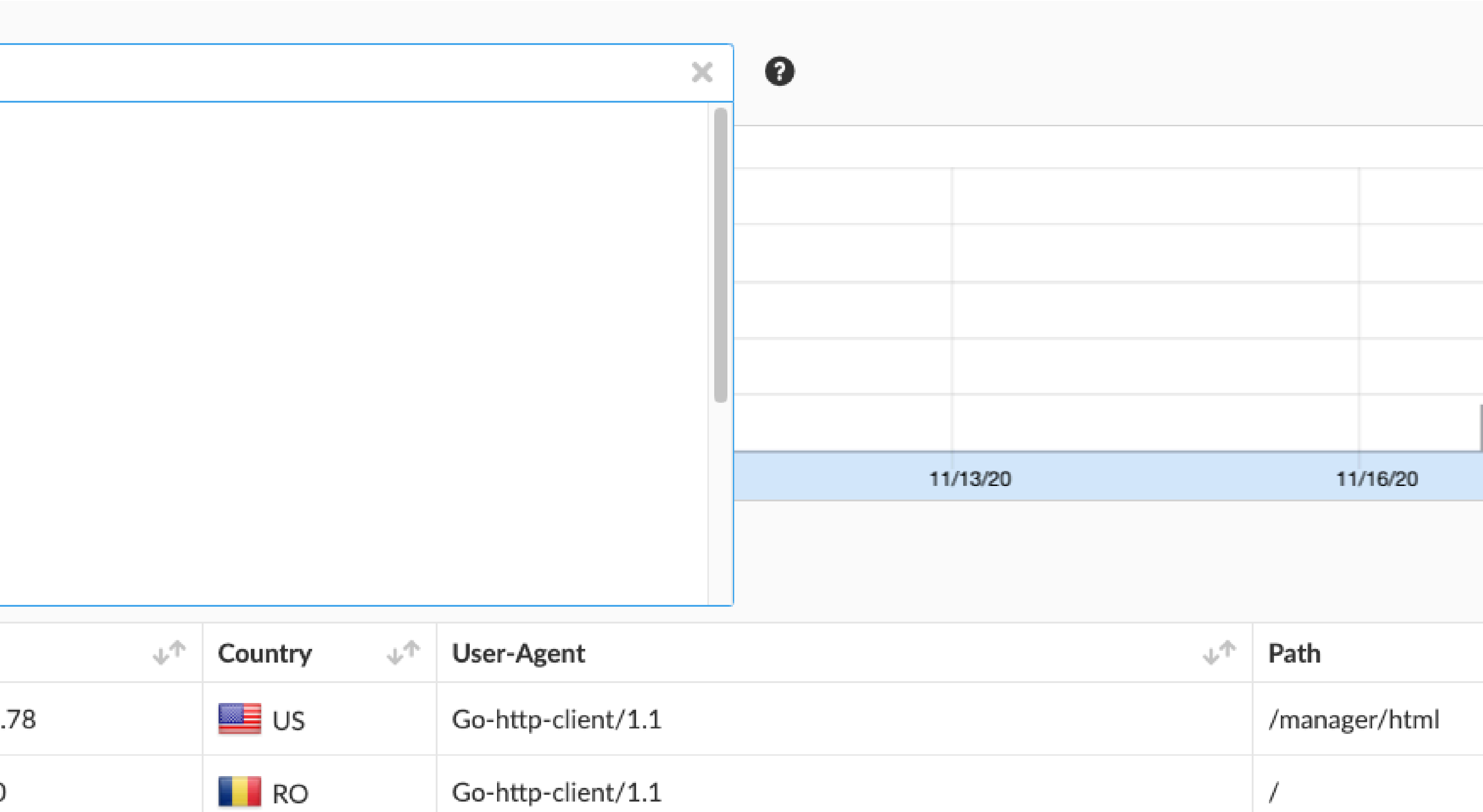


Each column on the timeline graph represents a dynamic period - hover over a column to reveal its start, end and event count.

The date filter can be adjusted by holding and selecting sections on the timeline graph.

Filters

Filter can be adjusted by using the filtering line:



Once set, the filters would apply on the graph and aggregation view.

You can dynamically update the date filter by selecting an area in the chart. Click in the chart area, hold the mouse button down, and draw a rectangle over the time frame of interest. The date filter is automatically updated to reflect your selection.

Aggregation view

	Request headers	Effect
_64; rv:71.0) Gecko/20100101 ...	Accept, Accept-Encoding, Accept-Language, Connection, Content-L...	Prevent
VebKit/535.35 (KHTML, like Ge...	Accept, Accept-Encoding, Accept-Language, Connection, Content-L...	Alert
VebKit/535.35 (KHTML, like Ge...	Accept, Accept-Encoding, Accept-Language, Connection, Content-L...	Alert
64; x64) AppleWebKit/537.36 (...	Accept-Encoding, Connection, Content-Length, Content-Type, User-...	Prevent
leWebKit/537.36 (KHTML, like ...	Accept, Accept-Encoding, Connection, Content-Length, Content-Typ...	Prevent
leWebKit/537.36 (KHTML, like ...	Accept, Accept-Encoding, Connection, Content-Length, Content-Typ...	Prevent
64; x64) AppleWebKit/537.36 (...	Accept-Encoding, Connection, Content-Length, User-Agent	Alert
	Connection	Prevent
64; x64) AppleWebKit/537.36 (...	Accept-Encoding, Connection, User-Agent	Alert
64; x64) AppleWebKit/537.36 (...	Accept-Encoding, Connection, Content-Type, User-Agent	Alert
64; x64) AppleWebKit/537.36 (...	Accept-Encoding, Connection, Content-Length, Content-Type, User-...	Alert
leWebKit/537.36 (KHTML, like ...	Accept, Accept-Encoding, Connection, Content-Length, Content-Typ...	Prevent
	Accept, Accept-Encoding, Connection, Content-Length, Content-Typ...	Prevent
	Accept, Accept-Encoding, Connection, Content-Length, User-Agent	Prevent
	Accept, Accept-Encoding, Connection, Content-Length, Content-Typ...	Prevent
64; x64) AppleWebKit/537.36 (...	Accept, Accept-Encoding, User-Agent	Alert
ertdavidgraham/masscan)	Accept, User-Agent	Alert

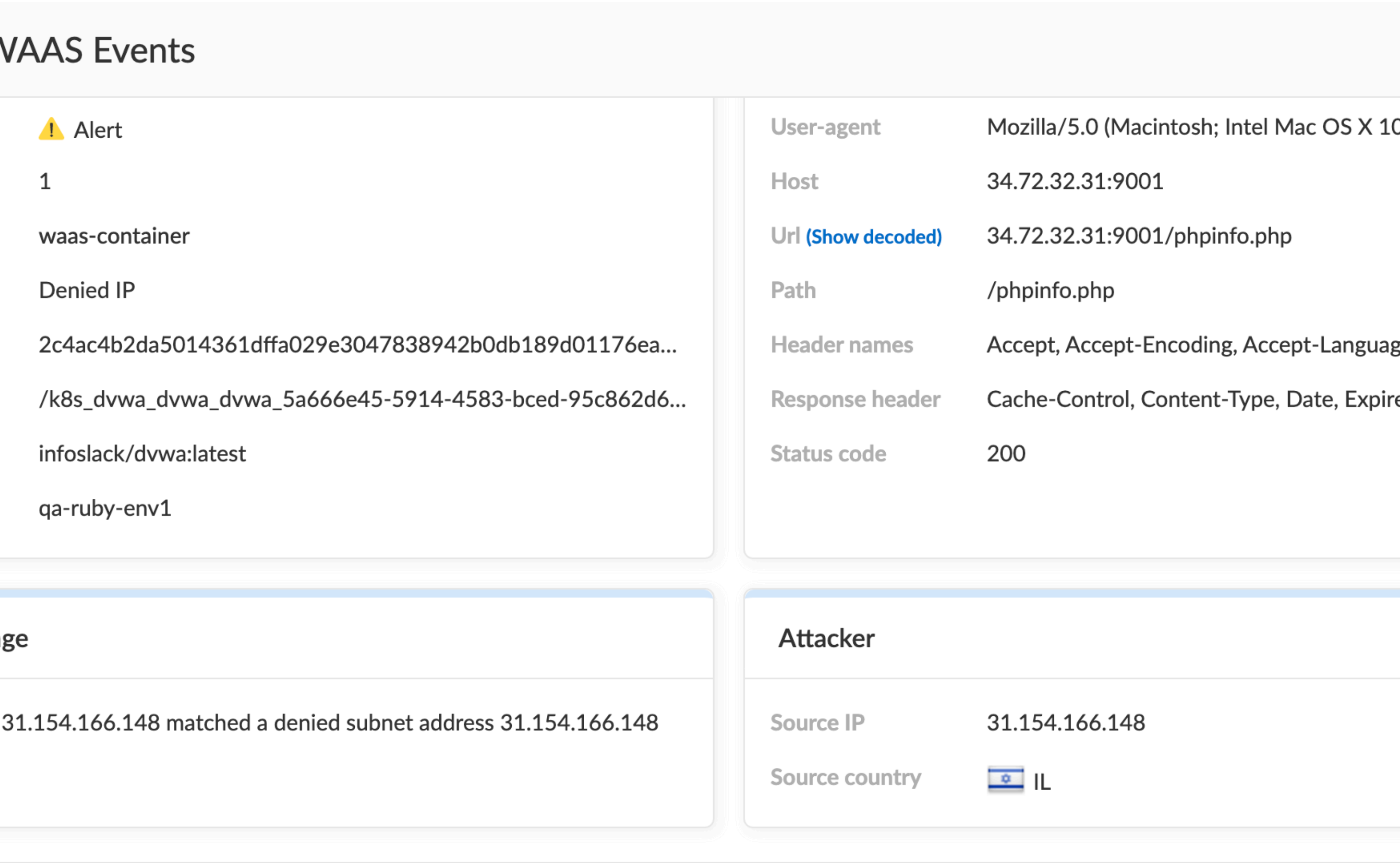
The aggregation view can be altered to group audits based on various data dimensions by clicking on the Group by button.

Users can add up to 6 dimensions to the aggregation and the Total column will be updated dynamically.

By default, aggregation view is sorted by the "Total" column. Sorting can be changed by clicking a column name.

Click on a line in the aggregation view to inspect the requests group by it.

Request view



Request view details all of the requests group by each line of the aggregated view.

Clicking on a column name will sort the table in the upper section and using the Columns button will add/remove columns.

For each request the following data points are available:

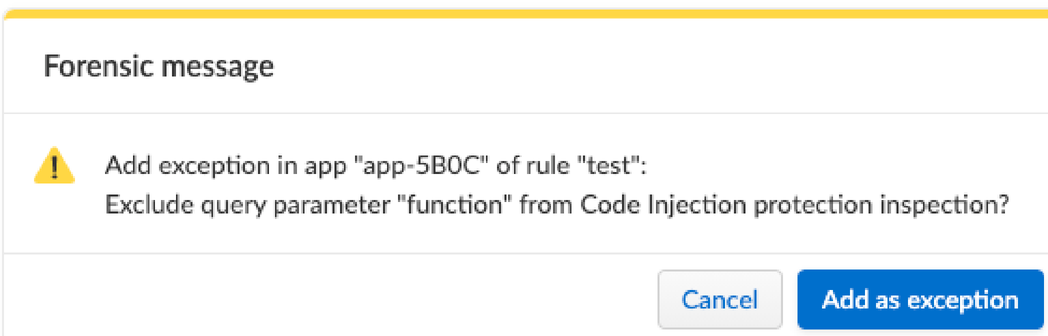
Audit data:

- **Time** - timestamp of the audit.
- **Effect** - effect set by policy.

- **Request Count** - If audits are received in close succession (less than 5 minutes apart) they are aggregated and grouped into one event. This field specifies the number of aggregated requests.
- **Rule Name** - name of the WAAS rule that matched the request and generated the event. Navigate to the configuration of the rule by clicking on the link.
- **Rule app ID** - corresponding app ID in the WAAS rule which triggered the event. Navigate to the configuration of the app ID by clicking on the link.
- **Attack Type** - attack type.
- **ATT&CK technique** - mapping to the techniques in the ATT&CK framework.
- **Container / Host / App / Function Details** - These fields include the id and name of the protected entity.

Forensics:

- **Forensic Message** - details on what caused the rule to trigger - payload content, location and additional relevant information.
- **Add as exception** - By clicking on the link, you can add an exception in the rule app ID for the attack type that triggered. The exception will be based on the location of the matched payload.



The "Add as exception" link may not be available for events created by rules and apps that no longer exist, as well as for events created in releases that predate the Iverson release.

HTTP request:

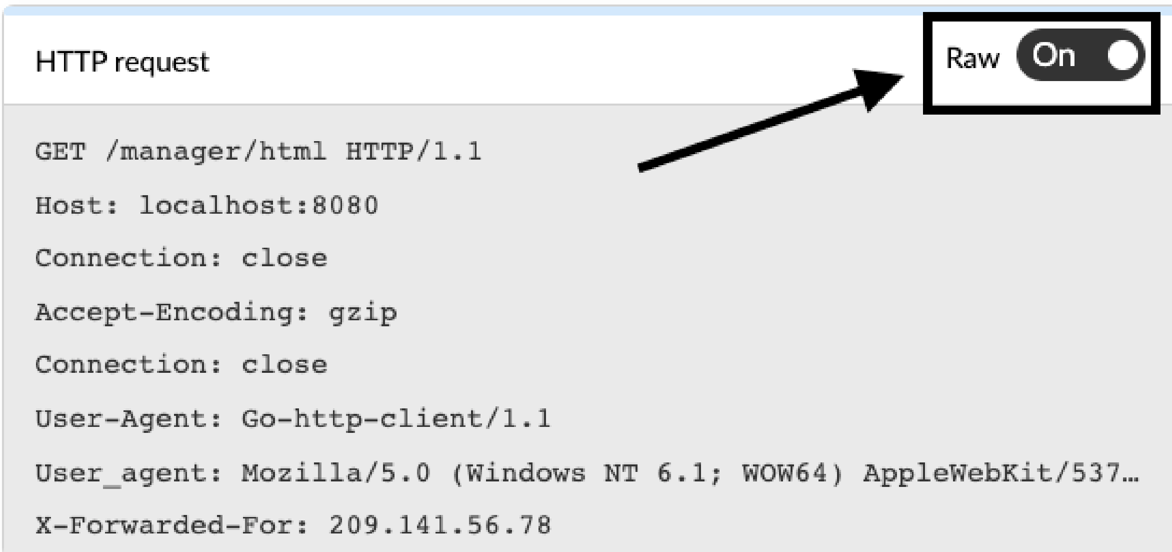
- **Method** - HTTP method used in the request.
- **User-Agent** - value of the User-Agent HTTP header.
- **Host** - hostname specified in the *Host* HTTP header or the host part of the URL.
- **URL** - full request urls (host and path) shown in a URL decoded or encoded form.
- **Path** - path element from the request URI.
- **Query** - query string.
- **Header Names** - list of the HTTP header names included in the request (sorted alphabetically).

Attacker:

- **Add IPs to Network List** - Adds the attacker IP either to a new network list or to an existing one. To access **Network Lists**, open Console, go to **Defend > WAAS** and select the **Network List** tab.

- **Source IP** - IP address from which the request originated. If an *X-Forwarded-For* header was included in the HTTP headers, source IP field will detail the first IP listed in the header value (true client IP).
- **Source Country** - source country associated with the source IP.
- **Connecting IPs** - entire connectivity chain, including true client IP and any transparent proxies listed in the HTTP request.

Users can user the *Raw* button to view the HTTP request in it's raw form:



API observations

Edit on GitHub

WAAS can automatically learn the API endpoints in your app, show an endpoint usage report, and let you export all discovered endpoints as an OpenAPI 3.0 spec file.

Enable API discovery

When API discovery is enabled, Defender inspects API traffic routed to the protected app. Defenders learn the endpoints in your API by analyzing incoming requests and generating a tree of API paths. Every 30 minutes, Defender sends Console a diff of what it has learned since its last update. Console merges the update with what it already knows about the API.

The API discovery subsystem attempts to ignore all HTTP traffic that doesn't resemble an API call. Defender uses some criteria for identifying which requests to inspect:

- Requests must have non-error response codes.
- Requests must not have an extension (.css, .html, etc).
- Requests content type must be textual (*text/*), application (*application/*), or empty.

API discovery is enabled by default. Learning is either on or off. (Compare this to container runtime protection, where there's a learning period, and after the learning period has elapsed, the model of "known good activity" is locked.)

To enable API discovery for a protected app:

STEP 1 | Log in to the Console, and go to **Defend > WAAS > {Container | Host | Out-of-band }**.

STEP 2 | Click **Add rule**.

STEP 3 | Select **API endpoint discovery**.

STEP 4 | **Save** the rule.

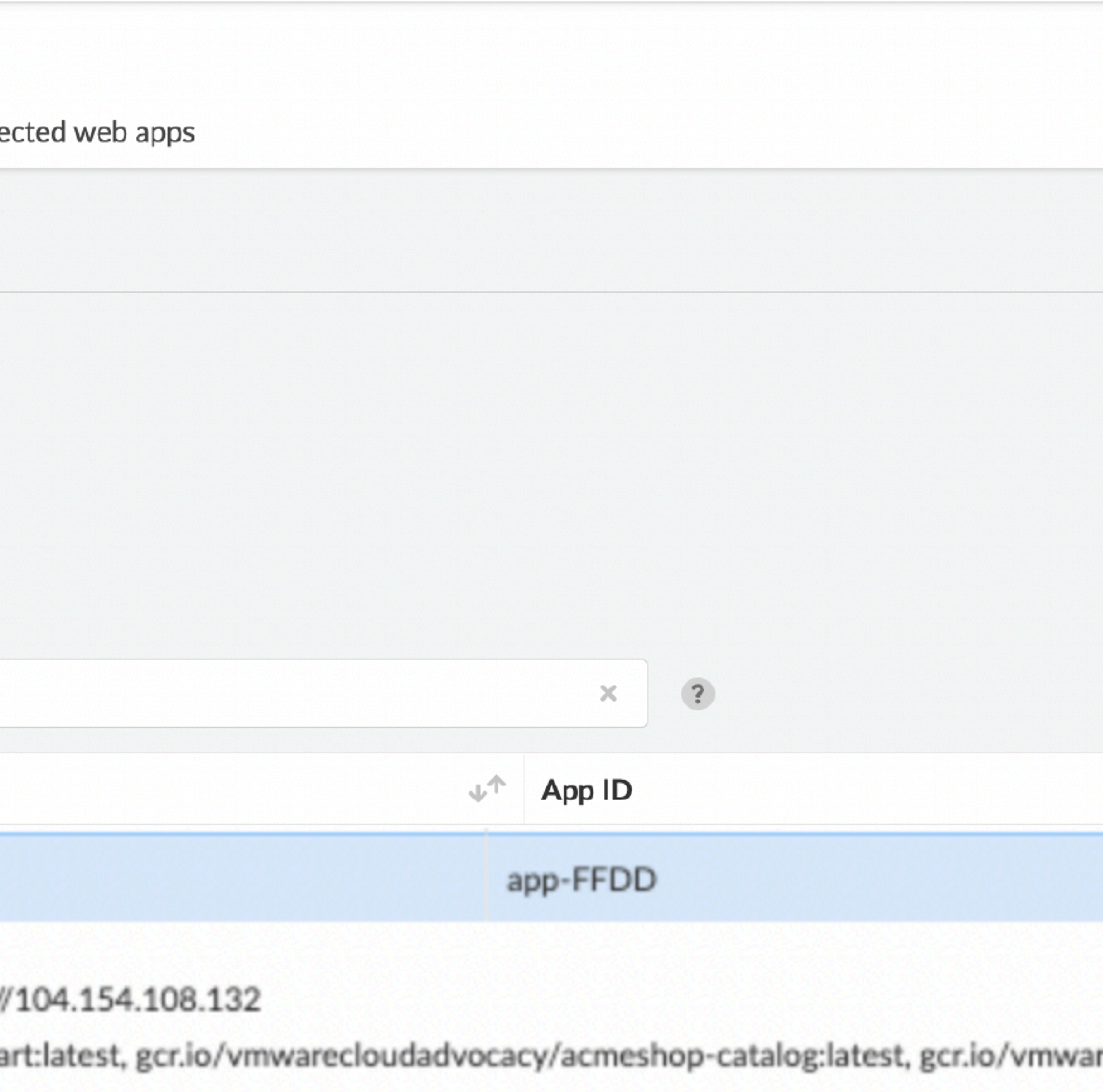
*If you have not enabled **API endpoint discovery** while creating a rule, you can also enable it after the rule creation.*

Inspect discovered endpoints

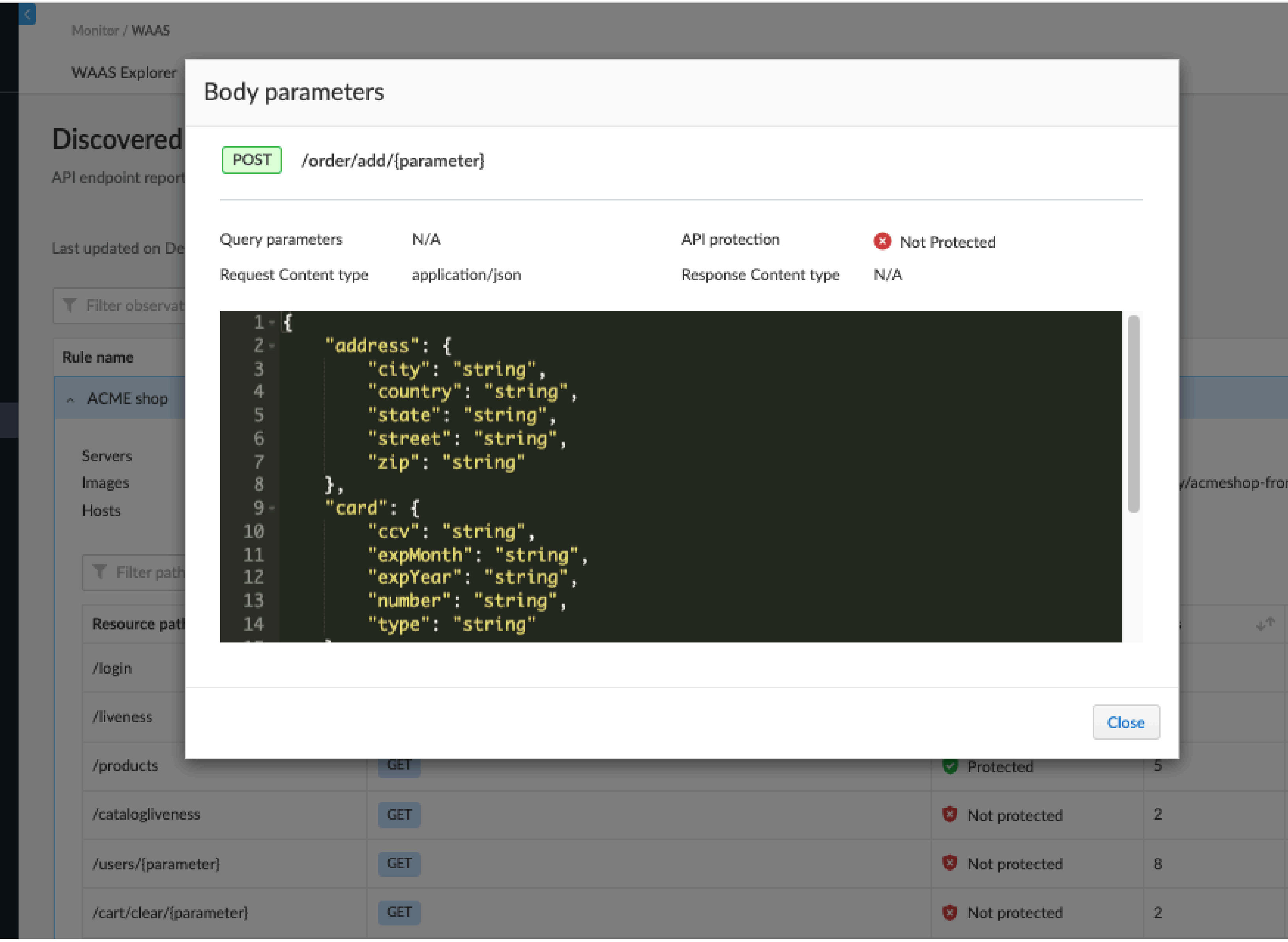
The endpoint report enumerates discovered APIs on a per-app basis. It shows information such as HTTP method, path (including path parameters), request content-type, query parameters, message JSON structure, hit count, and last seen date. To view the report, go to **Monitor** > **WAAS** > **API Observations** > **Discovered endpoints**.

To view the report for Out-of-band WAAS rules, go to **Monitor** > **WAAS** > **API Observations** > **Out-of-band observations**.

Select **Refresh** to force Console to poll Defenders for the latest data, analyze it, and present the results in the table.


ected web apps
App ID
app-FFDD
/104.154.108.132
art:latest, gcr.io/vmwarecloudadvocacy/acmeshop-catalog:latest, gcr.io/vmwar

Clicking on a line will present the recoded message body (currently, only JSON learning is supported)



You can export the discovered endpoints for an app as an OpenAPI spec file. Alternatively, you can select the 3 dots … and Delete, to delete everything that WAAS has learned about the API for an app so far.

If a rule with an app is deleted from the WAAS policy, its learned endpoints are also deleted.

API definition scan

Edit on GitHub

Prisma Cloud scans the API definition files and generates a report for any errors, or shortcomings such as structural issues, compromised security, best practices, and so on. API definition scan supports scanning OpenAPI 2.X and 3.X definition files in either YAML or JSON formats.

You can use the following methods to scan an API definition file:

- Upload API definition file to Console
- Run twistcli, a CLI tool aimed for CI/CD. Twistcli scans the API definition file and returns a full report with issues.
- Import an OpenAPI definition file into a WAAS app: When you import an OpenAPI definition file into a WAAS app, the Console automatically scans for issues. You can view the full report of the scan by navigating to **Monitor** > **WAAS** > **API definition scan**.

twistcli reference for scanning API definition files

Run the following command:

```
$ ./twistcli waas openapi-scan </path/to/file/example.yaml>
```

Syntax:

```
twistcli waas openapi-scan [command options] [arguments...]
```

OPTIONS:

- address value: Prisma Cloud Console URL. This is the value twistcli uses to connect to Console (required) (default: "https://127.0.0.1:8083")
- exit-on-error: Immediately exits scan if an error is encountered (not supported with --containerized)
- password value, -p value: Password for authenticating with Prisma Cloud Console. For Prisma Cloud Enterprise Edition, specify the secret key associated with the access key ID passed to --user [$TWISTLOCK_PASSWORD]
- project value: Target project
- tlscacert value: Path to Prisma Cloud CA certificate file
- token value: Token for authenticating with Prisma Cloud Console
- user value, -u value: User for authenticating with Prisma Cloud Console. For Prisma Cloud Enterprise Edition, specify an access key ID (default: "admin") [$TWISTLOCK_USER]

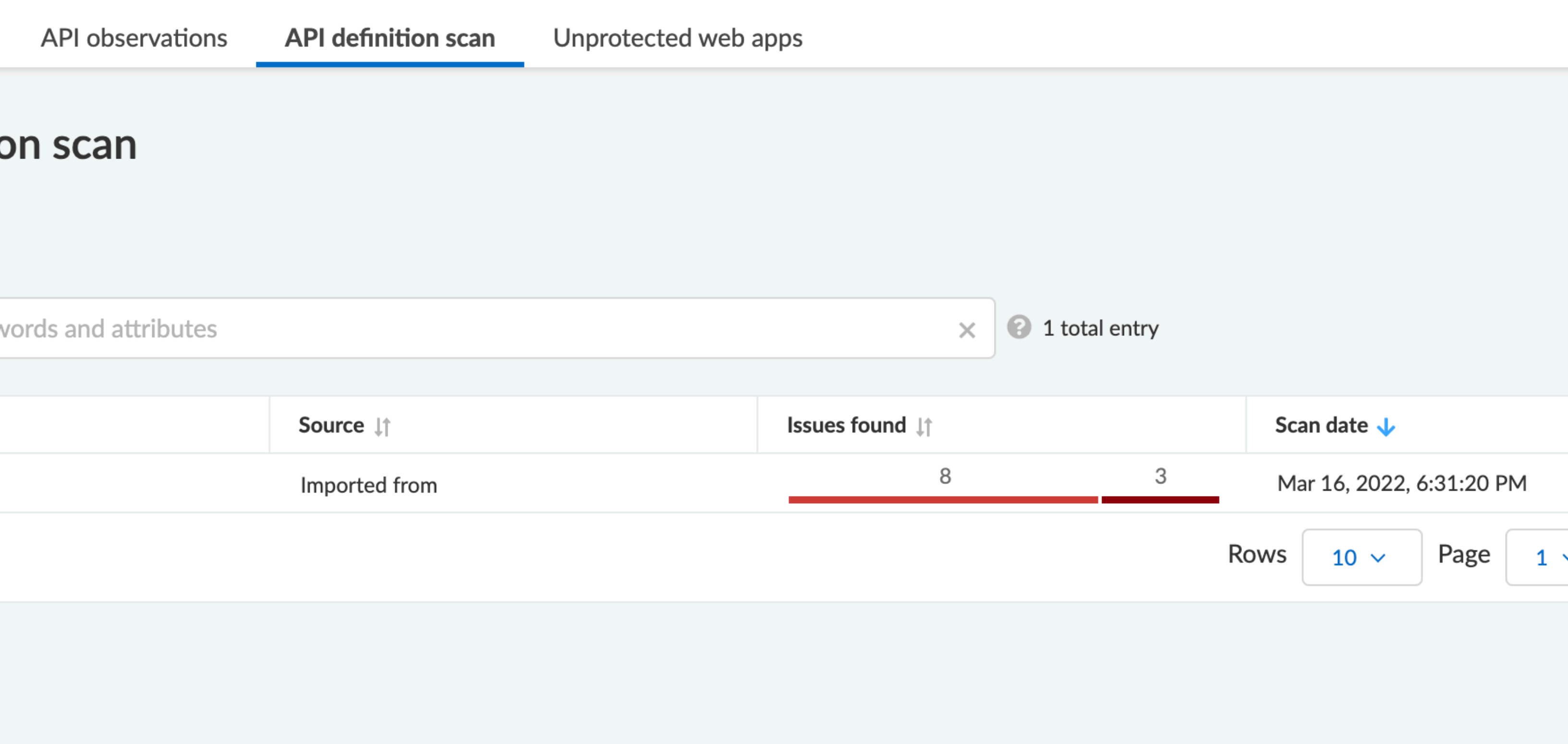
Upload API definition file

To import an API definition file, follow the steps below:

STEP 1 | Open the Console, and go to **Monitor > WAAS > API definition scan**.

STEP 2 | **Upload** an API definition scan file.

The following screenshot shows the API definition scan files:



You can also filter the API definition files by using the scan date, import source, or file name.

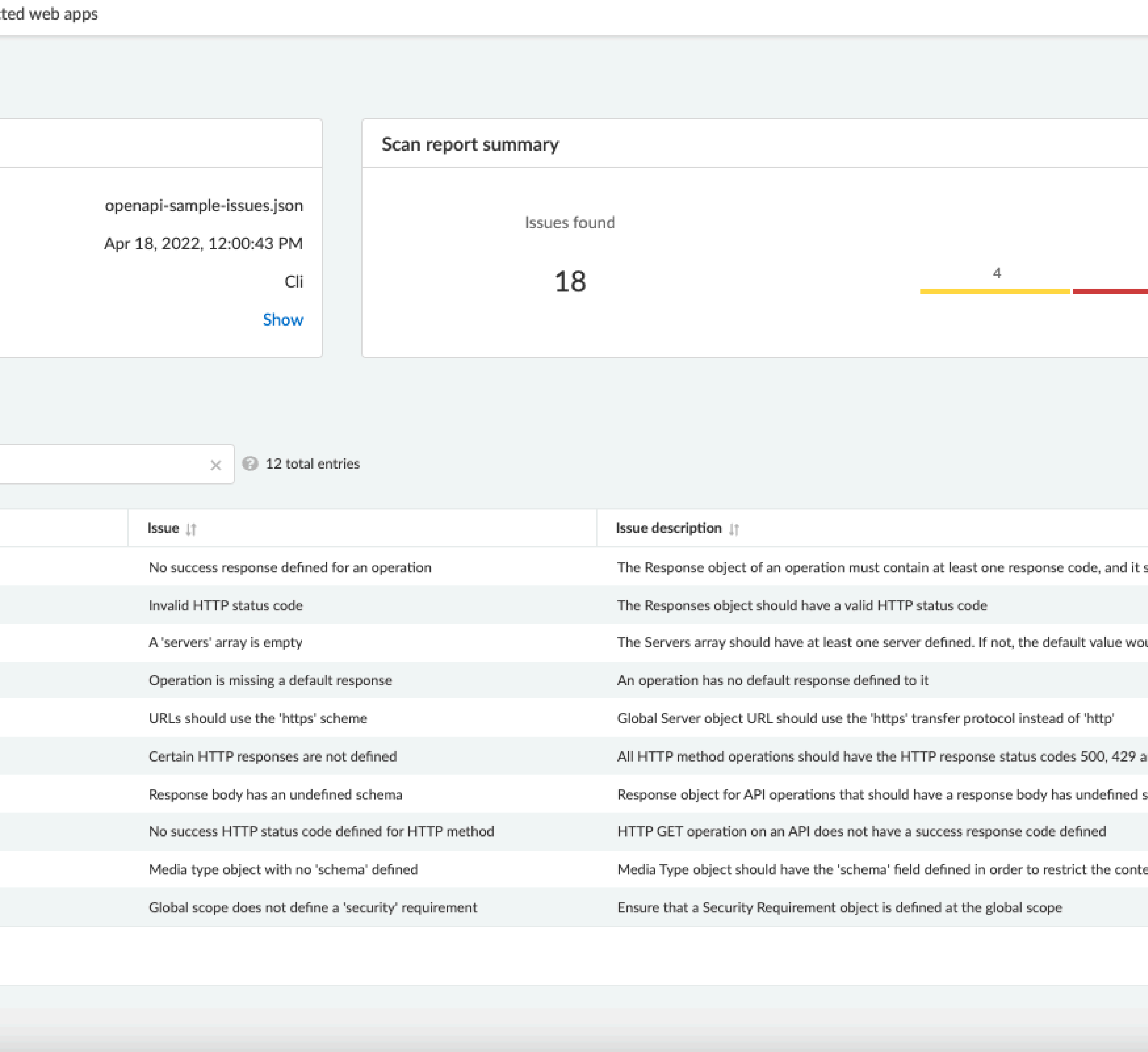
View API definition scan report details

STEP 1 | Open the Console, and go to go to **Monitor > WAAS > API definition scan**.

API definition scan reports are available along with the description of the file source such as twistcli scan, upload to the console, or WAAS app (where the file was imported).

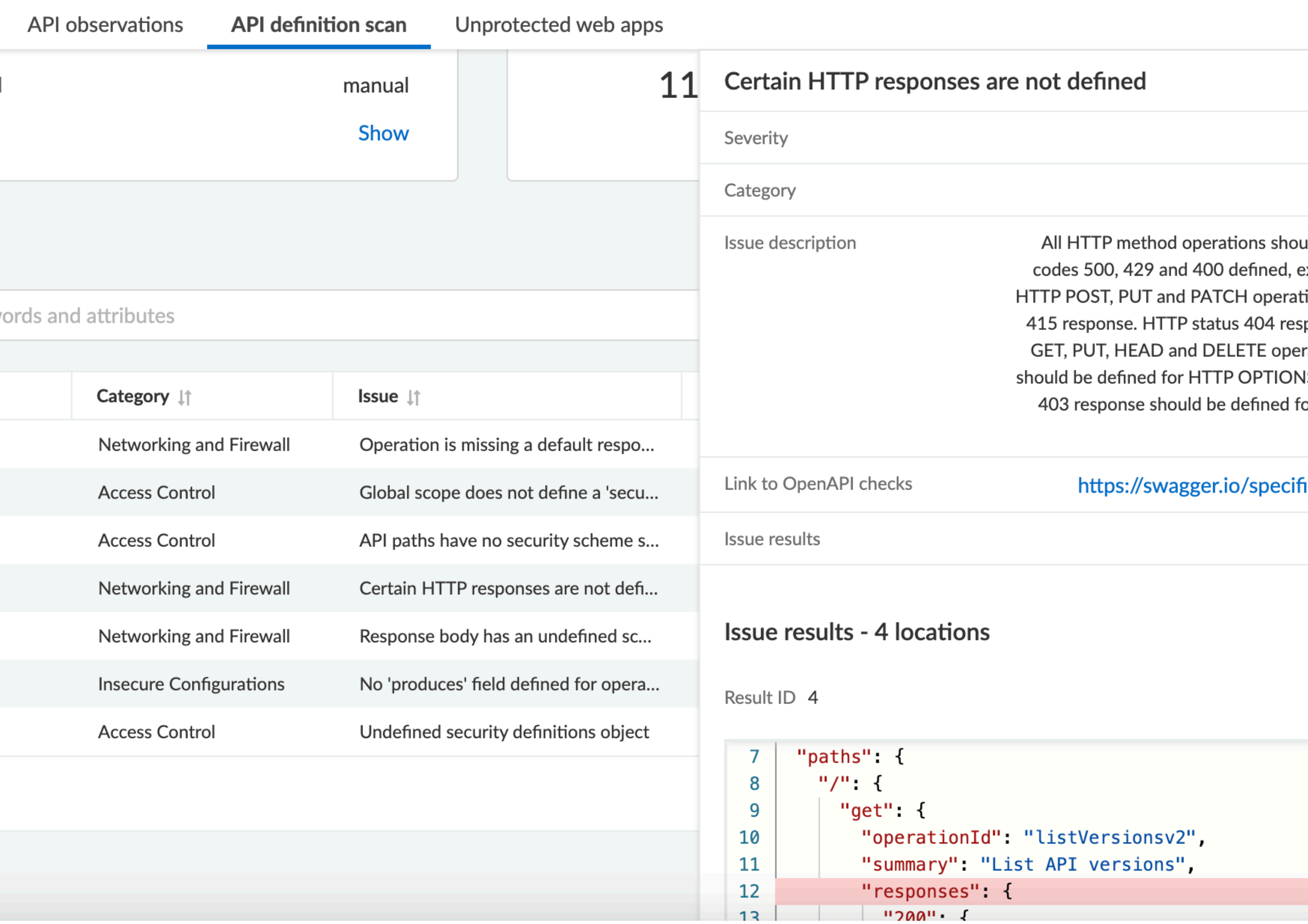
STEP 2 | In the **Actions** column, click **View**.

The following screenshot shows the severity of issues and their related categories:



STEP 3 | To view detailed information such as reference to the file, issue link, and so on for a specific issue, click on an issue under the **Findings** column.

The following screenshot shows a preview of various locations and details in the Openapi spec file for a selected issue:



WAAS custom rules

Edit on GitHub

WAAS custom rules offer an additional mechanism to protect your running web apps. Custom rules are expressions that give you a precise way to describe and detect discrete conditions in requests and responses. WAAS intercepts layer 7 traffic, passes it to Prisma Cloud for evaluation. Expressions let you inspect various facets of requests and responses in a programmatic way, then take action when they evaluate to true. Custom rules can be used in container, host, and app-embedded WAAS policies.

Besides your own custom rules, Prisma Labs ships and maintains rules for newly discovered threats. These systems rules are distributed via the Intelligence Stream. By default, they are shipped in a disabled state. You can review, and optionally activate them at any time. System rules cannot be modified. However, you can clone and customize them to fit your own specific needs.

Before using custom rules, ensure Console and Defender run the same version of Prisma Cloud Compute. The minimum required version for a Defender appears when you add a custom rule to a policy. For example, if a Console runs a newer version, but Defenders have not been upgraded, using functionality only available in the newer version will result in a WAAS error. If this occurs, upgrade Defenders to match their Console's version.

Expression grammar

Expressions let you examine the contents of requests and responses. The grammar lets you inspect various properties in an event. For example, you could write an expression that determines if an IP address fall inside a specific CIDR block.

Expressions support the following types:

- String.
- String list.
- String map.
- Integer.
- IP address (e.g. "192.168.0.1")
- CIDR block (e.g. "192.168.0.0/16")

Expressions have the following grammar:

*term (op term | in |)**

- **term** --

  integer | string | keyword | event | '(' expression ')' | unaryOp

- **op** --

  and | or | > | < | >= | # | = | !=

- **in** --

  '(' integer | string (',' integer | string))?*

  Can also be used to determine if an IP address is in a CIDR block: For example:

  req.ip in "192.168.0.0/16"
- **unaryOp** --

  not
- **keyword (similar to wildcards)** --

  startswith | contains

  contains can be used for:

  - Equality. For example: *req.header_names contains "X-Forwarded-For"*
  - Regular expression match for string lists. For example: *req.header_names contains /^X-Forwarded.*/*
  - Regular expression match for strings. For example: *req.body contains /^some-regex-text.*/*
- **string** --

  Strings must be enclosed in double quotes.
- **integer** --

  int
- **event** --

  req | resp
- **[ ]** --

  Selector operator. Selects a specific value by key from a map. Headers, cookies, body params, and query params are maps. The selection operation template is as following:

  req.<map>["<key>"]

  For example:

  req.headers["Content-Type"] contains "text/html"

Request events

Expressions can examine the following attributes of a request:

Attribute	Minimum Defender version	Type	Example
req.host	22.06	Map of String	*req.host contains /^.*ACME[1-9]{1,6}$/*
	21.04	Map of String	*req.headers["User-Agent"] contains /^.*ACME[1-9]{1,6}$/*

Attribute	Minimum Defender version	Type	Example
(for matching on "Host" header use req.host)			
req.header_names	21.04	String List	*req.header_names contains /^X-Forwarded.*/*
req.header_values	21.04	String List	*req.header_values contains "secretkey"*
req.cookies	21.04	Map of String	*req.cookies["yummy-cookie"] contains "flour"*
req.cookie_names	21.04	String List	*req.cookie_names contains "ga"*
req.cookie_values	21.04	String List	*req.cookie_values contains "admin"*
req.query_params	21.04	Map of String	*req.query_params["id"] contains "admin"*
req.query_param_names	21.04	String List	*req.query_param_names contains "ssn"*
req.query_param_values	21.04	String List	*req.query_param_values contains /\d{3}-?\d{2}-?\d{4}/*
req.body_param_values	21.04	String List	*req.body_param_values contains "username"*
req.http_method	21.04	String	*req.http_method = "POST"*
req.file_extension	21.04	String	*req.file_extension contains /pdf$/*
req.path	21.04	String	*req.path startswith "/admin/"*
req.ip	21.04	(written as string, parsed as IP if IP is valid)	*req.ip in "2.2.2.0/24" or req.ip = "8.8.8.8"*
req.country_code	21.04	String	*req.country_code = "US"*
req.body	21.04	String	*req.body contains /password/*
req.http_version	21.04	String	*req.http_version = "1.0"*

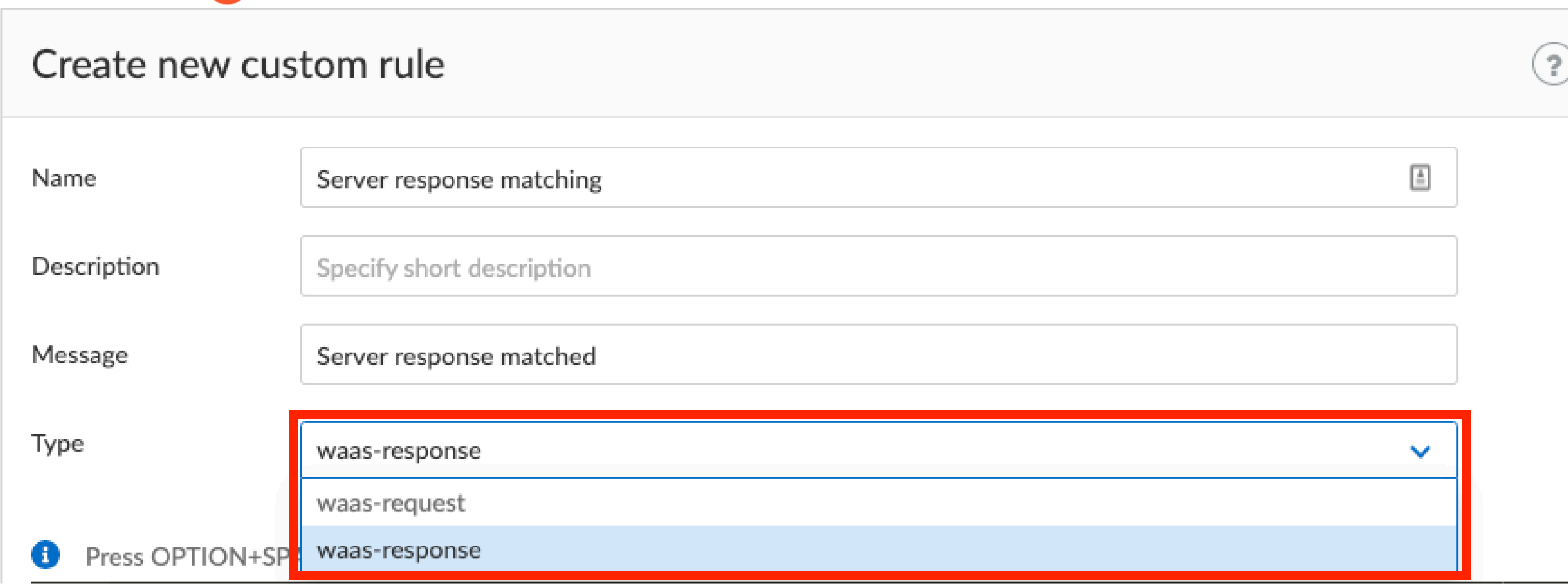
Attribute	Minimum Defender version	Type	Example
req.http_scheme	21.04	String	*req.http_scheme = "HTTPS"*

When gRPC is enabled, the req.body attribute may not be able to properly match on the body content if it is sent in binary form.

Response events

Expressions can examine the following attributes of a response.

To examine server responses in custom rules, the rule type must be set to waas-response



Attribute	Minimum Defender version	Type	Example
resp.status_code	21.04	Integer	*resp.status_code = 200*
resp.content_type	21.08	String	*resp.content_type = "application/ json"*
resp.body	21.08	String	*resp.body contains /^somesecret$/*
resp.headers	21.08	Map of String	*resp.headers["Set-Cookie"] contains /SESSIONID/*
resp.header_names	21.08	String List	*resp.header_names contains "Set-Cookie"*

Attribute	Minimum Defender version	Type	Example
resp.header_values	21.08	String List	*resp.header_values contains "ERROR"*

When gRPC is enabled, the resp.body attribute may not be able to properly match on the body content if it is sent in binary form.

Trasformation functions

The following transformations are available to users creating custom rules:

- **lowercase** - converts all charactes to lowercase.
- **compressWhitespace** - converts whitespace characters (32, \f, \t, \n, \r, \v, 160) to spaces (32) and then compresses multiple space characters into only one.
- **removeWhitespace** - removes all whitespace characters.
- **urlQueryDecode** - decodes URL query string.
- **urlPathDecode** - decodes URL path string (identical to **urlQueryDecode** except that it does not unescape + to space).
- **unicodeDecode** - normalizes unicode characters to their closest resemblance in ASCII format.
- **htmlEntityDecode** - decodes HTML components in a given string.
- **base64Decode** - decoes a base64-encoded string.
- **replaceComments** - replaces each occurence of a C-style comments (/* ... */) with a single space (multiple consecutive occurences of a space will not be compressed).
- **removeCommentSymbols** - removes each comment symbol (/*, */) from a string.
- **removeTags** - replaces encoded tag entities (*<*, *>*) with a single whitespace.

Effects

The following effects may be applied on HTTP requests/responses that match a WAAS custom rule:

- **Allow** - The request is passed to the protected application, all other detections are not applied (e.g app firewall, bot protection, API protection, etc.). No audit is generated.
- **Alert** - The request is passed to the protected application and an audit is generated for visibility.
- **Prevent** - The request is denied from reaching the protected application, an audit is generated and WAAS responds with an HTML page indicating the request was blocked.
- **Ban** - Can be applied on either IP or Prisma Session IDs. All requests originating from the same IP/Prisma Session to the protected application are denied for the configured time period (default is 5 minutes) following the last detected attack.

A message at the top of the page indicates the entity by which the ban will be applied (IP or Prisma Session ID). When the X-Forwarded-For HTTP header is included in the request headers, ban will apply based on the first IP listed in the header value (true client IP).

*For custom rules defined in **Out-of-band**, only **Allow** and **Alert** effects are allowed.*

Example expressions

The following examples show how to use the expression grammar:

Special expression to determine if an IP address falls within a CIDR block:

req.ip in "192.168.0.0/16"

Example of using a regular expression:

req.header_names contains /^X-Forwarded./*

Determine if the request method matches a method in the array. Currently, you can only create custom arrays as part of the *in* operator.

req.http_method in ("POST", "PUT")

Example of using *contains*:

req.header_values contains "text/html"

Example using a selector:

req.cookies["yummy-cookie"] contains "flour"

Example of an expression with three conditions. All conditions must evaluate to true for there to be a match.

req.http_method = "POST" and resp.status_code >= 400 and resp.status_code # 599

Example for detecting HTTP 1.0 requests sent to a path starting with /api/control/ with an "admin" cookie whose Base64 decoded value is set to "True".

req.http_version = "1.0" and lowercase(req.path) startswith "/api/control/" and base64Decode(req.cookies["admin"]) contains /^True$/`

Example for detecting successful login requests by checking the Set-Cookie header value using chained tranformation functions.

req.http_method = "POST" and resp.status_code = 200 and compressWhitespace(base64Decode(resp.headers["Set-Cookie"])) contains /SESSIONID/`

Write a WAAS custom rule

Expression syntax is validated when you save a custom rule.

STEP 1 | Open Console, and go to **Defend > WAAS** > **{Container | Host | App-Embedded | Out-of-band}**.

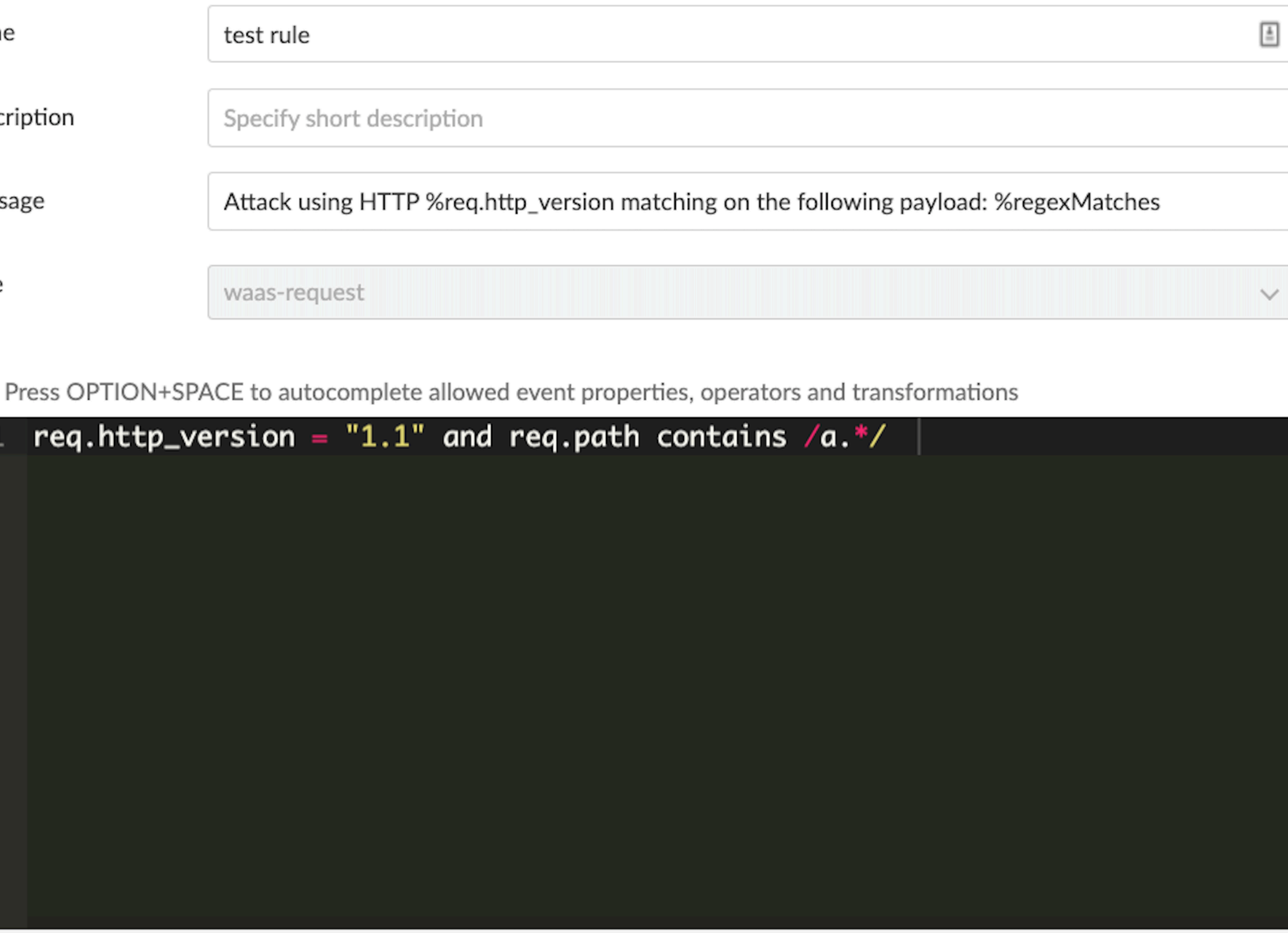
STEP 2 | Click **Add rule**.

STEP 3 | Enter a name for the rule.

STEP 4 | In **Message**, enter a audit message to be emitted when an event matches the condition logic in this custom rule.

Use the following syntax to view the matched groups: <Your text>: %regexMatches

Refer to the following screenshot:



STEP 5 | Select the rule type.

You can write expressions for requests or responses. What you select here scopes the vocabulary available for your expression.

STEP 6 | Enter your expression logic.

Press *OPTION + SPACE* to get a list of valid terms, expressions, operators, etc, for the given position.

Use the example expressions here as a starting point for your own expression.

STEP 7 | Click **Save**.

Your expression logic is validate before it's save to Console's database.

Activate WAAS custom rules

A custom rule is made up of one or more conditions. Attach a custom rule to a WAAS policy rule to activate it.

Custom rules are defined in **Defend > Custom rules > WAAS**. WAAS policy rules are defined in **Defend > WAAS > {Container | Host | App-Embedded | Out-of-band}**.

When attaching a custom rule to a WAAS policy rule, you specify the action to take when the expression evaluates to true (i.e. the expression matches). Supported actions are disable, alert, prevent, and ban.

Custom rules have priority over all other enabled WAAS protections. WAAS evaluates all custom rules that are attached, so you can get more than one audit if more than one custom rule matches.

Prerequisites: You have already set up WAAS to protect an app, and there's a rule for it under **Defend > WAAS > {Container | Host | App-Embedded | Out-of-band}**. For more information about setting up an app, see Deploy WAAS.

STEP 1 | Open Console, and go to **Defend > WAAS > {Container | Host | App-Embedded | Out-of-band}**.

STEP 2 | In the table, expand a rule.

STEP 3 | Under **App list**, click **Actions > Edit** for an app in the table.

STEP 4 | In the edit rule dialog, click the **Custom rules** tab.

STEP 5 | Click **Select rules**.

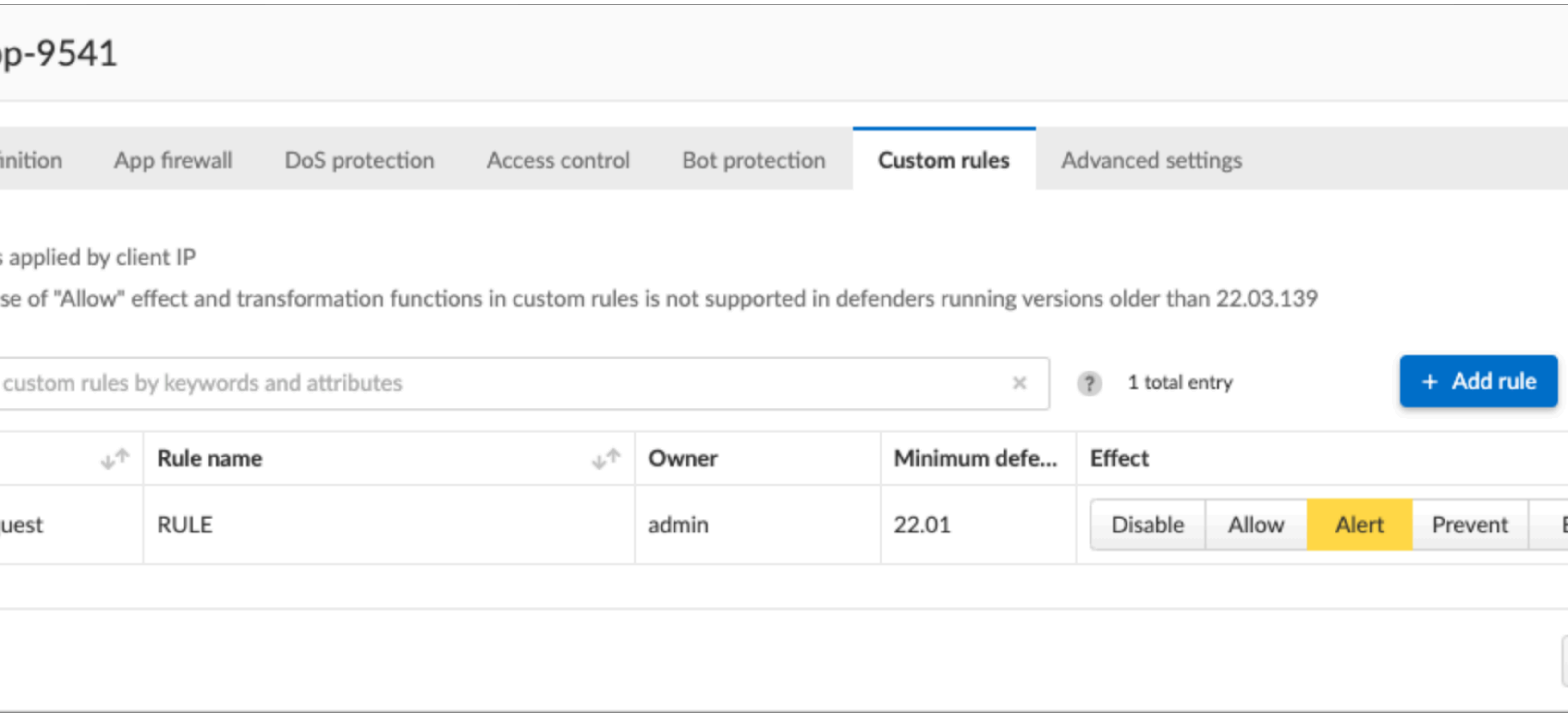
A list of available WAAS custom rules is displayed. Whenever a user creates a rule, the **owner** column is populated with the username. The owner column of virtual patches provided by Unit-42 researchers will have the value *system*.

Alternatively, you can click **Add rule** to author a new custom rule in place.

STEP 6 | Select one or more rules.

STEP 7 | Click **Apply**.

The minimum supported Defender version appears when you add the custom rule to a policy.



STEP 8 | Configure the effect for each custom rule.

By default, the effect is set to **Alert**.

STEP 9 | Click **Save**.

Detecting unprotected web apps

Edit on GitHub

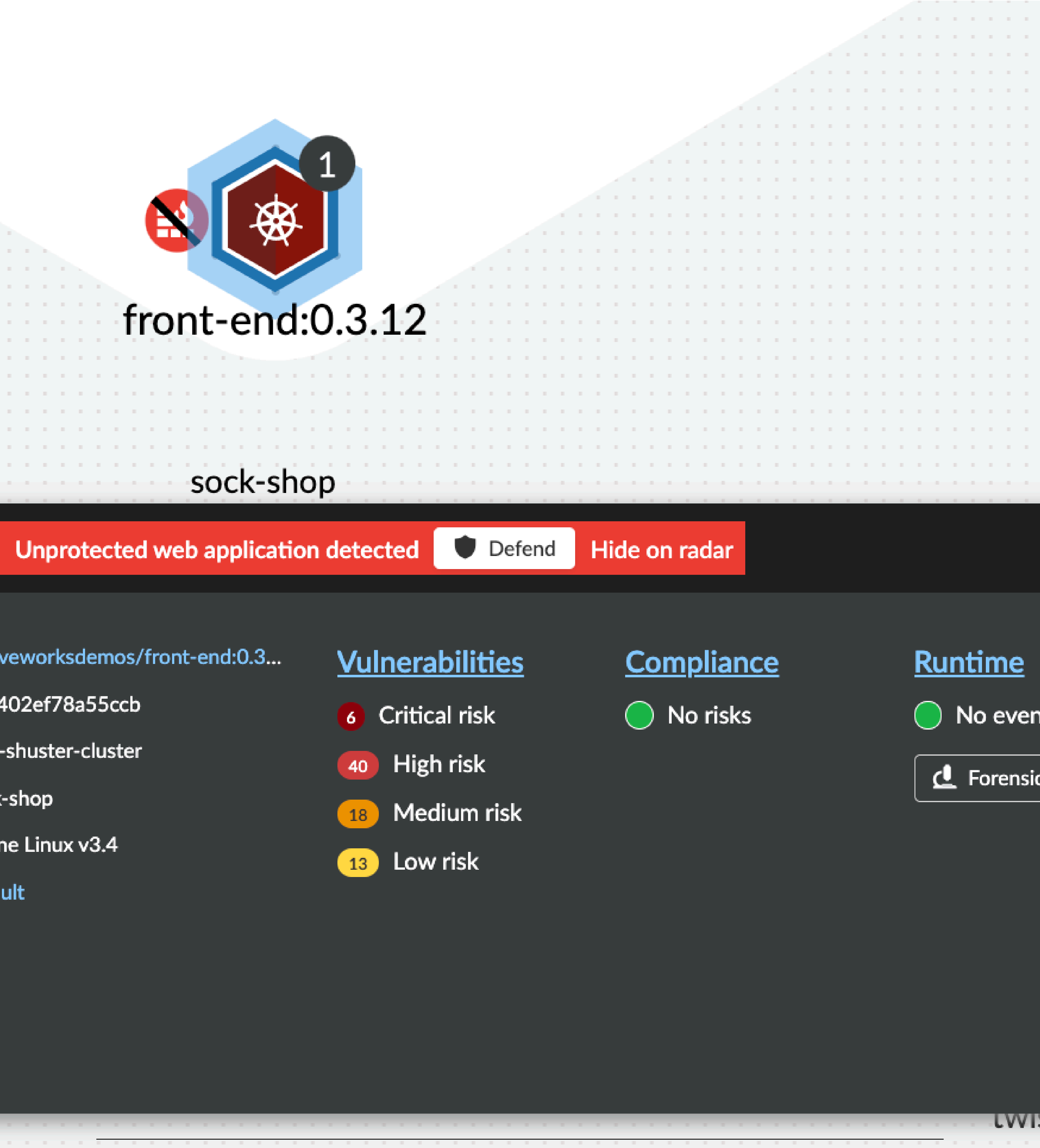
Prisma Cloud scans your environment for containers and hosts that run web apps, and reports any that aren't protected by WAAS.

During the scan, Prisma Cloud detects HTTP servers listening on exposed ports and flags them if they are not protected by WAAS.

Unprotected web apps are flagged on the radar view and are also listed in **Monitor > WAAS > Unprotected web apps**.

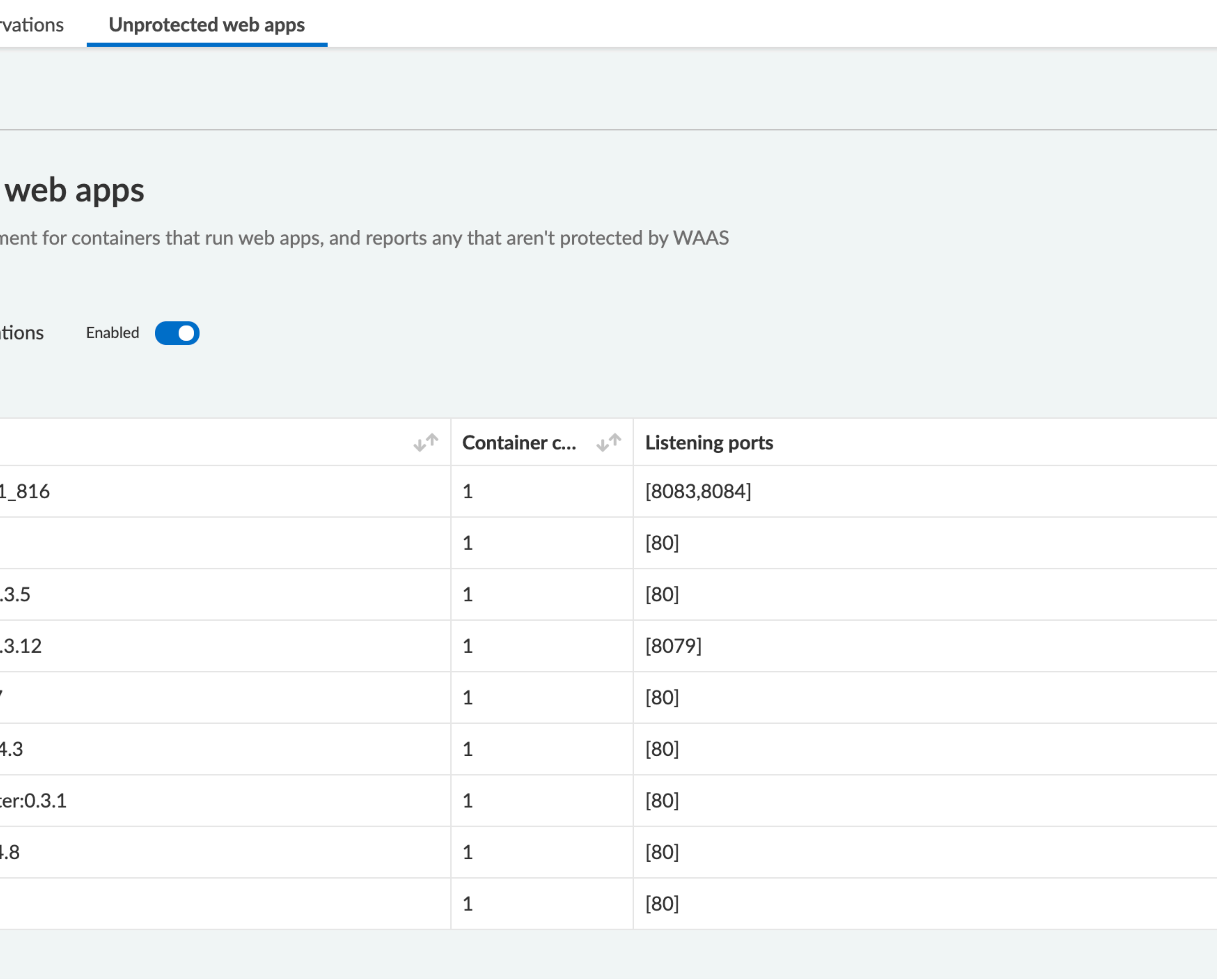
Unprotected web apps scan is available for containers

The following screenshot shows how Radar shows an unprotected web app:


1
front-end:0.3.12
sock-shop
Unprotected web application detected
Defend
Hide on radar
veworksdemos/front-end:0.3...
402ef78a55ccb
-shuster-cluster
-shop
ne Linux v3.4
ult
Vulnerabilities
6 Critical risk
40 High risk
18 Medium risk
13 Low risk
Compliance
No risks
Runtime
No even
Forensic

Report for unprotected web apps

The following screenshot shows how unprotected web apps are reported in **Monitor > WAAS > Unprotected web apps**:



In the *Containers* tab, the report lists the images containing unprotected web apps, the number of containers running those images, and the ports exposed in the running containers.

In the *Hosts* tab, the report lists the hosts on which unprotected web apps are running, the number of processes running those apps, process names and the ports exposed in the hosts.

This information can be used when adding new WAAS rules to protect containers and hosts.

Above the table is the date of the latest scan. The report can be refreshed by clicking the refresh button.

Users can export the list in CSV format. The CSV file has the following fields:

- **Containers** - Image, Host, Container, ID, Listening ports
- **Hosts** - ID, Unprotected processes

Disabling scans for unprotected web apps

By setting the *Scan for unprotected web applications* toggle to the **Disabled** position, users are able to disable periodic scanning for unprotected web applications and APIs.

The toggle in either the `Containers or `Hosts tabs will disable scanning of containers and hosts simultaneously when disabled.

WAAS Log Scrubbing

Edit on GitHub

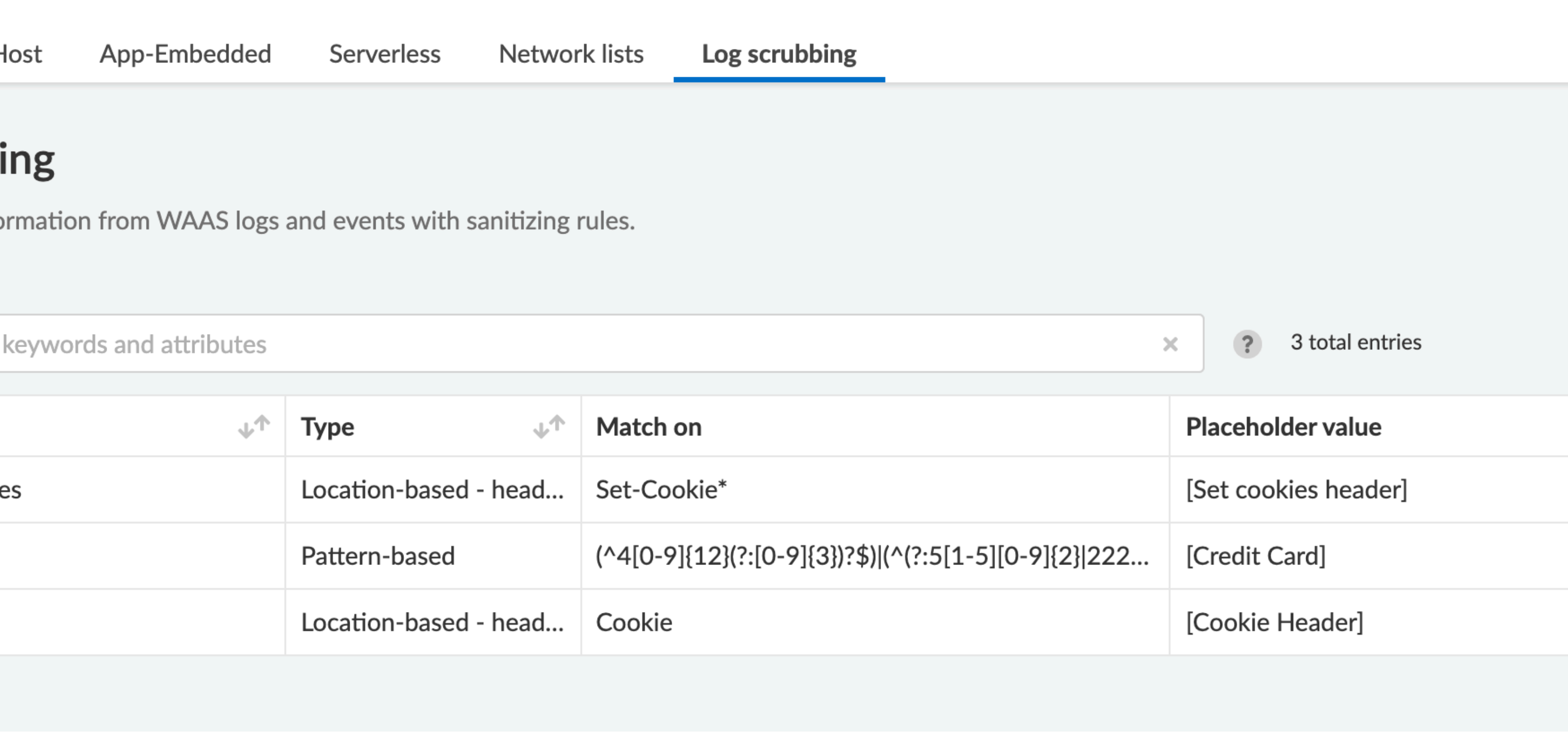
There may be sensitive data captured when WAAS events take place, such as access tokens, session cookies, PII or other information considered to be personal by various laws and regulations.

By using WAAS log scrubbing rules, users can mark data as sensitive based on regex patterns or its location in the HTTP request. This data is removed from the logs before events are recorded, and is replaced with placeholders entered by the user.

Add/Edit WAAS Scrubbing Rule

To create or edit log scrubbing rules, follow the steps below:

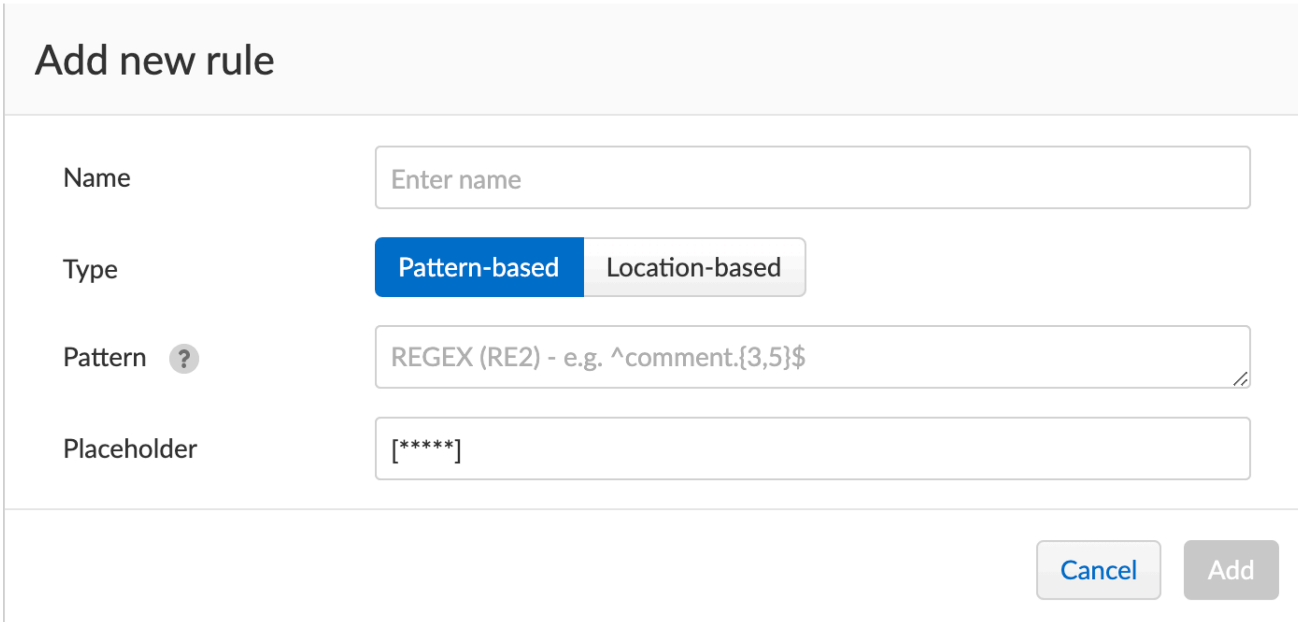
1. Open the Console, and go to **Defend > WAAS > Log Scrubbing**



2. Click on **Add rule** or select an existing rule.
3. Enter Rule Name.

4. Select rule type: pattern-based or location-based.

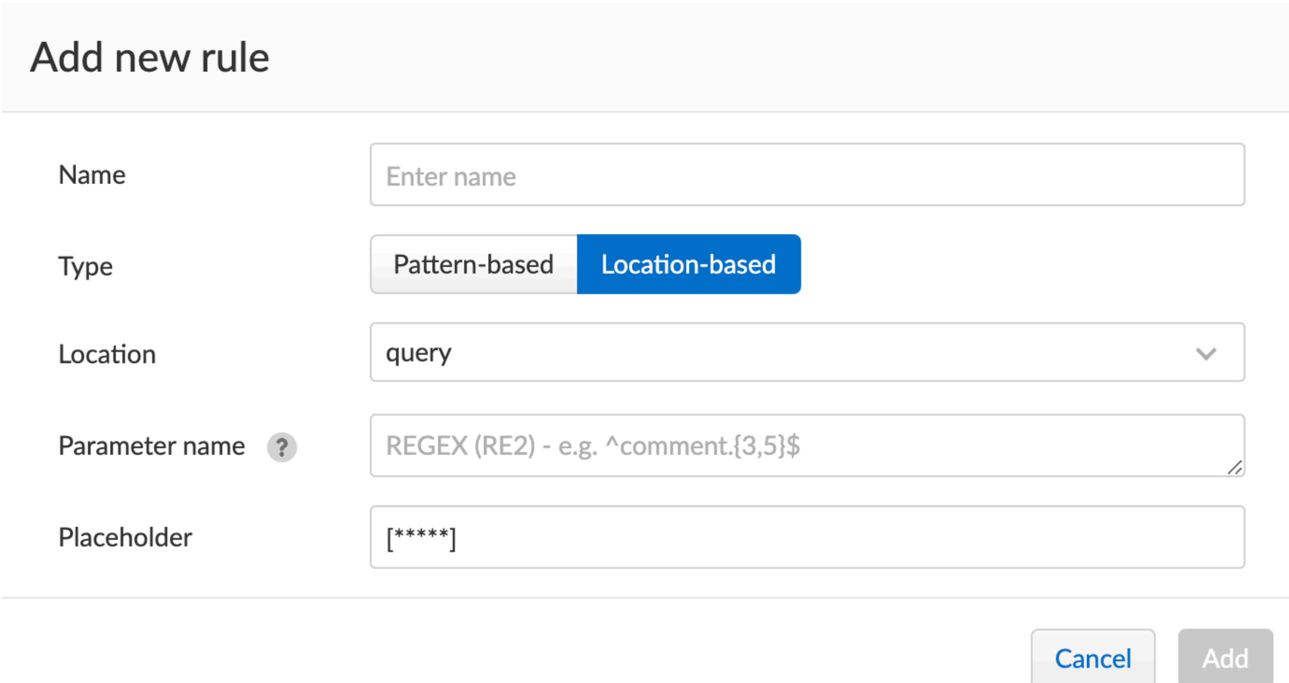
5. For pattern-based rules:



1. Provide match pattern in the form of a regular expression (re2), e.g. *^sessionID$*, *key-[a-zA-Z]{8,16}*.
2. Provide a placeholder string e.g. *[scrubbed sessionID]*.

Placeholder strings indicating the nature of the scrubbed data should be used as users will not be able to see the underlying scrubbed data.

6. For location-based rules

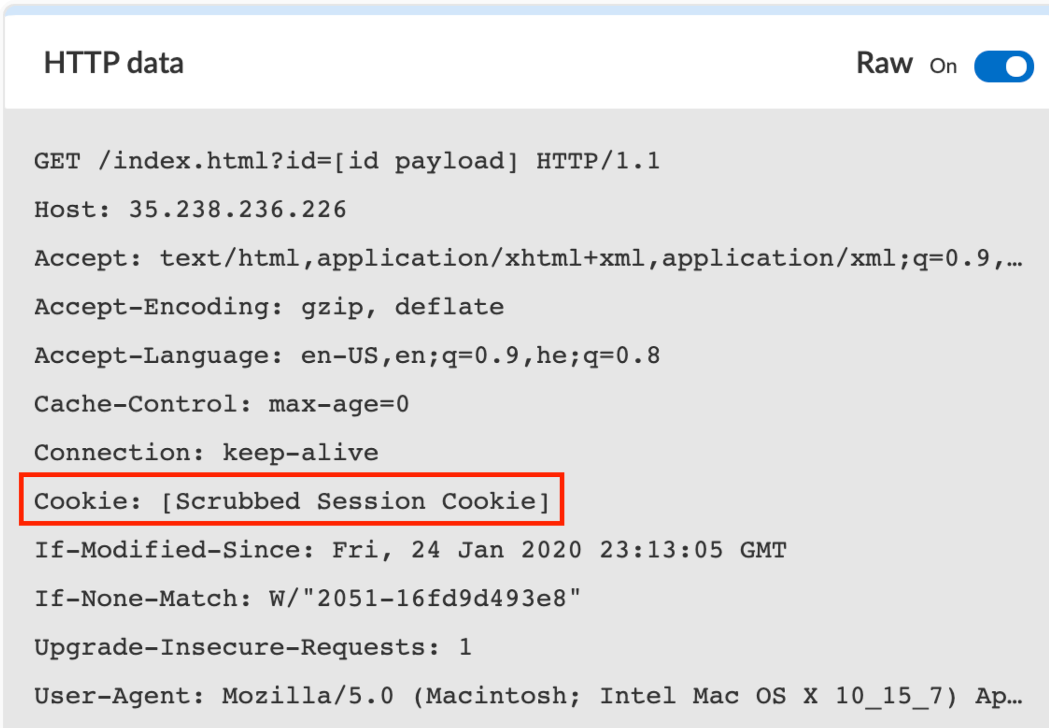


1. Select the location of the data to be scrubbed.
2. Provide location details:
    1. For *query* / *cookie* / *header* / *form/multipart* - provide match pattern in the form of a regular expression (re2), e.g. *^SCookie.*$*, *item-[a-zA-Z]{8,16}*.
    2. For *XML (body)* / *JSON (body)* - provide the path using Prisma Cloud's custom format e.g. */root/nested/id*.
3. Provide a placeholder string e.g. *[Scrubbed Session Cookie]*.

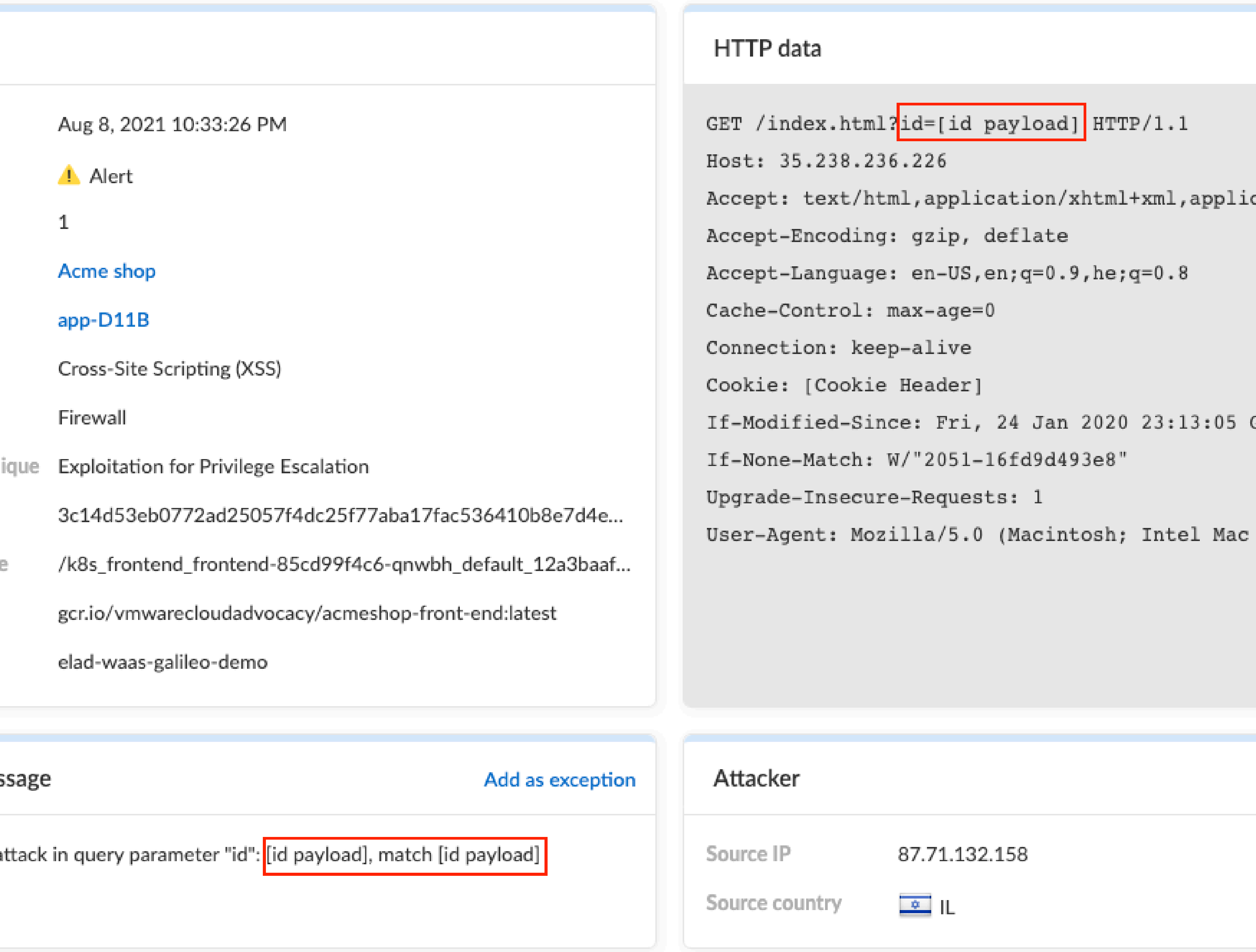
Placeholder strings indicating the nature of the scrubbed data should be used as users will not be able to see the underlying scrubbed data.

7. Click **Save**.

   Data will now be scrubbed from any WAAS event before it is written (either to the Defender log or syslog) and sent to the console:



If sensitive data triggers events, both the forensic message and the recorded HTTP request are scrubbed. Hence, placeholder strings indicating the nature of the scrubbed data should be used as users will not be able to see the underlying scrubbed data.



Firewalls

Edit on GitHub

Prisma Cloud provides layer 4 monitoring and enforcement, and layer 7 firewalling. For layer 7 firewalling, see WAAS.

- Cloud Native Network Firewall (CNNF)

Cloud Native Network Firewall (CNNF)

Edit on GitHub

Cloud Native Network Firewall (CNNF) is a Layer 4 container-aware virtual firewall and network monitoring tool. Network segmentation and compartmentalization is an important part of a comprehensive defense in depth strategy. CNNF works as an east-west firewall for containers and hosts. It limits damage by preventing attackers from moving laterally through your environment when they've already compromised your perimeter.

Container environments present security challenges that aren't suitably addressed by traditional tools. In a container environment, network traffic between nodes is usually encapsulated and encrypted in an overlay network. The IP addresses of the endpoints are ephemeral and largely irrelevant, so rules such as *from 192.168.1.100 to 192.168.1.200, allow tcp/27017* aren't useful because you usually don't know, or even care, about containers' IP addresses.

CNNF automatically discovers how entities in your environment communicate, and shows the connection patterns on Radar. Radar has a container view, which shows the network topology for your containerized apps. Radar also has a host view, which shows the network topology for hosts.

Using the connection patterns discovered and displayed on Radar, you can create rules that enforce how entities communicate with each other.

In addition to network topology, Radar has a data overlay that shows vulnerability, compliance, and runtime state for each node. Use the combined data to better asssess where risk should be mitigated.

Key capabilities

Coupled with Radar, CNNF lets you conceptualize connectivity, segment traffic, and compartmentalize attacks.

- CNNF lets you visualize the network topology of your containerized apps and your hosts. Radar paints nodes and edges on a 2D canvas to show how entities communicate with each other.
- CNNF lets you segment your microservices at the container level. CNNF also lets you segment your hosts. Create rules that specify how entities are allowed to talk to each other.
- CNNF lets you monitor the impact of your microsegmentation policy. Radar draws normal and abnormal traffic patterns on the canvas. Attempted connections that break from policy are highlighted on Radar and audits for these types of events are emitted.
- CNNF policy is portable. You can export all the rules from one Console and import them into another Console.

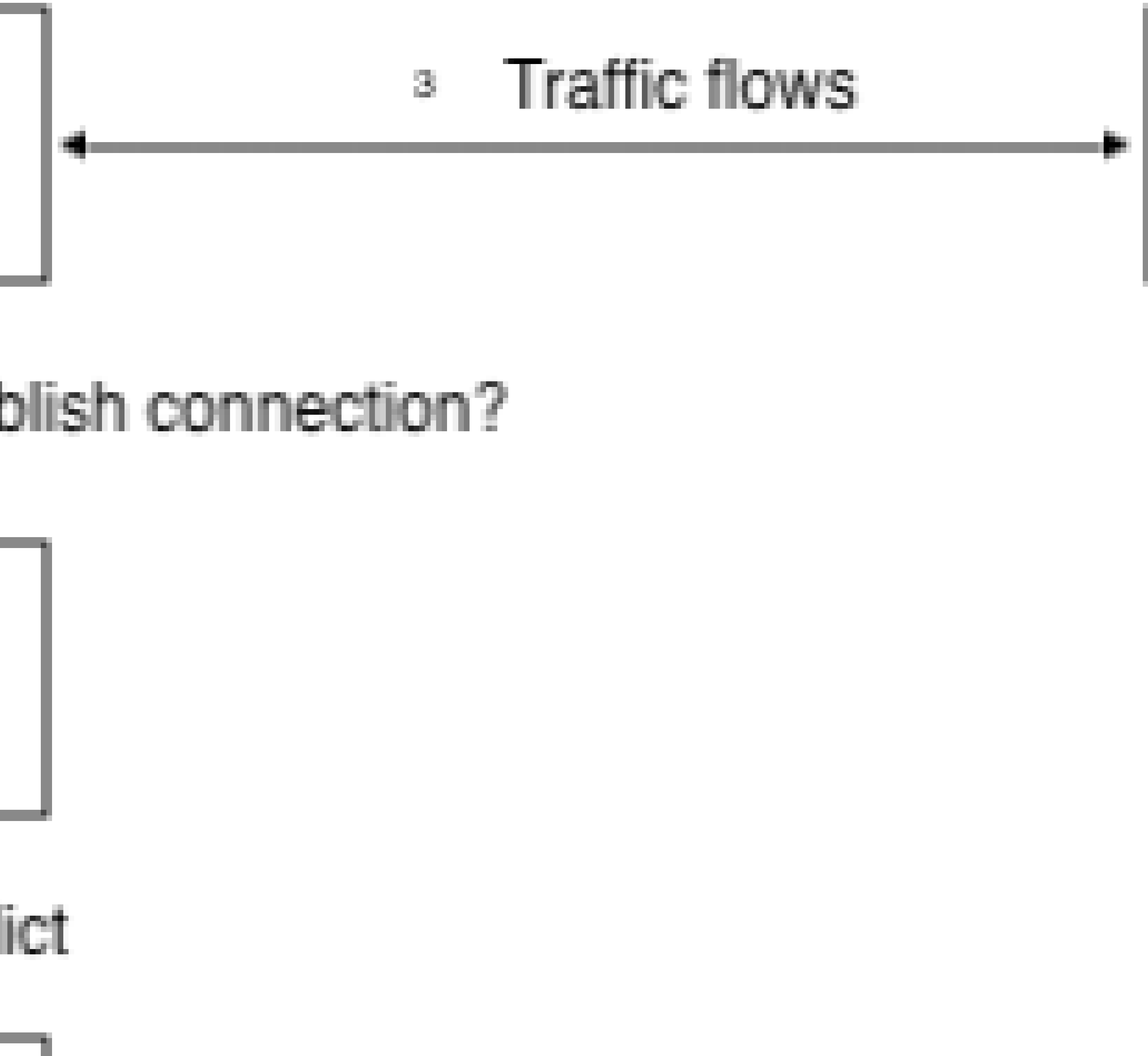
CNNF policy and enforcement is offered in Compute Edition only. For Enterprise Edition (SaaS) customers, a microsegmentation solution will be unveiled shortly.

Architecture

Defender enforces your CNNF policy in real-time.

Defender inspects and evaluates connections before they're set up, and either allows or denies connections from being established. After a connection is established, traffic flows directly between source and destination without any further oversight from Defender.

Defender adds iptables rules to observe TCP's three-way handshake. The three-way handshake sets up new connections using SYN messages. For each pod or container IP address, Defender adds an iptables rule with the target set to NFQUEUE. NFQUEUE is an iptables target which delegates the decision of how to handle a packet to a userspace program (in this case Defender). When SYN messages arrive, Defender evaluates them against policy to determine whether the connection is permitted. From this vantage point, Defender can raise alerts or block connections when anomalous activity is detected.


3
Traffic flows
blish connection?
ict

Enabling CNNF

CNNF runs in one of two modes: disabled or enabled.

- **Disabled** --

  CNNF displays limited traffic flows on Radar, including connections local to a node and outbound connections to the Internet. By default, CNNF ships in the disabled state.

- **Enabled** --

  CNNF monitors all connections, including connections across hosts and connections to any configured network object. CNNF enforces traffic flows according to your policy.

To enable CNNF:

STEP 1 | Open Console.

STEP 2 | Go to **Defend > Radar > Settings**.

STEP 3 | Enable CNNF for hosts and containers.

Interpreting Radar

Radar displays your microsegmentation policy, which is an aggregation of all your rules. Radar also displays attempted connections that raised alerts or were blocked, as well as attempted connections for which there were no rules.

Edges in the graph represent connections. Dotted edges show policy rules. Solid edges show actual observed connections. Clicking on an edge in Radar reveals more information about the connection.

Observed connections are matched against your policy.

- If there is a matching rule, the color of the port number reflects the matching rule's configured effect. Yellow port numbers represent connections that raised an alert. Orange port numbers represent connections that were blocked.
- If there's no matching rule, by default the connection is allowed. The port number is painted gray to show that the connection was observed, but there was no matching rule.

Port numbers painted in gray can indicate holes in your policy, and represent an opportunity to shore it up with additional rules.

If CNNF is disabled, Radar doesn't show outgoing connections to external IPs.

CNNF rules

CNNF rules let you explicitly allow or deny outbound connections from a source to a destination.

For containers, rules can be defined between:

- Image to image.
- Image to external network (where Prisma Cloud isn't running).
- Image to DNS domain.

For hosts, rules can be defined between:

- Host to host.
- Host to external network (where Prisma Cloud isn't running).

When external networks are declared, Prisma Cloud draws a node on the Radar canvas to represent it. If you create a rule that explicitly whitelists traffic between a source and an external network, an edge is drawn on Radar. If no external network is defined, and a connection is made to an external network, nothing is shown on Radar.

Currently, you can't mix DNS rules with image rules. For example, if you have a network object Image A and you define a DNS rule with it, the network object Image A can't have image rules as well. The following two rules can't be simultaneously defined:

- *Image A → DNS A (effect: alert)*
- *Image A → Image B (effect: alert)*

Evaluating rules in a policy

Your manually defined rules represent the full scope of your policy. When a connection is established between two entities in your environment, CNNF uses the following logic to process policy:

1. Apply the first manually-defined rule where both source and destination match.
2. If there are no matching rules, allow the connection.

Network objects

Rules are built around network objects. Network objects represent sources and destinations in your custom CNNF rules. You must declare the relevant network objects in your environment before you can create CNNF rules. Network objects can represent container images, subnets, DNS names, and hosts.

If you have a subnet network object, and you have a rule that blocks or audits on outgoing connections to the subnet for some ports, then blocking and auditing will take effect even if there are rules that allow some of those ports for images or apps that run on machines with IPs from that subnet. Unfortunately, Prisma Cloud cannot detect such "conflicts" when rules are created or updated.

Exporting and importing rules

You can export all manually defined rules. Rules are exported in JSON format and can be transferred between Consoles.

To export your policy, go to **Defend > CNNF**. Click **Export** from either the **Container** or **Host** tab. Whether you export from the **Container** or **Host** tab, the exported JSON will contain:

- The state of CNNF (disabled or enabled).
- Container policy (all rules).
- Host policy (all rules).
- Network entities.

When importing a CNNF policy, everything above will be overwritten by the imported policy.

Creating CNNF rules

Rules are displayed in Radar as dotted lines. When a connection is observed, the dotted line turns solid.

CNNF supports a maximum of 255 manual rules. Each rule can individually define an action (alert or prevent).

*If a rule alerts or prevents outgoing connections to a subnet, blocking/auditing will take effect even if there are rules that allow some of those ports for images/hosts that may be running on machines with IPs from subnets. The same is true for the All subnet (i.e. *.*.*.*/0).*

STEP 1 | Open Console.

STEP 2 | Go to **Defend > CNNF > {Container | Host}**.

STEP 3 | Click **Add rule**.

STEP 4 | Select a source.

If you don't have a network object for the source, click **Add new** in the drop-down list.

STEP 5 | Select a destination.

If you don't have a network object for the destination, click **Add new** in the drop-down list.

STEP 6 | Specify a port, port range, or wildcard.

STEP 7 | Specify an effect.

- **Allow** — Allows the connection.
- **Alert** — Allows the connection, but raises an alert.
- **Prevent** — Blocks the connection and raises an alert.

STEP 8 | Click **Save**.

Secrets

Edit on GitHub

Prisma Cloud integrates with the secrets management tools, such as Hashicorp Vault, CyberArk Enterprise Password Vault, AWS Secrets Manager, and Microsoft Azure Key Vault, to ensure the safe distribution of secrets. Compliance checks let you detect and prevent unsafe usage of secrets.

- Secrets manager
- Integrate with secrets stores
- Secrets Stores
- Inject secrets into containers
- Injecting secrets: end-to-end example

Secrets manager

Edit on GitHub

Containers often require sensitive information, such as passwords, SSH keys, encryption keys, and so on. You can integrate Prisma Cloud with many common secrets management platforms to securely distribute secrets from those stores to the containers that need them.

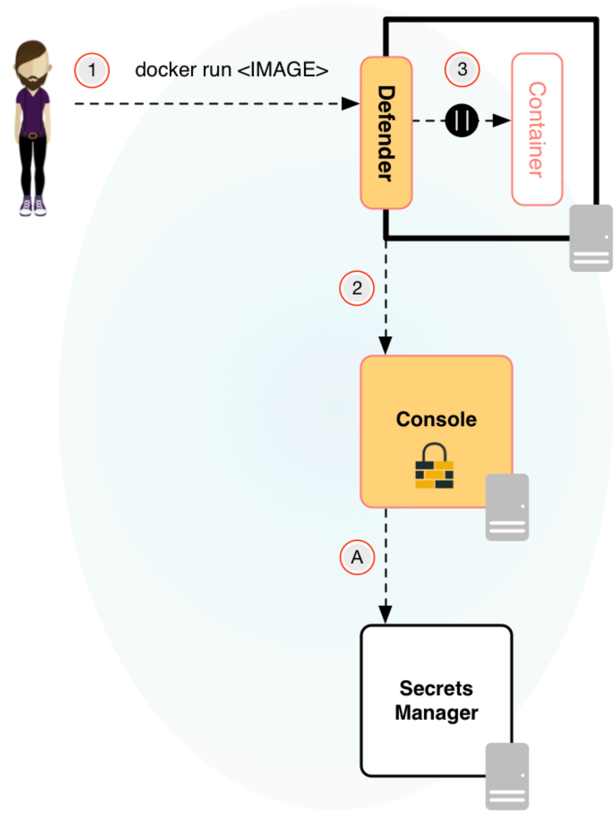
Enterprise secret stores reduce the risk of data breaches that could occur when sensitive information is littered across an organization, often in places where they should not be, such as email inboxes, source code repositories, developer workstations, and Dropbox. Secret stores provide a central and secure location for managing and distributing secrets to the apps that need them. They give you a way to account for all the secrets in your organization with audit trails that show how they are being used.

Orchestrators such as kubernetes and openshift, offer their own built-in secret stores with support for creating secrets, listing them, deleting them, and injecting them into containers. However, if you already have an enterprise secrets store, then you probably want to extend it to handle to your container environment. Utilizing the orchestrator's built-in secrets management capabilities when you already have an enterprise secrets store is unappealing because it represents another silo of sensitive information that needs to be carefully secured. It undermines the principal benefit of a secrets store, which is managing all sensitive data in a single location.

Theory of operation

When a user or orchestrator starts a container, Defender injects the secrets you specify into the container. In order for secret injection to work, all Docker commands must be routed through Defender, so be sure to download your client certs and setup your environment variables.

There are several moving parts in the solution:



A. Secret values are fetched, encrypted, and stored in Console's database when a rule is created or modified. Secret values in Console's database are periodically synced with the secrets store to provide resiliency in the case of connectivity outages and to optimize performance. All secrets cached on disk, both in Defender and in Console, are protected with 256 bit AES encryption. If you change a secret's value in the secrets store and you need it synced immediately, you can click the refresh button in the Console UI to refetch all secrets from their configured stores.

1. Operator (or orchestrator) starts a container with docker run.

2. Defender assesses the command against the policy installed in Console.

3.The secret is returned to Defender. Defender starts the container using the Docker API, which is exposed through the Docker daemon's local UNIX socket, and injects the secret into the container.

Capabilities

The Prisma Cloud secrets manager has the following capabilities:

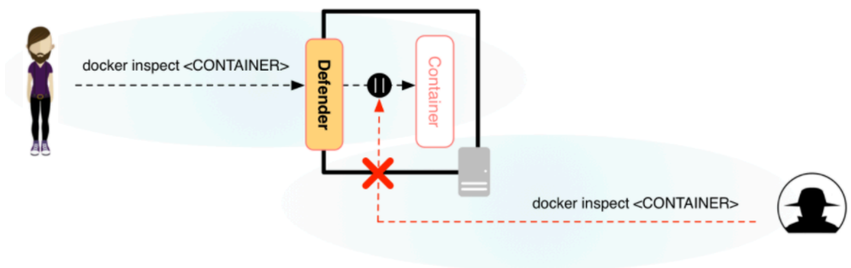
- Supports integration with common secrets management platforms.
- Manages the distribution of secrets from the secret store to your containers through policies. In Console, you create the rules that control which secrets get injected into which containers.
- Injects secrets into containers as either environment variables or files.
- Secrets injected as environment variables are presented in-the-clear from within the container. They are redacted from the outside to prevent exposure by the docker inspect command.

- Secrets injected as files are provided from an in-memory filesystem (on /run/secrets/<SECRET_NAME>) that is mounted into the container when it is created. When the container is stopped, the secrets directory is unmounted and deleted.

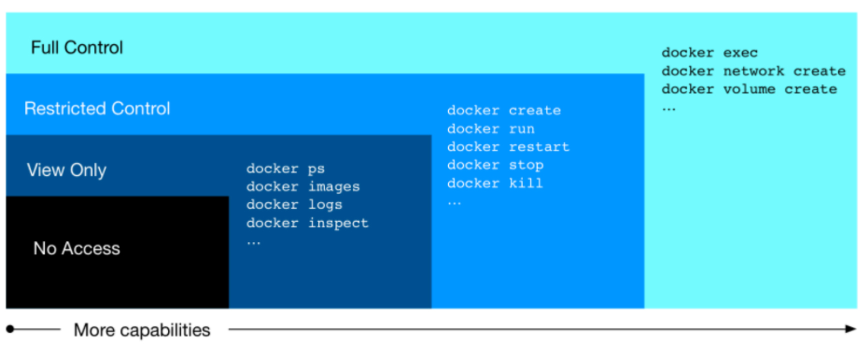
Best practices

There are a number of ways that secrets can be compromised.

Defender is bypassed. If Defender is bypassed, an attacker can execute Docker commands directly against the Docker daemon's local UNIX socket, and he will be able to expose your secrets. Be sure that your hosts are secured with least privilege access so that users can only run docker commands through Defender.



Limit lower privileged users to monitoring commands, such as docker ps and docker inspect. Prisma Cloud automatically encrypts secrets injected as environment variables when accessed from docker inspect. Restrict commands such as docker exec and docker run to just the operators that need them because these commands can reveal secrets injected into a container by giving the user shell access inside the container, where variables are in the clear. For example, docker exec printenv on a running container, or docker run <IMAGE_ID> printenv on an image, can reveal environment variables that are otherwise encrypted with docker inspect. The following diagram shows one way to grant access to Docker functions based on a user's role. This is the way that Docker Datacenter Universal Control Plane (UCP) grants permissions, and you can implement the same scheme with Prisma Cloud's access control rules.



Integrate with secrets stores

Edit on GitHub

To inject secrets into your containers, you must first integrate Prisma Cloud with your secrets manager, and then set up rules for injecting specific secrets into specific containers.

Prisma Cloud can integrate with the following secrets managers:

- AWS Secrets Manager
- AWS Systems Manager Parameters Store
- Azure Key Vault
- CyberArk Enterprise Password Vault
- HashiCorp Vault (versions 0.9.x and older, and versions 0.10 and later)

Refresh interval

By default, the refresh interval is disabled. That means if you change a secret's value in the secrets store, you must force Prisma Cloud to update its list of values. In Console, go to **Defend > Access > Secrets** and click **Refresh secrets** to force Prisma Cloud to fetch the latest values of all secrets from their configured stores.

You can also configure Prisma Cloud to periodically retrieve the latest values of all the secrets from their stores. In Console, go to **Manage > Authentication > Secrets**, click **Edit** next to the **Secrets refresh interval** field, and specify an integer value in hours. Setting the refresh interval to 0 disables automatic periodic refreshes.

Secrets Stores

Edit on GitHub

Integrate Prisma Cloud with the supported secrets management stores.

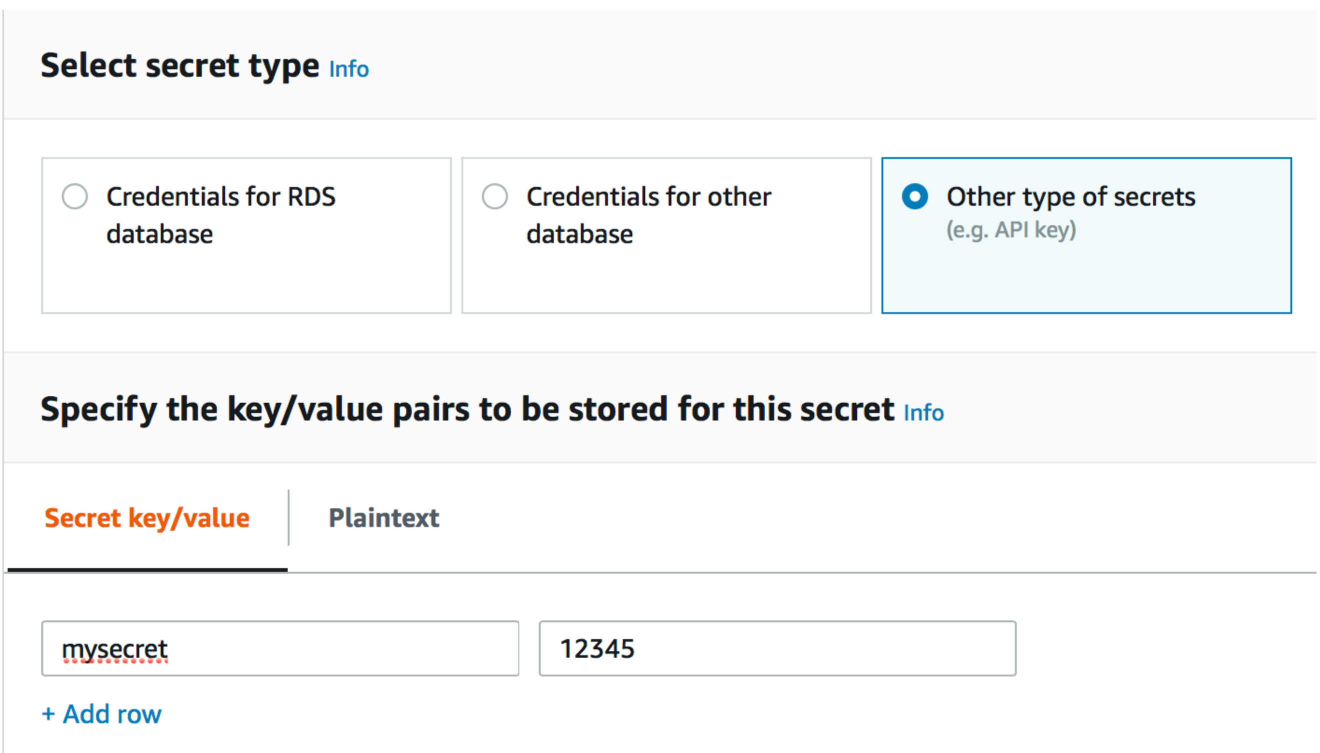
AWS Secrets Manager

Edit on GitHub

You can integrate Prisma Cloud with AWS Secrets Manager. First, configure Prisma Cloud to access AWS Secrets Manager, then create rules to inject the relevant secrets into the relevant containers.

Prerequisites:

- The service account Prisma Cloud uses to access the secrets store must have the following permissions:
  - secretsmanager:GetSecretValue
  - secretsmanager:ListSecrets
- You have created a secret in AWS Secrets Manager. Automatic rotation must be disabled. Prisma Cloud supports the key-value secret type only. When storing a new secret, select **Other type of secrets**, then **Secret key/value**.



STEP 1 | Open Prisma Cloud Console.

STEP 2 | Integrate Prisma Cloud with the secrets store.

1. Go to **Manage > Authentication > Secrets**, and click **Add store**.
2. Enter a name for the store. This name is used when you create rules to inject secrets into specific containers.
3. For **Type**, select **AWS Secrets Manager**, then fill out the rest of the form, including your credentials.
4. Fill out the rest of the form, specifying how to connect to the Secrets Manager.
5. Click **Add**.

   After clicking **Add**, Prisma Cloud tries connecting to your secrets manager. If successful, the dialog closes, and an entry is added to the table. Otherwise, connection errors are displayed directly in the configuration dialog.

   Next, inject a secret into a container.

AWS Systems Manager Parameters Store

Edit on GitHub

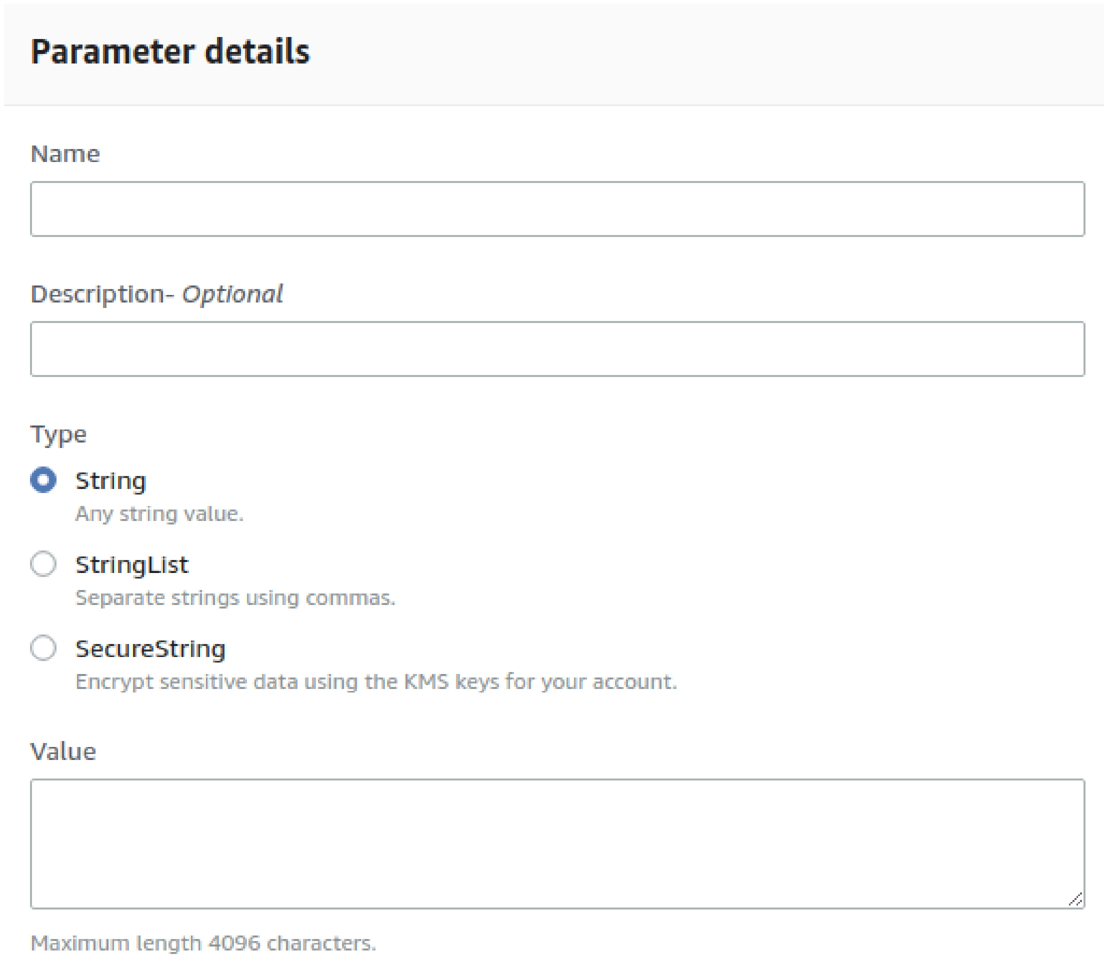
You can integrate Prisma Cloud with AWS Systems Manager Parameters Store. First configure Prisma Cloud to access the Parameters Store, then create rules to inject the relevant secrets into the relevant containers.

Prerequisites:

- The service account Prisma Cloud uses to access the Parameters Store must have the following permissions. These permissions are part of pre-existing policy named AmazonSSMReadOnlyAccess. For more information, see Configure User Access for Systems Manager.
  - ssm:Get*
  - ssm:List*
- You have created a secret in your Parameters Store. Prisma Cloud supports all parameter types. Note, however, that StringList is injected "as-is". For example, if the value you specify

for parameter of type StringList is *twistlock,test,value*, then the injected environment variable would look like this:

```
ENV_VAR=twistlock,test,value
```



STEP 1 | Open Prisma Cloud Console.

STEP 2 | Integrate Prisma Cloud with the store.

1. Go to **Manage > Authentication > Secrets**, and click **Add store**.
2. Enter a name for the store. This name is used when you create rules to inject secrets into specific containers.
3. For **Type**, select **AWS Systems Manager Parameters Store**.
4. Fill out the rest of the form, specifying how to connect to the store.
5. Click **Add**.

   After clicking **Add**, Prisma Cloud tries conecting to your store. If it is successful, the dialog closes, and an entry is added to the table. Otherwise, any connection errors are displayed directly in the configuration dialog.

   Next, inject a secret into a container.

Azure Key Vault

Edit on GitHub

You can integrate Prisma Cloud with Azure Key Vault. First configure Prisma Cloud to access your Key Vault, then create rules to inject the relevant secrets into their associated containers.

Prerequisites: You have created a secret in Key Vault.

STEP 1 | Create an Azure servicePrincipal in your Azure AD Tenant

1. Use AZ CLI to create a servicePrincipal and obtain the json credential file.
2. Authenticate to your Azure tenant.

```
$ az login
```

3. Create a servicePrincipal

```
$ az ad sp create-for-rbac
```

4. Save the resulting json output.+

```
{
  "appId": "xxxxxxxx-xxxxx-xxxx-xxxxxxxx",
  "displayName": "azure-cli-2018-11-01-xx-xx-xx",
  "name": "http://azure-cli-2018-11-01-xx-xx-xx",
  "password": "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx",
  "tenant": "xxxxxxxxxxxxxxxxxxxxxxxxxxx"
}
```

5. In the Azure Key Vault, add the servicePrincipal to the **Access Policies** with the following permissions:

```
secrets/get permission
secrets/list permission
```

STEP 2 | In the Prisma Cloud Console, go to **Manage > Authentication > Secrets**.

STEP 3 | Click **Add store**.

1. Enter a name for the vault. This name is used when you create rules to inject secrets into specific containers.
2. For **Type**, select **Azure Key Vault**.
3. For **Address**, enter **https://<vault-name>.vault.azure.net**. This address can be found in the Azure Key Vault's properties in the *DNS Name* element.
4. In **Credential**, click **Add new**.

   *If you create a credential in the credentials store (**Manage > Authentication > Credentials store**), your service principal authenticates with a password. To authenticate with a certificate,* create a cloud account.

5. Enter a name for the credentials.
6. In **Type**, select **Azure**.
7. In **Service Key**, enter the JSON credentials returned from the *az ad sp create-for-rbac* command.
8. Click **Save**.
9. Click **Add**.

   After adding the new store, Prisma Cloud tries conecting to your vault. If it is successful, the dialog closes, and an entry is added to the table. Otherwise, any connection errors are displayed directly in the configuration dialog.

   Next, inject a secret into a container.

CyberArk Enterprise Password Vault

Edit on GitHub

You can integrate Prisma Cloud with CyberArk Enterprise Password Vault (EPV). To retrieve passwords from the vault, Prisma Cloud uses the CyberArk Central Credential Provider (CCP) web service. Prisma Cloud supports CyberArk CCP version 12.1.0 with Digital Vault version 12.2.0. To integrate with CyberArk EPV, first configure Prisma Cloud to access CyberArk Enterprise Password Vault, then create rules to inject the relevant secrets into the relevant containers.

STEP 1 | In Console, go to **Manage > Authentication > Secrets**.

STEP 2 | Click **Add store**.

1. Enter a name for the vault. This name is used when you create rules to inject secrets into specific containers.
2. For **Secret type**, select **CyberArk Enterprise Password Vault**.
3. In **Settings**, fill out the form as follows:
   1. Address: the address and port of the Central Credential Provider web service.
   2. Application ID: The application ID that Prisma Cloud should use to issue each password request. To configure this for CCP 12.1, see here.
   3. CA certificate (Optional): for an application configured to authenticate using a client certificate, the certificate of the CA that signed the CyberArk server's certificate in

PEM format. For more information about this authentication method for CCP 12.1 see here.

4. Client certificate (Optional): for an application configured to authenticate using a client certificate, the client certificate in PEM format.

4. Click **Add**.

After clicking **Add**, Prisma Cloud tries conecting to your vault. If it is successful, the dialog closes, and an entry is added to the table. Otherwise, any connection errors are displayed directly in the configuration dialog.

Next, inject a secret into a container.

HashiCorp Vault

Edit on GitHub

You can integrate Prisma Cloud with HashiCorp Vault. Prisma Cloud supports the K/V Secrets Engine v2 in Vault 0.10.x, and K/V Secrets Engine v1 in Vault 0.9.x and older. Prisma Cloud does not support Secrets Engine v1 in Vault 0.10.x.

First configure Prisma Cloud to access HashiCorp Vault, then create rules to inject the relevant secrets into the relevant containers.

STEP 1 | In Console, go to **Manage > Authentication > Secrets**.

STEP 2 | Click **Add store**.

1. Enter a name for the vault. This name is used when you create rules to inject secrets into specific containers.
2. For **Type**, select **HashiCorp Vault**. Choose the version that matches the version of Vault installed in your environment.
3. Fill out the rest of the form, specifying how to connect to your vault.
4. Click **Add**.

After clicking **Add**, Prisma Cloud tries conecting to your vault. If it is successful, the dialog closes, and an entry is added to the table. Otherwise, any connection errors are displayed directly in the configuration dialog.

Next, inject a secret into a container.

Inject secrets into containers

Edit on GitHub

To inject secrets into your containers, first integrate Prisma Cloud with your secrets manager, and then set up rules for injecting specific secrets into specific containers.

Use the same procedure for injecting secrets in a Kubernetes cluster. Set up your rules to target specific containers, images, or labels. Make sure Kubernetes uses dockerd and Prisma Cloud is running in local socket mode.

Injecting secrets into containers

After integrating your secrets store with Prisma Cloud, specify which secrets should be injected into which containers. To do this, create the appropriate rules in Console.

Secrets can be injected as environment variables or as files.

For secrets injected as environment variables: if there is a collision between a predefined environment variable and an injected secret, the value of the environment variable will always be the value of the secret.

For security reasons, secrets injected as environment variables are only exposed to the container's main process and children of the main process.

For secrets injected as files: they can be found in */run/secrets/<SECRET_NAME>*, where the contents of the file contain the secret's value. By default, secrets can only be read by root users in the container space. If you run your containers as non-root users, configure the injection rule to make the secrets readable by all users. Prisma Cloud can set the access permissions of the injected secrets file to read-only for the 'others' class of users. For more information about access permissions and 'others', see the chmod man page.

Secrets injection currently only works with image labels, not container or host labels.

Prerequisite: You've already created a secret in your store or vault.

STEP 1 | In Console, go to **Defend > Access > Secrets**.

STEP 2 | Click **Add new secrets rule**.

STEP 3 | Specify a name for your rule.

STEP 4 | Specify how your secret(s) should be injected. You can choose between environment variables and files.

If you choose files, you can select how the files are injected into the container. By default, the files are readable by root users only. If your containers run as non-root users, select **All Users**. The **All Users** option makes the files readable by any user by setting read permission for the others class of users.

STEP 5 | Create a list of secrets from your store that you want to inject into your container(s). Under **Add secret**:

1. Specify a secret name.

   When you inject secrets as environment variables, this field specifies the environment variable name.

   When you inject secrets as files, this field specifies the file name.
2. Specify the store where the secret is stored. The drop-down list contains any store that you integrated with Prisma Cloud.
3. Specify the secret's path and key.
4. Click **Add Secret**. It is added to the list of secrets.
5. Repeat steps a through d for as many secrets that must be included in your rule.

STEP 6 | Specify a scope for your rule with one or more collections. By default, the **All** collection is selected, which injects all secrets into all containers on all hosts.

STEP 7 | Click **Add**.

STEP 8 | Verify that your secrets are properly injected.

For example, assuming your rule targets the alpine container and secrets are injected as environment variables, run the following commands:

*Default rules target all resources in the environment. The **Containers**, **Images**, **Hosts**, and **Labels** fields are set to wildcards. If your rule is set up this way, then your secrets will be injected into the alpine container.*

```
$ docker run -it alpine:latest /bin/sh
/ # printenv
```

If your secrets are injected as files, and you left **Target directory** unspecified in your rule, then your secrets are injected into */run/secrets/*, where <SECRET-NAME> is the name of the injected file, as specified in your rule.

```
$ docker run -it alpine:latest /bin/sh
/ # cat /run/secrets/<SECRET-NAME>
```

Injecting secrets: end-to-end example

Edit on GitHub

This article presents a step-by-step guide for testing Prisma Cloud's secret manager. You will set up HashiCorp Vault, store a secret in it, inject the secret into a running container, then validate that it can be seen from within the container.

Setting up Vault

Set up HashiCorp Vault in development mode.

STEP 1 | Download Vault from https://www.vaultproject.io/downloads.html.

STEP 2 | Unzip the package, then copy the vault executable to a directory in your *PATH*.

STEP 3 | Verify that vault is installed. Run the following command:

```
$ vault -help
```

STEP 4 | Start Vault in development mode.

```
$ vault server -dev -dev-listen-address='<VAULT_HOST_IPADDR>:8200'

==> WARNING: Dev mode is enabled!

In this mode, Vault is completely in-memory and unsealed.
Vault is configured to only have a single unseal key. The root
token has already been authenticated with the CLI, so you can
immediately begin using the Vault CLI.

The only step you need to take is to set the following
environment variables:

    export VAULT_ADDR='http://10.240.0.53:8200'

The unseal key and root token are reproduced below in case you
want to seal/unseal the Vault or play with authentication.

Unseal Key: Hb0dBfYh3ieHRmf28ohu5xh0DKfmP4aNa8JS5/jNsWQ=
Root Token: 29e3e12b-09b4-af6c-6e87-cbd9fbcb51bd
```

Storing a secret in HashiCorp Vault

Store a secret in Vault.

STEP 1 | Open a shell and ssh to the host running Vault.

STEP 2 | Set the Vault address in your environment.

```
$ export VAULT_ADDR='http://<VAULT_HOST_IPADDR>:8200'
```


STEP 3 | Create a secret.

For Vault 0.10 or later:

```
vault kv put secret/mySecret1 "pass=1234567"
```

For Vault 0.9.x or older:

```
$ vault write secret/mySecret1 "pass=1234567"
```

STEP 4 | Read the secret back to validate it was properly stored.

For Vault 0.10 or later:

```
$ vault kv get secret/mySecret1
```

For Vault 0.9.x or older:

```
$ vault read secret/mySecret1
```

Integrating Prisma Cloud and Vault

Follow the steps in Integrating Prisma Cloud with HashiCorp Vault.

Creating a rule in Console

Follow the steps in Injecting secrets into containers.

Validating the secret is injected

Start a container and verify that your secret is properly injected.

+ The same procedure can be followed for injecting secrets in Kubernetes cluster. Make sure Kubernetes uses dockerd and Prisma Cloud runs in local socket mode.

Prerequisites: Defender must be running on the machine where you start your container.

STEP 1 | Start a container:

```
$ docker run -ti ubuntu /bin/bash
```

STEP 2 | Validate your secrets have been injected into this container.

If you injected your secrets as environment variables, run:

```
# printenv
```

If you injected your secrets as files, run:

```
# ls /run/secrets
# cat /run/secrets/<SECRET_NAME>
```

STEP 3 | Exit the shell inside the container.

```
# exit
```

STEP 4 | If your secrets are injected as environment variables, validate that they are encrypted when you run docker inspect.

Start a container, and run it in the background:

```
$ docker run -dit ubuntu /bin/bash
<CONTAINER_ID>
```

```
$ docker inspect <CONTAINER_ID>
```


Alerts

Edit on GitHub

Prisma Cloud lets you surface critical policy breaches by sending alerts to any number of channels. Alerts ensure that significant events are put in front of the right audience at the right time.

- Alert mechanism
- AWS Security Hub
- Cortex XDR alerts
- Cortex XSOAR alerts
- Email alerts
- Google Cloud Pub/Sub
- Google Cloud Security Command Center
- IBM Cloud Security Advisor
- JIRA Alerts
- PagerDuty alerts
- ServiceNow alerts
- ServiceNow alerts
- Slack Alerts
- Splunk alerts
- Webhook alerts

Alert mechanism

Edit on GitHub

Prisma Cloud generates alerts to help you focus on the significant events that need your attention. Because alerts surface policy violations, you need to put it in front of the right audience and in a timely manner. To meet this need, you can create alert profiles that send events/ notifications to the alert notification providers your internal teams use to triage and address these violations.

Alert profiles are built on the following constructs:

- **Alert provider** --

  Specifies the notification provider or channel to which you want to send alerts. Prisma Cloud supports multiple options such as email, JIRA, Cortex, PagerDuty.

You can create any number of alert profiles, where each profile gives you granular control over who should receive the notifications and for what types of alerts.

- **Alert settings** --

  Specifies the configuration settings required to send the alert to the alert provider or messaging medium.

- **Alert triggers** --

  Specifies what alerts you want to send to the provider included in the profile. Alerts are generated when the rules included in your policy are violated, and you can choose whether you want to send a notification for the detected issues. For example, on runtime violations, compliance violations, cloud discovery or WAAS.

Not all triggers are available for all alert providers.

Frequency

Most alerts trigger on a policy violation, and are aggregated by the audit aggregation period or frequency that you define as a global setting. Vulnerability, compliance, and cloud discovery alerts work differently, as described below.

Vulnerability alerts

Image vulnerabilities are checked for image in the registry and deployed. The number of known vulnerabilities in a resource is not static over time. As the Prisma Cloud Intelligence Stream is updated with new data, new vulnerabilities might be uncovered in resources that were previously considered clean. The first time a resource (image, container, host, etc.) enters the environment, Prisma Cloud assesses it for vulnerabilities. Thereafter, every resource is periodically rescanned. Prisma Cloud sends an alert that reports on all newly detected vulnerabilities within the last 24 hours.

Vulnerability alerts from registry scans only trigger for the 50 most recent images, as sorted by last modified date. The limit is designed to optimize Console resource consumption in large environments.

- **Immediate alerts** — You can configure sending alerts immediately when the number of vulnerabilities for the resource increased, which can happen in one of the following scenarios:
  - Deploy a new image/host with vulnerabilities.
  - Detect new vulnerabilities when re-scanning existing image/host, in that case an alert is dispatched again for this resource with all its vulnerabilities.

    Immediate alerts are not supported for registry scan vulnerabilities.

  The ability to send immediate vulnerability alerts is configurable for each alert profile and is disabled by default.

  Immediate alerts do not affect the vulnerabilities report that is generated every 24 hours. The report will include all vulnerabilities that were detected in the last 24 hours, including those sent as an immediate alert.

Compliance alerts

Compliance alerts are sent in one of two ways. Each alert channel that has compliance alert triggers ("Container and image compliance", "Host compliance"), only uses one of these ways.

Compliance reports

This form of compliance alert works under the idea that resources in your system can only be in one of two states: compliant or non-compliant. When your system is non-compliant, Prisma Cloud sends an alert. As long as there are non-compliant resources, Prisma Cloud sends an alert every 24 hours. Compliance reports list each failed check, and the number of resources that failed the check in the latest scan and the previous scan. For detailed information about exactly which resources are non-compliant, use Compliance Explorer.

For example:

- Scan period 1: You have non-complaint container named *crusty_pigeon*. You'll be alerted about the container compliance issues.
- Scan period 2: Container *crusty_pigeon* is still running. It's still non-compliant. You'll be alerted about the same container compliance issues.

The following screenshot shows an example compliance email alert:

PRISMA CLOUD

For full details, please see Compliance Explorer on Prisma Cloud Compute console

Container compliance

		Previous	Curre
520	Do not share the host's UTS namespace	67	76
515	Do not share the host's process namespace	51	48
59	Do not share the host's network namespace	106	115
519	Do not set mount propagation mode to shared	3	6
55	Do not mount sensitive host system directories on containers	78	84
597	Secrets in clear text environment variables	11	8
598	Container app is running with weak settings	17	14
51	Verify AppArmor profile, if applicable	13	22
54	Do not use privileged containers	31	34
528	Use PIDs cgroup limit	190	172
525	Restrict container from acquiring additional privileges	175	158
599	Container is running as root	173	154
510	Limit memory usage for container	110	92
521	Do not disable default seccomp profile	166	153

Image compliance

		Previous	Curre
425	Private keys stored in image	12	10
406	Add HEALTHCHECK instruction to the container image	87	56

Host compliance

		Previous	Curre
435	Private keys stored in function	72	71
24	Do not use insecure registries	9	12

This method applies to the following alert channels: email, Cortex XSOAR.

Compliance scans

This form of compliance alert is emitted whenever there is an increasment in the number of compliance issues detected on a resource. The first time a resource (image, container, host, etc) enters the environment, Prisma Cloud assesses it for compliance issues. If a compliance issue violates a rule in the policy, and the rule has been configured to trigger an alert, an alert is dispatched. Thereafter, every time a resource is rescanned (periodically or manually), and there is an increasment in the resource's compliance issues, an alert is dispatched again for this resource with all its compliance issues.

This method applies to the following alert channels: Webhook, Splunk, and ServiceNow.

Cloud discovery alerts

Cloud discovery alerts warn you when new cloud native resources are discovered in your environment so that you can inspect and secure them with Prisma Cloud. Cloud discovery alerts are available on the email and XSOAR channels only. For each new resource discovered in a scan, Prisma Cloud lists the cloud provider, region, project, service type (i.e. AWS Lambda, Azure AKS) and resoure name (my-aks-cluster).

WAAS alerts

WAAS alerts are generated for the following—WAAS Firewall (App-Embedded Defender), WAAS Firewall (container), WAAS Firewall (host), WAAS Firewall (serverless), WAAS Firewall (Out-of-band), WAAS health

Management

When you set up alerts for Defender health events. These events tell you when Defender unexpectedly disconnects from Console. Alerts are sent when a Defender has been disconnected for more than 6 hours.

CNNS

Cloud Native Network Segmentation (CNNS)

Runtime

Runtime alerts are generated for the following categories—Container runtime, App-Embedded Defender runtime,Host runtime, Serverless runtime, and Incidents.

For runtime audits, there's a limit of 50 runtime audits per aggregation period (seconds, minutes, hours, days) for all alert providers.

Access

Access alerts are for the audits of users who accessed the management console (Admission audits) and Kubernetes audits.

Code repository

Code repository vulnerabilities

Set up alert notifications to an external integration using an alert profile

1. Navigate to **Compute > Manage > Alerts**.
2. Set the default frequency for alert notifications.

   The value you set for **General Settings** applies for all alert notifications except for vulnerability, compliance, and cloud discovery. For vulnerability, compliance, and cloud discovery the default frequency varies by integration and is displayed when you select the alert triggers for which you want to send notifications in step 4. The default for all other alert notifications is 1 second, and you can change it to 1 minute, 10 minutes, 1 hour, or 1 day.

3. Enter a name for the profile.

   Select the provider from the list. The supported providers are : Cortex, Email, Google Pub/Sub, Google CSCC, IBM Cloud Security Advisor, Jira, PagerDuty, ServiceNow, AWS Security Hub, Slack, Splunk, Webhook
4. Select the triggers.

   The triggers are grouped by category.

   For each category you can select the event for which you want to send a notification and select the rules for the respective trigger. The frequency for vulnerability, compliance, and cloud discovery varies by provider and is enabled when you select one or more triggers within the alert category (see above for a description of each category).
5. Set up the configuration for integrating with the provider.

   Use the instructions for the provider of your choice.
6. Review the summary.
7. Send a test alert.
8. Verify the status of the alert profile.

   Check that the alert profile you created displays in the table and the connection status is green. If not, edit the profile to set it up properly and verify that the test alert is successful.

AWS Security Hub

Edit on GitHub

AWS Security Hub aggregates, organizes, and prioritizes security alerts from multiple AWS services and AWS Partner Network solutions, including Prisma Cloud, to give you a comprehensive view of security across your environment.

Permissions

The minimum required permissions policy to integrate Prisma Cloud with AWS Security Hub is **AWSSecurityHubFullAccess**. Whether using IAM users, groups, or roles, be sure the entity Prisma Cloud uses to access AWS Security Hub has this minimum permissions policy.

This procedure shows you how to set up integration with an IAM user (configured as a service account). In AWS IAM, create a service account that has the **AWSSecurityHubFullAccess** permissions policy. You will need the service account's access key ID and secret access key to integrate with Prisma Cloud.

Enabling AWS Security Hub

STEP 1 | Log into your AWS tenant and enter **Security Hub** in the **Find services** search, then select **Security Hub**.

STEP 2 | Click **Enable Security Hub**.

STEP 3 | Enable the Prisma Cloud integration.

1. Choose Integrations from the Security Hub menu.
2. Accept findings from Palo Alto Networks: Prisma Cloud Compute.

   See AWS documentation

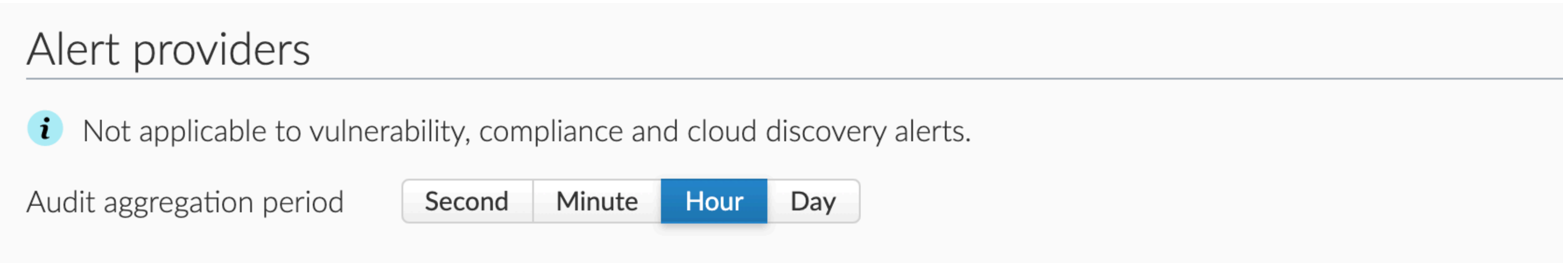
Configuring alert frequency

You can configure the rate at which alerts are emitted. This is a global setting that controls the spamminess of the alert service. Alerts received during the specified period are aggregated into a single alert. For each alert profile, an alert is sent as soon as the first matching event is received. All subsequent alerts are sent once per period.

STEP 1 | Open Console, and go to **Manage > Alerts**.

STEP 2 | In **Aggregate audits every**, specify the maximum rate that alerts should be sent.

You can specify **Second**, **Minute**, **Hour**, **Day**.



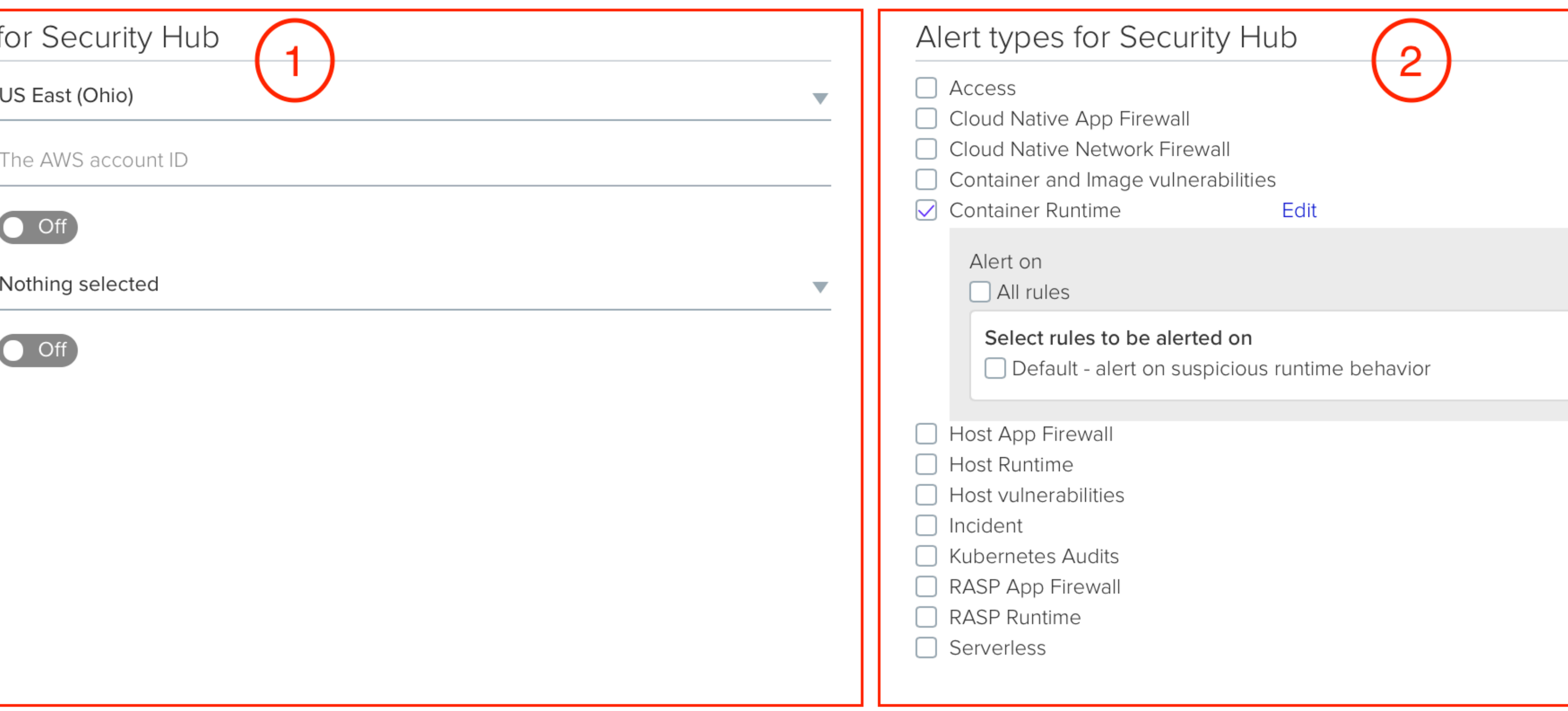
Sending alerts to Security Hub

Alert profiles specify which events should trigger the alert machinery, and to which channel alerts are sent. You can send alerts to any combination of channels by creating multiple alert profiles.

Alert profiles consist of two parts:

(1) Alert settings — Who should get the alerts, and on what channel? Configure Prisma Cloud to integrate with your messaging service and specify the people or places where alerts should be sent. For example, configure the email channel and specify a list of all the email addresses where alerts should be sent. Or for JIRA, configure the project where the issue should be created, a long with the type of issue, priority, assignee, and so on.

(2) Alert triggers — Which events should trigger an alert to be sent? Specify which of the rules that make up your overall policy should trigger alerts.



If you use multi-factor authentication, you must create an exception or app-specific password to allow Console to authenticate to the service.

Create new alert profile

Create a new alert profile.

STEP 1 | In **Manage > Alerts**, click **Add profile**.

STEP 2 | Enter a name for your alert profile.

STEP 3 | In **Provider**, select **AWS Security Hub**.

Configure the channel

Configure the channel.

STEP 1 | In **Region**, select your region.

STEP 2 | Enter your **Account ID**, which can be found in the AWS Management Console under **My Account > Account Settings**.

STEP 3 | Select or create credentials, which Prisma Cloud uses to integrate with AWS Security Hub.

You can use an IAM user, IAM role, or AWS STS.

STEP 4 | Click **Send Test Alert** to test the connection. An alert is sent immediately.

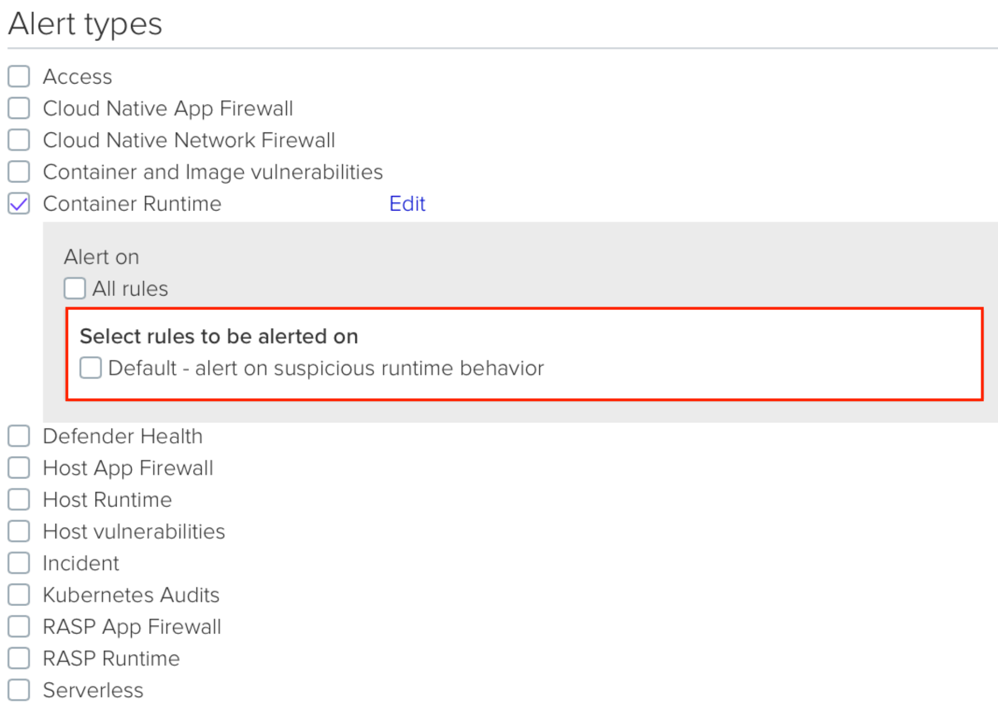
Configure the triggers

Configure how the alert is triggered.

STEP 1 | Under **Alert Types**, check the boxes types of events that should trigger an alert.

STEP 2 | For additional configuration options, click **Edit**.

STEP 3 | To specify specific rules that should trigger an alert, deselect **All rules**, and then select any individual rules.



STEP 4 | Click **Save**.

Cortex XDR alerts

Edit on GitHub

Cortex XDR is a detection and response app that natively integrates network, endpoint and cloud data to stop sophisticated attacks. Prisma Cloud can send runtime alerts to XDR when your policies are violated. Prisma Cloud can be configured to send data when an entire policy, or even specific rules, are violated.

Prisma Cloud uses webhooks to send the alerts to Cortex XDR. When an event occurs, Prisma Cloud notifies the web service with an HTTP POST request that contains a JSON body.

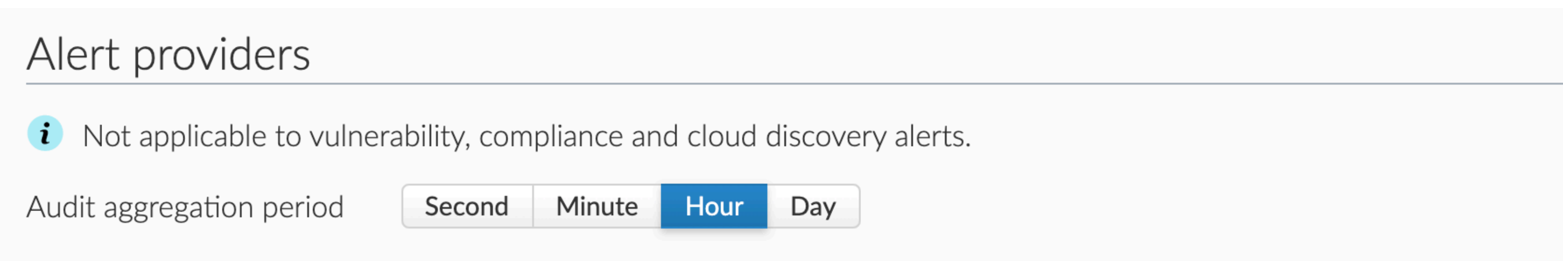
Configuring alert frequency

You can configure the rate at which alerts are emitted. This is a global setting that controls the spamminess of the alert service. Alerts received during the specified period are aggregated into a single alert. For each alert profile, an alert is sent as soon as the first matching event is received. All subsequent alerts are sent once per period.

STEP 1 | Open Console, and go to **Manage > Alerts**.

STEP 2 | In **Aggregate audits every**, specify the maximum rate that alerts should be sent.

You can specify **Second**, **Minute**, **Hour**, **Day**.



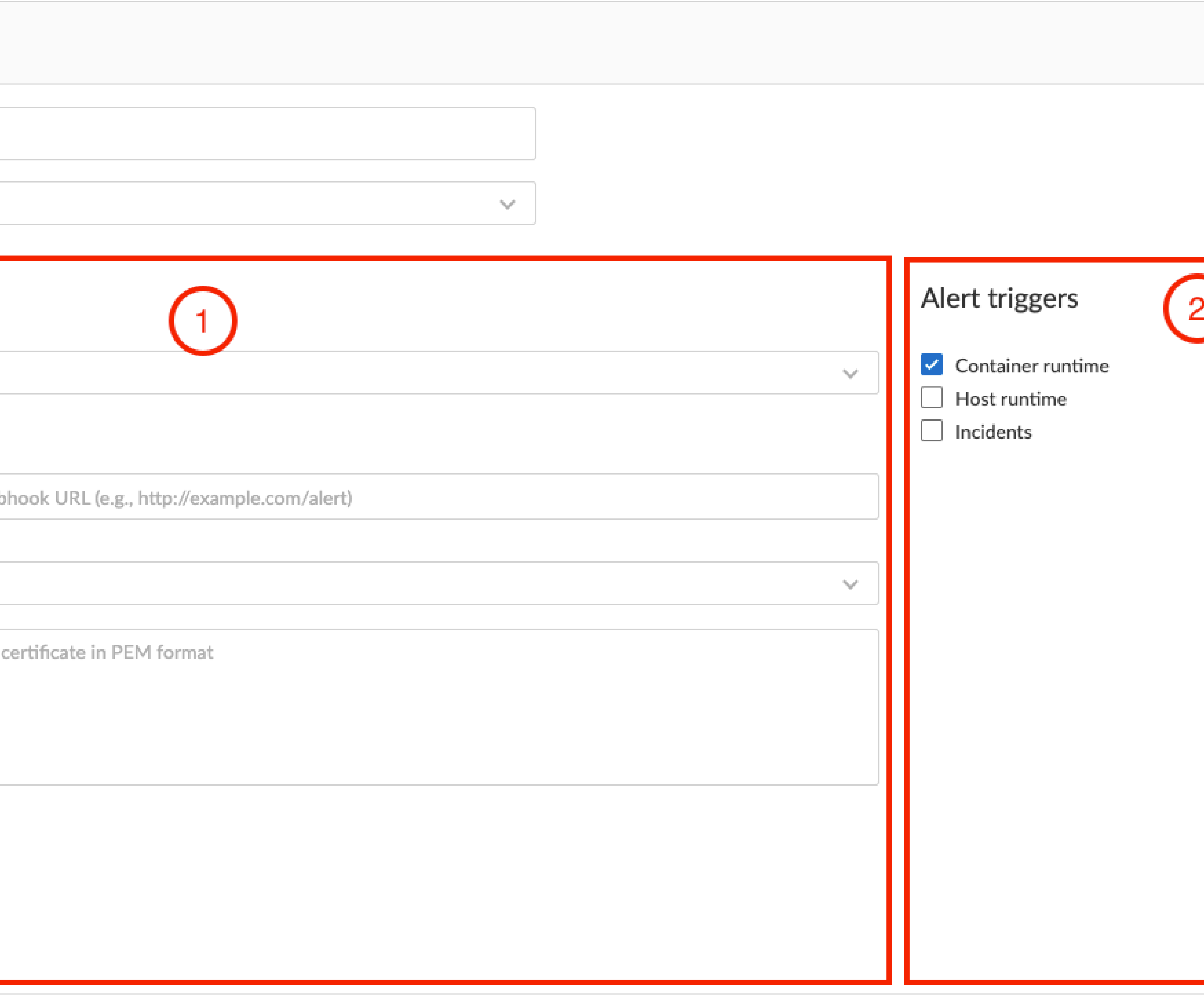
Send alerts to XDR

Alert profiles specify which events should trigger the alert machinery, and to which channel alerts are sent. You can send alerts to any combination of channels by creating multiple alert profiles.

Alert profiles consist of two parts:

(1) Alert settings — Who should get the alerts, and on what channel? Configure Prisma Cloud to integrate with your messaging service and specify the people or places where alerts should be sent. For example, configure the email channel and specify a list of all the email addresses where alerts should be sent. Or for JIRA, configure the project where the issue should be created, a long with the type of issue, priority, assignee, and so on.

(2) Alert triggers — Which events should trigger an alert to be sent? Specify which of the rules that make up your overall policy should trigger alerts.



If you use multi-factor authentication, you must create an exception or app-specific password to allow Console to authenticate to the service.

Create new alert channel

Create a new alert channel for Cortex XDR.

STEP 1 | In **Manage > Alerts**, click **Add profile**.

STEP 2 | Enter a name for your alert profile.

STEP 3 | In **Provider**, select **Cortex**.

STEP 4 | In **Application**, select **XDR**.

Configure the channel

Configure the new channel.

STEP 1 | In **Incoming webhook URL**, enter the Cortex XDR endpoint where Prisma Cloud should submit the alerts.

STEP 2 | (Optional) In **Credential**, specify a basic auth credential if your endpoint requires authentication.

STEP 3 | (Optional) In **CA Certificate**, enter a CA cert in PEM format.

When using a CA cert to secure communication, only one-way SSL authentication is supported. If two-way SSL authentication is configured, alerts will not be sent.

STEP 4 | Click **Send Test Alert** to test the connection. An alert is sent immediately.

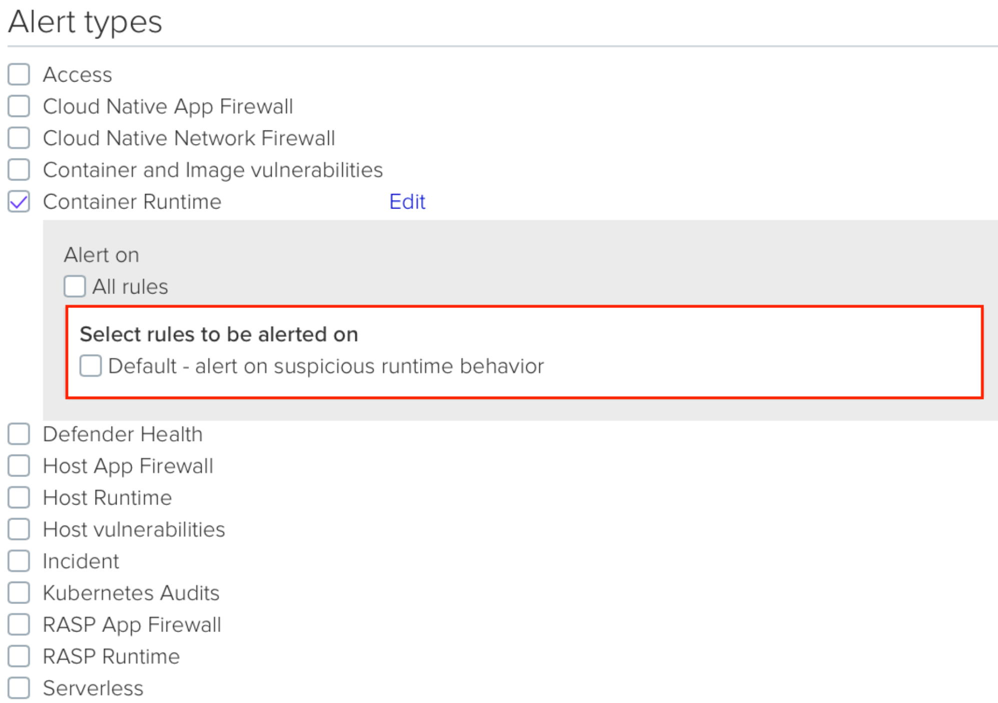
Configure the triggers

Configure how the alert is triggered.

STEP 1 | Under **Alert Types**, check the boxes types of events that should trigger an alert.

STEP 2 | For additional configuration options, click **Edit**.

STEP 3 | To specify specific rules that should trigger an alert, deselect **All rules**, and then select any individual rules.



STEP 4 | Click **Save**.

Cortex XSOAR alerts

Edit on GitHub

Cortex XSOAR is a security orchestration, automation, and response (SOAR) platform. Prisma Cloud can send alerts, vulnerabilities, and compliance issues to XSOAR when your policies are violated. Prisma Cloud can be configured to send data when an entire policy, or even specific rules, are violated.

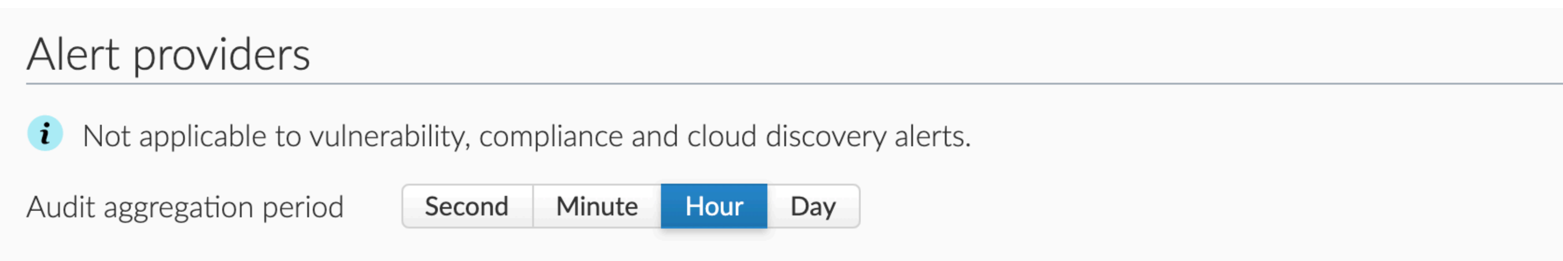
Configuring alert frequency

You can configure the rate at which alerts are emitted. This is a global setting that controls the spamminess of the alert service. Alerts received during the specified period are aggregated into a single alert. For each alert profile, an alert is sent as soon as the first matching event is received. All subsequent alerts are sent once per period.

STEP 1 | Open Console, and go to **Manage > Alerts**.

STEP 2 | In **Aggregate audits every**, specify the maximum rate that alerts should be sent.

You can specify **Second**, **Minute**, **Hour**, **Day**.



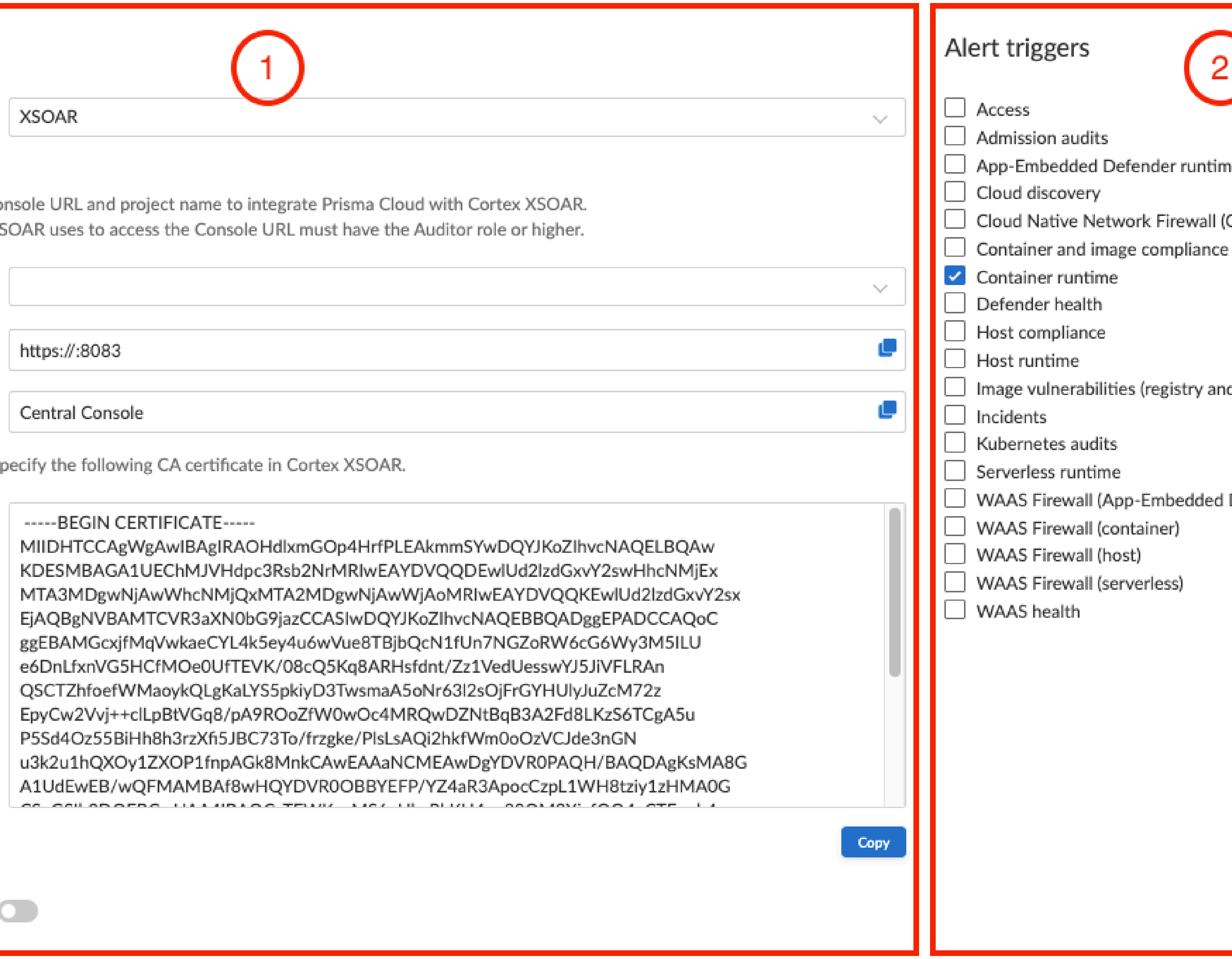
Send alerts to XSOAR

Alert profiles specify which events should trigger the alert machinery, and to which channel alerts are sent. You can send alerts to any combination of channels by creating multiple alert profiles.

Alert profiles consist of two parts:

(1) Alert settings — Who should get the alerts, and on what channel? Configure Prisma Cloud to integrate with your messaging service and specify the people or places where alerts should be sent. For example, configure the email channel and specify a list of all the email addresses where alerts should be sent. Or for JIRA, configure the project where the issue should be created, a long with the type of issue, priority, assignee, and so on.

(2) Alert triggers — Which events should trigger an alert to be sent? Specify which of the rules that make up your overall policy should trigger alerts.



If you use multi-factor authentication, you must create an exception or app-specific password to allow Console to authenticate to the service.

Create new alert profile

Create a new alert profile.

STEP 1 | In **Manage > Alerts**, click **Add profile**.

STEP 2 | Enter a name for your alert profile.

STEP 3 | In **Provider**, select **Cortex**.

STEP 4 | In **Application**, select **XSOAR**.

Configure the channel

Configure the channel.

STEP 1 | In **Console Name**, choose the console name XSOAR should use to access your Prisma Cloud console.

STEP 2 | Copy the **Console URL** and save it for creating the integration in XSOAR.

STEP 3 | Copy the **Project Name**, and save it for creating the integration in XSOAR (the project name would only appear if you are using the projects feature).

STEP 4 | Copy the **CA certificate** and save it for creating the integration in XSOAR.

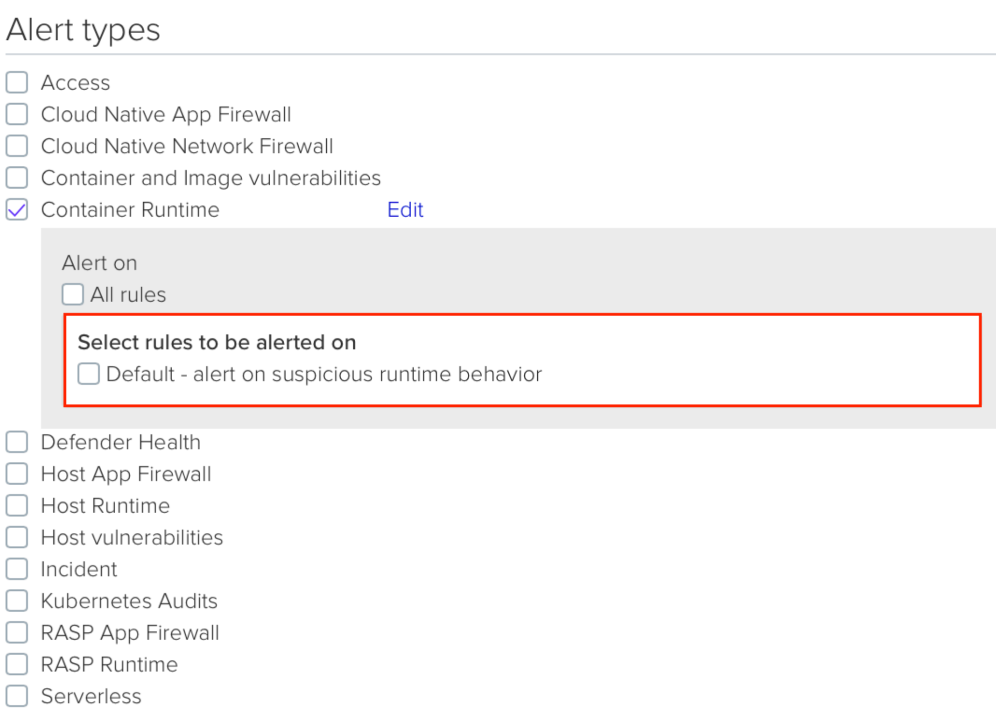
Configure the triggers

Configure how the alert is triggered.

STEP 1 | Under **Alert Types**, check the boxes types of events that should trigger an alert.

STEP 2 | For additional configuration options, click **Edit**.

STEP 3 | To specify specific rules that should trigger an alert, deselect **All rules**, and then select any individual rules.



STEP 4 | Click **Save**.

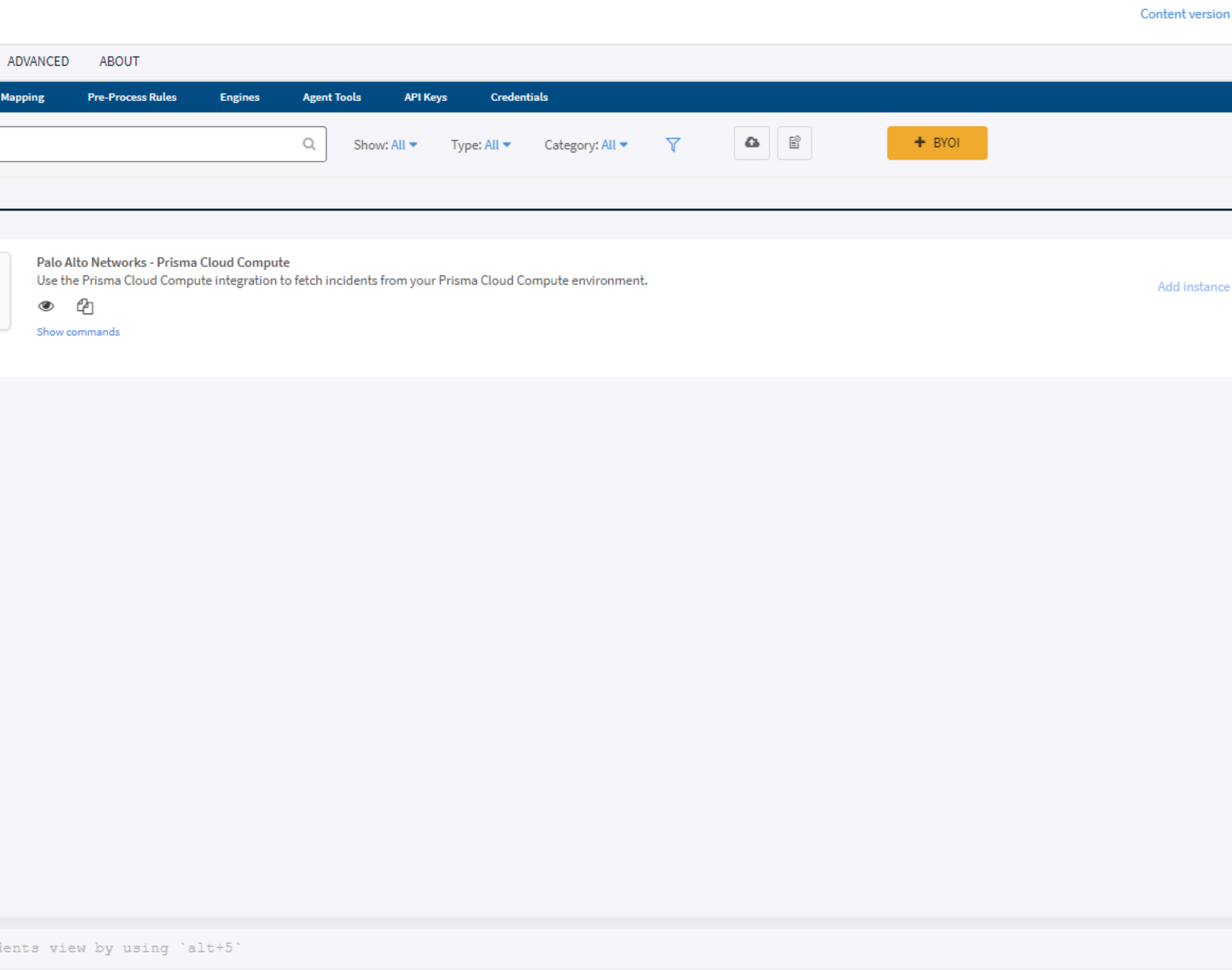
Configure XSOAR

Create a new Prisma Cloud Compute integration in XSOAR.

STEP 1 | Log into Cortex XSOAR.

STEP 2 | Go to **Settings > Integrations**.

STEP 3 | Search for **Prisma Cloud Compute** and click **Add instance**.



STEP 4 | Under the **Settings**:

1. **Name**: Enter the name for the integration.
2. Check the **Fetch incidents** checkbox.
3. **Prisma Cloud Compute Console URL and Port**: Paste the URL of the console that you copied from Prisma Cloud.
4. (optional) **Prisma Cloud Compute Project Name**: Enter the name of the project in Prisma Cloud.
5. **Credentials**: Enter the Prisma Cloud username that XSOAR should use to communicate with your Prisma Cloud console.
6. **Password**: Enter the password for the username you provided.
7. **Prisma Cloud Compute CA Certificate**: Paste the CA Certificate you copied from Prisma Cloud, or enter your own CA Certificate (if using a custom certificate to access your Prisma Cloud console).

STEP 5 | Click **Test** to check the connection to Prisma Cloud console.

STEP 6 | Click **Done** to save the integration.

STEP 7 | Go to **Incidents** to see the alerts received from Prisma Cloud.

Email alerts

Edit on GitHub

Prisma Cloud can send email alerts when your policies are violated. Audits in **Monitor > Events** are the result of a policy violation. Prisma Cloud can be configured to notify the appropriate party by email when an entire policy, or even specific rules, are violated.

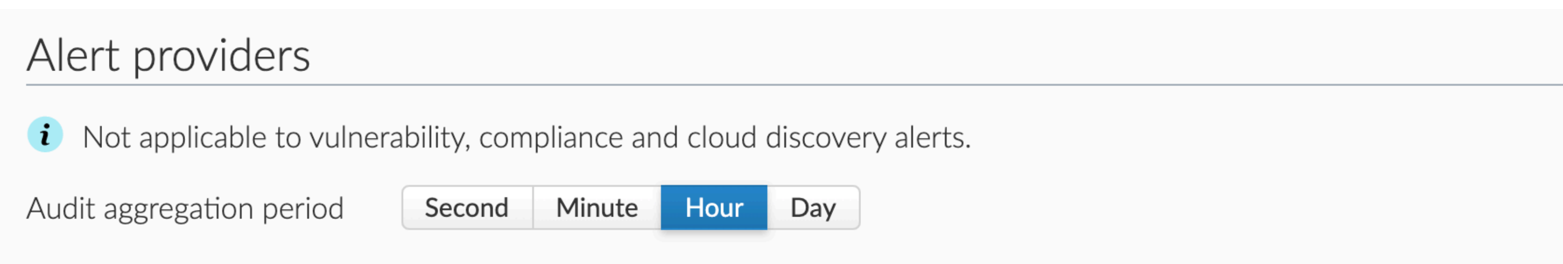
Configuring alert frequency

You can configure the rate at which alerts are emitted. This is a global setting that controls the spamminess of the alert service. Alerts received during the specified period are aggregated into a single alert. For each alert profile, an alert is sent as soon as the first matching event is received. All subsequent alerts are sent once per period.

STEP 1 | Open Console, and go to **Manage > Alerts**.

STEP 2 | In **Aggregate audits every**, specify the maximum rate that alerts should be sent.

You can specify **Second**, **Minute**, **Hour**, **Day**.

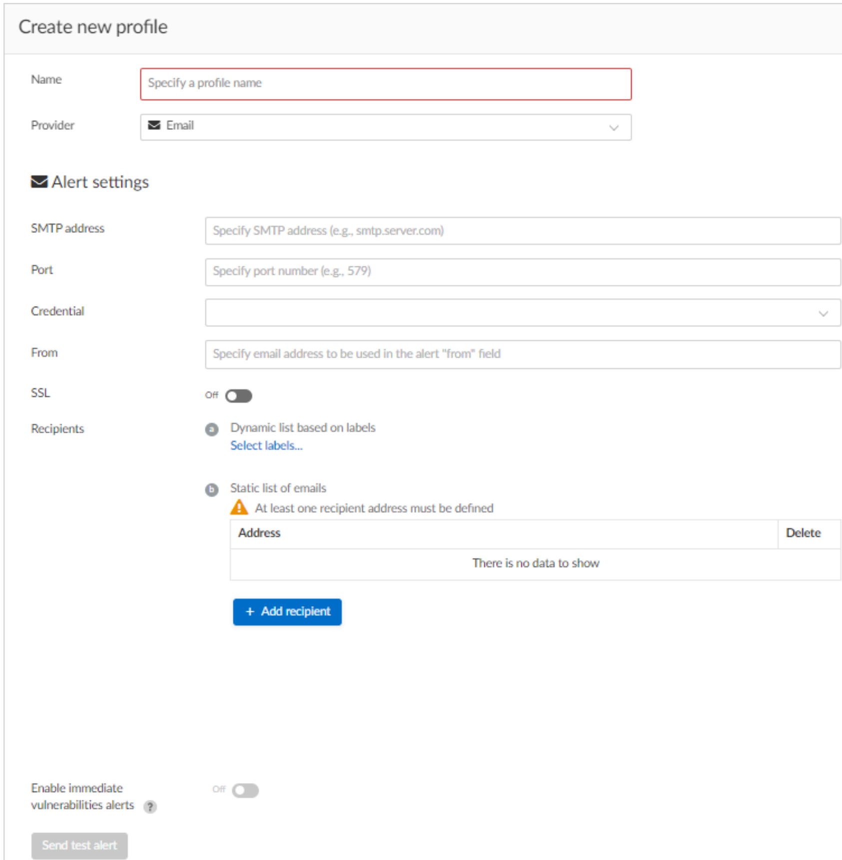


Sending email alerts

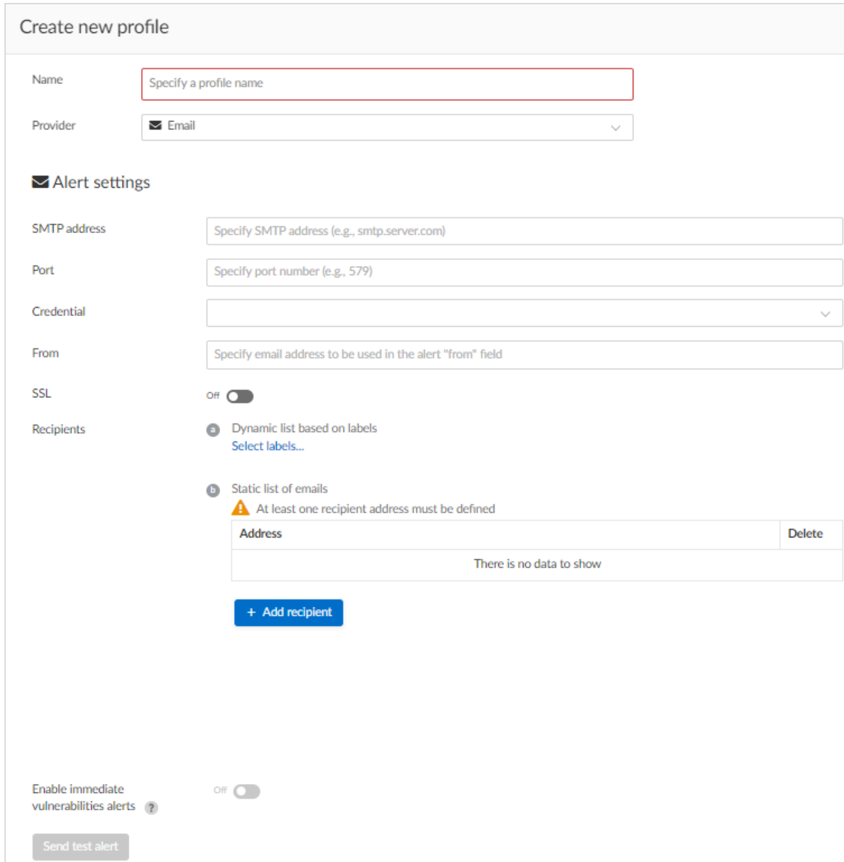
Alert profiles specify which events should trigger the alert machinery, and to which channel alerts are sent. You can send alerts to any combination of channels by creating multiple alert profiles.

Alert profiles consist of two parts:

(1) Alert settings — Who should get the alerts, and on what channel? Configure Prisma Cloud to integrate with your messaging service and specify the people or places where alerts should be sent. For example, configure the email channel and specify a list of all the email addresses where alerts should be sent. Or for JIRA, configure the project where the issue should be created, a long with the type of issue, priority, assignee, and so on.



(2) Alert triggers — Which events should trigger an alert to be sent? Specify which of the rules that make up your overall policy should trigger alerts.



If you use multi-factor authentication, you must create an exception or app-specific password to allow Console to authenticate to the service.

Create new alert profile

Create a new alert profile.

STEP 1 | In **Manage > Alerts**, click **Add profile**.

STEP 2 | Enter a name for your alert profile.

STEP 3 | In **Provider**, select **Email**.

Configure the channel

Configure the channel.

STEP 1 | In **SMTP address**, specify the hostname for your outgoing email server.

STEP 2 | In **Port**, specify the port for email submissions.

STEP 3 | In **Credential**, create the credentials required to access the email account that sends alerts. This isn't a required field.

1. Click **Add new**.
2. Select **Basic authentication**.
3. Enter a username and password.

STEP 4 | If you're using SMTPS (your SMTP connection is secured by SSL), set **SSL** to **On**.

STEP 5 | Set up your recipients.

1. Click **Add recipient**, and enter an email address. Every email alert profile must have at least one recipient, even if you're using alert labels.
2. (Optional) Specify recipients using alert labels.

STEP 6 | Click **Send Test Alert** to test the connection to your SMTP server.

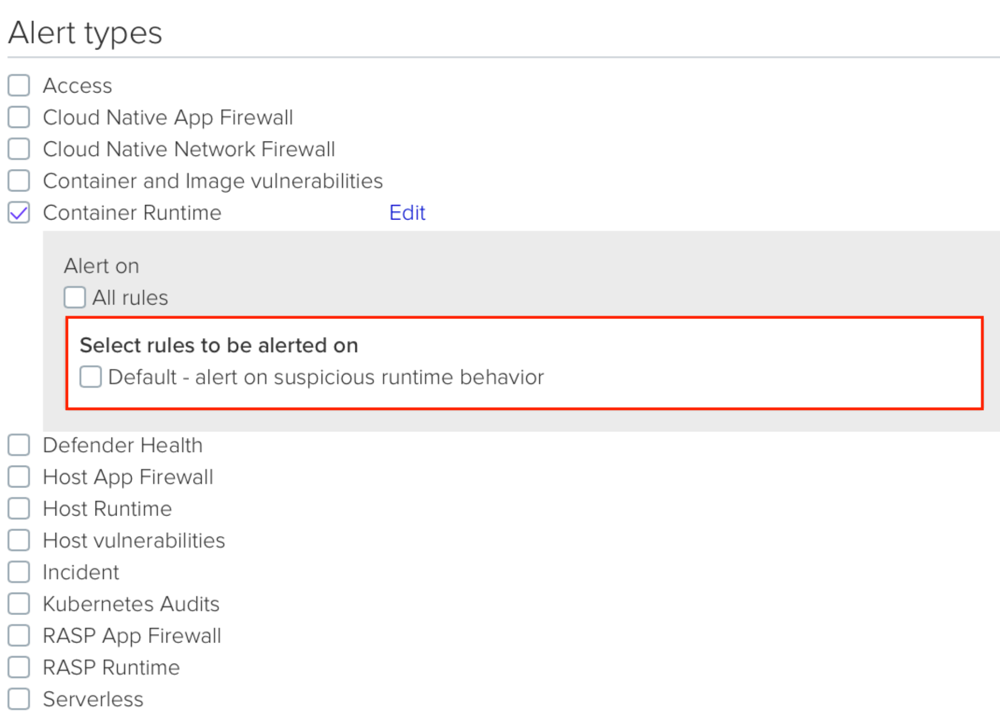
Configure the triggers

Configure how the alert is triggered.

STEP 1 | Under **Alert Types**, check the boxes types of events that should trigger an alert.

STEP 2 | For additional configuration options, click **Edit**.

STEP 3 | To specify specific rules that should trigger an alert, deselect **All rules**, and then select any individual rules.



STEP 4 | Click **Save**.

Google Cloud Pub/Sub

Edit on GitHub

Google Cloud Pub/Sub is a durable, scalable event ingestion and delivery system. It provides asynchronous messaging that decouples senders from receivers, and enables highly available communication between independently written applications.

Prisma Cloud can send alerts to Google Cloud Pub/Sub topics, where a topic is a message feed.

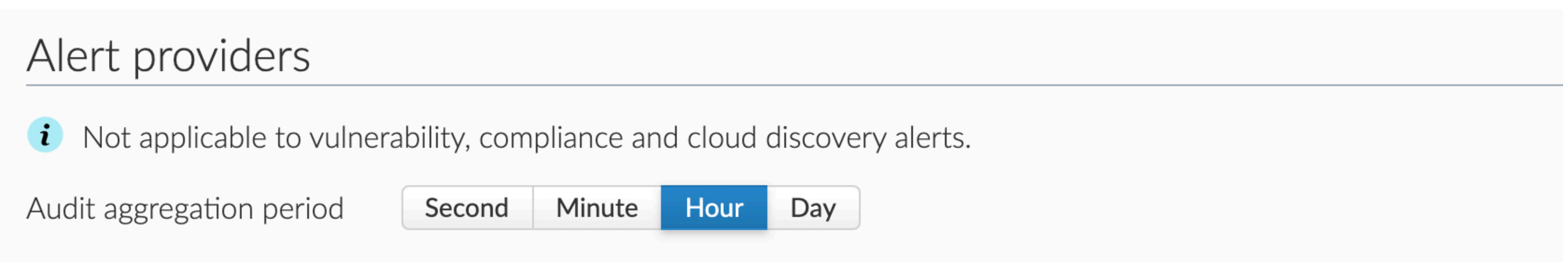
Configuring alert frequency

You can configure the rate at which alerts are emitted. This is a global setting that controls the spamminess of the alert service. Alerts received during the specified period are aggregated into a single alert. For each alert profile, an alert is sent as soon as the first matching event is received. All subsequent alerts are sent once per period.

STEP 1 | Open Console, and go to **Manage > Alerts**.

STEP 2 | In **Aggregate audits every**, specify the maximum rate that alerts should be sent.

You can specify **Second**, **Minute**, **Hour**, **Day**.



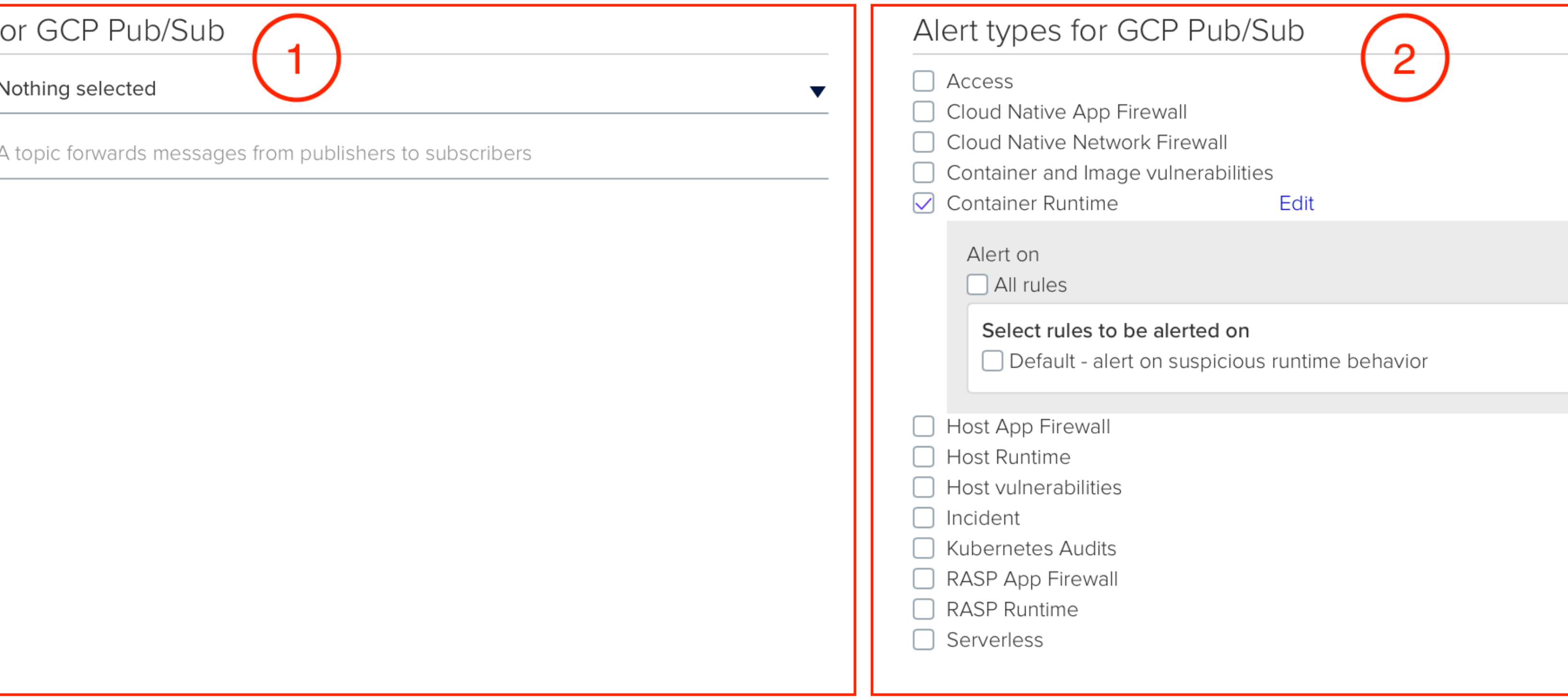
Sending alerts to Google Cloud Pub/Sub

Alert profiles specify which events should trigger the alert machinery, and to which channel alerts are sent. You can send alerts to any combination of channels by creating multiple alert profiles.

Alert profiles consist of two parts:

(1) Alert settings — Who should get the alerts, and on what channel? Configure Prisma Cloud to integrate with your messaging service and specify the people or places where alerts should be sent. For example, configure the email channel and specify a list of all the email addresses where alerts should be sent. Or for JIRA, configure the project where the issue should be created, a long with the type of issue, priority, assignee, and so on.

(2) Alert triggers — Which events should trigger an alert to be sent? Specify which of the rules that make up your overall policy should trigger alerts.



If you use multi-factor authentication, you must create an exception or app-specific password to allow Console to authenticate to the service.

Create new alert profile

Create a new alert profile.

Prerequisite: You've set up a Cloud Pub/Sub topic.

STEP 1 | In **Manage > Alerts**, click **Add profile**.

STEP 2 | Enter a name for your alert profile.

STEP 3 | In **Provider**, select **GCP Pub/Sub**.

Configure the channel

Configure the channel.

STEP 1 | In **Credential**, click **Add new** or select an existing service account.

To create a new GCP credential, see here.

STEP 2 | Enter a Cloud Pub/Sub topic.

STEP 3 | Click **Send Test Alert** to test the connection.

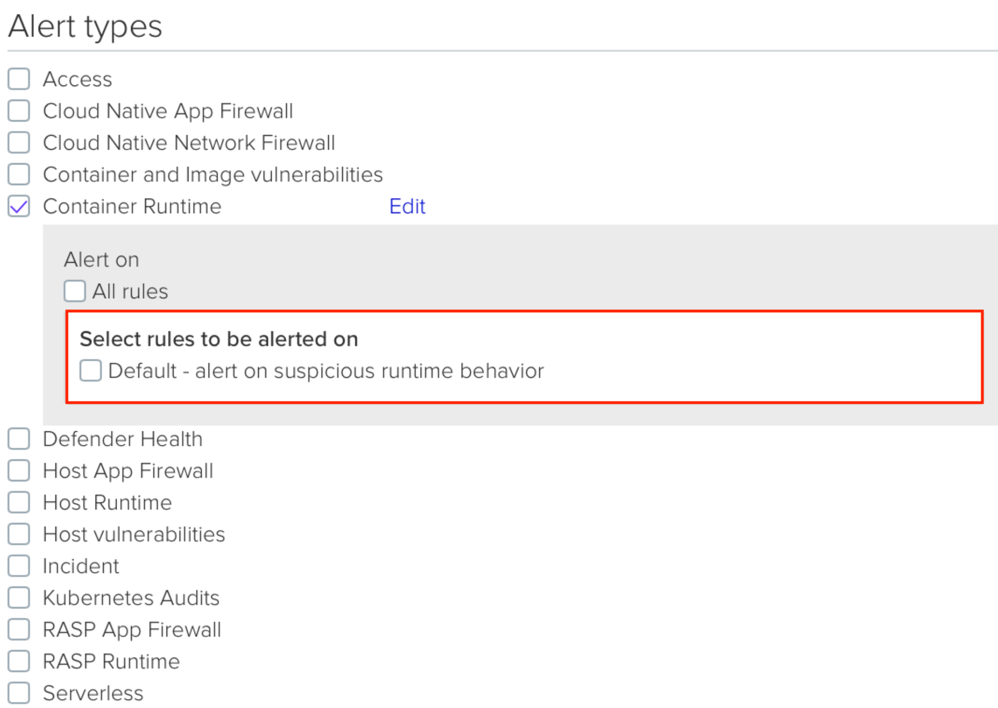
Configure the triggers

Configure how the alert is triggered.

STEP 1 | Under **Alert Types**, check the boxes types of events that should trigger an alert.

STEP 2 | For additional configuration options, click **Edit**.

STEP 3 | To specify specific rules that should trigger an alert, deselect **All rules**, and then select any individual rules.



STEP 4 | Click **Save**.

Google Cloud Security Command Center

Edit on GitHub

Prisma Cloud can be configured as a security source that provides security findings to Google Cloud Security Command Center (SCC). This lets you see all security tool findings in a single place.

Prisma Cloud is a registered Google Cloud Platform Marketplace partner.

Configuring Google Cloud Security Command Center

In Google Cloud Platform (GCP), create a service account in your project that has the **Cloud Security Command Center API** enabled. You will need the service account keys, API, and Organization ID to enable this feature.

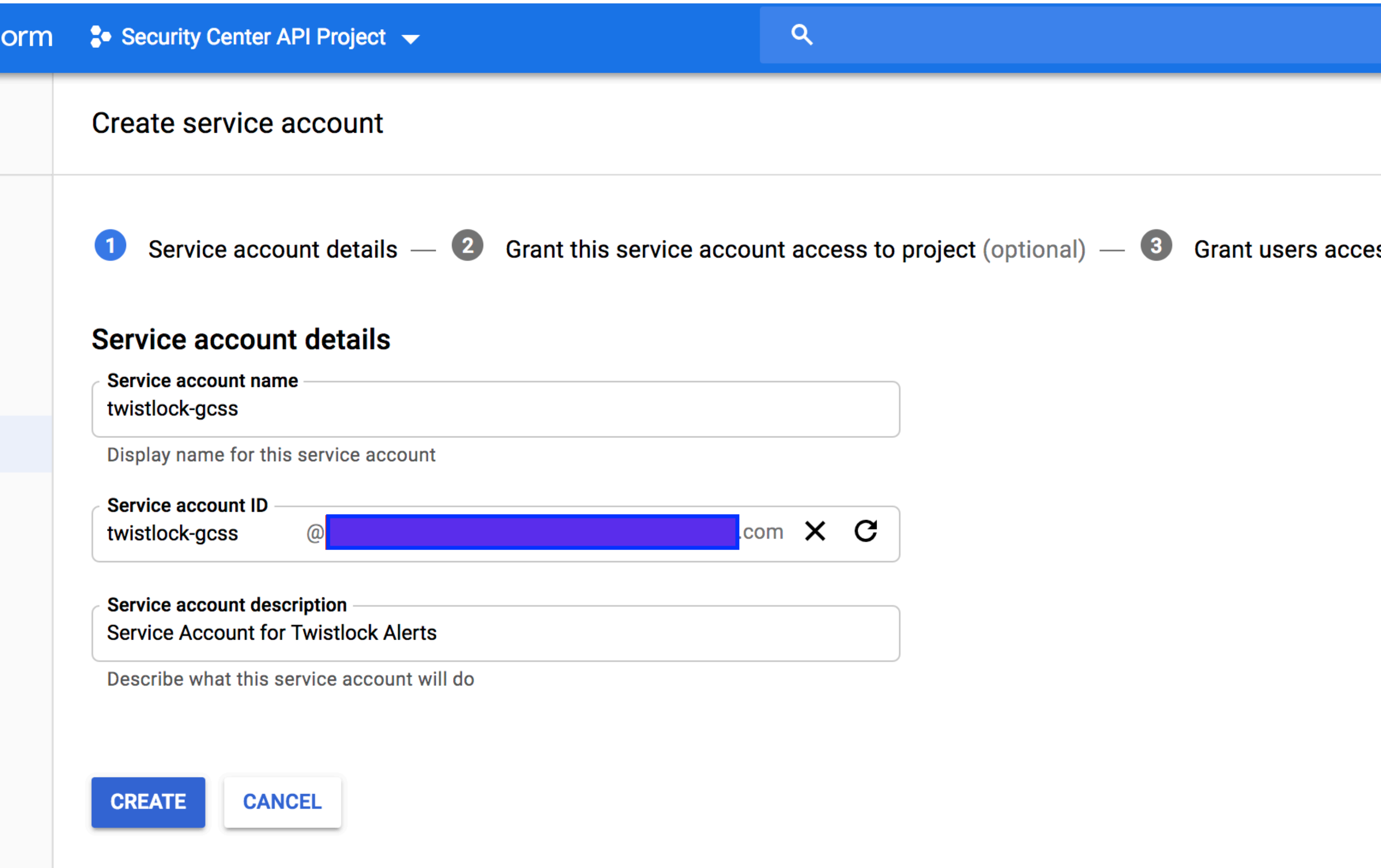
You should have already enabled and onboarded Prisma Cloud as a Security Source in Google Security Command Center. Prisma Cloud supports the alpha and beta versions of Google Security Command Center. The following instructions show how to configure the beta version.

STEP 1 | Log into your GCP tenant and select the project that has the Cloud Security Command Center API enabled.

STEP 2 | Go to **IAM & admin > Service accounts**.

STEP 3 | Click **Create Service Account**.

STEP 4 | Enter a name and description for the service account.



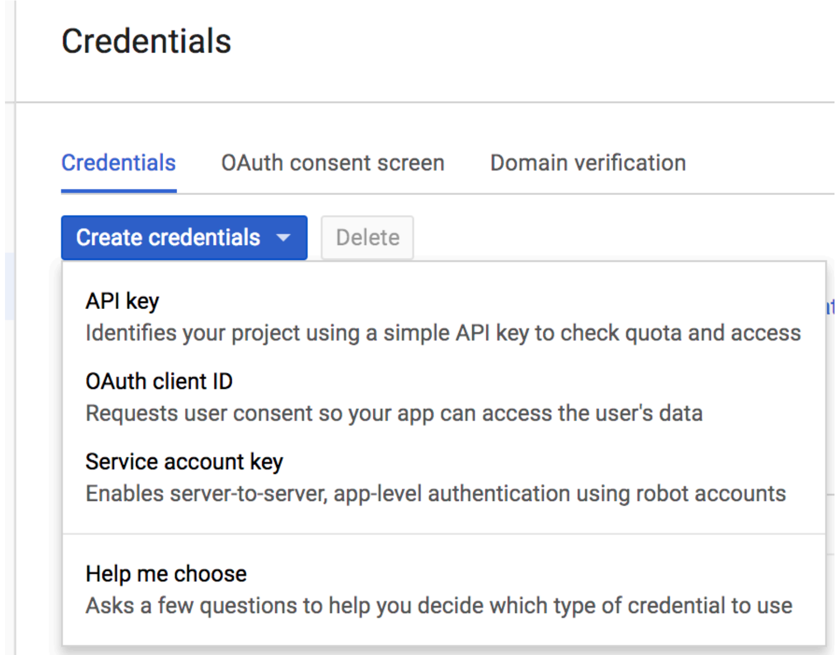
STEP 5 | **Grant this service account access to project (optional)** click **continue**. Do not grant a role to the account at this time.

STEP 6 | **Grant user account to this service account** click **create key**.

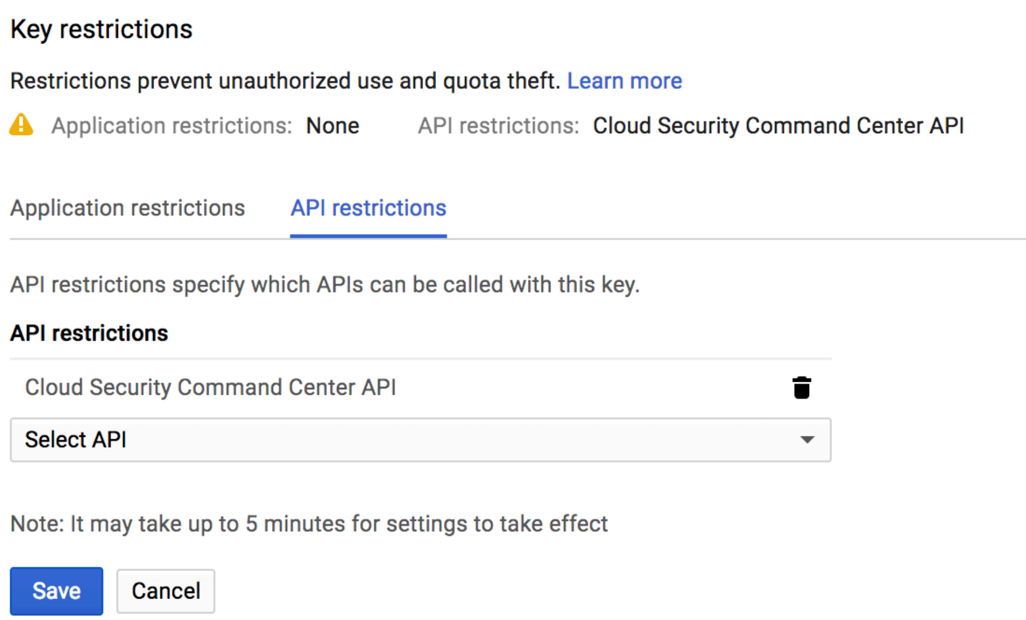
STEP 7 | Set key type to **JSON**, and click **create**. Save the downloaded JSON key.

STEP 8 | Go to the project's **APIs & Services > Credentials**.

STEP 9 | Click **Create credentials > API key**.



STEP 10 | Save the API key. We recommended that you restrict the key to the **Cloud Security Command Center API**.



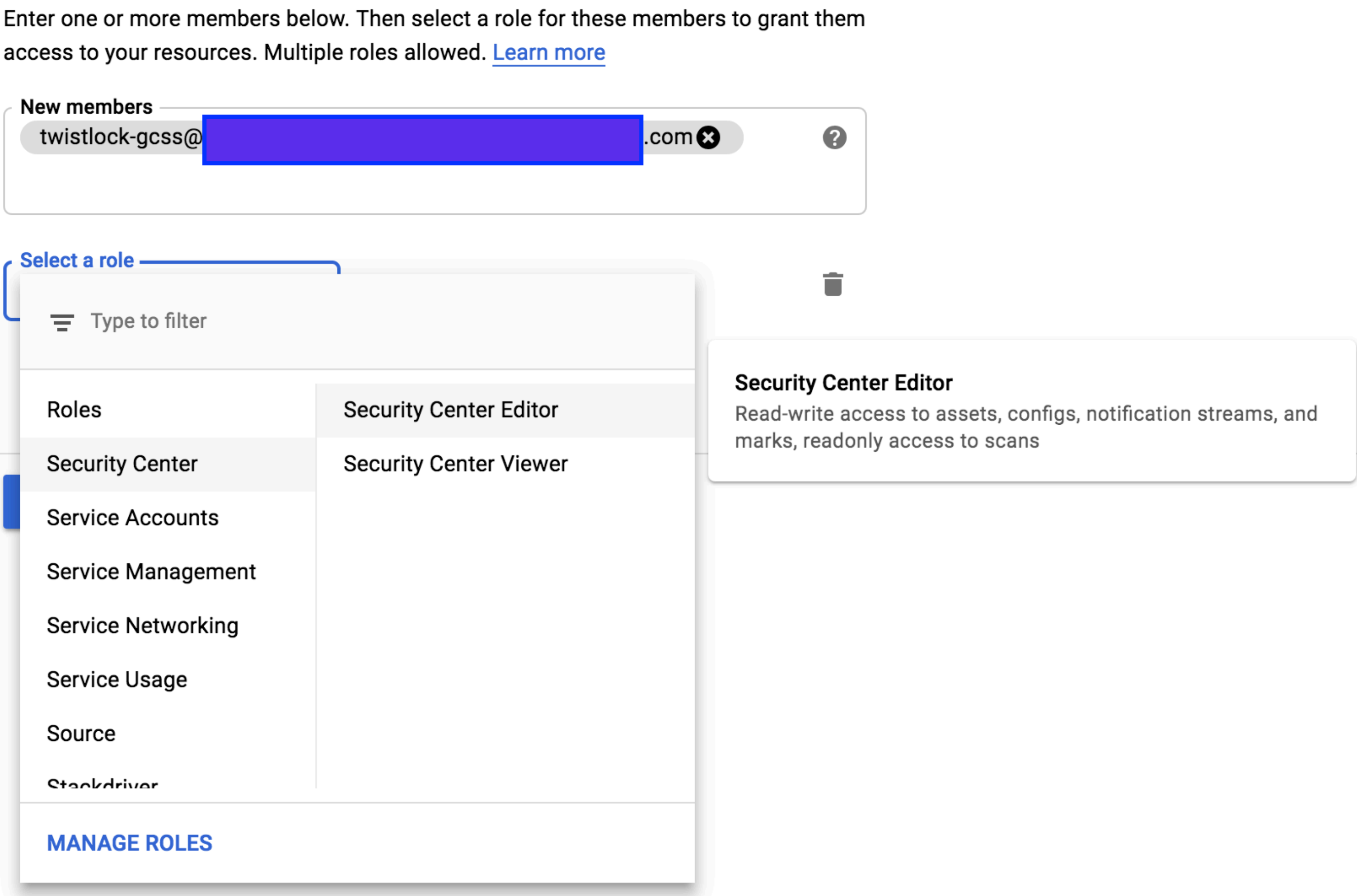
STEP 11 | Go to the Google tenant's organizational **IAM & admin**.

This setting is configured at the organizational level, not the project level.

STEP 12 | In the **IAM** window click **+Add**.

STEP 13 | Paste in the name of the service account that has been created.

STEP 14 | Select Role: **Security Center > Security Center Editor**.



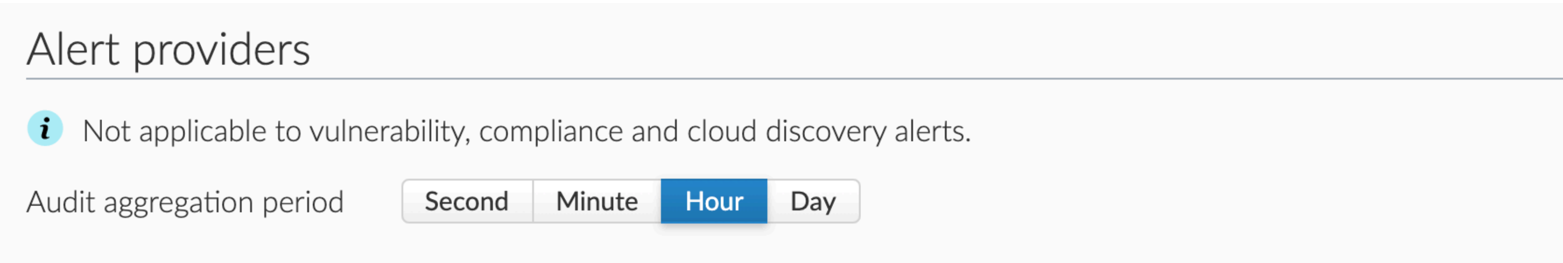
Configuring alert frequency

You can configure the rate at which alerts are emitted. This is a global setting that controls the spamminess of the alert service. Alerts received during the specified period are aggregated into a single alert. For each alert profile, an alert is sent as soon as the first matching event is received. All subsequent alerts are sent once per period.

STEP 1 | Open Console, and go to **Manage > Alerts**.

STEP 2 | In **Aggregate audits every**, specify the maximum rate that alerts should be sent.

You can specify **Second**, **Minute**, **Hour**, **Day**.



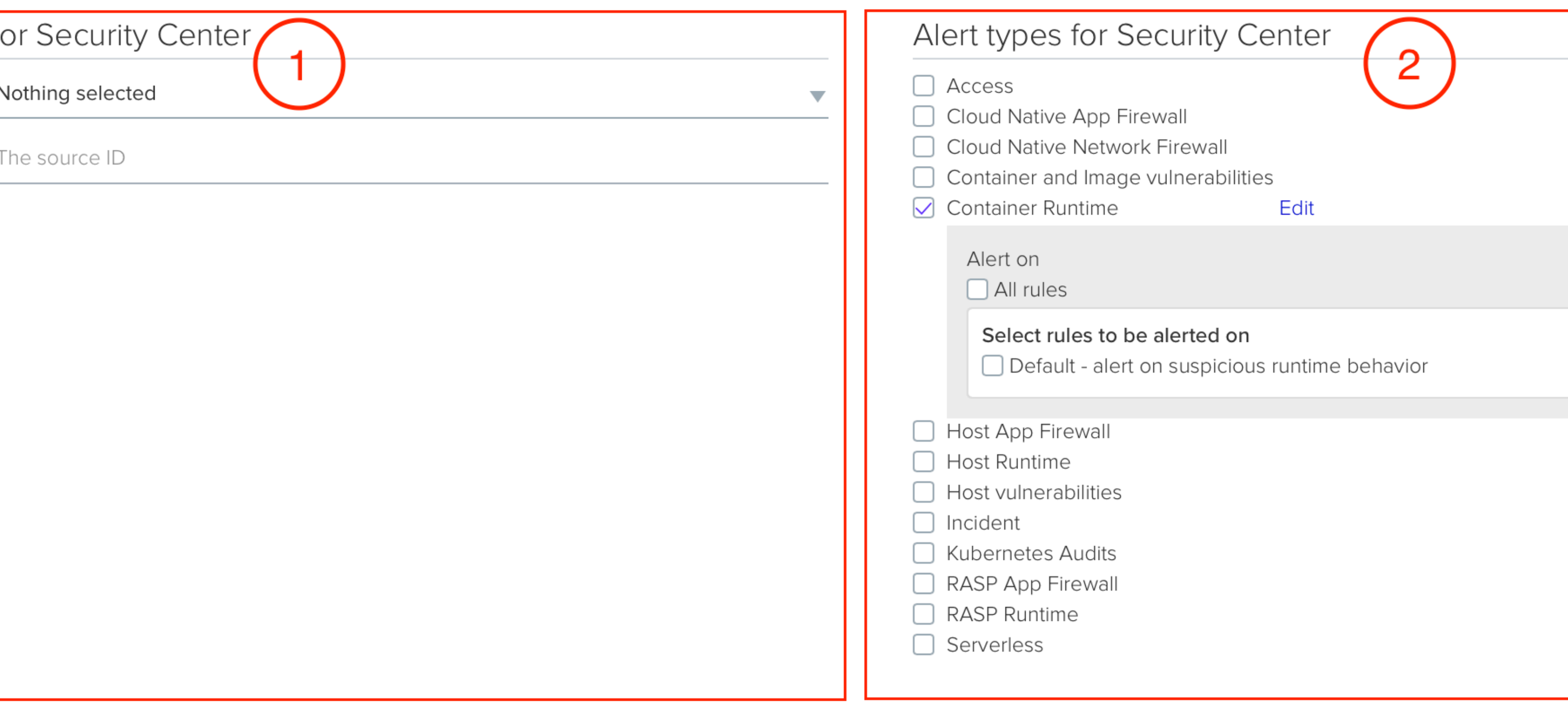
Sending alerts to Google Cloud SCC

Alert profiles specify which events should trigger the alert machinery, and to which channel alerts are sent. You can send alerts to any combination of channels by creating multiple alert profiles.

Alert profiles consist of two parts:

(1) Alert settings — Who should get the alerts, and on what channel? Configure Prisma Cloud to integrate with your messaging service and specify the people or places where alerts should be sent. For example, configure the email channel and specify a list of all the email addresses where alerts should be sent. Or for JIRA, configure the project where the issue should be created, a long with the type of issue, priority, assignee, and so on.

(2) Alert triggers — Which events should trigger an alert to be sent? Specify which of the rules that make up your overall policy should trigger alerts.



If you use multi-factor authentication, you must create an exception or app-specific password to allow Console to authenticate to the service.

Create new alert profile

Create a new alert profile.

STEP 1 | In **Manage > Alerts**, click **Add profile**.

STEP 2 | Enter a name for your alert profile.

STEP 3 | In **Provider**, select **Security Center**.

Configure the channel

Configure the channel.

STEP 1 | In **Credential**, click **Add new** or select an existing service account.

To create a new GCP credential, see here.

STEP 2 | In **Source Name**, enter the resource path for a source that's already been created.

The source name has the following format:

```
organizations/<organization_id>/sources/<source_id>
```

Where organization_id and source_id are numeric identifiers. For example:

```
organizations/111122222444/sources/43211234
```

STEP 3 | Click **Send Test Alert** to test the connection.

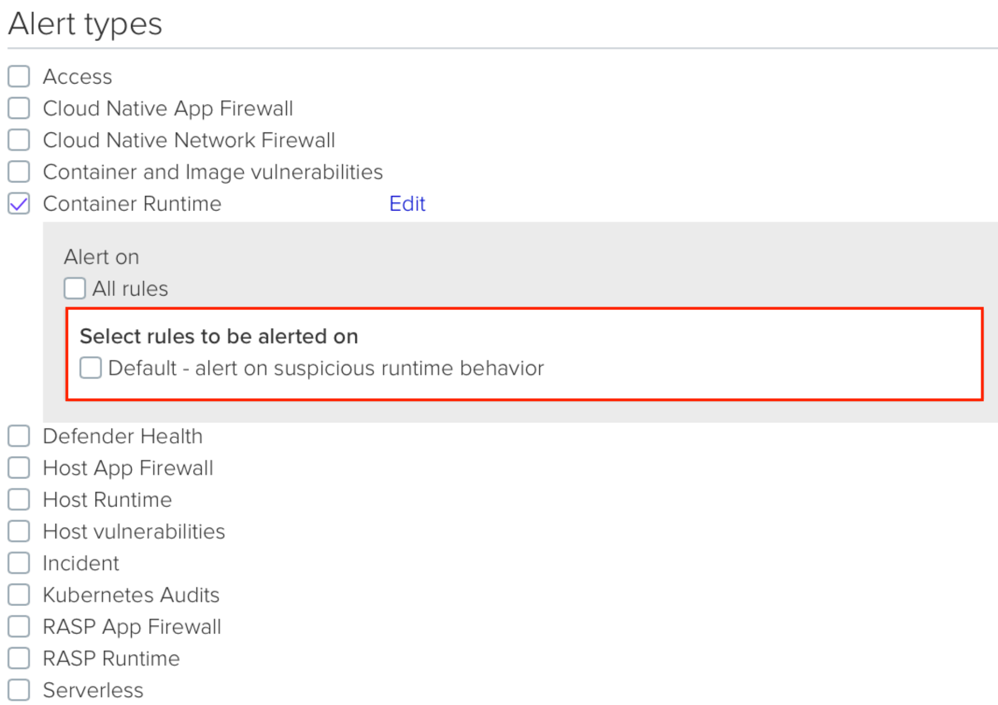
Configure the triggers

Configure how the alert is triggered.

STEP 1 | Under **Alert Types**, check the boxes types of events that should trigger an alert.

STEP 2 | For additional configuration options, click **Edit**.

STEP 3 | To specify specific rules that should trigger an alert, deselect **All rules**, and then select any individual rules.



STEP 4 | Click **Save**.

IBM Cloud Security Advisor

Edit on GitHub

IBM Cloud Security Advisor is a centralized security dashboard. Prisma Cloud can be configured to send security findings to your service dashboard.

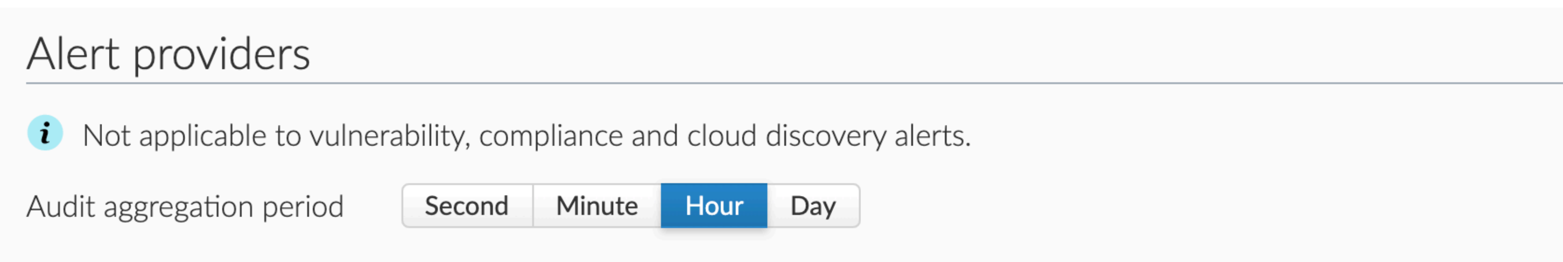
Configuring alert frequency

You can configure the rate at which alerts are emitted. This is a global setting that controls the spamminess of the alert service. Alerts received during the specified period are aggregated into a single alert. For each alert profile, an alert is sent as soon as the first matching event is received. All subsequent alerts are sent once per period.

STEP 1 | Open Console, and go to **Manage > Alerts**.

STEP 2 | In **Aggregate audits every**, specify the maximum rate that alerts should be sent.

You can specify **Second**, **Minute**, **Hour**, **Day**.



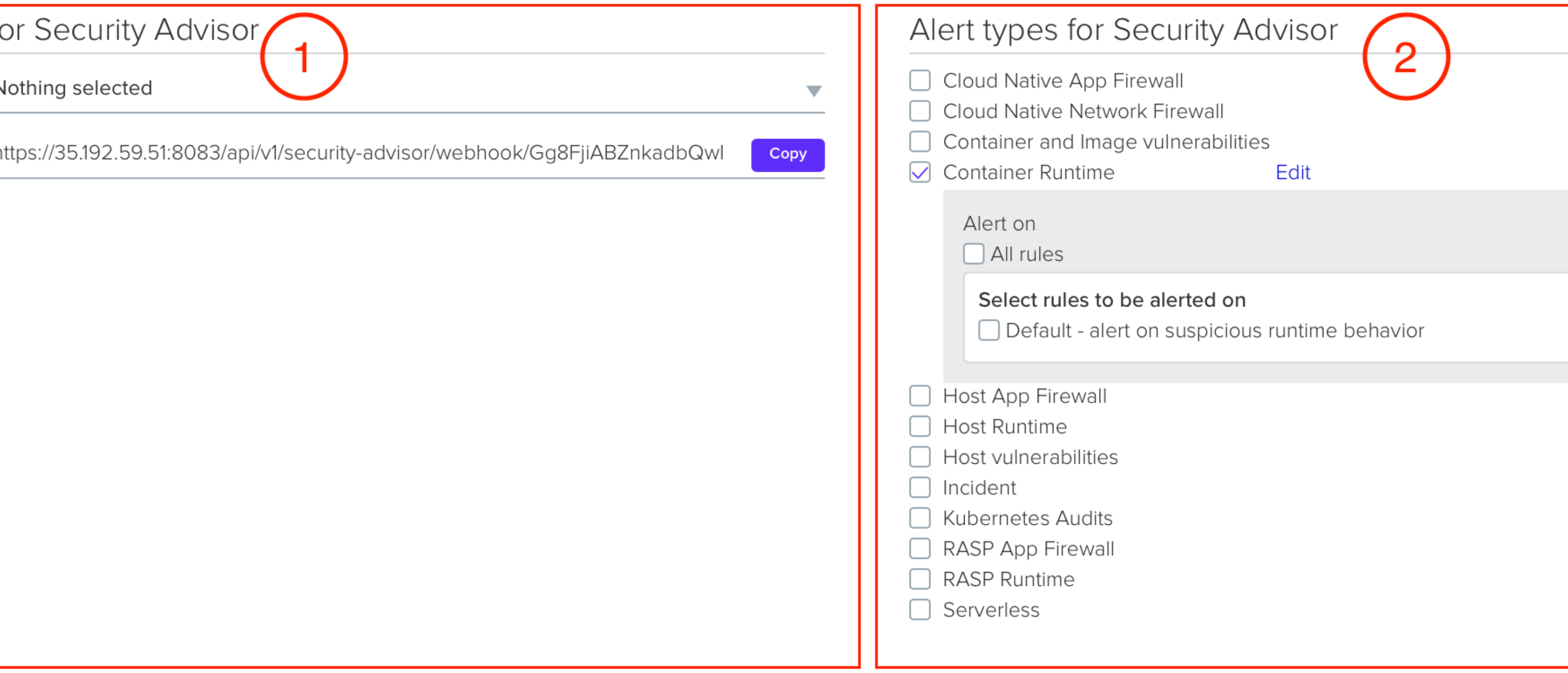
Sending alerts to Security Advisor

Alert profiles specify which events should trigger the alert machinery, and to which channel alerts are sent. You can send alerts to any combination of channels by creating multiple alert profiles.

Alert profiles consist of two parts:

(1) Alert settings — Who should get the alerts, and on what channel? Configure Prisma Cloud to integrate with your messaging service and specify the people or places where alerts should be sent. For example, configure the email channel and specify a list of all the email addresses where alerts should be sent. Or for JIRA, configure the project where the issue should be created, a long with the type of issue, priority, assignee, and so on.

(2) Alert triggers — Which events should trigger an alert to be sent? Specify which of the rules that make up your overall policy should trigger alerts.



If you use multi-factor authentication, you must create an exception or app-specific password to allow Console to authenticate to the service.

Create new alert profile

Create a new alert profile.

STEP 1 | In **Manage > Alerts**, click **Add profile**.

STEP 2 | Enter a name for your alert profile.

STEP 3 | In **Provider**, select **IBM Cloud Security Advisor**.

Configure the channel

Configure the channel.

STEP 1 | In **Credential**, click **Add new** or select an existing service account.

To create a new IBM Cloud credential, see here.

STEP 2 | Copy the configuration URL, and set it aside. You’ll need it for the next step.

STEP 3 | Go to the Security Advisor dashboard, and then follow the steps in Prisma Cloud partner integration to complete the setup process.

STEP 4 | Click **Send Test Alert** to test the connection.

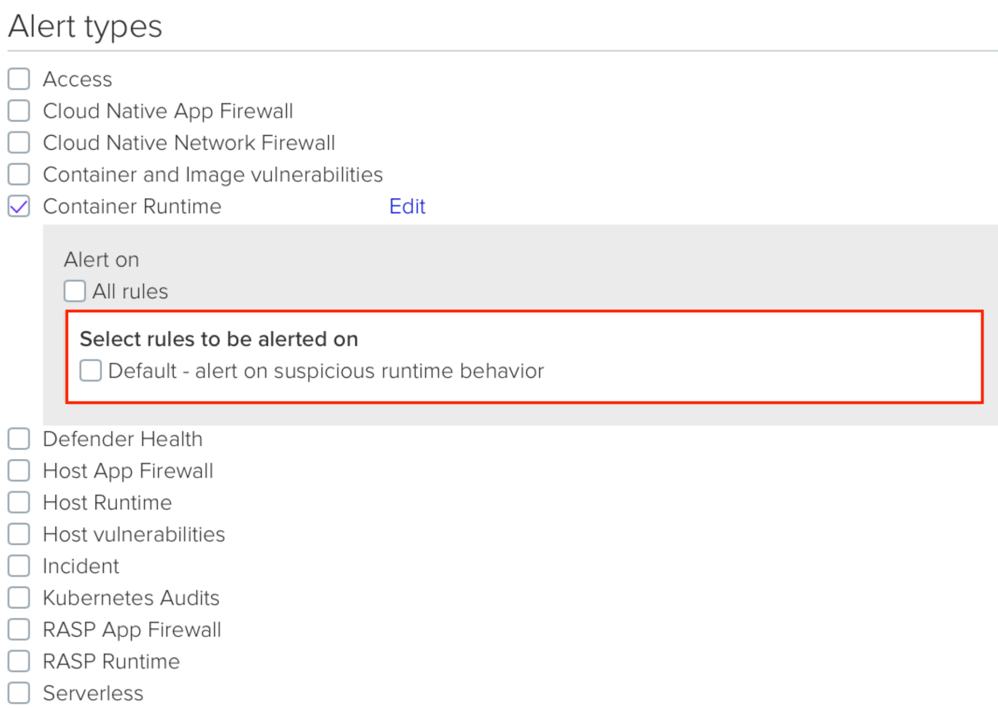
Configure the triggers

Configure how the alert is triggered.

STEP 1 | Under **Alert Types**, check the boxes types of events that should trigger an alert.

STEP 2 | For additional configuration options, click **Edit**.

STEP 3 | To specify specific rules that should trigger an alert, deselect **All rules**, and then select any individual rules.



STEP 4 | Click **Save**.

JIRA Alerts

Edit on GitHub

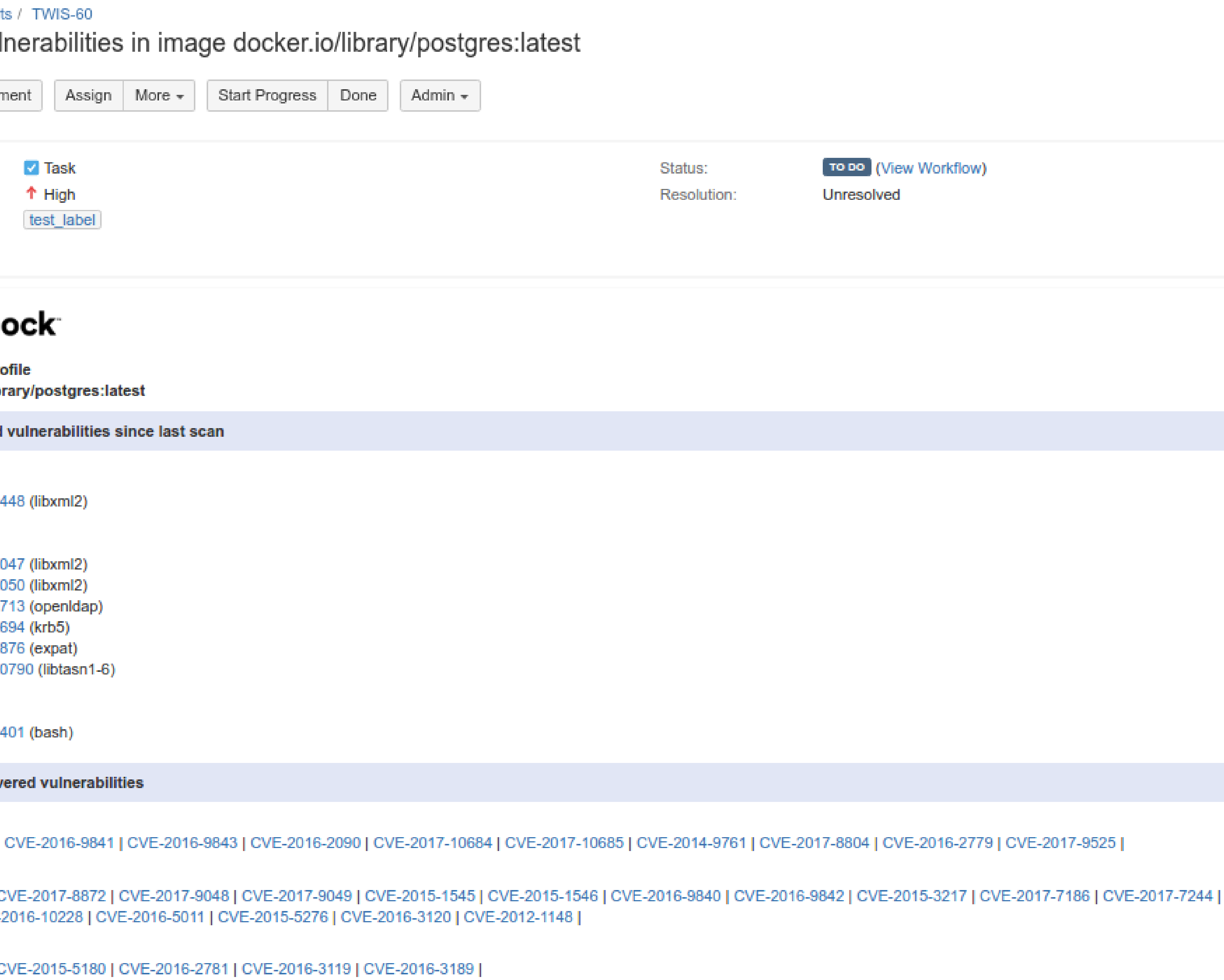
Prisma Cloud continually scans your environment for vulnerabilities using the threat data in the Intelligence Stream. Prisma Cloud can open JIRA issues when new vulnerabilities are detected in your environment. This mechanism lets you implement continuous vulnerability assessment and remediation by hooking directly into the developer's workflow.

New JIRA issues are opened when new vulnerabilities are found. Issues are opened on a per-image basis. Each JIRA issue lists the new vulnerabilities discovered, and a list of vulnerabilities that have already been reported but were still detected.

JIRA issues are opened based on policy. For example, an issue would be created when all of the following conditions are met:

- You have a rule that alerts on critical vulnerabilities,
- The rule is associated with your JIRA alert profile,
- The Prisma Cloud scanner finds a critical vulnerability in an image in your environment.

The following screenshot shows an example JIRA issue opened by Prisma Cloud.



Intelligent issue routing

You can leverage image labels to intelligently route alerts to the right team, and eliminate manual ticket triage. For example, if team-a is responsible for image-a, and a vulnerability is found in image-a, you could set up the alert to flow directly to team-a's JIRA queue.

Intelligent routing depends on a Prisma Cloud feature called alert labels, where you define labels that Prisma Cloud should watch. When rules trigger, Prisma Cloud extracts the value of the label from the resource, and applies it to the next phase of alert processing. For JIRA alerts, you can use labels to specify the JIRA project key, JIRA labels, and JIRA issue assignee.

For example, if you have an image with the following labels:

```
group=front-end-group
team=client-team
business-app=my-business-app
```

You could configure Prisma Cloud to open issues about this specific image in the JIRA project defined by the *group* label.

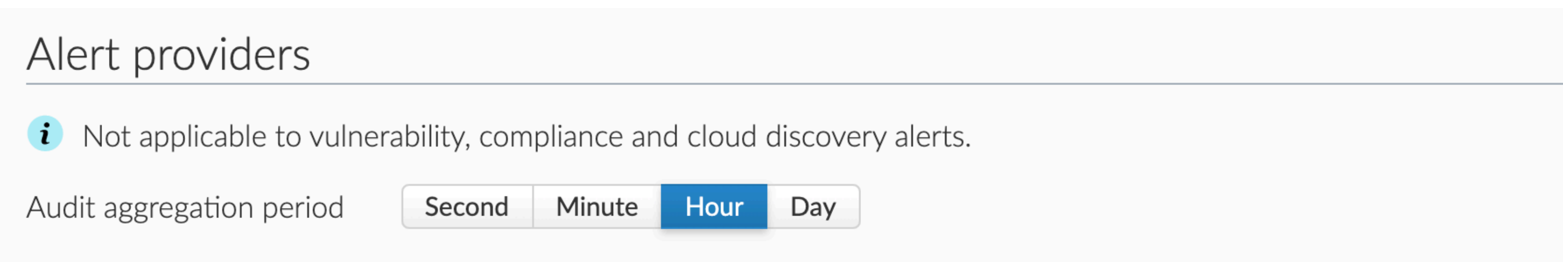
Configuring alert frequency

You can configure the rate at which alerts are emitted. This is a global setting that controls the spamminess of the alert service. Alerts received during the specified period are aggregated into a single alert. For each alert profile, an alert is sent as soon as the first matching event is received. All subsequent alerts are sent once per period.

STEP 1 | Open Console, and go to **Manage > Alerts**.

STEP 2 | In **Aggregate audits every**, specify the maximum rate that alerts should be sent.

You can specify **Second**, **Minute**, **Hour**, **Day**.



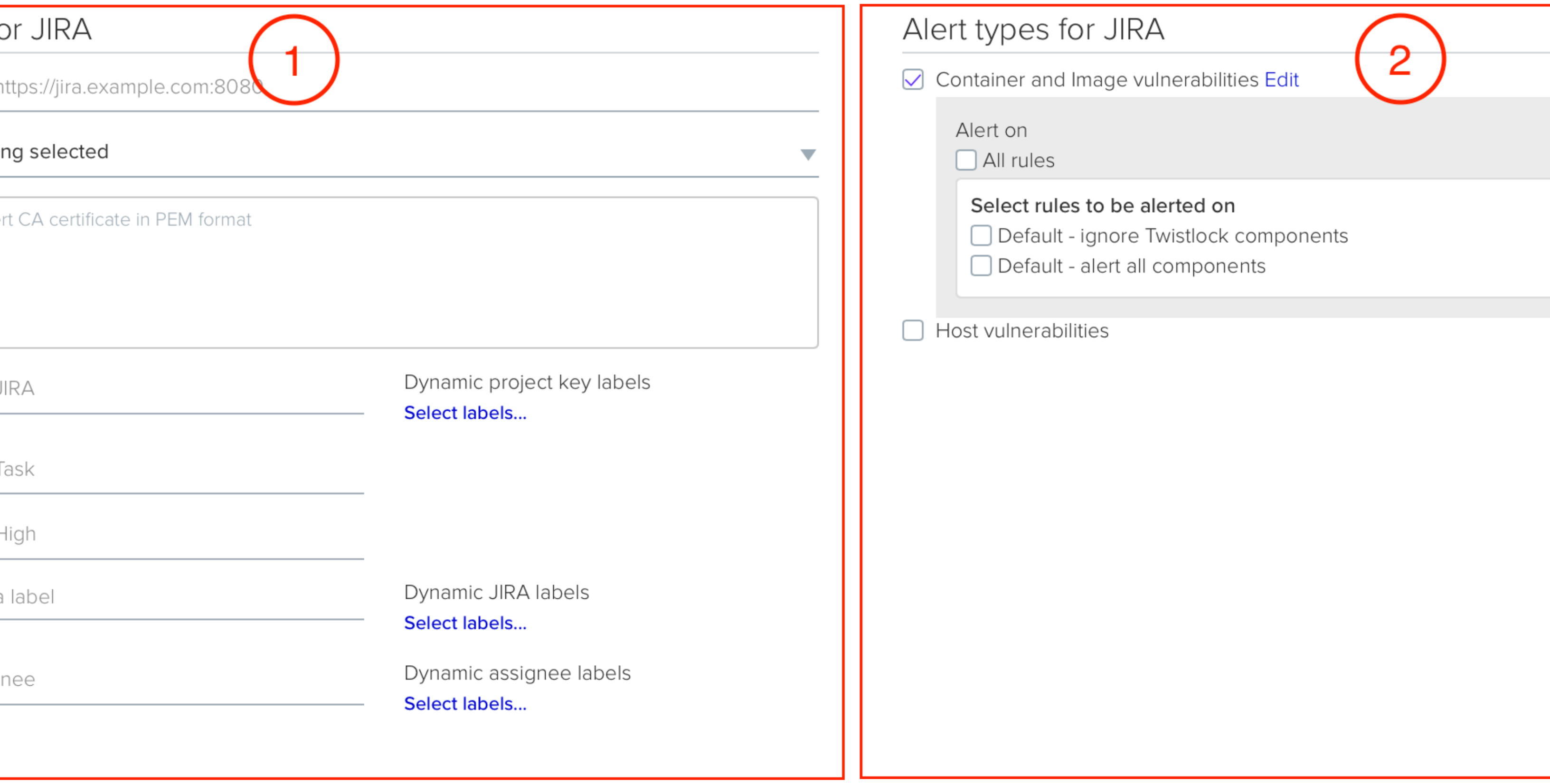
Integrating Prisma Cloud with JIRA

Alert profiles specify which events should trigger the alert machinery, and to which channel alerts are sent. You can send alerts to any combination of channels by creating multiple alert profiles.

Alert profiles consist of two parts:

(1) Alert settings — Who should get the alerts, and on what channel? Configure Prisma Cloud to integrate with your messaging service and specify the people or places where alerts should be sent. For example, configure the email channel and specify a list of all the email addresses where alerts should be sent. Or for JIRA, configure the project where the issue should be created, a long with the type of issue, priority, assignee, and so on.

(2) Alert triggers — Which events should trigger an alert to be sent? Specify which of the rules that make up your overall policy should trigger alerts.



If you use multi-factor authentication, you must create an exception or app-specific password to allow Console to authenticate to the service.

Create new alert profile

Create a new alert profile.

STEP 1 | In **Manage > Alerts**, click **Add profile**.

STEP 2 | Enter a name for your alert profile.

STEP 3 | In **Provider**, select **JIRA**.

Configure the channel

Configure the channel.

STEP 1 | In **Base URL**, specify the location of your JIRA service.

STEP 2 | In **Credential**, create the credentials required to access the account.

1. Click **Add new**.
2. Select **Basic authentication**.
3. Enter a username and password.

   If you are using Jira Cloud, this will be an email address and API token respectively. You can generate your API token here.

4. Click **Save**.

STEP 3 | In **CA certificate**, enter a copy of the CA certificate in PEM format.

STEP 4 | In **Project key**, enter a project key.

Alternatively, you can dynamically specify the project key based on a label. When an alert fires, the project key is taken from the label of the resource that triggered the action. To do so, click **Select labels...**, and choose a label that you know will contain the project key. If there are no labels in the drop-down list, go to **Manage > Alerts > Alert Labels**, and define them.

STEP 5 | Enter an issue type.

STEP 6 | Enter a priority.

STEP 7 | Enter a comma delimited list of JIRA labels to apply to the issue.

You can dynamically define the list from a label. Click **Select labels...**, and select one or more labels.

STEP 8 | Enter an assignee for the new issue.

You can dynamically define the assignee from a label. Click **Select labels...**, and select one or more labels.

STEP 9 | Click **Send Test Alert** to test the connection.

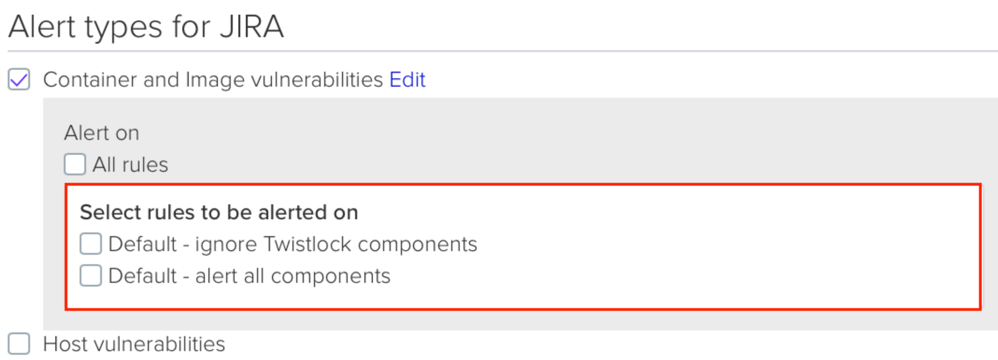
Configure the triggers

Configure how the alert is triggered.

STEP 1 | Under **Alert Types**, check the boxes types of events that should trigger an alert.

STEP 2 | For additional configuration options, click **Edit**.

STEP 3 | To specify specific rules that should trigger an alert, deselect **All rules**, and then select any individual rules.



STEP 4 | Click **Save**.

PagerDuty alerts

Edit on GitHub

You can configure Prisma Cloud to route alerts to PagerDuty. When Prisma Cloud detects anomalies, it generates alerts. Alerts are raised when the rules that make up your policy are violated.

Configuring PagerDuty

Create a new Prisma Cloud service, and get an integration key.

STEP 1 | Log into PagerDuty.

STEP 2 | Go to **Configuration** > **Services**.

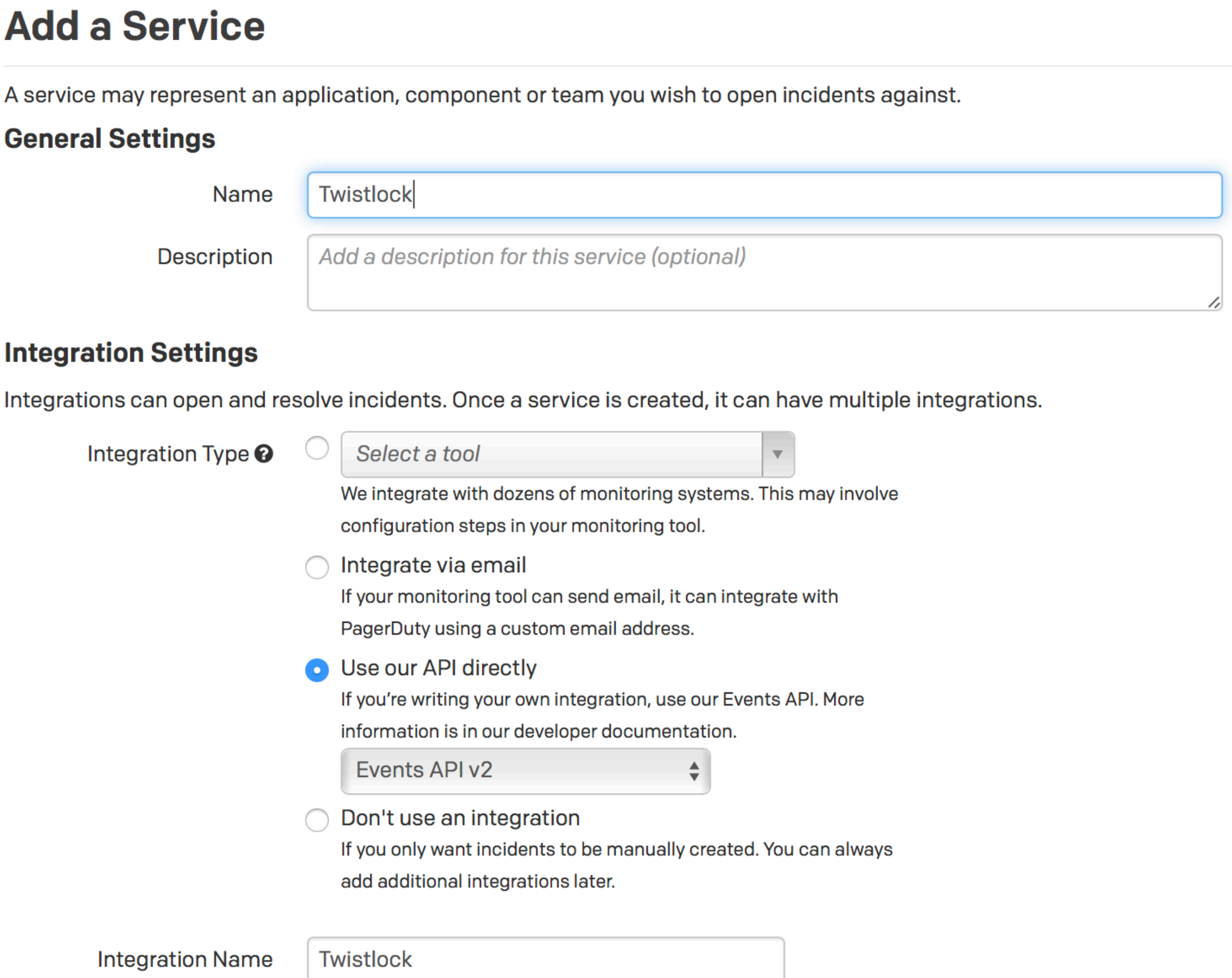
STEP 3 | Click **New Service**.

STEP 4 | Under **General Settings**:

1. **Name**: Enter **Prisma Cloud**.

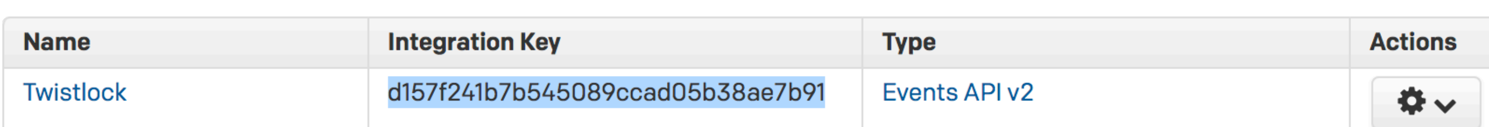
STEP 5 | Under **Integration Settings**:

1. **Integration Type**: Select **Use our API directly**, the select **Events API v2**.
2. **Integration Name**: Enter **Prisma Cloud**.



STEP 6 | Click **Add Service**. You're taken to **Integrations** tab for the Prisma Cloud service.

STEP 7 | Copy the **Integration Key**, and set it aside. You'll use it to configure the integration in Prisma Cloud Console.



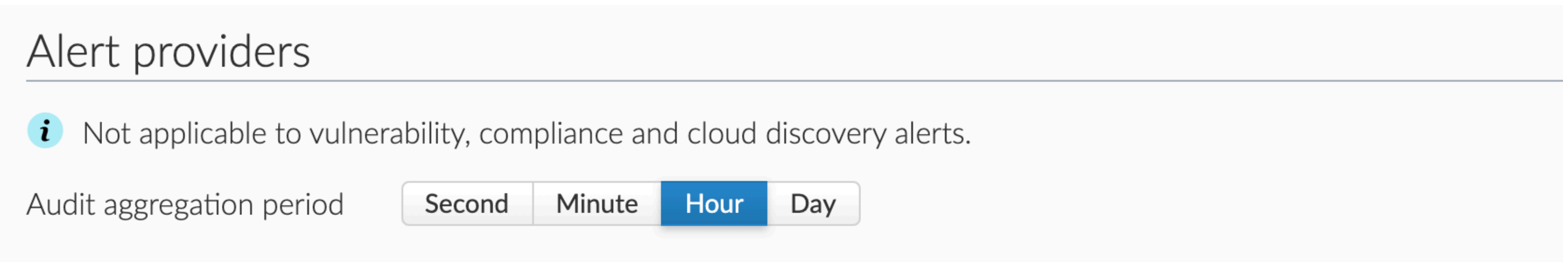
Configuring alert frequency

You can configure the rate at which alerts are emitted. This is a global setting that controls the spamminess of the alert service. Alerts received during the specified period are aggregated into a single alert. For each alert profile, an alert is sent as soon as the first matching event is received. All subsequent alerts are sent once per period.

STEP 1 | Open Console, and go to **Manage > Alerts**.

STEP 2 | In **Aggregate audits every**, specify the maximum rate that alerts should be sent.

You can specify **Second**, **Minute**, **Hour**, **Day**.



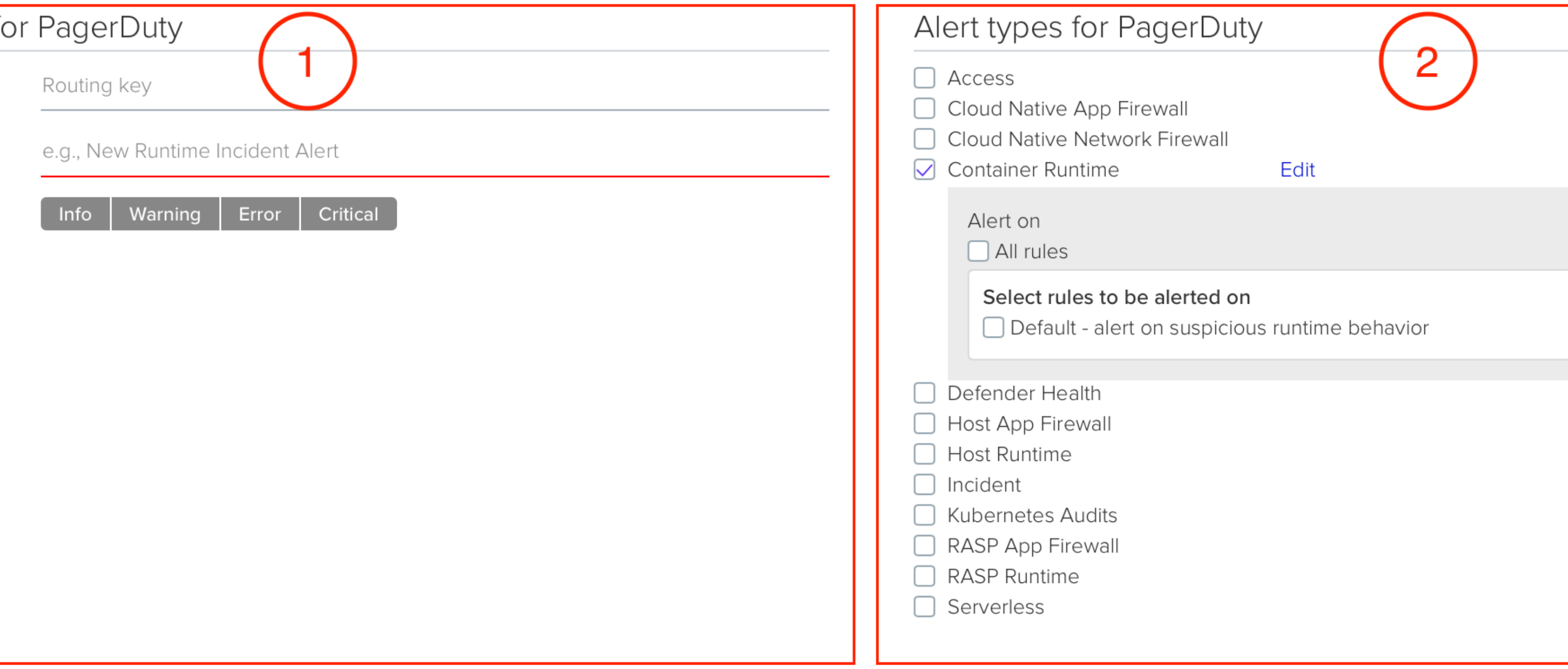
Sending alerts to PagerDuty

Alert profiles specify which events should trigger the alert machinery, and to which channel alerts are sent. You can send alerts to any combination of channels by creating multiple alert profiles.

Alert profiles consist of two parts:

(1) Alert settings — Who should get the alerts, and on what channel? Configure Prisma Cloud to integrate with your messaging service and specify the people or places where alerts should be sent. For example, configure the email channel and specify a list of all the email addresses where alerts should be sent. Or for JIRA, configure the project where the issue should be created, a long with the type of issue, priority, assignee, and so on.

(2) Alert triggers — Which events should trigger an alert to be sent? Specify which of the rules that make up your overall policy should trigger alerts.



If you use multi-factor authentication, you must create an exception or app-specific password to allow Console to authenticate to the service.

Create new alert profile

Create a new alert profile.

STEP 1 | In **Manage > Alerts**, click **Add profile**.

STEP 2 | Enter a name for your alert profile.

STEP 3 | In **Provider**, select **PagerDuty**.

Configure the channel

Configure the channel.

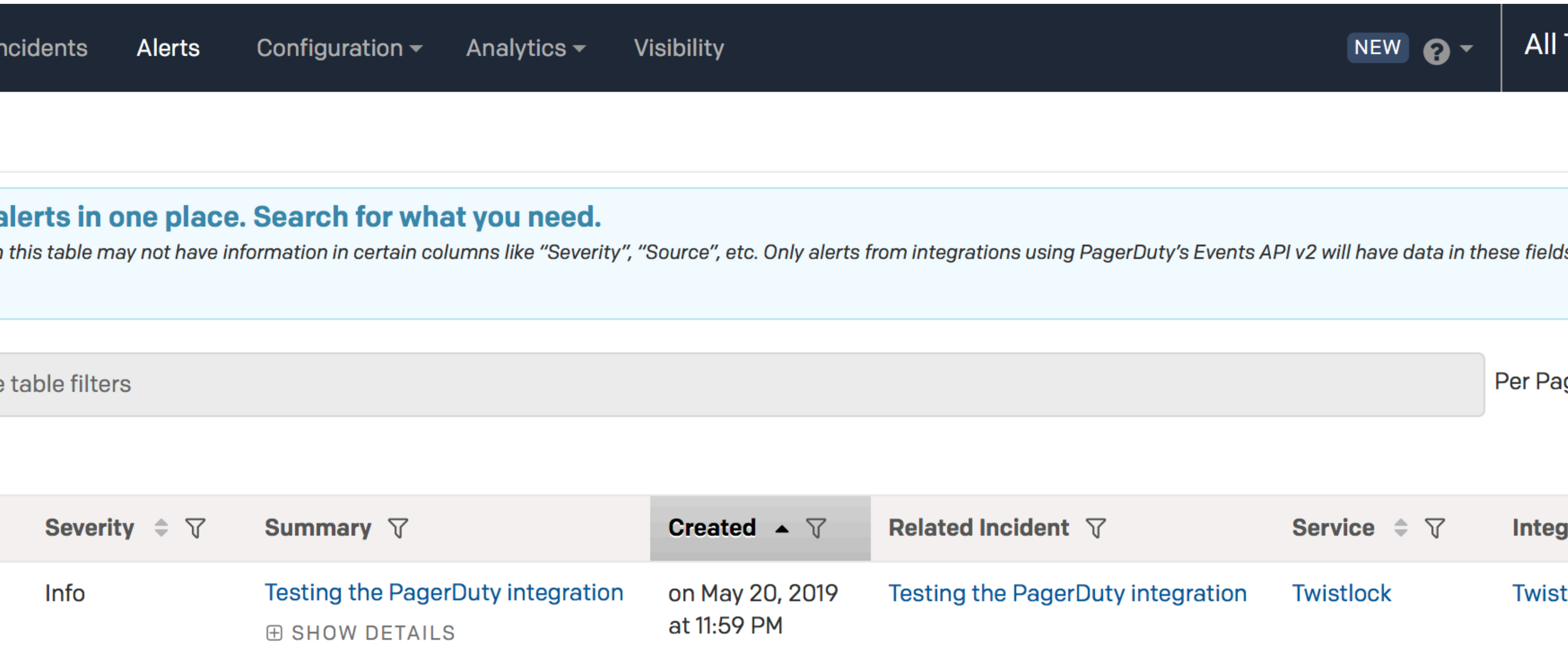
STEP 1 | In **Routing Key**, enter the integration key you copied from PagerDuty.

STEP 2 | In **Summary**, enter a brief description, which will appear in the PagerDuty dashboard alongside your alerts.

STEP 3 | For **Severity**, select the urgency of the alert.

STEP 4 | Click **Send Test Alert** to validate the integration.

If the integration is set up properly, you will see a sample alert in PagerDuty. In the PagerDuty dashboard, click **Alerts**.



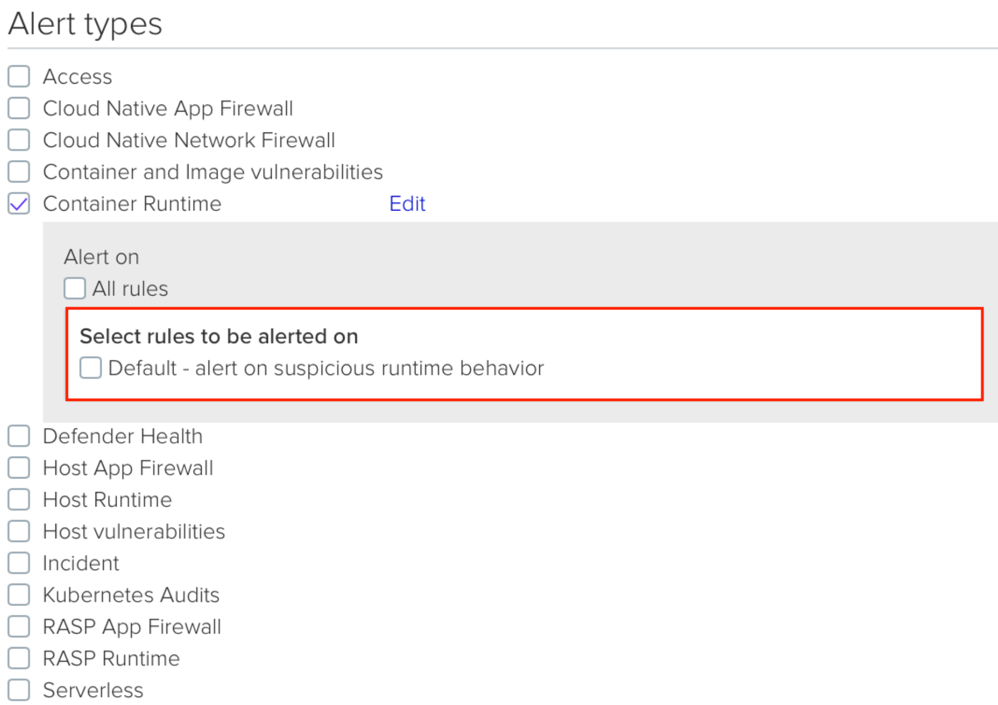
Configure the triggers

Configure how the alert is triggered.

STEP 1 | Under **Alert Types**, check the boxes types of events that should trigger an alert.

STEP 2 | For additional configuration options, click **Edit**.

STEP 3 | To specify specific rules that should trigger an alert, deselect **All rules**, and then select any individual rules.



STEP 4 | Click **Save**.

ServiceNow alerts

Edit on GitHub

ServiceNow is a workflow management platform. It offers a number of security operations applications. You can configure Prisma Cloud to route alerts to ServiceNow's Security Incident Response application.

Prisma Cloud audits are mapped to a ServiceNow security incident as follows:

- Audits and incidents are mapped to individual ServiceNow security incidents.
- Vulnerabilities are aggregated by resource (currently image) and mapped to individual ServiceNow security incidents. ServiceNow short description field lists the resource. ServiceNow description field lists the details of each finding.
- Compliance issues are aggregated by resource (image/container/host) and mapped to individual ServiceNow security incidents. ServiceNow short description field lists the resource. ServiceNow description field lists the details of each finding.

Compliance alerts will be sent to ServiceNow in real time (right after compliance scan), unlike the other alert providers which send compliance alerts every 24 hours.

Compliance alerts will be sent if the resource is new, or if there's a difference in the number of compliance issues for this resource after its scan. All the compliance issues of the resource will be sent (not only the new ones).

ServiceNow security incident	Field description	Prisma Cloud audit data
State	The current state of the security incident. Upon security incident creation, this field defaults to Draft.	Draft (automatically set by ServiceNow)
Priority	Select the order in which to address this security incident, based on the urgency. If this value is changed after the record is saved, it can affect the Business impact calculation.	Vulnerabilities: Max severity from the image's new vulnerabilities. ServiceNow's priorities map one-to-one to Prisma Cloud severities (Critical - Critical, High - High, Medium - Medium, Low - Low). Compliance: Max severity from the image/container/host's compliance issues. ServiceNow's priorities map one-to-one to Prisma Cloud severities (Critical - Critical, High - High, Medium - Medium, Low - Low).

ServiceNow security incident	Field description	Prisma Cloud audit data
		Incidents and audits: runtime audits priority set in the alert profile.
Business impact	Select the importance of this security incident to your business. The default value is Non-critical. If, after the security incident record has been saved, you change the value in the Priority and/or Risk fields, the Business impact is recalculated.	Automatically calculated by ServiceNow
Assignment group	The group to which this security incident is assigned.	Assignment group set in the alert profile
Assigned to	The individual assigned to analyze this security incident.	Assignee set in the alert profile
Short description	A brief description of the security incident.	Vulnerabilities: Prisma Cloud Compute vulnerabilities for image <image name> Compliance: Prisma Cloud Compute compliance issues for image/container/host <image/container/host name> Incidents and audits: Prisma Cloud Compute Audit - <audit type> - <message>
Category		Set to "None"
Sub-category		Set to "None"
Description	Description	Vulnerabilities: • Related resource details • CVEs IDs list (with each CVE’s details) • Project • Collections Compliance: • Related resource details

ServiceNow security incident	Field description	Prisma Cloud audit data
		• Compliance issues list (with each issue's details) • Project • Collections Incidents and audits: • Description • Related resource • Collections • Project • Time created • Then add all the other fields this type of Incident/ Audit has Note that the **Project** field will specify **Central Console** even when projects aren't enabled. Note that the **Collections** field will exist only for the following runtime audits: Admission Audits, Docker Audits, App Embedded Audits, Host Activities, Host Log Inspection, WAAS audits, Incidents, Defender Disconnected.

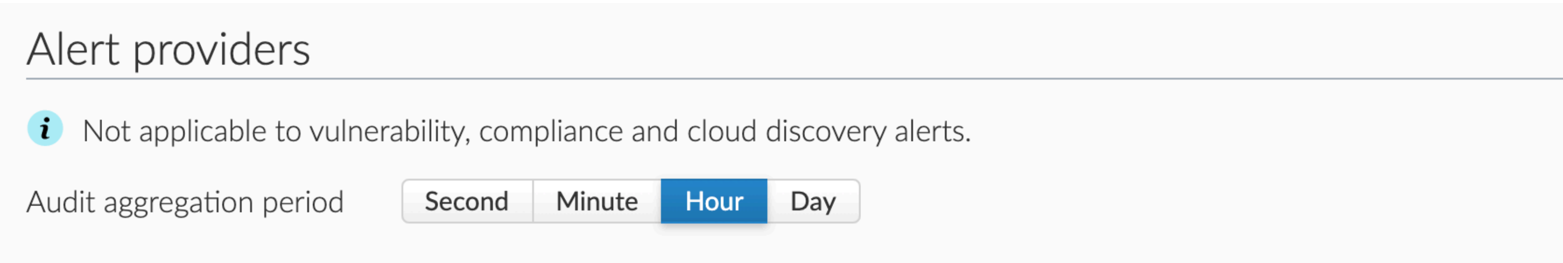
Configuring alert frequency

You can configure the rate at which alerts are emitted. This is a global setting that controls the spamminess of the alert service. Alerts received during the specified period are aggregated into a single alert. For each alert profile, an alert is sent as soon as the first matching event is received. All subsequent alerts are sent once per period.

STEP 1 | Open Console, and go to **Manage > Alerts**.

STEP 2 | In **Aggregate audits every**, specify the maximum rate that alerts should be sent.

You can specify **Second**, **Minute**, **Hour**, **Day**.



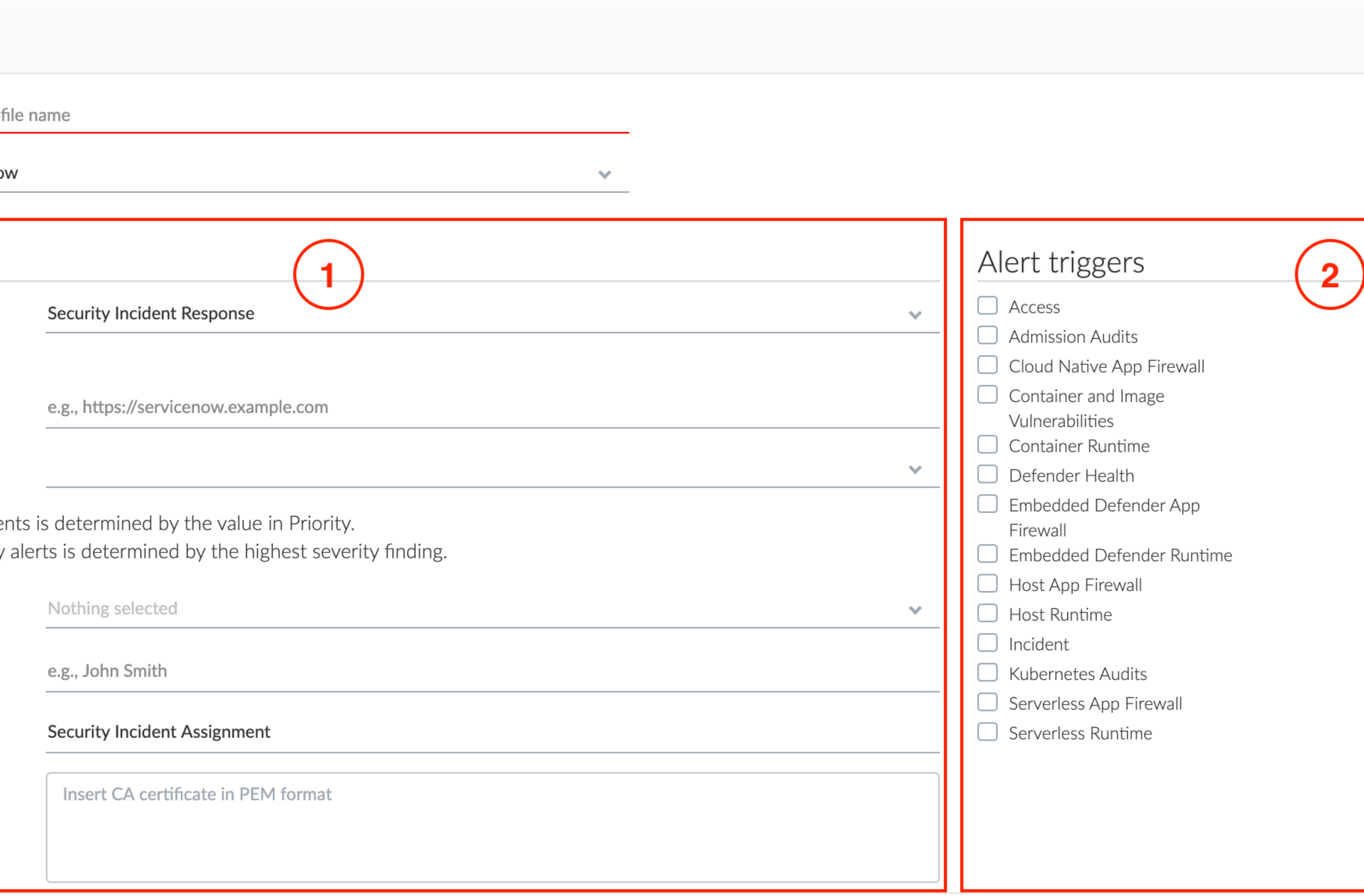
Sending findings to ServiceNow

Alert profiles specify which events trigger the alert machinery, and to which channel alerts are sent. You can send alerts to any combination of channels by creating multiple alert profiles.

Alert profiles consist of two parts:

(1) Alert settings — Who should get the alerts, and on what channel? Configure Prisma Cloud to integrate with ServiceNow and specify the people or places where alerts should be sent. You can specify assignees and assignment groups.

(2) Alert triggers — Which events should trigger an alert to be sent? Specify which of the rules that make up your overall policy should trigger alerts.



Create new alert profile

Create a new alert profile.

STEP 1 | In **Manage > Alerts**, click **Add profile**.

STEP 2 | Enter a name for your alert profile.

STEP 3 | In **Provider**, select **ServiceNow**.

Configure the channel

Configure Prisma Cloud to send alerts to ServiceNow, then validate the setup by sending a test alert.

Prerequisites: You've created a service account in ServiceNow with a base role of web_service_admin.

STEP 1 | In **Application**, select **Security Incident Response**.

STEP 2 | In **URL**, specify the base URL of your ServiceNow tenant.

For example, *https://ena03291.service-now.com*

STEP 3 | In **Credential**, click **Add New**.

1. In **Type**, select **Basic authentication**.

   This is currently the only auth method supported.
2. Enter a username and password.

STEP 4 | (Optional) In **Assignee**, enter the name of a user in ServiceNow that will be assigned the security incident.

This value isn't case sensitive.

STEP 5 | (Mandatory) In **Assignment Group**, enter the name of a group in ServiceNow that will be assigned the security incident. The default value is **Security Incident Assignment**.

If **Assignment Group** is set without speciying **Assignee**, the first user from the group is set on the security incident (ServiceNow's logic).

If the **Assignee** set in the profile isn't a part of the **Assignment Group**, the security incident won't be created (ServiceNow's logic).

STEP 6 | (Optional) In **CA certificate**, enter a CA certificate in PEM format. Relevant only for on-premises deployments of ServiceNow.

STEP 7 | Click **Send Test Alert**. If everything looks good, and you get an alert in ServiceNow, save the profile.

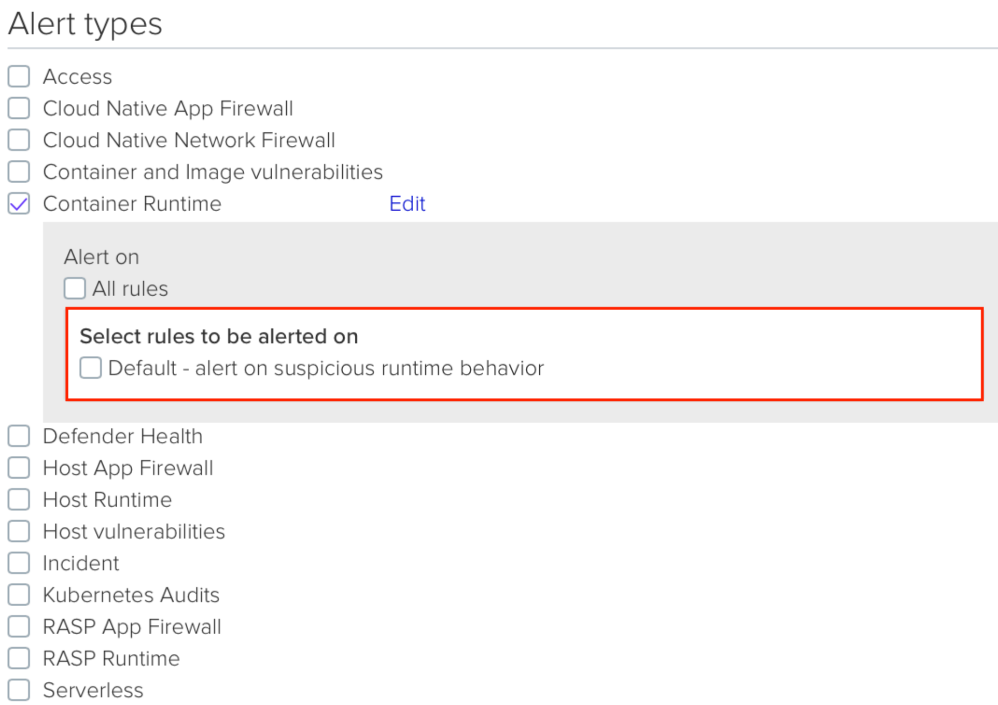
Configure the triggers

Configure how the alert is triggered.

STEP 1 | Under **Alert Types**, check the boxes types of events that should trigger an alert.

STEP 2 | For additional configuration options, click **Edit**.

STEP 3 | To specify specific rules that should trigger an alert, deselect **All rules**, and then select any individual rules.



STEP 4 | Click **Save**.

ServiceNow alerts

Edit on GitHub

ServiceNow is a workflow management platform. It offers a number of security operations applications. You can configure Prisma Cloud to route alerts to ServiceNow's Vulnerability Response application.

To integrate Prisma Cloud with ServiceNow, you'll need to create a ServiceNow endpoint to consume findings from the Prisma Cloud scanner. The endpoint is created using ServiceNow's Scripted REST API mechanism.

Each vulnerability found by the Prisma Cloud scanner is mapped to a ServiceNow vulnerable item. Scanner data is mapped to vulnerable items as follows:

*Vulnerable items contain all CVEs reported by the Prisma Cloud scanner only if the corresponding CVEs also exist in ServiceNow's vuln DB. If a CVE doesn't exist in ServiceNow, the **Vulnerability (Reference)** field won't list it.*

ServiceNow vulnerablity item field	Field description	Prisma Cloud scanner data
Source	The scanner that found this vulnerable item.	Prisma Cloud Compute
Vulnerability (Reference)	ID of the vulnerability associated with this vulnerable item.	Reference to CVE ID (if exists in ServiceNow's vulnerabilities DB)
State	This field defaults to Open, but you can change it to Under Investigation if the vulnerability is ready for immediate remediation.	Open (automatically set by ServiceNow)
Assignment group	Group selected to work on this vulnerability group.	Assignment group set in the alert profile
Assigned to	Individual from the selected assignment group that works on this vulnerability.	Assignee set in the alert profile
Created	The date this vulnerable item was created in your instance.	Creation date of the vulnerable item (automatically set by ServiceNow)
Additional comments	Any relevant information.	Vulnerabilities: • Image name • Severity

ServiceNow vulnerablity item field	Field description	Prisma Cloud scanner data
		• Package • Package version • Fix status • Project • Collections

Configuring ServiceNow

Create a ServiceNow endpoint to collect findings from the Prisma Cloud scanner.

Prerequisites: Prisma Cloud Console is running.

STEP 1 | In ServiceNow, create a Scripted REST API. Name it **Prisma Vulnerabilities Report**.

For more information, see the official documentation here.

STEP 2 | Create a new resource in your scripted REST service.

STEP 3 | In **Name**, enter **report_findings**.

STEP 4 | In **HTTP method**, select **POST**.

STEP 5 | Download the script that implements the endpoint from Prisma Cloud Console.

1. Log into Prisma Cloud Console.
2. Go to **Manage > Alerts > Add Profile**.
3. Click **Add Profile**.
4. In **Provider**, select **ServiceNow**.
5. In **Application**, select **Vulnerability Response**.
6. In **Scripted REST API**, click **Copy**.
7. In ServiceNow, paste the script into **Script**.

STEP 6 | Click **Submit** to create the resource.

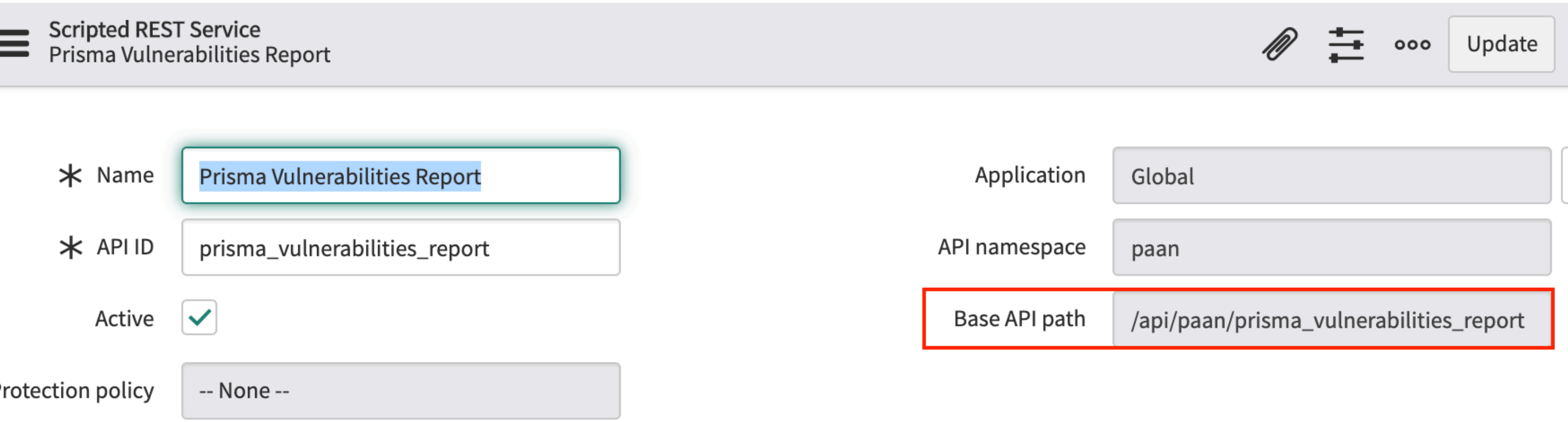
STEP 7 | Construct the URL for your resource (endpoint), then copy it, and set it aside. You'll need when you configure Prisma Cloud to send findings to ServiceNow.

The format for the base URL is: *https://<SERVICENOW>/<BASE_API_PATH>*

For example: *https://ena03291.service-now.com/api/paan/prisma_vulnerabilities_report*

Where:

- SERVICENOW — URL for your ServiceNow instance.
- BASE_API_PATH — Path to the scripted API service you just created.



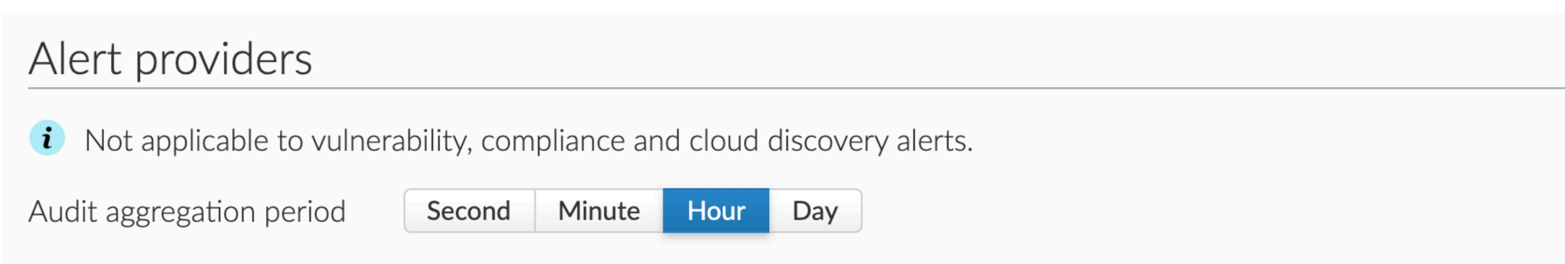
Configuring alert frequency

You can configure the rate at which alerts are emitted. This is a global setting that controls the spamminess of the alert service. Alerts received during the specified period are aggregated into a single alert. For each alert profile, an alert is sent as soon as the first matching event is received. All subsequent alerts are sent once per period.

STEP 1 | Open Console, and go to **Manage > Alerts**.

STEP 2 | In **Aggregate audits every**, specify the maximum rate that alerts should be sent.

You can specify **Second**, **Minute**, **Hour**, **Day**.



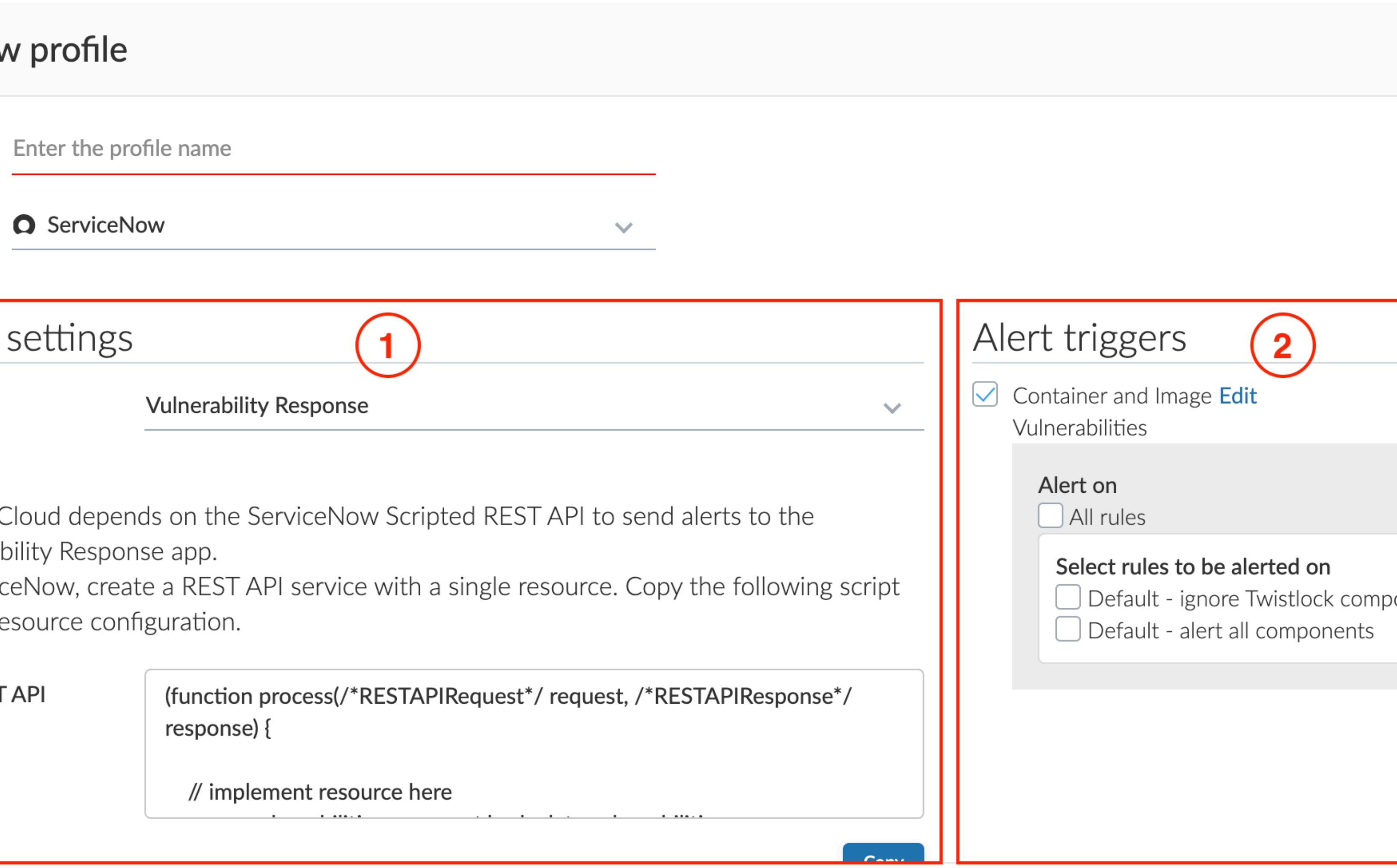
Sending findings to ServiceNow

Alert profiles specify which events trigger the alert machinery, and to which channel alerts are sent. You can send alerts to any combination of channels by creating multiple alert profiles.

Alert profiles consist of two parts:

(1) Alert settings — Who should get the alerts, and on what channel? Configure Prisma Cloud to integrate with ServiceNow and specify the people or places where alerts should be sent. You can specify assignees and assignment groups.

(2) Alert triggers — Which events should trigger an alert to be sent? Specify which of the rules that make up your overall policy should trigger alerts. For the Vulnerability Response application, you can send vulnerability and compliance alerts only.



Create new alert profile

Create a new alert profile.

STEP 1 | In **Manage > Alerts**, click **Add profile**.

STEP 2 | Enter a name for your alert profile.

STEP 3 | In **Provider**, select **ServiceNow**.

Configure the channel

Configure Prisma Cloud to send alerts to ServiceNow, then validate the setup by sending a test alert.

Prerequisites: You've created a service account in ServiceNow with a base role of web_service_admin.

STEP 1 | In **Application**, select **Vulnerability Response**.

STEP 2 | In **Scripted API URL**, enter the url of the vulnerabilities reporting api defined in ServiceNow (see ServiceNow config above). e.g. https://ven03718.service-now.com/api/paan/prisma_vulnerabilities_report

STEP 3 | In **Credential**, click **Add New**.

1. In **Type**, select **Basic authentication**.

   This is currently the only auth method supported.
2. Enter a username and password.

STEP 4 | (Optional) In **Assignee**, enter the name of a user in ServiceNow that will be assigned the Vulnerable Items.

The assignee name isn't case-sensitive.

STEP 5 | (Optional) In **Assignment Group**, enter the name of a group in ServiceNow that will be assigned the Vulnerable Items.

STEP 6 | (Optional) In **CA certificate**, enter a CA certificate in PEM format. Relevant only for on-premises deployments of ServiceNow.

STEP 7 | Click **Send Test Alert**. If everything looks good, and you get an alert in ServiceNow, save the profile.

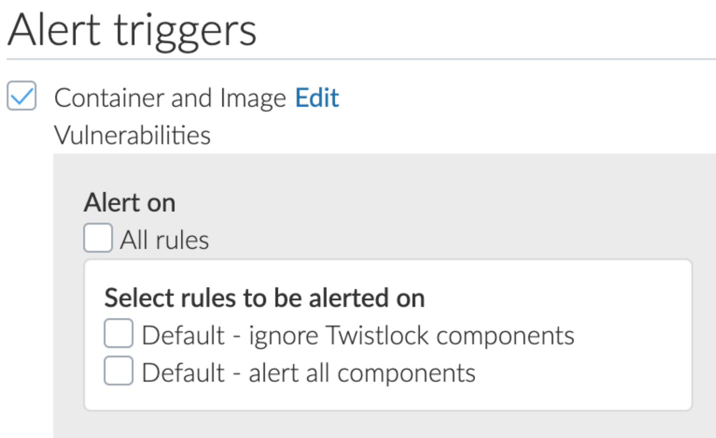
Configure the triggers

Configure how the alert is triggered.

STEP 1 | Under **Alert Types**, check the boxes types of events that should trigger an alert.

STEP 2 | For additional configuration options, click **Edit**.

STEP 3 | To specify specific rules that should trigger an alert, deselect **All rules**, and then select any individual rules.



STEP 4 | Click **Save**.

Map Vulnerable Items to Configuration Items (optional)

Adjust your Scripted REST API to map each vulnerable item to its configuration item (CI) in ServiceNow's CMDB.

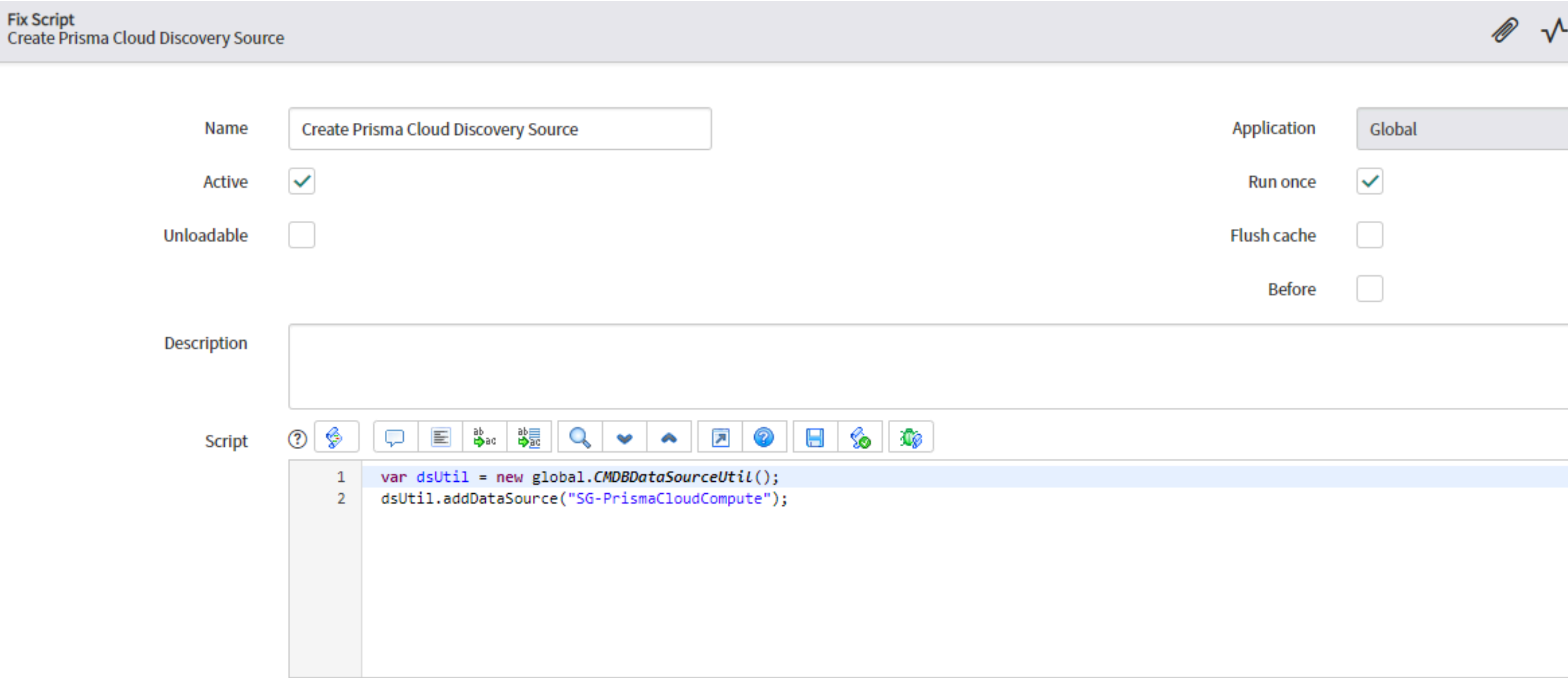
STEP 1 | Create a Discovery Source in ServiceNow for Prisma Cloud Compute:

1. Navigate to **System Definition > Fix Script**.
2. Click **New**.
3. Name your Fix Script appropriately.
4. Ensure **Run once** is selected .
5. Execute the following code with the appropriate value for your Discovery Source name: “SG-PrismaCloudCompute”.

```
Source var dsUtil = new global.CMDBDataSourceUtil();
```

```
dsUtil.addDataSource("SG-PrismaCloudCompute");
```

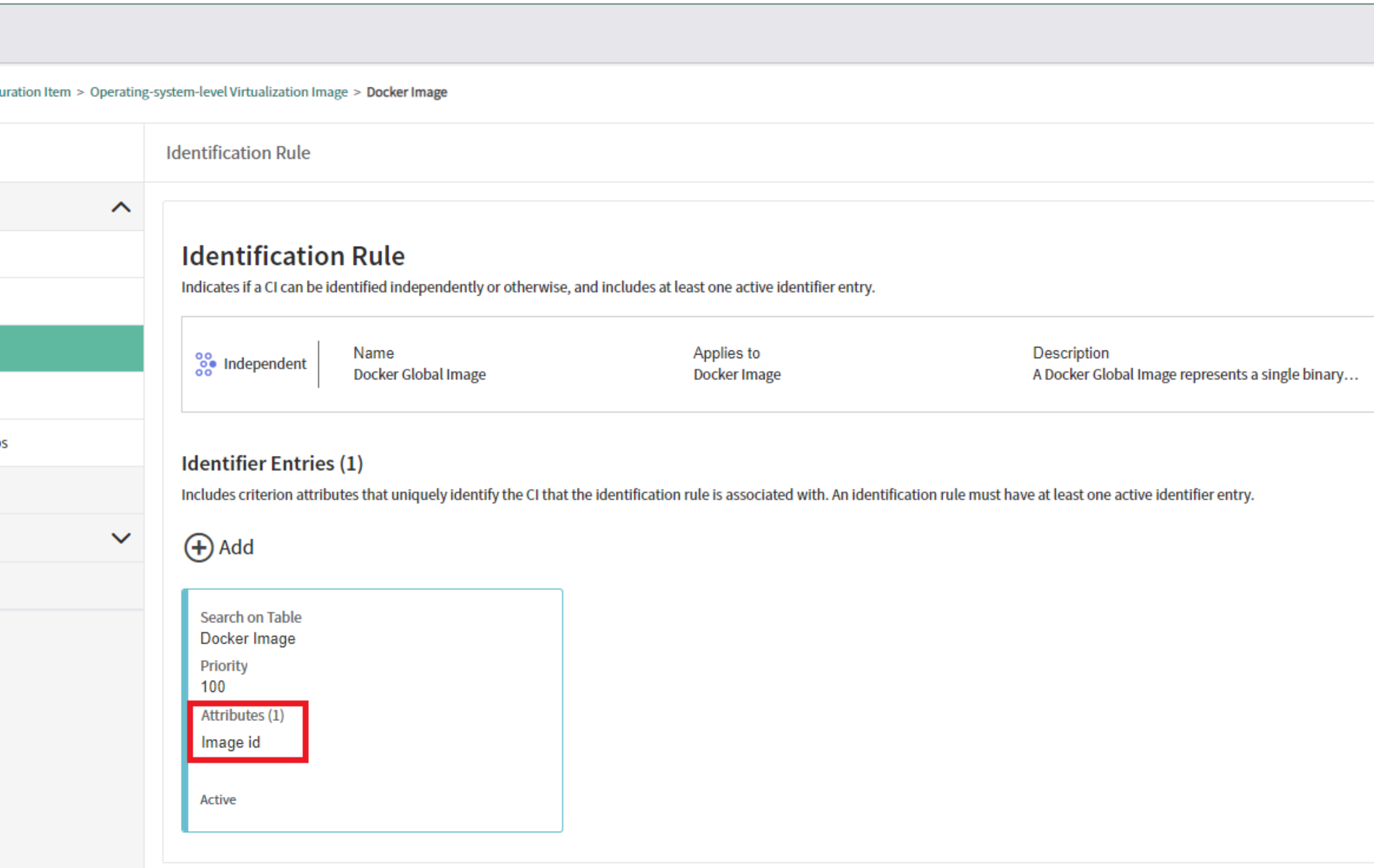
6. Your fix script should look like this:



7. Ensure your discovery source has fewer than 40 characters.
8. After you have saved your fix script, navigate to it again and click **Run Fix Script**.

STEP 2 | Create an Identification Rule in ServiceNow:

1. Navigate to **Configuration > CI Class Manager**.
2. In the CI Classes hierarchy, choose **Docker Image**.
3. Navigate to **Class Info > Identification Rule**.
4. Add an identification rule to identify the image by the **Image id** attribute.



STEP 3 | Use the enhanced script in place of the standard script you copy from Console when setting up the Scripted REST API.

Suggested script

The following script maps vulnerable items to configuration items. Use it in place of the script you copy from Console when setting up the Scripted REST API in Configuring ServiceNow. To use the script, you must first set up a discovery source and identification rule.

The script in this section extends the standard script to:

- Implement CI mapping — Finds the relevant CI (from type docker image), or creates one if it doesn't exist, and references it in the Vulnerable Item.

- Create a vulnerability placeholder — Creates an empty vulnerability in ServiceNow's vulnerabilities DB when the CVE ID sent by Prisma Cloud Compute can't be found in ServiceNow.

The following listing shows the script in its entirety. Copy and use this listing when setting up the vulnerable item to configuration item mapping.

```
(function process( /*RESTAPIRequest*/ request, /*RESTAPIResponse*/
 response) {

    var vulnerabilities = request.body.data.vulnerabilities;
    response.setContentType('application/JSON');
    var writer = response.getStreamWriter();

    for (var i in vulnerabilities) {
        var vulnItemRecord = new
 GlideRecord('sn_vul_vulnerable_item');
        var vulnEntryRecord = new GlideRecord('sn_vul_entry');
        var userGroupsRecord = new GlideRecord('sys_user_group');
        var vulnerability = vulnerabilities[i];
        // the id field is the name (a string) of the cve
        if (!vulnEntryRecord.get('id', vulnerability.cve)) {
            // The following code inserts the placeholder
 vulnerability in sn_vul_nvd_entry. The other attributes will be
 filled once the NVD import run's
            var nvd_entry = new GlideRecord("sn_vul_nvd_entry");
            nvd_entry.initialize();
            nvd_entry.setValue("id", vulnerability.cve);
            var vulEntry = nvd_entry.insert();
            vulnEntryRecord = new GlideRecord('sn_vul_entry');
            vulnEntryRecord.get(vulEntry);
        }

        if (!userGroupsRecord.get('name',
 vulnerability.assignment_group)) {
            userGroupsRecord.sys_id = "";
        }

        vulnItemRecord.initialize();

        // The following block of code is to create a CI using IRE
        if (vulnerability.imageName && vulnerability.imageID) {
          // Step 1: construct the payload
          // Step 2: Encode the payload as JSON
          // Step 3: create CI using createOrUpdateCIEnhanced API.
 This API requires discovery source that needs to be created
         var payload = {
             "items": [{
                  "className": "cmdb_ci_docker_image", // update the
 cmdb unmatched class name here
                  "values": {
                    "image_id": vulnerability.imageID, // update the
 correct values that needs to be populated and any additional fields
                    "name": vulnerability.imageName
                  },
```

```
                  "sys_object_source_info": { // optional, used to
 optimize the fetch to get CIs from this specific source only
                      "source_native_key": vulnerability.imageID, //
 unique key/id for the item from the source
                      "source_name": "SG-PrismaCloudCompute" // The
 discovery source of the CI information
                  }
              }]
          };

       var inputPayload = new JSON().encode(payload);
       var cmdb =
 SNC.IdentificationEngineScriptableApi.createOrUpdateCI("SG-
PrismaCloudCompute",inputPayload); // CMDB discovery source name
       var output = JSON.parse(cmdb);

 vulnItemRecord.cmdb_ci.setDisplayValue(output.items[0].sysId); //
 This will  assign CMDB_ci item to vuln item.

       } else if (vulnerability.imageName != "Prisma Test Alert") {
            gs.log("missing image name or image id");
       }

       vulnItemRecord.Description = vulnerability.description;
       vulnItemRecord.assigned_to = vulnerability.assigned_to;
       // sys_id is the unique id of a record (like an internal
 service now GUID), used to link records in different tables

 vulnItemRecord.assignment_group.setDisplayValue(userGroupsRecord.sys_id);
       vulnItemRecord.source = vulnerability.source;

 vulnItemRecord.vulnerability.setDisplayValue(vulnEntryRecord.sys_id);

 vulnItemRecord.comments.setJournalEntry(vulnerability.comments);
       vulnItemRecord.insert();
       vulnItemRecord.query();
       writer.writeString(JSON.stringify(vulnItemRecord));
    }

    response.setStatus(201);

})(request, response);
```

The following excerpt shows the part of the listing that implements the CI mapping:

```
        // The following block of code is to create a CI using IRE
        if (vulnerability.imageName && vulnerability.imageID) {
          // Step 1: construct the payload
          // Step 2: Encode the payload as JSON
          // Step 3: create CI using createOrUpdateCIEnhanced API.
 This API requires discovery source that needs to be created
         var payload = {
             "items": [{
                  "className": "cmdb_ci_docker_image", // update the
 cmdb unmatched class name here
                  "values": {
```



```
                   "image_id": vulnerability.imageID, // update the
correct values that needs to be populated and any additional fields
                   "name": vulnerability.imageName
                 },
                 "sys_object_source_info": { // optional, used to
optimize the fetch to get CIs from this specific source only
                     "source_native_key": vulnerability.imageID, //
unique key/id for the item from the source
                     "source_name": "SG-PrismaCloudCompute" // The
discovery source of the CI information
                 }
             }]
         };

      var inputPayload = new JSON().encode(payload);
      var cmdb =
SNC.IdentificationEngineScriptableApi.createOrUpdateCI("SG-
PrismaCloudCompute",inputPayload); // CMDB discovery source name
      var output = JSON.parse(cmdb);

vulnItemRecord.cmdb_ci.setDisplayValue(output.items[0].sysId); //
This will  assign CMDB_ci item to vuln item.

      } else if (vulnerability.imageName != "Prisma Test Alert") {
           gs.log("missing image name or image id");
      }
```

The following excerpt shows the part of the listing that creates the vulnerability placeholder:

```
       if (!vulnEntryRecord.get('id', vulnerability.cve)) {
           // The following code inserts the placeholder
vulnerability in sn_vul_nvd_entry. The other attributes will be
filled once the NVD import run's
           var nvd_entry = new GlideRecord("sn_vul_nvd_entry");
           nvd_entry.initialize();
           nvd_entry.setValue("id", vulnerability.cve);
           var vulEntry = nvd_entry.insert();
           vulnEntryRecord = new GlideRecord('sn_vul_entry');
           vulnEntryRecord.get(vulEntry);
       }
```


Slack Alerts

Edit on GitHub

Prisma Cloud lets you send alerts to Slack channels and users.

Configuring Slack

To integrate Prisma Cloud with Slack, you must enable incoming webhooks. Prisma Cloud uses incoming webhooks to post messages to Slack.

STEP 1 | Log into the page where you manage apps for your Slack workspace.

STEP 2 | In the **Search App Directory** box, enter **Incoming Webhooks**, and hit **Return**.

STEP 3 | Click on the result.

STEP 4 | Click the green **Add Configuration** button.

STEP 5 | Enter the channel where you want Prisma Cloud to post.

STEP 6 | Click **Add Incoming Webooks Integration**.

STEP 7 | Copy the **Webhook URL** and set it aside. You will use it when configuring Prisma Cloud.

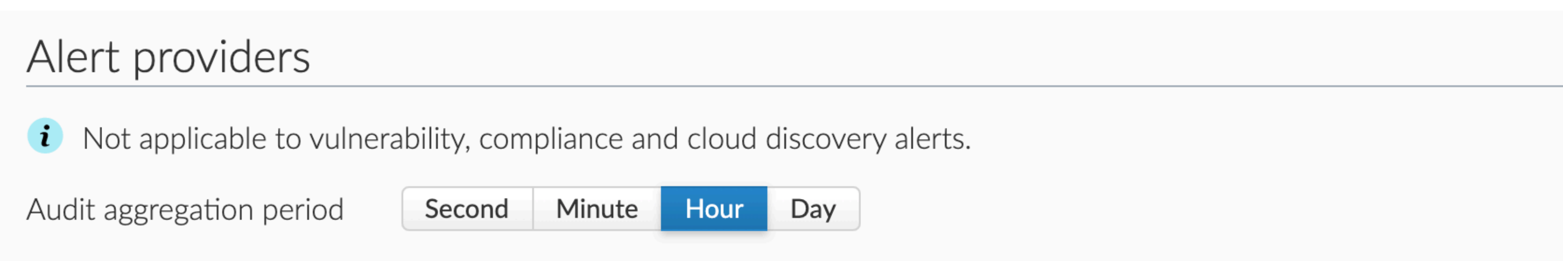
Configuring alert frequency

You can configure the rate at which alerts are emitted. This is a global setting that controls the spamminess of the alert service. Alerts received during the specified period are aggregated into a single alert. For each alert profile, an alert is sent as soon as the first matching event is received. All subsequent alerts are sent once per period.

STEP 1 | Open Console, and go to **Manage > Alerts**.

STEP 2 | In **Aggregate audits every**, specify the maximum rate that alerts should be sent.

You can specify **Second**, **Minute**, **Hour**, **Day**.

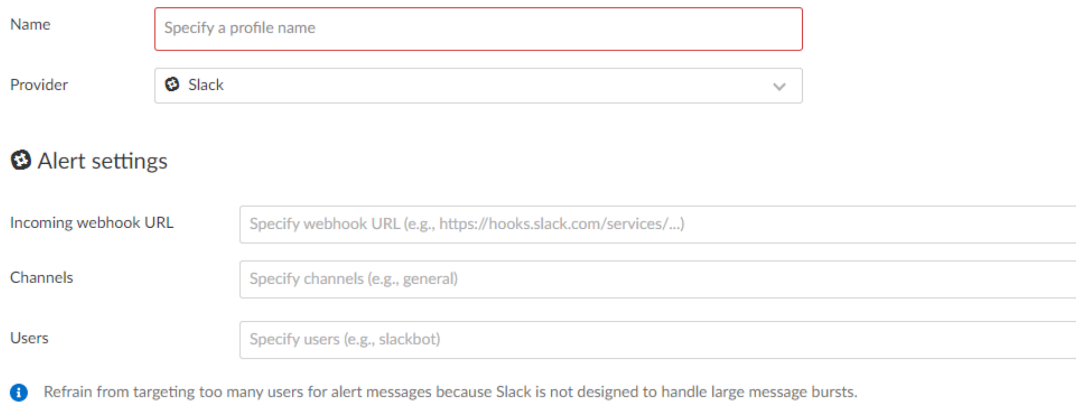


Sending alerts to Slack

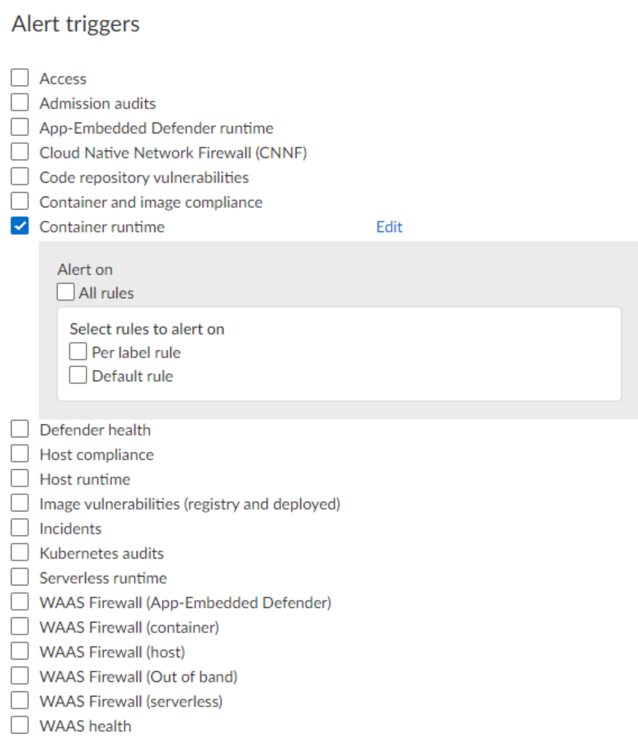
Alert profiles specify which events should trigger the alert machinery, and to which channel alerts are sent. You can send alerts to any combination of channels by creating multiple alert profiles.

Alert profiles consist of two parts:

(1) Alert settings — Who should get the alerts, and on what channel? Configure Prisma Cloud to integrate with your messaging service and specify the people or places where alerts should be sent. For example, configure the email channel and specify a list of all the email addresses where alerts should be sent. Or for JIRA, configure the project where the issue should be created, a long with the type of issue, priority, assignee, and so on.



(2) Alert triggers — Which events should trigger an alert to be sent? Specify which of the rules that make up your overall policy should trigger alerts.



If you use multi-factor authentication, you must create an exception or app-specific password to allow Console to authenticate to the service.

Create new alert profile

Create a new alert profile.

STEP 1 | In **Manage > Alerts**, click **Add profile**.

STEP 2 | Enter a name for your alert profile.

STEP 3 | In **Provider**, select **Slack**.

Configure the channel

Configure the channel.

STEP 1 | In **Incoming Webhook URL**, enter the URL you generated in the previous section.

STEP 2 | Specify how to route alerts. Enter values for one or both of the following fields.

1. In **Channels**, enter the Slack channel where you want to post alerts.
2. In **Users**, enter the Slack users to whom you want to send alerts.

STEP 3 | Click **Send Test Alert** to test the connection.

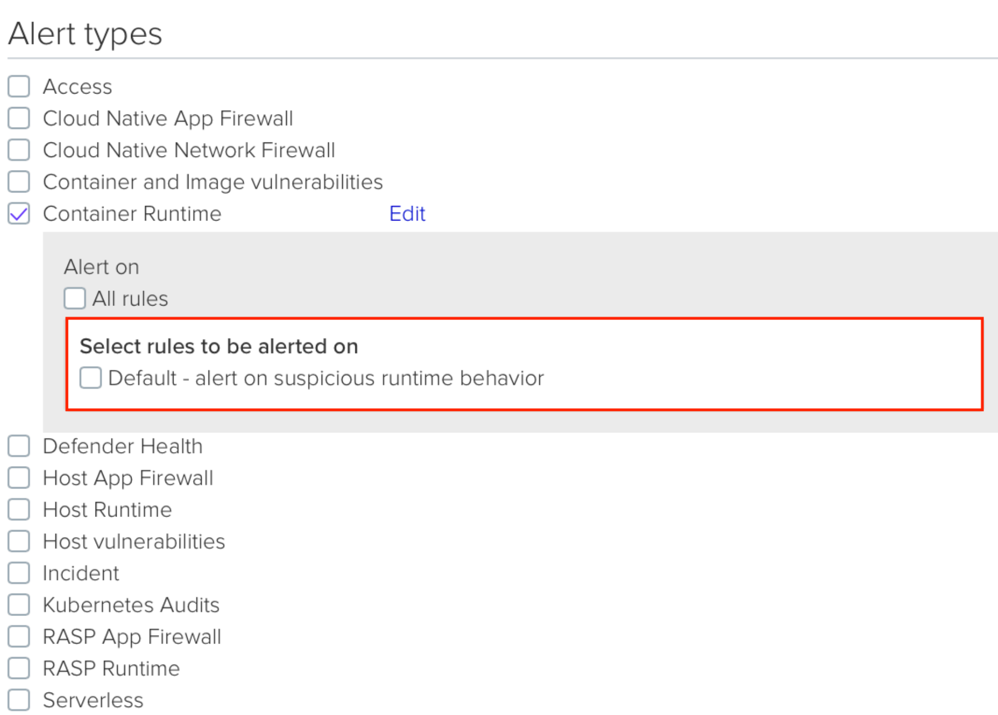
Configure the triggers

Configure how the alert is triggered.

STEP 1 | Under **Alert Types**, check the boxes types of events that should trigger an alert.

STEP 2 | For additional configuration options, click **Edit**.

STEP 3 | To specify specific rules that should trigger an alert, deselect **All rules**, and then select any individual rules.



STEP 4 | Click **Save**.

Splunk alerts

Edit on GitHub

Splunk is a software platform to search, analyze, and visualize machine-generated data gathered from websites, applications, sensors, and devices.

Prisma Cloud continually scans your environment for vulnerabilities, Complience, Runtime behavior, WAAS valiolations and more. You can now monitor your Prisma Cloud alerts in Splunk using a native integration.

Sending alerts to Splunk

Follow the instructions below to send alerts from your Prisma Cloud Console to Splunk Enterprise or Splunk Cloud Platform.

Set up Splunk HTTP Event Collector (HEC) to view alert notifications from Prisma Cloud in Splunk

Splunk HEC lets you send data and application events to a Splunk deployment over the HTTP and HTTPS protocols. This helps consolidate alert notifications from Prisma Cloud into Splunk, so that your operations team can review and take action on the alerts.

STEP 1 | To set up HEC, use instructions in Splunk documentation. note that the default **source type** is **_json**

STEP 2 | Select **Settings > Data inputs > HTTP Event** Collector and make sure you see HEC added in the list and that the status shows that it is **Enabled**.

Set up the Splunk integration in Prisma Cloud Compute edition

STEP 1 | Log in to Prisma Cloud Console

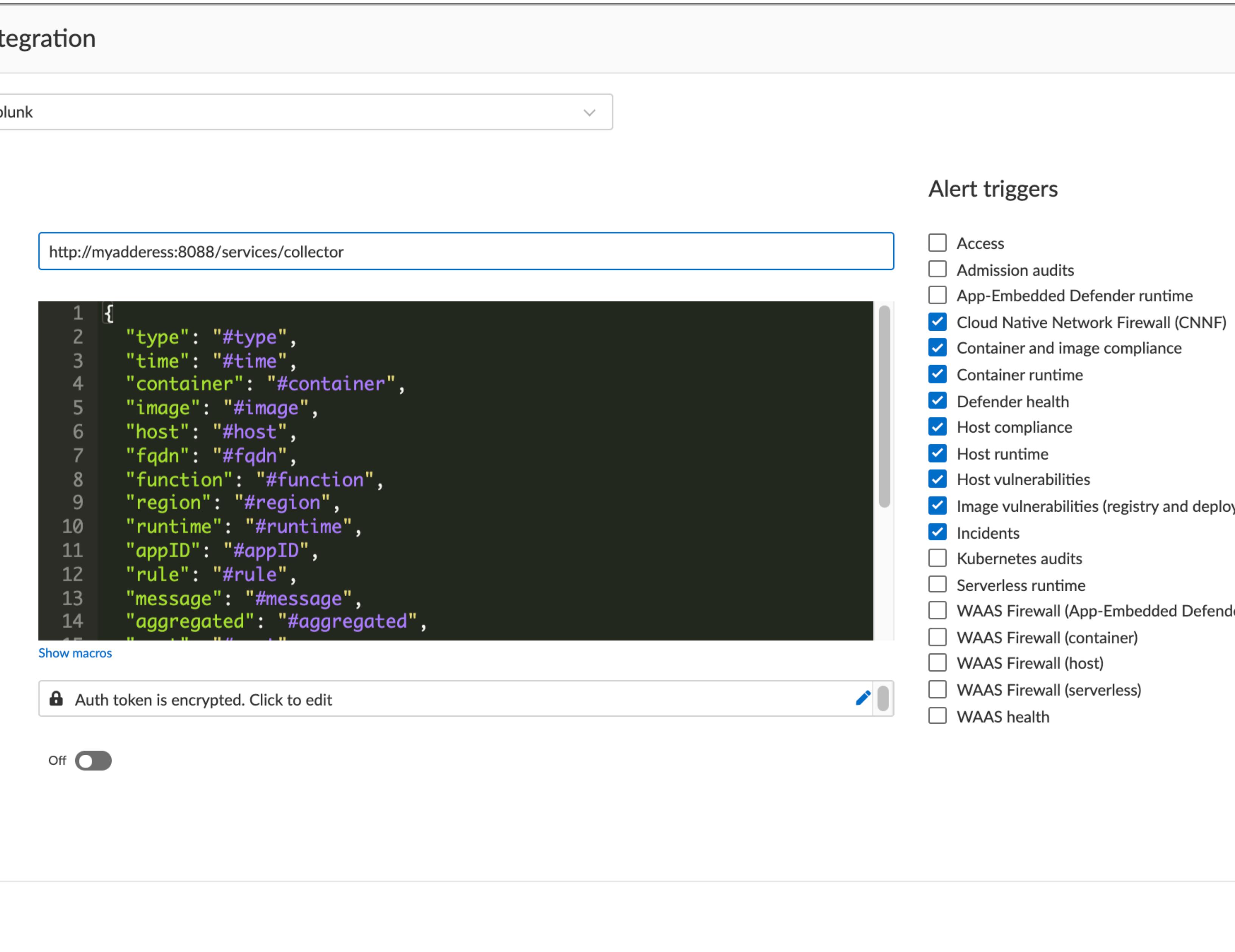
STEP 2 | Go to **Manage > Alerts > Manage** tab

STEP 3 | Click on **+ Add profile** to create a dedicated alert profile for Splunk

1. Enter a name for your alert profile
2. In **Provider**, select **Splunk**
    1. In **Splunk HTTP event collector URL**, enter the Splunk HEC URL that you set up earlier.
    2. In Custom JSON, enter the structure of the JSON payload, or use the default JSON.

       For more details about the type of data in each field, click **Show macros**.
    3. Enter **Auth Token**

       The integration uses token-based authentication between Prisma Cloud and Splunk to authenticate connections to Splunk HEC. A token is a 32-bit number that is presented in Splunk.
3. In **Alert triggers** section, select the triggers that you would like Splunk to be alerted by
4. Click **Send test alert** to test the connection. You can view the test message in Splunk



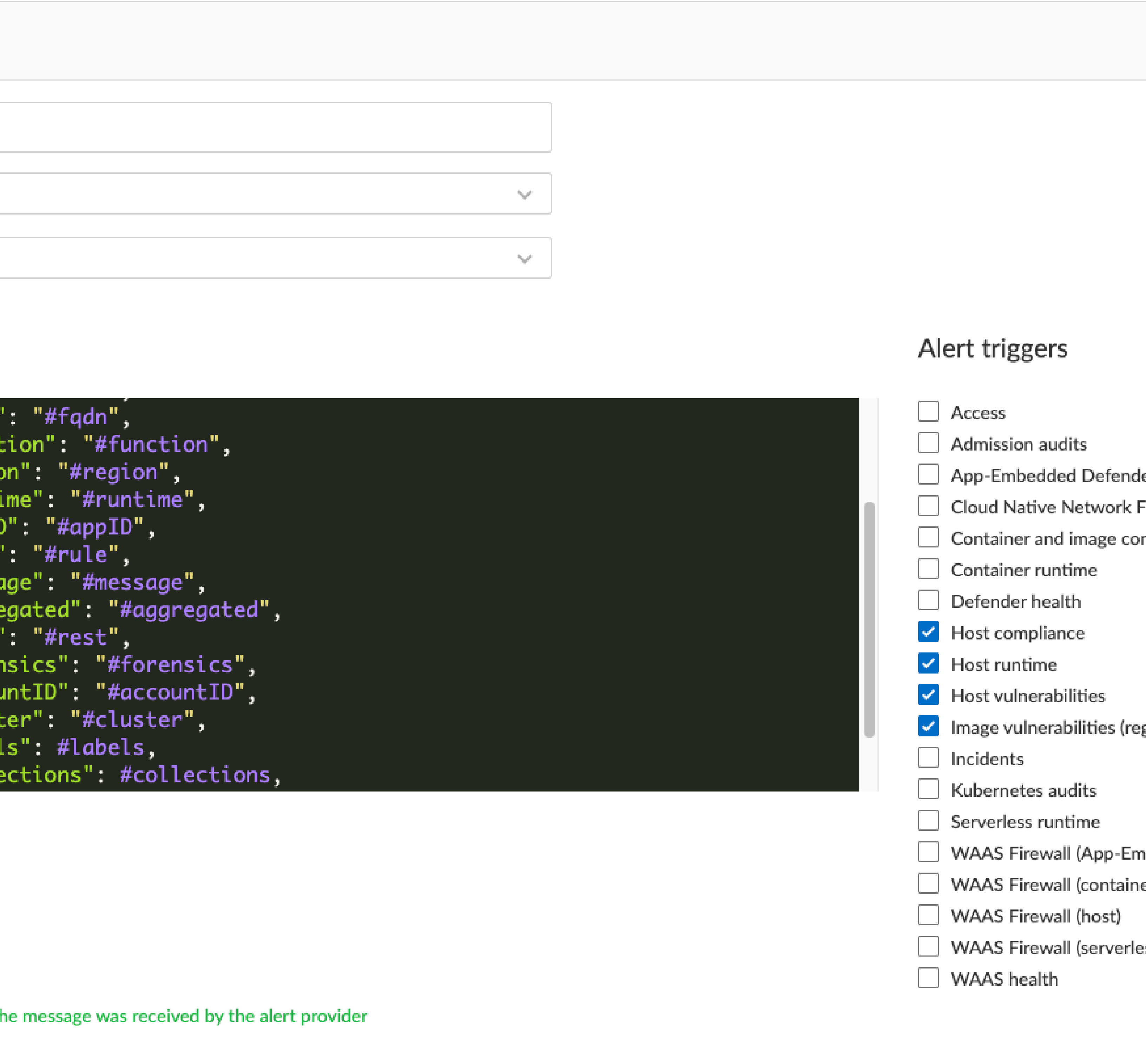
Set up the Splunk integration in Prisma Cloud Enterprise edition (SAAS)

Prisma Cloud Compute in SAAS uses the same notification settings set up in the platform for CSPM alerts. These configurations are setup in the platform under **Settings > Integrations**, and can be used in Compute by importing them as an Alert Profile. Any changes to the provider settings will need to be done on the platform side.

STEP 1 | Integrate Prisma Cloud with Splunk

STEP 2 | Importing platform configurations inside Compute:

1. Navigate to **Manage > Alerts > Manage** tab in Compute, click on "Add Profile"
2. From the Provider drop down, select **Prisma Cloud**
3. In the Integrations field, select the configuration you set up with Splunk in step 1
4. Select triggers to be sent to this channel
5. Click Save


Alert triggers
": "#fqdn",
tion": "#function",
on": "#region",
ime": "#runtime",
D": "#appID",
": "#rule",
age": "#message",
egated": "#aggregated",
": "#rest",
nsics": "#forensics",
untID": "#accountID",
ter": "#cluster",
ls": #labels,
ections": #collections,
Access
Admission audits
App-Embedded Defende
Cloud Native Network F
Container and image co
Container runtime
Defender health
Host compliance
Host runtime
Host vulnerabilities
Image vulnerabilities (reg
Incidents
Kubernetes audits
Serverless runtime
WAAS Firewall (App-Em
WAAS Firewall (containe
WAAS Firewall (host)
WAAS Firewall (serverle
WAAS health
he message was received by the alert provider

Message structure

Both integrations with Splunk, via Prisma Cloud SAAS and Enterprise eddition, generate the same event format.

JSON schema

The JSON scema includes the following default fields:

- app: Prisma Cloud Compute Alert Notification
- message: contains the alert content in a JSON format as defined in the **Custom JSON** field
- sender: Prisma Cloud Compute Alert Notification
- sentTs: Event sending timestamp as Unix time
- type: alert

```
{
   app: Prisma Cloud Compute Alert Notification
   message: { [+] }
   sender: Prisma Cloud Compute Alert Notification
   sentTs: 1637843439
   type: alert
}
```

You can learn more about the Alert JSON macros and customizations in the Webhook Alert documentation

Webhook alerts

Edit on GitHub

Prisma Cloud offers native integration with a number of services, including email, JIRA, and Slack. When no native integration is available, webhooks provide a mechanism to interface Prisma Cloud's alert system with virtually any third party service.

A webhook is an HTTP callback. When an event occurs, Prisma Cloud notifies your web service with an HTTP POST request. The request contains an JSON body that you configure when you set up the webhook. A webhook configuration consists of:

- URL
- Custom JSON body
- Credentials
- CA Certificate

Custom JSON body

You can customize the body of the POST request with values of interest. The content of the JSON object in the request body is defined using predefined macros. For example:

```
{
  "type":#type,
  "host":#host,
  "details":#message
}
```

When an event occurs, Prisma Cloud replaces the macros in your custom JSON with real values, and then submits the request.

```
{
  "type":"ContainerRuntime",
  "host":"host1",
  "details":"/bin/cp changed binary /bin/busybox MD5:XXXXXXX"
}
```

All supported macros are described in the following table. Not all macros are applicable to all alert types.

Rule	Description
#type	Audit alert type. For example, 'Container Runtime'.
#time	Audit alert time. For example, 'Jan 21, 2018 UTC'.
#container	Impacted container.
#image	Impacted image.

Rule	Description
#host	Hostname for the host where the audit occurred.
#fqdn	Fully qualified domain name for the host where the audit occurred.
#function	Serverless function where the audit occurred.
#region	Region where the audit occurred. For example 'N. Virginia'.
#runtime	Language runtime in which the audit occurred. For example, 'python3.6'.
#appID	Serverless or Function name.
#rule	Rule which triggered the alert.
#message	Associated alert message.
#aggregated	All fields in the audit message as a single JSON object.
#rest	All subsequent alerts that occurred during the aggregation period, in JSON format.
#forensics	API link to download the forensics data for the incident.
#accountID	The cloud account ID in which the audit was detected.
#cluster	The cluster in which the audit was detected.
#labels	A list of the alert labels of the resource in which the audit was detected.
#collections	A list of the associated collections for the resource where the issue was detected.
#complianceIssues	The compliance issues detected in the latest scan of the resource. A single alert includes compliance issues for a single resource.
#vulnerabilities	The new vulnerabilities detected in the latest scan. A single alert includes vulnerabilities for all resources where new vulnerabiltiies were found, so the vulnerabilities macro includes the data about the resources as well. All other macros, except type and time, will be empty.

The *#vulnerabilities* and *#complianceIssues* macros include inner structures. Below is an example of their content. Notice that the structure is subject to minor changes between versions.

```
{
    "vulnerabilities": [
    {
      "imageName": "ubuntu@sha256:c95a8e48bf...", [only for image
 vulnerabilities]
      "imageID": "sha256:f643c72bc25212974c1...", [only for image
 vulnerabilities]
      "hostname": "console.compute.internal", [only for host
 vulnerabilities]
      "distribution": "Ubuntu 20.04.1 LTS",
      "labels": {
        "key1": "value1",
        "key2": "value2"
      },
      "collections": [
        "All",
        "collection1",
        "collection2"
      ],
      "newVulnerabilities": [
        {
          "severity": "High",
          "vulnerabilities": [
            {
              "cve": "CVE-2020-1971",
              "severity": "high",
              "link": "https://people.canonical.com/~ubuntu-security/
cve/2020/CVE-2020-1971",
              "status": "Fixed in: 1.0.1f-1ubuntu2.27+esm2",
              "packages": "openssl",
              "packageVersion": "1.0.1f-1ubuntu2.27"
            },
            ... more vulnerabilities
          ]
        },
        {
          "severity": "Low",
          "vulnerabilities": [
            {
              "cve": "CVE-2019-25013",
              "severity": "low",
              "link": "https://people.canonical.com/~ubuntu-security/
cve/2019/CVE-2019-25013",
              "status": "needed",
              "packages": "libc-dev-bin,libc6-dev,libc6,libc-bin",
              "packageVersion": "2.31-0ubuntu9.1",
              "sourcePackage": "glibc"
            },
            ... more vulnerabilities
          ]
        }
      ]
    },
```



```
    ... more images/hosts
  ]
}
```

```
{
    "complianceIssues": [
    {
      "title": "(CIS_Docker_v1.2.0 - 4.1) Image should be created
 with a non-root user",
      "id": "41",
      "description": "It is a good practice to run the container as a
 non-root user, if possible...",
      "type": "image",
      "category": "Docker",
      "severity": "high"
    },
      "title": "Private keys stored in image",
      "id": "425",
      "description": "",
      "type": "image",
      "category": "Twistlock Labs",
      "severity": "high",
      "cause": "Found: /usr/share/npm/node_modules/agent-base/..."
    },
    ... more compliance issues
  ]
}
```

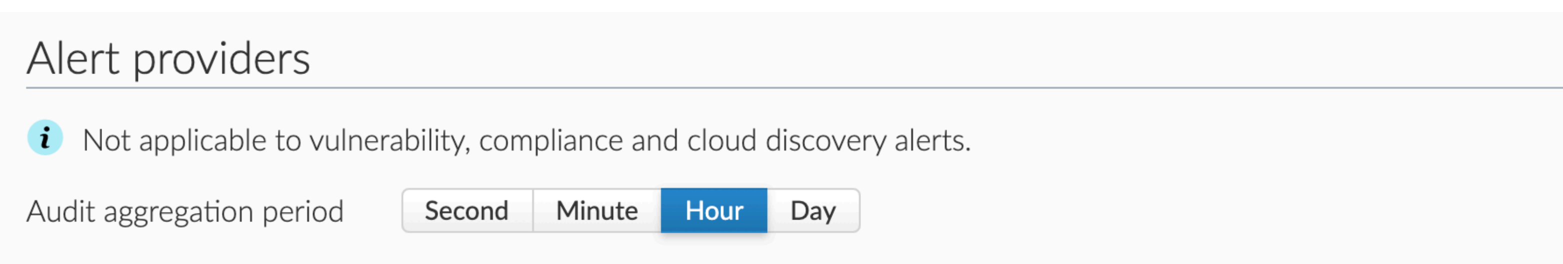

Configuring alert frequency

You can configure the rate at which alerts are emitted. This is a global setting that controls the spamminess of the alert service. Alerts received during the specified period are aggregated into a single alert. For each alert profile, an alert is sent as soon as the first matching event is received. All subsequent alerts are sent once per period.

STEP 1 | Open Console, and go to **Manage > Alerts**.

STEP 2 | In **Aggregate audits every**, specify the maximum rate that alerts should be sent.

You can specify **Second**, **Minute**, **Hour**, **Day**.

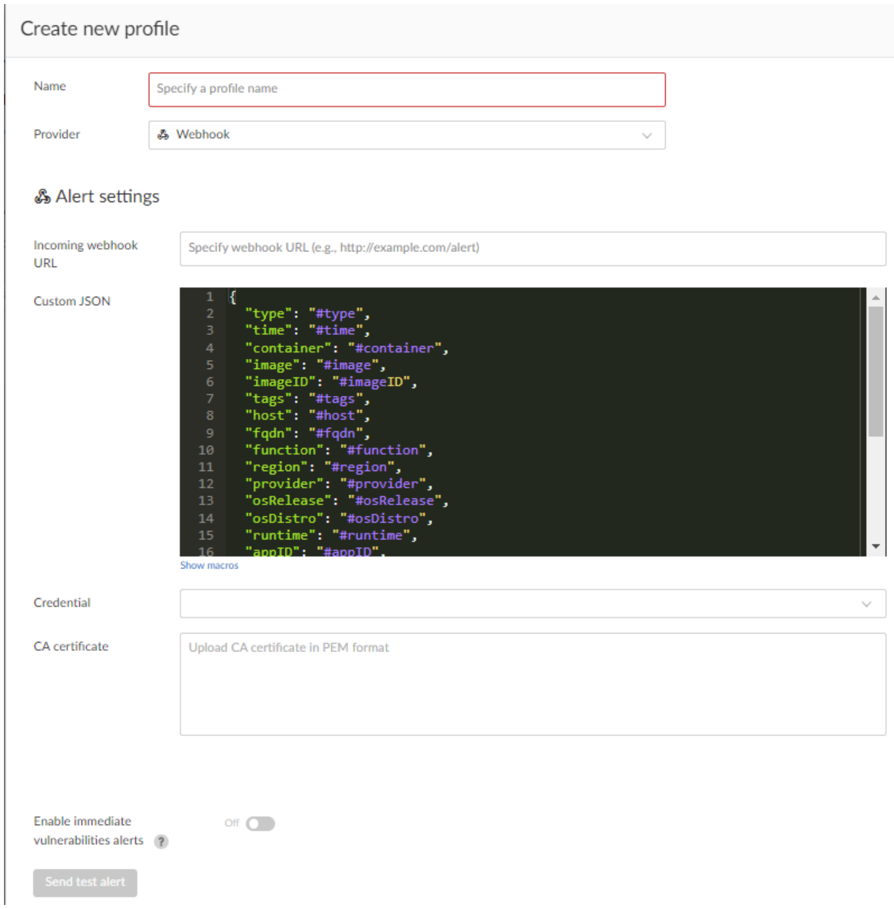


Sending alerts to a webhook

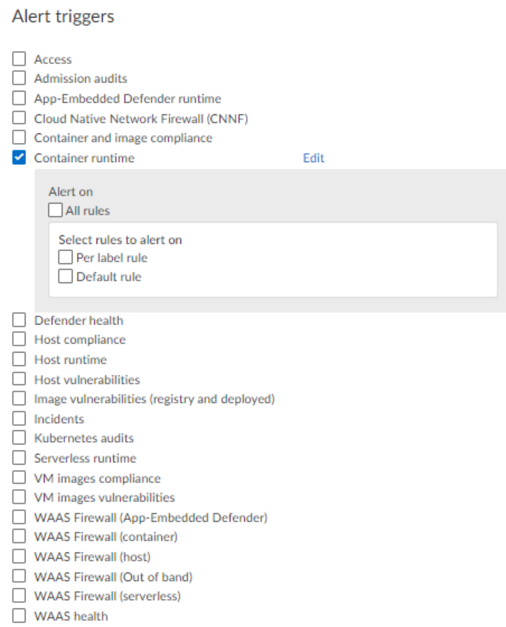
Alert profiles specify which events should trigger the alert machinery, and to which channel alerts are sent. You can send alerts to any combination of channels by creating multiple alert profiles.

Alert profiles consist of two parts:

(1) Alert settings — Who should get the alerts, and on what channel? Configure Prisma Cloud to integrate with your messaging service and specify the people or places where alerts should be sent. For example, configure the email channel and specify a list of all the email addresses where alerts should be sent. Or for JIRA, configure the project where the issue should be created, a long with the type of issue, priority, assignee, and so on.



(2) Alert triggers — Which events should trigger an alert to be sent? Specify which of the rules that make up your overall policy should trigger alerts.



If you use multi-factor authentication, you must create an exception or app-specific password to allow Console to authenticate to the service.

Create new alert channel

Create a new alert channel.

Prerequisites: You have a service to accept Prisma Cloud’s callback. For purely testing purposes, consider PostBin or RequestBin.

STEP 1 | In **Manage > Alerts**, click **Add profile**.

STEP 2 | Enter a name for your alert profile.

STEP 3 | In **Provider**, select **Webhook**.

Configure the channel

Configure the channel.

STEP 1 | In **Webhook incoming URL**, enter the endpoint where Prisma Cloud should submit the alert.

STEP 2 | In **Custom JSON**, Enter the structure of the JSON payload that your web application is expecting.

For more details about the type of data in each field, click **Show macros**.

STEP 3 | (Optional) In **Credential**, specify a basic auth credential if your endpoint requires authentication.

STEP 4 | (Optional) In **CA Certificate**, enter a CA cert in PEM format.

When using a CA cert to secure communication, only one-way SSL authentication is supported. If two-way SSL authentication is configured, alerts will not be sent.

STEP 5 | Click **Send Test Alert** to test the connection. An alert is sent immediately.

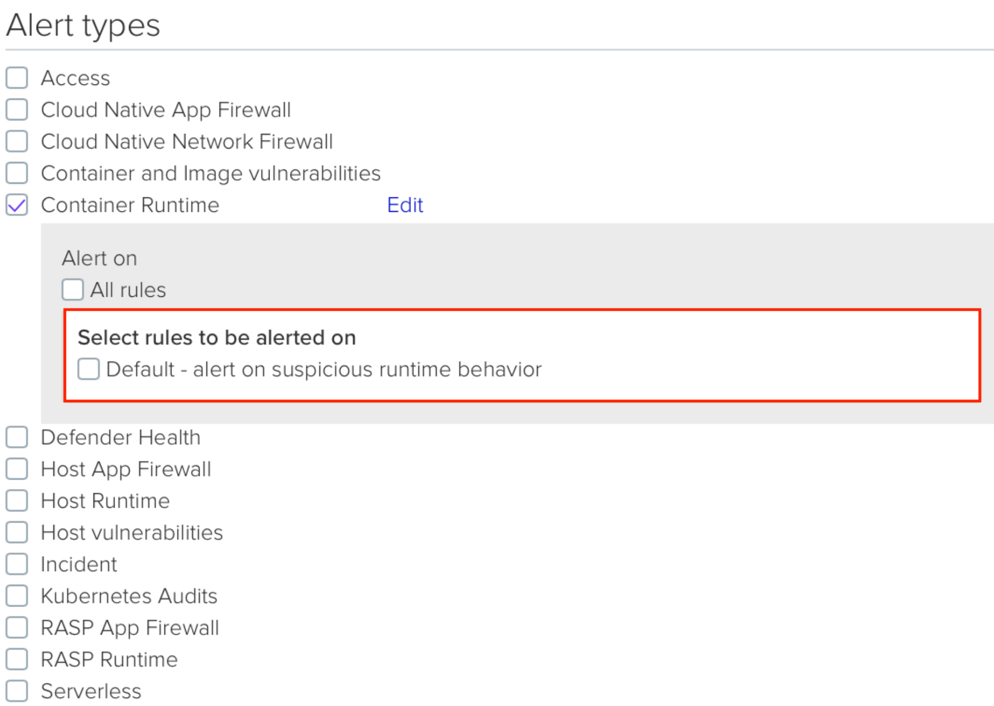
Configure the triggers

Configure how the alert is triggered.

STEP 1 | Under **Alert Types**, check the boxes types of events that should trigger an alert.

STEP 2 | For additional configuration options, click **Edit**.

STEP 3 | To specify specific rules that should trigger an alert, deselect **All rules**, and then select any individual rules.



STEP 4 | Click **Save**.

Audit

Edit on GitHub

Prisma Cloud creates and stores audit event records (audits) for all major subsystems. Audits can be reviewed in Monitor > Events, or they can be retrieved from the Prisma Cloud API. If you have a centralized syslog collector, you can integrate Prisma Cloud with your existing infrastructure by configuring Prisma Cloud to send all audit events to syslog in RFC5424-compliant format.

- Event viewer
- Host activity
- Administrative activity audit trail
- Annotate audit event records
- Delete audit logs
- Syslog and stdout integration
- Log rotation
- Throttling audits
- Prometheus
- Kubernetes auditing

Event viewer

Edit on GitHub

Prisma Cloud creates and stores audit event records (audits) for all major subsystems. Audits can be reviewed in **Monitor > Events**, or they can be retrieved from the Prisma Cloud API. If you have a centralized syslog collector, you can integrate Prisma Cloud with your existing infrastructure by configuring Prisma Cloud to send all audit events to syslog in RFC5424-compliant format.

You can review some of the limits on storing audit events.

When you're reviewing audits in a dialog, the list of audits isn't updated in real-time. To retrieve all the latest data, close the dialog. If the ***Refresh*** *button is decorated with a red indicator, click it to refresh the view with the latest data, then reopen the dialog.*

Access audits

Access to any container resource protected by Defender is logged and aggregated in Console. You can also configure Prisma Cloud to record audits for sudo, SSH, and other events that are executed on hosts protected by Defender. This audit trail links access to system components to individual users. Access events can be viewed in Console under **Monitor > Events**.

Runtime audits

Prisma Cloud records an audit every time a runtime sensor (process, network, file system, and system call) detects activity that deviates from the sum of the predictive model plus any runtime rules you've defined. For example, a file system audit event is emitted when Prisma Cloud detects malware in a container. Runtime events for containers can be viewed in Console under **Monitor > Events**. Runtime events for hosts can be viewed in Console under **Monitor > Events**.

Firewall audits

Web Application and API Security (WAAS) is a layer 7 filtering engine that ensures only safe, clean traffic ever reaches your web app. Audits are generated when WAAS detects an attack, such as SQL injection or cross-site scripting. WAAS audits can be viewed under **Monitor > Events**.

Admin activity

All Prisma Cloud administrative activity can viewed under **Manage > View Logs**.

Prisma Cloud limits viewing of audit trails to those with a job-related need. To view audit events, you must log into Console. Only users with Administrator, Operator, Defender Manager, or Auditor roles can view audit data in Console. Similarly, only users with the above-mentioned roles can retrieve audit data from the Prisma Cloud API.

Host activity

Edit on GitHub

Prisma Cloud lets you audit security-related activity on hosts protected by Defender.

Runtime rules specify the type of activity to capture. The default host runtime rule, *Default - alert on suspicious runtime behavior*, assesses interactive user activity. You can create additional runtime rules to control which type of events are captured on which hosts.

The following types of activity can be assessed and captured.

- **Docker** — Docker commands that alter state: create, run, exec, commit, save, push, login, export, kill, start, stop, and tag.
- **Read-only Docker events** — When you configure Prisma Cloud to capture Docker commands, you can optionally capture commands that simply read state. These include *docker ps* and *docker images*.
- **New sessions spawned by sshd** — Self-explanatory.
- **Commands run with sudo or su** — Self-explanatory.
- **Log activity from background apps** — Processes run by services on the host that could raise security concerns. Activities include: service restart, service install, service modified, cron modified, system update, system reboot, package source modified, package source added, iptables changed, secret modified, accounts modified, and sensitive files modified.

Whereas Defender's runtime system surfaces suspect activity by sifting through events, Defender's forensics system presents a raw list of all spawned processes.

Enabling audits for local events

To enable audits for host activity, create a new host runtime rule. After making your changes, you can view all audits in **Monitor > Events** with the **Host Activities** filter.

Auditing begins after a rule is created. Any events that occurred before the rule was created are not recorded.

STEP 1 | Open Console.

STEP 2 | Go to **Defend > Runtime > Host Policy**.

STEP 3 | Click **Add rule**, and give it a name.

STEP 4 | In **Hosts**, specify the hosts for which this rule applies.

STEP 5 | In the **Activities** tab, enable the events for which you want audits.

STEP 6 | Click **Save**.

Administrative activity audit trail

Edit on GitHub

All Prisma Cloud administrative activities are logged.

Changes to any settings (including previous and new values), changes to any rules (create, modify, or delete), changes to the credentials (create,modify, or delete), and all logon activity (success and failure) are logged. For every event, both the user name and source IP are captured.

Audit records for App-Embedded runtime audits, Trust audits, Container network firewall audits, and Host network firewall audits are retained for up to 25,000 entries or 50 MB, whichever limit is met first.

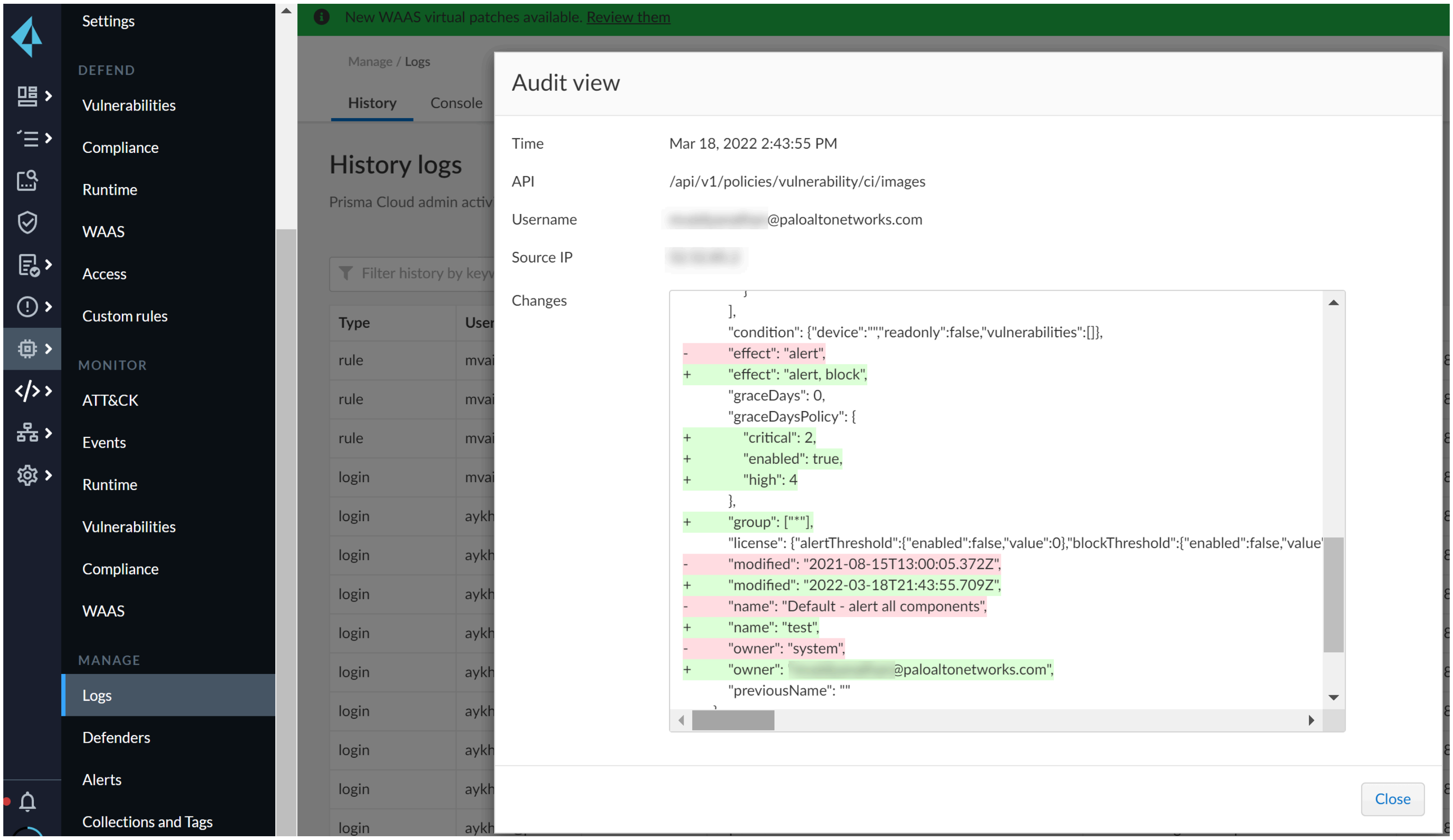
For login activity, the following events are captured:

- Every login attempt from the login page, including failures.
- Every failed attempt to authenticate to the API. Successfully authenticated calls to the API are not recorded.

The full set of log data is available to anyone with a user role of auditor or higher.

To view the administrative history, open Console, then go to **Manage > Logs > History**.

Settings, credentials, and rule events show how a configuration has changed. You can review the API endpoint, and a diff of the previous and current JSON objects. The following screenshot shows the changes to a vulnerability management rule:



Use the API reference to view information on the API endpoint. The */api/22.01/policies/vulnerability/ci/images* endpoint creates and modifies vulnerability rules for images scanned in the CI process. In this case, user *name obscured* has changed the threshold for the grace period for fixing Critical and High severity CVEs to 2 days and 4 days respectively after the vulnerability was published or disclosed.

Annotate audit event records

Edit on GitHub

Prisma Cloud lets you surface and display designated labels in events and reports. For example, you might already use labels to classify resources according to team name or cost center. With *alert labels*, you can specify which of these key-value pairs are appended to events (audits, incidents, syslog, alerts) and reports.

Labels are key-value string pairs that can be attached to objects such as images, containers, or pods. In Console, specify a list of Docker and Kubernetes labels that contain the metadata you want to append to Prisma Cloud events. When an event fires, if the associated object has any of the specified labels, they are appended to the event.

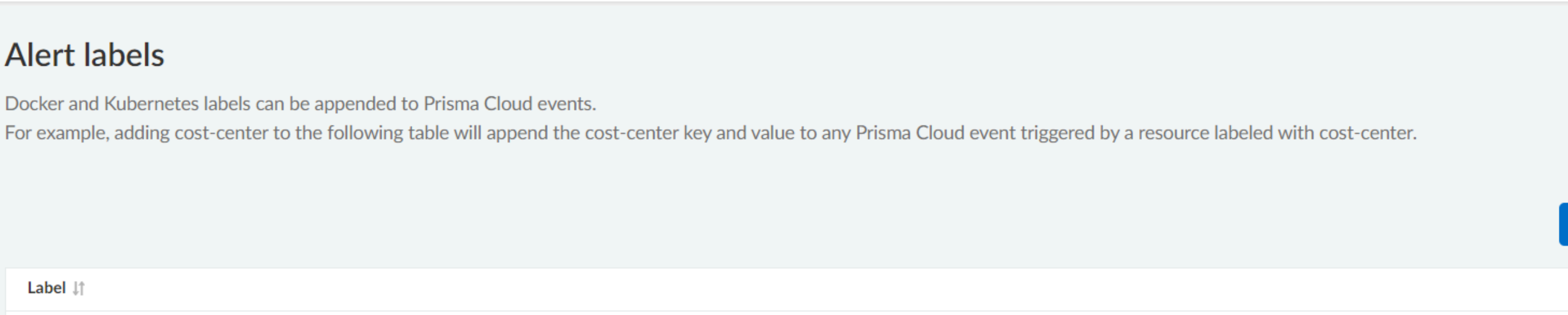
Specifying labels to append to Prisma Cloud events

Specify which labels to append to Prisma Cloud events.

STEP 1 | Open Console.

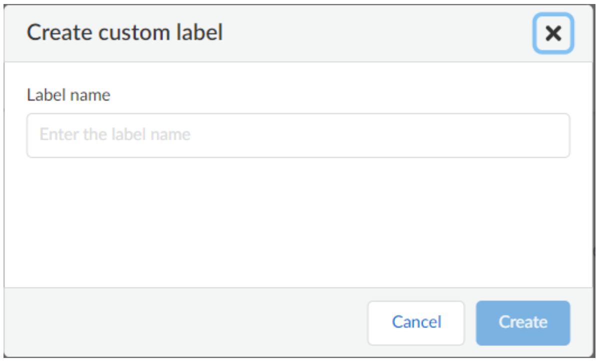
STEP 2 | Go to **Manage > Alerts > Alert Labels**.

STEP 3 | Click **Add Label**.



STEP 4 | Enter the name of the label to be appended to Prisma Cloud events.

STEP 5 | Click **Create**.



Email alerts

The contents of a label can be used as a dynamic target for email alerts. Specify the labels that contain a comma delimited list of email addresses, and when an event fires, the recipients will be notified.

Before setting up your email alerts, be sure you've specified a list of labels to be appended to Prisma Cloud events, where at least one label contains a comma-delimited list of email addresses.

Kubernetes labels don't support special characters, such as @, which are required to specify email addresses. Therefore, only Docker labels can be used as a dynamic address list for email alerts.

Configure email alerts

JIRA alerts

The contents of a label can be used to dynamically specify project keys, JIRA labels, and assignees for new JIRA issues.

Before setting up your JIRA alerts, be sure you've specified a list of labels to be appended to Prisma Cloud events, where the labels contain the type of information you need to dynamically route JIRA issues to the right team.

Configure JIRA alerts

Delete audit logs

Edit on GitHub

Delete audits from the log using the Prisma Cloud API.

Delete all access audit events

Deleting audit log entries is done through API calls only.

Path

```
DELETE /api/v1/audits/access?type=[type]
```

Description

Deletes all access events of a specific type. In case type is not provided all access audits for every type will be removed. The possible 'types' for this command are:

- *docker*: Docker access audit
- *kubernetes*: Kubernetes access audit (to Kubernetes master)
- *sshd*: SSH audit to host
- *sudo*: sudo commands audit on host

Status codes

- *200* - no error
- *400* - bad request was provided

Example request

```
curl -X DELETE  -u admin:<Password>  'https://<localhost>:8443/api/
v1/audits/access?type=docker'
```

Example response

```
{}
```

Delete access audit event

Deleting audit log entries is done through API calls only.

Path

```
DELETE /api/v1/audits/access/[id]
```

Description

Deletes an access event with specific id.

Status codes

- *200* - no error

- *400* - bad request was provided

Example request

```
curl -X DELETE -u admin:<Password>  'https://<localhost>:8443/api/v1/
audits/access/580fd342b8aaba1000ec47be'
```

Example response

```
{}
```

The current set up enables user to delete entries at the access layer for each runtime sensor. To learn more on API calls, see the API reference.

Syslog and stdout integration

Edit on GitHub

You can configure Prisma Cloud to send audit event records (audits) to syslog and/or stdout for Console and Defender based on whether you have Prisma Cloud Compute Edition or Prisma Cloud Enterprise Edition.

With the Prisma Cloud Compute Edition, you can configure Prisma Cloud to send audit event records (audits) to syslog and/or stdout.

Syslog integration must be turned on manually. Open Console, go to **Manage > Alerts > Logging**, then set **Syslog** to **Enabled**. Prisma Cloud connects to the syslog socket on */dev/log*. Stdout integration can be enabled from the same tab.

When you enable syslog or stdout integration, you can optionally enable verbose output. Verbose output records vulnerability and compliance issues in your environment. It also records all process activity.

In general, enabling verbose output is not recommended because of the substantial overhead. You can retrieve this data much more efficiently from the Prisma Cloud API. Nevertheless, sometimes this capability is expressly required for integration with SIEM tools.

Do not enable both syslog and stdout on hosts with systemd. With systemd, anything sent to stdout gets logged to syslog. With both syslog and stdout enabled, you would get duplicate messages in syslog.

Sending syslog messages to a network endpoint

Writing to */dev/log* sends logs to the local host's syslog daemon. The syslog daemon can then be optionally configured to forward those logs to a remote syslog or SIEM server. If you don't have access to the underlying host, you can configure Prisma Cloud Console to send log messages directly to your remote system.

In most cases, you won't need to specify a network endpoint in order to send syslog messages to your SIEM tool. If you already have log collectors on your hosts, simply enable syslog. Your log collectors will stream Prisma Cloud syslog messages to your SIEM tool.

Some things to keep in mind:

- Console sends logs directly to your remote server. When configuring Console with the remote server, validate that the address you enter is actually reachable from the host where Console runs. Otherwise, you risk losing log messages.
- Because Console sends messages directly to your remote server, and not through the local syslog daemon, you don't get some of syslog's built-in benefits, such as buffering, which protects against network outages and service failures.
- The classic syslog implementation sends logs over UDP. This is considered a bad practice if your logs have any value. UDP is connectionless. Packets are sent to their destination without confirming that they were received. TCP's stateful connections and retransmission capabilities make it more appropriate for shuttling logs to a SIEM.

STEP 1 | Log into Console.

STEP 2 | Go to **Manage > Alerts > Logging**.

STEP 3 | Set **Syslog** to **Enabled**.

STEP 4 | In **Send syslog messages over the network to**, click **Edit**, and then specify a destination.

Appending custom strings to syslog messages

You can configure Prisma Cloud Compute to append a custom string to all Console and Defender syslog messages.

Custom strings are set in the event message as a key-value pair, where the key is "id", and the value is your custom string. The following screenshot shows a Defender event, where the custom string is "koko".

Configuring a custom string is useful when you have multiple Prisma Cloud Compute deployments (i.e. multiple Compute Consoles) and you're aggregating all messages in a single log management system. The custom string serves as a marker that lets you correlate specific events to specific deployments.

STEP 1 | Open Console.

STEP 2 | Go to **Manage > Alerts > Logging**.

STEP 3 | Set **Syslog** to **Enabled**.

STEP 4 | For **Identifier**, click **Edit**, and enter a string.

Console events

Both Console and Defender emit messages. Console syslog messages are tagged as *Twistlock-Console* in the logs.

The data emitted to syslog and stdout is exactly the same.

Console syslog event types

The following table describes each message type and sub-type.

Syslog Type	Sub Type	Description
image_scan	—	This represents an image scan.
—	containerCompliance	This represents any Compliance findings within the image scan.
—	vulnerability	This represents any Vulnerability findings within the image scan.

Syslog Type	Sub Type	Description
container_scan	—	This represents a Container scan.
—	container	This represents any Compliance findings within the container scan.
vm_scan	—	This represents a VM scan.
—	containerCompliance	This represents any Compliance findings within the vm scan.
—	vulnerability	This represents any Vulnerability findings within the vm scan.
host_scan	—	This represents a Host scan.
—	containerCompliance	This represents any Compliance findings within the host scan.
—	vulnerability	This represents any Vulnerability findings within the host scan.
scan_summary	—	This represents a scan summary. The type of summary is dependent upon subtype below.
—	image	This represents a summary of image Vulnerability and Compliance issues.
—	container	This represents a summary of container Vulnerability and Compliance issues.
—	vm	This represents a summary of vm Vulnerability and Compliance issues.
—	host	This represents a summary of host Vulnerability and Compliance issues.
—	code_repository_scan	This represents a summary of code repository Vulnerability and Compliance issues.
—	registry_scan	This represents a summary of registry Vulnerability and Compliance issues.
—	cloud_scan	This represents a summary of cloud accounts with Compliance Compliance issues.
management_audit	—	This represents any management audit. This is broken out in the subtypes listed below.

Syslog Type	Sub Type	Description
—	login	This represents a login audit.
—	profile	This represents a profile state change audit.
—	settings	This represents a settings change audit.
—	rule	This represents a rule change audit.
—	user	This represents a user change audit.
—	group	This represents a group change audit.
—	credential	This represents a credential change audit.
—	tag	This represents a tag change audit.
kubernetes_audit	—	This represents a Kubernetes audit.
admission_audit	—	This represents an Admission Controller audit.
serverless_runtime_audit	—	This represents a Serverless runtime audit.
serverless_app_firewall_audit	—	This represents a Serverless WAAS audit.
app_embedded_runtime_audit	—	This represents an app embedded runtime audit.
app_embedded_app_firewall_audit	—	This represents an app embedded WAAS audit.
defender_disconnected	—	This represents when a Defender is disconnected.

Image scan

Records when Prisma Cloud scans an image.

Example image scan message:

```
Jul 30 18:51:32 aqsa-root Twistlock-Console[1]:
  time="2019-07-30T18:51:32.214136319Z"
  type="scan_summary"
  log_type="image"

 image_id="sha256:cd14cecfdb3a657ba7d05bea026e7ac8b9abafc6e5c66253ab327c7211fa6
  image_name="aqsa/internal:tag5"
  vulnerabilities="297"
  compliance="1"
```

Container scan

Records when Prisma Cloud scans a container.

Example container scan message:

```
Jul 30 22:06:15 aqsa-root Twistlock-Console[1]:
  time="2019-07-30T22:06:15.804842461Z"
  type="container_scan"
  log_type="container"

 container_id="d29ac3222f430ccf6a7d730db5cec3363d4c608680de881e26e13f9011e36d13
  container_name="twistlock_console"
  image_name="twistlock/private:console_19_07_353"
  compliance="6"
```

Host scan

Records when Prisma Cloud scans a host. Defenders scan the hosts they run on.

Example host scan:

```
Jul 30 22:09:53 aqsa-root Twistlock-Console[1]:
  time="2019-07-30T22:09:53.390680962Z"
   type="scan_summary"
   log_type="host"
   hostname="aqsa-root.c.cto-sandbox.internal"
   vulnerabilities="89"
   compliance="17"
```

Code repository scan

Records when Prisma Cloud scans a code repository.

Example scan:

```
Jul  7 23:34:09 ip-172-31-55-106 Twistlock-Console[1]:
  time="2020-07-07T23:34:09.25109843Z"
  type="scan_summary"
  last_update_time="2020-07-07 23:21:00.203 +0000 UTC"
  log_type="code_repository_scan"
  source="github"
  repository_name="jerryso/apper"
  vulnerable_files="1"
  vulnerabilities="25"
  collections="All"
```

Individual compliance issues

Records a compliance finding. These messages are tagged with *log_type="compliance"*, and are generated as a byproduct of container scans, image scans, host scans, and registry scans.

Compliance issues are only recorded when **Detailed output for vulnerabilities and compliance** is enabled in **Manage > Alerts > Logging** (to see this option, syslog must be enabled).

A syslog entry is generated for each compliance issue. This can result in a significant amount of data, which is why verbose output is disabled by default.

You must have a rule that alerts on compliance issues for an entry to be written to syslog. It might just be the *Default - alert all components* rule, or another custom rule. This option does not simply log all compliance issues irrespective of the rules that are in place.

Example image compliance issue:

```
Jul 30 22:18:53 aqsa-root Twistlock-Console[1]:
  time="2019-07-30T22:18:53.23838464Z"
  type="image_scan"
  log_type="containerCompliance"
  compliance_id="41"
  severity="high"
  description="(CIS_Docker_CE_v1.1.0 - 4.1) Image should be created
 with a non-root user"
  rule="Default - ignore Prisma Cloud components"
  host="aqsa-root.c.cto-sandbox.internal"

 image_id="sha256:a92d9a54137dccb6f78161d4468b21ae4bebe4fc3c772845253a2f8d80a5d
  image_name="twistlock/private:defender_19_03_311"
```

Example container compliance issue:

```
Jul 30 22:22:56 aqsa-root Twistlock-Console[1]:
 time="2019-07-30T22:22:56.871490132Z"
 type="container_scan"
 log_type="containerCompliance"
 compliance_id="526"
 severity="medium"
 description="(CIS_Docker_CE_v1.1.0 - 5.26) Check container health at
 runtime"
 rule="Default - alert on critical and high"
 host="aqsa-root.c.cto-sandbox.internal"
 container_id="22b745b2220f3f128a1cf57d2ffff328a02ba380930ebf83fca9f26d4d2b8aa4
 container_name="serene_cray"
```

Example host compliance issue:

```
Jul 30 22:09:53 aqsa-root Twistlock-Console[1]:
 time="2019-07-30T22:09:53.390585517Z"
 type="host_scan"
 log_type="compliance"
 compliance_id="6518"
 severity="high"
 description="(CIS_Linux_1.1.0 - 5.1.8) Ensure at/cron is restricted
 to authorized users"
 rule="Default - alert on critical and high"
 host="aqsa-root.c.cto-sandbox.internal"
```

Individual vulnerability issues

Records a vulnerability finding. These messages are tagged with *log_type="vulnerability"*, and are generated as a byproduct of image scans, host scans, and registry scans.

Vulnerability issues are only recorded when **Detailed output for vulnerabilities and compliance** is enabled in **Manage > Alerts > Logging**.

A syslog entry is generated for each vulnerability for each package. This can result in a significant amount of data, which is why verbose output is disabled by default.

For example, consider a rule that raises an alert when vulnerabilities of medium severity or higher are found in an image. If there are eleven packages that violate this rule, there will be eleven syslog entries, one for each package.

You must have a rule that alerts on vulnerabilities for an entry to be written to syslog. It might just be the *Default - alert all components* rule, or another custom rule. This option does not simply log all vulnerability data irrespective of the rules that are in place.

Example image vulnerability issue:

```
Jul 30 22:19:11 aqsa-root Twistlock-Console[1]:
  time="2019-07-30T22:19:11.264627256Z"
  type="image_scan"
  log_type="vulnerability"
  vulnerability_id="410"
  description="Image contains vulnerable Python components"
  cve="CVE-2019-11236"
  severity="medium"
  package="urllib3"
  package_version="1.24.1"
  vendor_status="fixed in 1.24.3"
  rule="test"
  host="aqsa-root.c.cto-sandbox.internal"

 image_id="sha256:196601f91030425db810fa57104b041e414b9b963923ad574e74700c3ea82
  image_name="weaveworksdemos/user-db:0.4.0"
```

Example registry image vulnerability issue:

```
Jul 30 22:03:56 aqsa-root Twistlock-Console[1]:
  time="2019-07-30T22:03:56.930640366Z"
  type="registry_scan"
  log_type="vulnerability"
  vulnerability_id="410"
  description="Image contains vulnerable Python components"
  cve="CVE-2019-11236"
  severity="medium"
  package="urllib3"
  package_version="1.24.1"
  vendor_status="fixed in 1.24.3"
  rule="test"
  host="aqsa-root.c.cto-sandbox.internal"

 image_id="sha256:11cd0b38bc3ceb958ffb2f9bd70be3fb317ce7d255c8a4c3f4af30e298aa1
  image_name="aqsa/internal:tag7"
```

Example host vulnerability issue:

```
Jul 30 22:09:53 aqsa-root Twistlock-Console[1]:
  time="2019-07-30T22:09:53.390181271Z"
  type="host_scan"
  log_type="vulnerability"
  vulnerability_id="46"
```

```
    description="Image contains vulnerable OS packages"
    cve="CVE-2017-8845"
    severity="low"
    package="lzo2"
    package_version="2.08-1.2"
    vendor_status="deferred"
    rule="Default - alert all components" host="aqsa-root.c.cto-
sandbox.internal"
```

Admin activity

Changes to any settings (including previous and new values), changes to any rules (create, modify, or delete), and all logon activity (success and failure) are logged. For every event, both the user name and source IP are captured.

Example admin acitivty audit:

```
Jul 30 21:58:16 aqsa-root Twistlock-Console[1]:
    time="2019-07-30T21:58:16.80522678Z"
    type="management_audit"
    log_type="login"
    username="aqsa"
    source_ip="137.83.195.96"
    api="/api/v1/authenticate"
    status="successful login attempt"
```

Defender events

Defender syslog messages are tagged as *Twistlock-Defender* in logs. The data emitted to syslog and stdout is exactly the same.

App-embedded, Serverless, and Windows Defenders do not support Syslog.

Defender syslog event types

The following table describes each event type and sub-type.

Syslog Type	Sub Type	Description
container_runtime_audit		This represents a Container Runtime Audit. Details of Audit type is listed as subtype below.
—	processes	This represents a Container process runtime audit.
—	network	This represents a Container network runtime audit.
—	filesystem	This represents a Container filesystem runtime audit.
host_activity_audit		This represents a Host activity audit.
host_network_firewall_audit		This represents a Host WAAS audit.

Syslog Type	Sub Type	Description
container_app_firewall_audit		This represents a Container WAAS audit.
host_runtime_audit		This represents a Host Runtime Audit. Each audit type is listed as subtype below.
—	processes	This represents a Host process runtime audit.
—	network	This represents a Host network runtime audit.
—	kubernetes	This represents a Host Kubernetes runtime audit.
—	filesystem	This represents a Host filesystem runtime audit.
incident	—	This represents an Incident. Host and Container incidents are differentiated by "host" or "container_id".

Container runtime audit

Activity that breaches your runtime rules or the automatically generated allow lists in your models generates audits. The *log_type* field specifies the runtime sensor that detected the anomaly (filesystem, processes. syscalls, or network).

Example container runtime audit: The following process audit shows that busybox was unexpectedly launched, and an alert was raised.

```
Jul 30 22:41:25 aqsa-root Twistlock-Defender[13460]:
  time="2019-07-30T22:41:25.448709847Z"
  type="container_runtime_audit"

 container_id="73c2e8267f9b80ea152403c36c377476d24e43e211bb098300a317b3d1c472e4
  container_name="/dreamy_rosalind"
 image_id="sha256:94e814e2efa8845d95b2112d54497fbad173e45121ce9255b93401392f538
  image_name="ubuntu:18.04"
  effect="alert"
  msg="High rate of reg file access events, reporting aggregation
 started;
  last event: /usr/lib/apt/methods/gpgv wrote a suspicious file to /
tmp/apt.conf.2ZH7tP.
  Command: /usr/lib/apt/methods/gpgv"
  log_type="filesystem"
  custom_labels="io.kubernetes.pod.namespace:default"
  account_id="prisma-cloud-compute"
  cluster="cluster1"
```

Host runtime audit

Activity that breaches your runtime rules or the automatically generated allow lists in your host services models generates audits.

Example host runtime audit:

```
Jul 30 22:47:12 aqsa-root Twistlock-Defender[13460]:
  time="2019-07-30T22:47:12.325487039Z"
  type="host_runtime_audit"
  service_name="ssh"
  effect="alert"
  msg="Outbound connection by /usr/lib/apt/methods/http to an
 unexpected port: 80 IP: 91.189.91.26. Low severity audit, event is
 automatically added to the runtime model"
  log_type="network"
  account_id="prisma-cloud-compute"
  cluster="cluster1"
```

Access audit

Docker commands run on hosts protected by Defender.

With user access events, you can determine who performed an action, and on which resource.

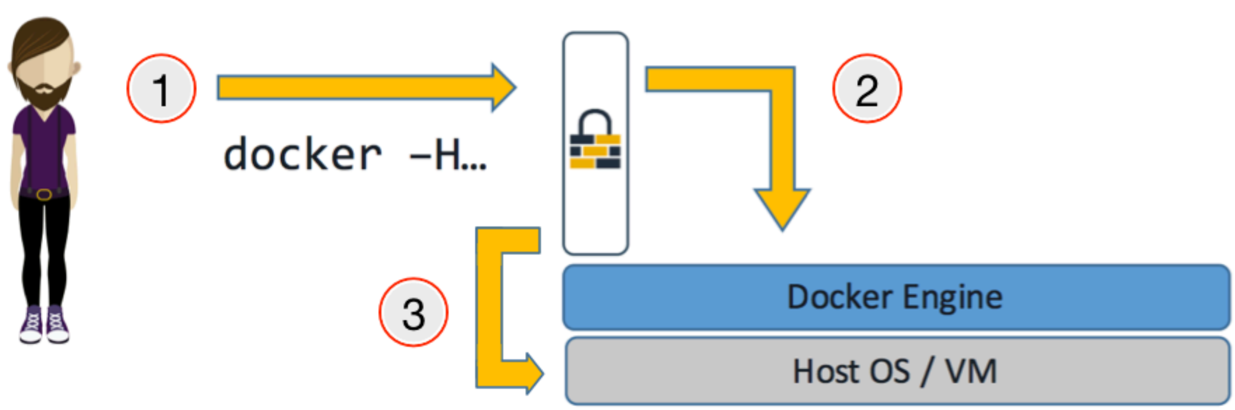
For example:

- [Bruce] [started container X] in the [DEV environment] (allowed).
- [Bruce] [stopped container Y] in the [PROD environment] (denied).

All Docker commands issued to the Docker daemon are intercepted and inspected by Defender to determine if they comply with the policy set in Console.

The following diagram illustrates how Defender operates on the management plane:

1. Bruce, a developer, issues a command, docker -H.
2. Defender checks the command against the policies defined in the Console. If the command is allowed, Defender forwards it to the Docker daemon for execution. If the command is denied, the user is notified.
3. An event is recorded in syslog.



Access audits have the following fields:

- type=access_audit

- user=[String] Identity of the person who ran the command
- action=[String] Docker command requested - API invoked
- action_type=[String] Action type
- allow=[Boolean] true/false - Action was allowed or not.
- rule=[String] Rule matched

Example:

```
Jul 30 23:02:23 aqsa-root Twistlock-Defender[13460]:
  time="2019-07-30T23:02:23.179494498Z"
  type="access_audit"
  user="aqsa"
  action="docker_ping"
  action_type="docker"
  allow="true"
  rule="Default - allow all"
```

App firewall audit (WAAS)

All events associated with WAAS (Web-Application and API Security) rules for container, hosts and app-embedded generate audits.

WAAS serverless events are not registered in the syslog. Events audits will be registered to the syslog in future releases.

WAAS Container and Host rule audits are written to the Defender host's syslog. WAAS App-Embedded rule audits are written to the console's host's syslog.

Message fields for WAAS audit would change based on the deployment type as follows:

- **container_id=[String]** Container id in which the event triggered
- **container_name=[String]** Container name on which the action was performed
- **image_name=[String]** Image name on which the action was performed
- **custom_labels=[String]** User-defined Alert Labels (**Mange > Alerts > Alert Labels**)
- **cluster=[String]** Cluster name in which the event triggered
- **hostname=[String]** host in which the event triggered
- **cluster=[String]** Cluster name in which the event triggered
- **app_id=[String]** app_id in which the event triggered
- **time=[String]** request timestamp
- **type=[String]** type of app_firewall_audit
- **effect=[String]** "alert", "prevent", "ban"
- **msg=[String]** Audit message detailing the event
- **log_type=[String]** Attack Type
- **source_ip=[String]** source IP address from the request originated
- **source_country=[String]** country associated with source IP address

- **connecting_ips=[CSV]** list of IPs included in the *X-Forwarded-For* header
- **request_method=[String]** HTTP Request Method
- **request_user_agents=[String]** user-agent string parsed from the *User-Agent* header
- **request_host=[String]** HTTP hostname in the request
- **request_url=[String]** request url
- **request_path=[String]** request path
- **request_query=[String]** request query string
- **request_header_names=[String]** ordered list of HTTP request headers
- **response_header_names=[String]** ordered list of HTTP response headers
- **status_code=[String]** HTTP response status code in the server response

In addition, message structure is subject for the following changes:

- Fields containing empty values are omitted from the message i.e. if a HTTP message does not contain a query field the request_query field will not be present in the message.
- **connecting_ips** - present only if *X-Forwarded-For* Header is present in the request.
- **status_code** - present only for audits created for the "Track Server Error Response Codes" and "Detect Information Leakage" protections
- **response_header_names** - present only for audits created for the "Track Server Error Response Codes" and "Detect Information Leakage" protections.
- **source_country** - present only if resolution was successful.
- **container_name** - will be replaced by **host_id** or **function_id**

Example:

```
Jul 16 20:10:16 cnaf-nightly-build Twistlock-Defender[1947]:
  time="2020-07-16T20:10:16.706085135Z"
  type="container_app_firewall_audit"

 container_id="0a16b4e4dbefc6ef8cc6a08d038e775a8523ad053416730f01eafbf2dee2e693
  container_name="/nginx"
  image_name="nginx:latest"
  effect="prevent"
  msg="Client exceeded violations within 1m. Banning client for 5m"
  log_type="violations exceeded"
  source_ip="12.34.56.78"
  source_country="IL"
  connecting_ips="11.22.33.44"
  request_method="HEAD"
  request_user_agents="curl/7.54.0"
  request_host="www.example.com"
  request_url="www.example.com/?id=../etc/passwd"
  request_path="/"
  request_query="id=../etc/passwd"
  request_header_names="X-Forwarded-For,User-Agent,Accept"
  response_header_names="Set-Cookie,Date,Content-Type,Content-Length
 X-Frame-Options"
  status_code="404"
```


Process activity audit

Records all processes spawned in a container.

Process audits are only recorded when **Detailed output of all runtime process activity** is enabled in **Manage > Alerts > Logging**.

Note that process activity that breaches your runtime policy is separately audited. For more infomration, see the container runtime audit section.

This audit has the following fields:

- type=process
- pid=Process ID
- path=Path to the executable in the container file system
- interactive=Whether the process was spawned from a shell session: true or false
- container-id=Container ID

Example: This audit shows that busybox was spawned in the container with ID 8c5b3fe0037d.

```
Jul 30 22:06:03 aqsa-root Twistlock-Defender[13460]:
  time="2019-07-30T22:06:03.515319204Z"
  type="process"
  pid="20859"
  path="/bin/df"
  interactive="false"

 container_id="3491b03544a51c60e176e54a5077161f14dbc850bf069cf7a096db028e9981de
```

Incidents

Incidents are logical groupings of events, related by context, that reveal known attack patterns.

Example container incident:

```
Jul 30 22:41:24 aqsa-root Twistlock-Defender[13460]:
  time="2019-07-30T22:41:24.987209676Z"
  type="incident"

 container_id="73c2e8267f9b80ea152403c36c377476d24e43e211bb098300a317b3d1c472e4
  image_name="ubuntu:18.04"
  host="aqsa-root.c.cto-sandbox.internal"
  incident_category="hijackedProcess"
  custom_labels="io.kubernetes.pod.namespace:default"
  account_id="prisma-cloud-compute"
  cluster="cluster1"
```

Example host incident:

```
Mar  5 00:26:42 itay-ThinkPad-P50 Twistlock-Defender[22797]:
  time="2018-03-05T00:26:42.894707831+02:00"
  type="incident"
  service_name="http-service"
  host="itay-ThinkPad-P50"
  incident_category="serviceViolation"
```



```
audit_ids="5a9c72a223d020590de74db5"
account_id="prisma-cloud-compute"
cluster="cluster1"
```

Rate limiters

Depending on your configuration, Prisma Cloud can produce a lot of logs, especially in environments with many hosts, images, and containers. By default, most syslog daemons throttle logging with a rate limiter.

If you have a large environment (hundreds of Defenders with tens of images per host) AND you have configured Prisma Cloud for verbose syslog output, you will need to tune the rate limiter. Otherwise, you might find that logs are missing.

For example, on RHEL 7, you must tune both systemd-journald's *RateLimitInterval* and *RateLimitBurst* settings and rsyslog's *imjournalRatelimitInterval* and *imjournalRatelimitBurst* settings. For more information about RedHat settings, see How to disable log rate-limiting in Red Hat Enterprise Linux 7.

Truncated log messages

Very long syslog events can get truncated. For example, changing settings in Console generates management_audits events, which show a diff between old settings and new settings. For policies changes, the diff can be big. Linux log managers limit the number of characters logged per line, and so long messages, such as management audits, can be truncated.

If you've got truncated log messages, increase the log manager's default string size limit. There are several types log managers, but rsyslog is popular with most distributions. For rsyslog, the default log string size is 1024 characters per line. To increase it, open */etc/rsyslog.conf* and set the maximum message size:

```
$MaxMessageSize 20k
```


Log rotation

Edit on GitHub

Both Console and Defender call *log-rotate* every 30 minutes. The options passed to log-rotate are described below.

Defender

The default path for Defender's log file is */var/lib/twistlock/log/defender.log*.

It is configured as follows:

- Truncate the original log file in place after creating a copy, instead of moving the old log file. (*copytruncate*)
- Have 10 backup files rotated. If rotation exceeds 10 files, the oldest rotated file is deleted. (*rotate 10*)
- Don't generate an error in case a log file doesn't exist. (*missingok*)
- Don't rotate the log in case it's empty. (*notifempty*)
- Rotate the log only if its size is 100M or more. (*size 100M*)
- Compress the rotated logs. (*compress*)

Console

The default path for Console's log file is */var/lib/twistlock/log/console.log*. It is configured as follows:

- Truncate the original log file in place after creating a copy, instead of moving the old log file. (*copytruncate*)
- Have 10 backup files rotated. If rotation exceeds 10 files, the oldest rotated file is deleted. (*rotate 10*)
- Don't generate an error in case a log file doesn't exist. (*missingok*)
- Don't rotate the log in case it's empty. (*notifempty*)
- Rotate the log only if its size is 100M or more. (*size 100M*)
- Compress the rotated logs. (*compress*)

DB logs

We log CRITICAL/ERROR messages to enable critical DB diagnostics.

This is automatically done by Prisma Cloud and is non-configurable.

Throttling audits

Edit on GitHub

When your runtime models aren't completely tuned, you can get a barrage of false positives. It's difficult for operators to parse through so many audits, especially when most of it is noise. And the volume and rate of audits can degrade your system.

To address the problem, Console presents a cross-section of the most important audits, while dropping redundant audits. Prisma Cloud collects, collates, and throttles audits on a per-profile (model) basis, with a maximum of 100 audits per profile, sorted by recency. Every audit is categorized by Type and Attack Type, where a Type can have one or more Attack Types. For example, the Network Type has the following Attack Types (not a complete list):

Type	Attack Type	Description
Network	Feed DNS	DNS query of a high risk domain based on data in the Intelligence Stream.
Network	Unexpected Listening Port	Container process is listening on an unexpected port.
Network	etc.	etc.

When there's a large number of incoming audits, Prisma Cloud temporarily applies throttling. When more than five audits of the same Attack Type are received over a short period of time, those audits are dropped. A running count of all audits (dropped and not dropped) is updated periodically. If no audits are received after a grace period, throttling is disabled. Throttling is reset every 24 hours. That is, if throttling is applied for all day 0, and five audits of a given attack have already been received, then no new audits for that Attack Type are displayed for 24 hours. At the 24 hour period mark, throttling is disabled, and any new audits are collected, collated, and presented, until throttling is reapplied.

Throttling is applied to audits in the following systems:

- **Monitor > Events > Container Audits**
- **Monitor > Events > Host Audits**
- **Monitor > Events > Cloud Native App Firewall**
- **Monitor > Events > WAAS for Hosts**

Note that a comprehensive list of audits can always be found in the Defender logs. If syslog and/ or stdout integration is enabled, all audits are always emitted there too. Finally, if you set up alerts on all container runtime rules, you'll get all audits to your alert channel; nothing is dropped or throttled.

Finally, if audits are being throttled, it's a symptom of a larger issue. You should tune your runtime models.

Prometheus

Edit on GitHub

Prometheus is a monitoring platform that collects metrics from targets by scraping their published endpoints. Prisma Cloud can be configured to be a Prometheus target.

You can use Prometheus to monitor time series data across your environment and show high-level, dashboard-like, stats to visualize trends and changes. Prisma Cloud's instrumentation lets you track metrics such as the total number of connected Defenders and the total number of container images in your environment being protected by Defender.

Metrics

Metrics are a core Prometheus concept. Instrumented systems expose metrics. Prometheus stores the metrics in its time-series database, and makes them easily available to query to understand how systems behave over time.

Prisma Cloud has two types of metrics:

- Counters: Single monotonically increasing values. A counter's value can only increase or be reset to zero.
- Gauges: Single numerical values that can arbitrarily go up or down.

Prisma Cloud metrics

All Prisma Cloud metrics are listed in the following table. Vulnerability and compliance metrics are updated every 24 hours. The rest of the metrics are updated every 10 minutes.

Note that *_vulnerabilities and *_compliance metrics report how many entities (images, containers, hosts, etc) are at risk by the highest severity issue that impacts them. In other words, images_critical_vulnerabilities is not a total count of critical vulnerabilities in the images in your environment. Rather, it is a total count of images where the highest severity CVE is critical. For a thorough explanation of how this type of metric is used, see Vulnerability Explorer.

Metric	Type	Description
totalDefenders	Gauge	Total number of Defenders connected to Console. Connected and disconnected Defenders can be reviewed in Console under **Manage > Defenders > Manage**.
activeDefenders	Gauge	Total number of all Defenders for which a license is allocated, regardless of whether it is currently connected to Console or not.
images_critical_vulnerabilities	Gauge	Total number of containers impacted by critical vulnerabilities.

Metric	Type	Description
images_high_vulnerabilities	Gauge	Total number of containers impacted by high vulnerabilities.
images_medium_vulnerabilities	Gauge	Total number of containers impacted by medium vulnerabilities.
images_low_vulnerabilities	Gauge	Total number of containers impacted by low vulnerabilities.
hosts_critical_vulnerabilities	Gauge	Total number of hosts impacted by critical vulnerabilities.
hosts_high_vulnerabilities	Gauge	Total number of hosts impacted by high vulnerabilities.
hosts_medium_vulnerabilities	Gauge	Total number of hosts impacted by medium vulnerabilities.
hosts_low_vulnerabilities	Gauge	Total number of hosts impacted by low vulnerabilities.
serverless_critical_vulnerabilities	Gauge	Total number of serverless functions impacted by critical vulnerabilities.
serverless_high_vulnerabilities	Gauge	Total number of serverless functions impacted by high vulnerabilities.
serverless_medium_vulnerabilities	Gauge	Total number of serverless functions impacted by medium vulnerabilities.
serverless_low_vulnerabilities	Gauge	Total number of serverless functions impacted by low vulnerabilities.
images_critical_compliance	Gauge	Total number of images impacted by critical compliance issues.
images_high_compliance	Gauge	Total number of images impacted by high compliance issues.
images_medium_compliance	Gauge	Total number of images impacted by medium compliance issues.
images_low_compliance	Gauge	Total number of images impacted by low compliance issues.
containers_critical_compliance	Gauge	Total number of containers impacted by critical compliance issues.

Metric	Type	Description
containers_high_compliance	Gauge	Total number of containers impacted by high compliance issues.
containers_medium_compliance	Gauge	Total number of containers impacted by medium compliance issues.
containers_low_compliance	Gauge	Total number of containers impacted by low compliance issues.
hosts_critical_compliance	Gauge	Total number of hosts impacted by critical compliance issues.
hosts_high_compliance	Gauge	Total number of hosts impacted by high compliance issues.
hosts_medium_compliance	Gauge	Total number of hosts impacted by medium compliance issues.
hosts_low_compliance	Gauge	Total number of hosts impacted by low compliance issues.
serverless_critical_compliance	Gauge	Total number of serverless functions impacted by critical compliance issues.
serverless_high_compliance	Gauge	Total number of serverless functions impacted by high compliance issues.
serverless_medium_compliance	Gauge	Total number of serverless functions impacted by medium compliance issues.
serverless_low_compliance	Gauge	Total number of serverless functions impacted by low compliance issues.
active_app_firewalls	Gauge	Total number of active app firewalls (WAAS).
app_firewall_events	Gauge	Total number of app firewall (WAAS) events.
protected_containers	Gauge	Total number of protected containers.
container_runtime_events	Gauge	Total number of container runtime events.
host_runtime_events	Gauge	Total number of host runtime events.
access_events	Gauge	Total number of access events.
api_requests	Counter	Total number of requests to the Prisma Cloud API.

Metric	Type	Description
defender_events	Counter	Total number of events sent by all Defenders to Console.

Integrating Prisma Cloud with Prometheus

The Prometheus server scrapes endpoints at configurable time intervals. Prisma Cloud refreshes vulnerability and compliance data every 24 hours. All other data is refreshed every 10 minutes. Regardless of the value you set for the Prometheus scrape interval, new Prisma Cloud data is only available at our refresh rates.

This procedure shows you how to enable Prisma Cloud's Prometheus integration and spin up a Prometheus server running in a container. If you already have a Prometheus server in your environment, all you need is the Prisma Cloud scrape configuration.

STEP 1 | Enable Prisma Cloud's Prometheus instrumentation.

1. Log into Prisma Cloud Console.
2. Go to **Manage > Alerts > Logging**.
3. Set **Prometheus instrumentation** to **Enabled**.

STEP 2 | Prepare a scrape configuration file for the Prometheus server.

1. Create a new file named *prometheus.yml*, and open it for editing.
2. Enter the following configuration, where:
    - CONSOLE_ADDRESS is the DNS name or IP address for Prisma Cloud Console.
    - USER is a Prisma Cloud user, with the minimum role of Auditor.
    - PASS is the user's password.

```
global:
  scrape_interval:     15s # Set the scrape interval to
 every 15 seconds. Default is every 1 minute.
  evaluation_interval: 15s # Evaluate rules every 15
 seconds. The default is every 1 minute.

# Prisma Cloud scrape configuration.
scrape_configs:
  - job_name: 'twistlock'
    static_configs:
    - targets: ['CONSOLE_ADDRESS:8083']
    metrics_path: /api/v1/metrics
    basic_auth:
      username: 'USER'
      password: 'PASS'
```

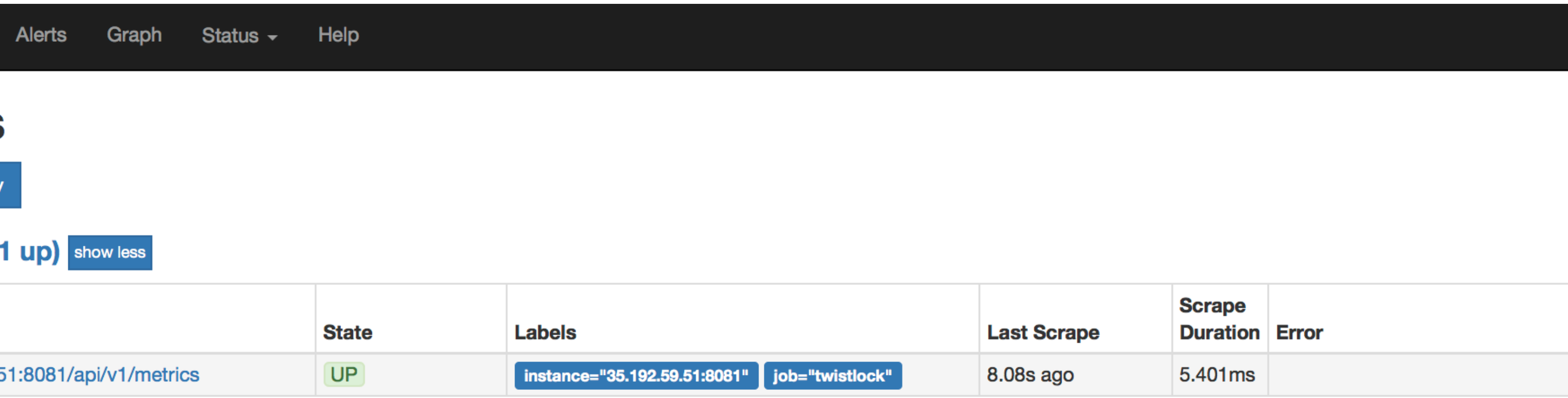
STEP 3 | Start the Prometheus server with the scrape configuration file.

```
$ docker run \
  --rm \
  --network=host \
```



```
  -p 9090:9090 \
  -v /PATH_TO_YML/prometheus.yml:/etc/prometheus/prometheus.yml \
  prom/prometheus
```

STEP 4 | Validate that the Prisma Cloud integration is properly set up In a new browser window, go to http://<PROMETHEUS_HOST>:9090/targets.



For testing, restart Console to get results immediately instead of waiting for the first 10 minute window to elapse.

Using Prometheus with Projects

If you want to use Prometheus with projects, modify the scrape configuration file with an additional job for each Prisma Cloud Console.

If you are using tenant projects, enable Prometheus instrumentation in both the Central and Supervisor Consoles.

The following listing shows an example configuration that scrapes two Consoles:

- Central Console.
- A supervisor Console for a tenant project.

```
global:
  scrape_interval:     15s # Set the scrape interval to every 15
 seconds. Default is every 1 minute.
  evaluation_interval: 15s # Evaluate rules every 15 seconds. The
 default is every 1 minute.

# Prisma Cloud scrape configuration.
scrape_configs:
  - job_name: 'Central Console'
    static_configs:
    - targets: [CONSOLE_ADDRESS:8083]
    metrics_path: /api/v1/metrics
    basic_auth:
      username: 'USER01'
      password: 'PASS01'
  - job_name: 'Tenant Console'
    static_configs:
```



```
    - targets: [CONSOLE_ADDRESS:8083]
    metrics_path: /api/v1/metrics
    scheme: http
    params:
      project: [TENANT_PROJECT_NAME]
    basic_auth:
      username: 'USER02'
      password: 'PASS02'
```

Where:

- CONSOLE_ADDRESS — DNS name or IP address for your Central Console
- USER01 — Prisma Cloud user with access to Central Console
- PASS01 — USER01's password
- USER02 — Prisma Cloud user with access to the tenant project
- PASS02 — USER02's password
- TENANT_PROJECT_NAME — name of the tenant project

The value in job_name does not need to match anything else. You can set it to anything.

Create a simple graph

Create a graph that shows the number of deployed Defenders.

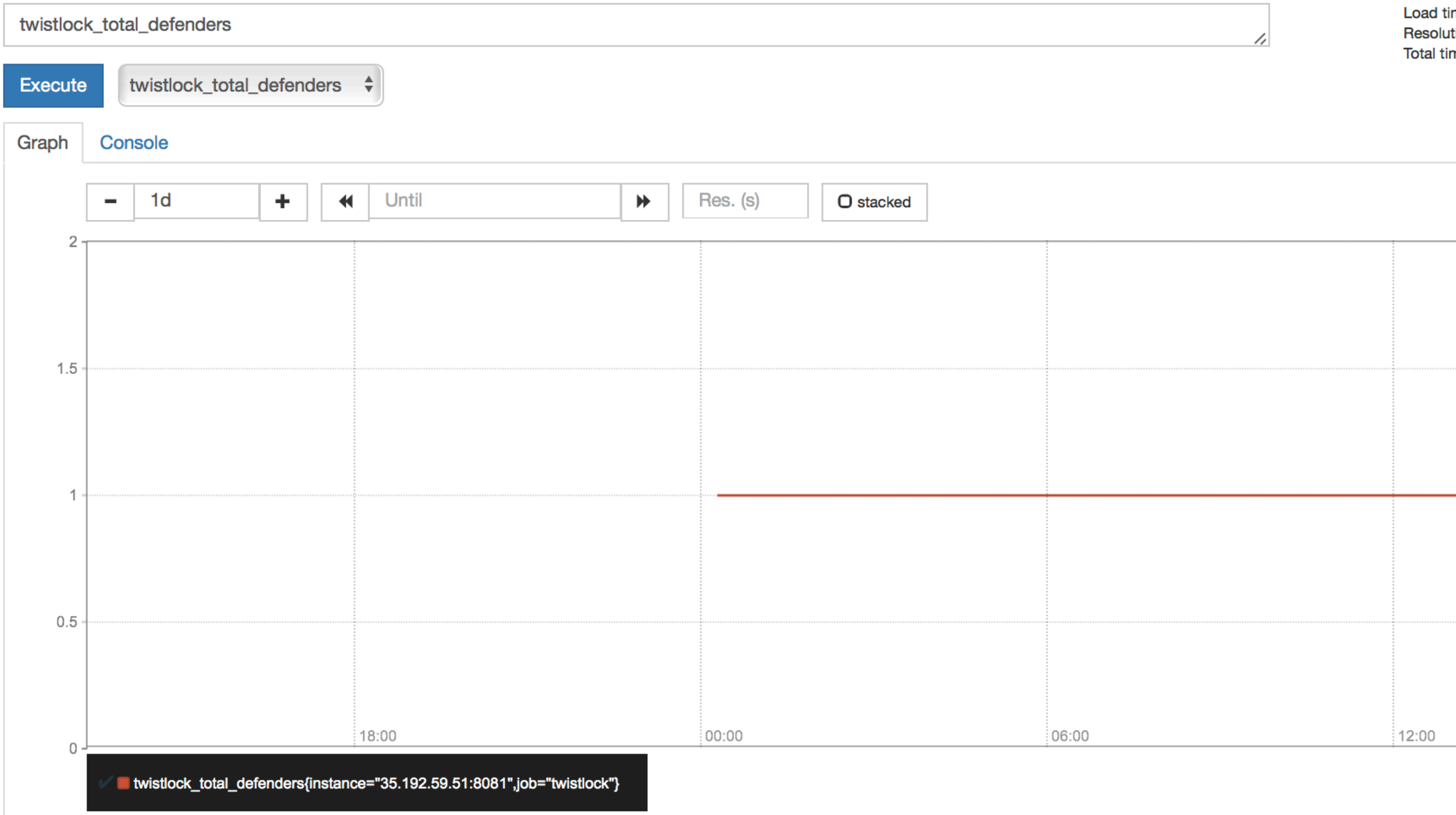
STEP 1 | Go to http://<PROMETHEUS_HOST>:9090/graph

STEP 2 | Click **Add Graph**.

STEP 3 | In the drop-down list, select **twistlock_total_defenders**.

STEP 4 | Click **Execute**. In the **Console** tab, you will see the value for total number of Defenders connected to Console.

STEP 5 | Open the **Graph** tab to see a visual representation of how the number of Defenders has changed over time.



Kubernetes auditing

Edit on GitHub

The Kubernetes auditing system records the activities of users, administrators, and other components, that have affected the cluster. Prisma Cloud can ingest, analyze, and alert on security-relevant events. Write custom rules or leverage Prisma Cloud Labs prewritten rules to assess the incoming audit stream and surface suspicious activity.

Audits types are limited to the ones been configured by the audit policy of the cloud provider.

Rule library

Custom rules are stored in a central library, where they can be reused. Besides your own rules, Prisma Cloud Labs also distributes rules via the Intelligence Stream. These rules are shipped in a disabled state by default. You can review, and optionally apply them at any time.

Your Kubernetes audit policy is defined in **Defend > Access > Kubernetes**, and formulated from the rules in your library. There are four types of rules, but the only one relevant to the audit policy is the *kubernetes-audit* type. Custom rules are written and managed in Console under **Defend > Custom rules > Runtime** with an online editor. The compiler checks for syntax errors when you save a rule.

Expression grammar

Expressions let you examine contents of a Kubernetes audit. Expressions have the following grammar:

*expression: term (op term | in)**

- **term** --

  integer | string | keyword | event | '(' expression ')' | unaryOp term
- **in** --

  '(' integer | string (',' integer | string)*)?
- **op** --

  and | or | > | < | >= | # | = | !=
- **unaryOp** --

  not
- **keyword** --

  startswith | contains
- **string** --

  Strings must be enclosed in double quotes
- **integer** --

  int

- **event** --

  process, file system, or network

Kubernetes audit events

When Prisma Cloud receives an audit, it is assessed against your policy. Like all policies in Prisma Cloud, rule order is important. Rules are processed top to bottom, and processing stops at the first match. When a rule matches, an alert is raised.

Write rules to surface audits of interest. Rules are written with the jpath function. The jpath function extracts fields from JSON objects, which is the format of a Kubernetes audit. The extracted string can then be compared against strings of interest. The primary operators for jpath expressions are '=', 'in', and 'contains'. For non-trivial examples, look at the Prisma Cloud Lab rules.

The argument to jpath is a single string. The right side of the expression must also be a string. A basic rule with a single jpath expression has the following form:

```
jpath("path.in.json.object") = "something"
```

Let's look at some examples using the following JSON object as our example audit.

```
{
   "user":{
      "uid":"1234",
      "username":"some-user-name",
      "groups":[
         "group1",
         "group2"
      ]
   },
   "stage":"ResponseComplete"
}
```

To examine a user's UID, use the following syntax. This expression evaluates to true.

```
jpath("user.uid") = "1234"
```

To examine the username, use the following syntax:

```
jpath("user.username") = "some-user-name"
```

To examine the stage field, use the following syntax:

```
jpath("stage") = "ResponseComplete"
```

To examine the groups list field, use the following syntax:

```
jpath("user.groups") contains "group1"
```

Or alternatively:

```
jpath("user.groups") in ("group1","group2")
```


Integrating with self-managed clusters

Prisma Cloud supports self-managed clusters. See here for supported Kubernetes versions. You can deploy clusters with any number of tools, including kubeadm.

Prerequisites: You've already deployed a Kubernetes cluster.

Configure the API server

Configure the API server to forward audits to Prisma Cloud.

To configure the audit webhook backend:

- Create an audit policy file that specifies the events to record and the data events should contain.
- Create a configuration file that defines the backend details and configurations.
- Update the API server config file to point to your audit policy and configuration files.

If your API server runs as a pod, then the audit policy and configuration files must be placed in a directory mounted by the API server pod. Either place the files in an already mounted directory, or create a new one.

If flags/objects related to AuditSink/dynamic auditing were previously added to your API server configuration, remove them. Otherwise, this setup won't work.

STEP 1 | Specify the audit policy.

Create a file called *audit-policy.yaml* with the following recommended policy:

```
apiVersion: audit.k8s.io/v1 # This is required.
kind: Policy
# Generate audit events only for ResponseComplete or panic stages
 of a request.
omitStages:
  - "RequestReceived"
  - "ResponseStarted"
rules:
  # Audit on pod exec/attach events
  - level: Request
    resources:
    - group: ""
      resources: ["pods/exec", "pods/attach"]

  # Audit on pod creation events
  - level: Request
    resources:
    - group: ""
      resources: ["pods"]
    verbs: ["create"]

  # Audit on changes to the twistlock namespace (defender
 daemonset)
  - level: Request
    verbs: ["create", "update", "patch", "delete"]
    namespaces: ["twistlock"]
```



```
  # Default catch all rule
  - level: None
```

More details can be found here.

STEP 2 | Create a configuration file.

Create a configuration file named *audit-webhook.yaml*.

For the server address, *<console_url_webhook_suffix>*, do the following

Step 1. Perform GET /api/v1/settings/kubernetes-audit and get the suffix. example response: { "webhookUrlSuffix": "Rov4TLMx1UiaJuP99OyulwQVUT0=", "lastPollingTime": null }

Step 2. Append the suffix to your console URL

For example : https://1.1.1.1:8083/api/v1/kubernetes/webhook/Rov4TLMx1UiaJuP99OyulwQVUT0=

```
apiVersion: v1
kind: Config
preferences: {}
clusters:
- name: <cluster_name>
  cluster:
    server: <console_url_webhook_suffix> # compute console endpoint
 as stated above
contexts:
- name: webhook
  context:
    cluster: <cluster_name>
    user: kube-apiserver
current-context: webhook
```

STEP 3 | Move the config files into place.

Move both *audit-policy.yaml* and *audit-webhook.yaml* to a directory that holds your API server config files. If the API server runs as a pod, move the files to a directory that is accessible to the pod. Accessible directories can be found in the API server config file under *mounts*.

Alternatively, create a new directory and add it to *mounts*. For more information, see here.

STEP 4 | Add flags.

Configure the API server to use the policy and configuration files you just created. Add the following flags to the API server config file:

```
spec:
  containers:
  - command:
    # Existing flags
    ...
    # New flags for Prisma Cloud:
    - --audit-policy-file=<PATH-TO-API-SERVER-CONFIG-FILES>/audit-
policy.yaml
```

```
    - --audit-webhook-config-file=<PATH-TO-API-SERVER-CONFIG-
FILES>/audit-webhook.yaml
```

When changing the kube-apiserver config file, the API server automatically restarts. It can take a few minutes for the API server to resume operations.

Integrating with Google Kubernetes Engine (GKE)

On GKE, Prisma Cloud retrieves audits from Stackdriver, polling it every 10 minutes for new data.

Note that there can be some delay between the time an event occurs in the cluster and when it appears in Stackdriver. Due to Twistock's polling mechanism, there's another delay between the time an audit arrives in Stackdriver and when it appears in Prisma Cloud.

See here for GKE cluster versions supported by Prisma Cloud.

Prerequisites: You've created a service account with one of the following authorization scopes:

- https://www.googleapis.com/auth/logging.read
- https://www.googleapis.com/auth/logging.admin
- https://www.googleapis.com/auth/cloud-platform.read-only
- https://www.googleapis.com/auth/cloud-platform

STEP 1 | Open Console.

STEP 2 | Go to **Defend > Access > Kubernetes**.

STEP 3 | Set **Kubernetes auditing** to **Enabled**.

STEP 4 | Click **Add settings** to configure how Prisma Cloud connects to your cloud provider's managed Kubernetes service.

1. Set **Provider** to **GKE**.
2. Select your GKE credential.

   If there are no accounts to select, add one to the credentials store.
3. (Optional) Specify clusters to collect audit data, allows to limit the collected data
4. Specify project IDs. If unspecified, the project ID where the service account was created is used
5. (Optional) Specify Advanced filter - specify a filter to reduce the amount of data transferred

   Do not use the *resource.type* or *timestamp* filters because Prisma Cloud uses them internally.
6. Click **Add**.

STEP 5 | Click **Save**.

CA bundle

If you're sending audit data to Prisma Cloud's webhook over HTTPS, you must specify a CA bundle in the AuditSink object.

If you've customized Console's certificate, you can get a copy from **Manage > Authentication > System-certificates > TLS certificate for Console**. Paste the certificate into a file named *server-cert.pem*, then run the following command:

```
$ openssl base64 -in server-cert.pem -out base64-output -A
```

In the AuditSingle object, set the value of caBundle to the contents of the base64-output file.

Testing your setup

Write a new rule, or select a prewritten rule from the inventory, and add it your audit policy. This setup installs a rule that fires when privileged pods are created in the cluster.

STEP 1 | Open Console, and go to **Defend > Access > Kubernetes**.

STEP 2 | Add a Prisma Cloud Labs prewritten rule.

1. Click **Select rules**.
2. If you're integrated with a managed cluster, select **Prisma Cloud Labs - Privileged pod creation**. If you're integrated with GKE, select **Prisma Cloud Labs - GKE - privileged pod creation**.

   There are separate rules for standard Kubernetes and GKE because the structure of the audits are different. Therefore, the logic for parsing the audit JSON is different.
3. Click **Save**.

STEP 3 | Create a pod deployment file named *priv-pod.yaml*, and enter the following contents.

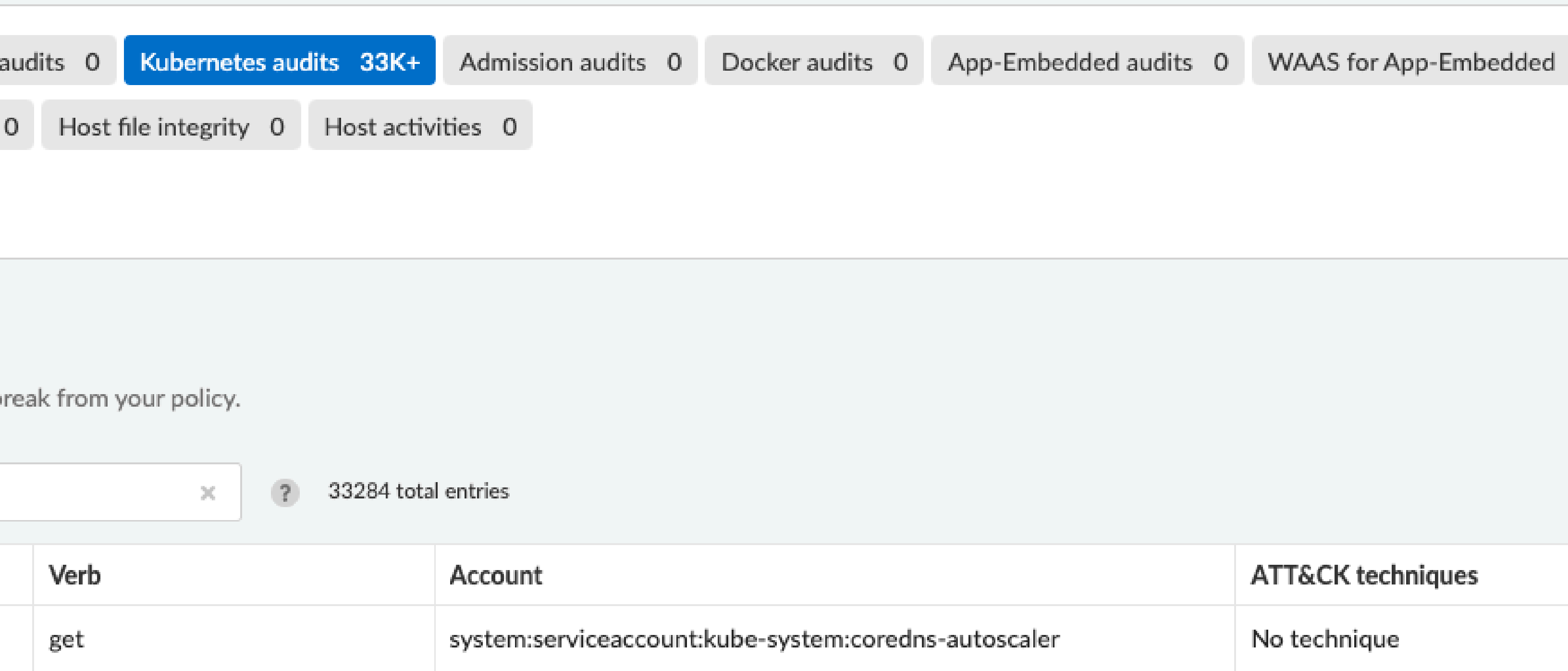
```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  containers:
  - name: nginx
    image: nginx
    ports:
    - containerPort: 80
    securityContext:
      privileged: true
```

STEP 4 | Create the privileged pod.

```
$ kubectl apply -f priv-pod.yaml
```


STEP 5 | Verify an audit was created.

Go to **Monitor > Events**, and select the **Kubernetes Audits** filter.

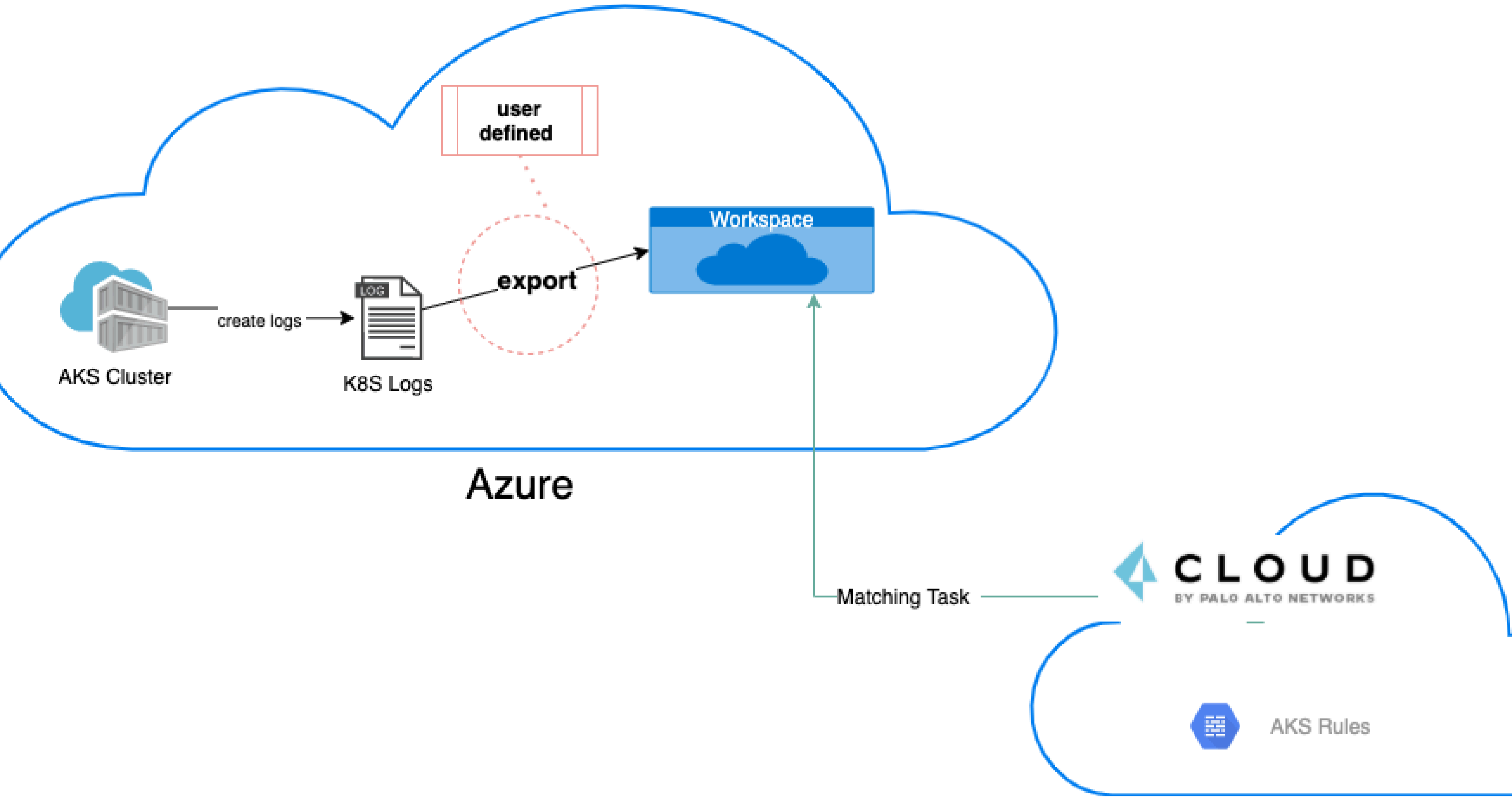


Integrating with Azure Kubernetes Service (AKS)

With AKS, Prisma Cloud retrieves audits from "Log Analytics workspace", polling it every 10-15 minutes for new data.

You will have to enable exporting AKS logs into Azure Workspace, and Prisma Cloud will extract the logs from there. You only need to export AKS resource logs of the category kube-audit (see here). Also, there can be some delay between the time an event occurs in the cluster and when it appears in Workspace. Due to Prisma Cloud's polling mechanism, there's another delay between the time an audit arrives in the Workspace and when it appears in Prisma Cloud.

Prisma Cloud supports only AKS cluster versions that allow log exporting.



STEP 1 | Open Console.

STEP 2 | Go to **Defend > Access > Kubernetes**.

STEP 3 | Set **Kubernetes auditing** to **Enabled**.

STEP 4 | Click **Add settings** to configure how Prisma Cloud connects to your cloud provider's managed Kubernetes service.

1. Set **Provider** to **AKS**.
2. Select your AKS credential.

   If there are no accounts to select, add one to the credentials store.
3. (Optional) specify clusters to collect audit data, allows to limit audit data.
4. Specify the Workspace Name.

   We recommend that you use the free 7 day retention period workspace.
5. Specify a list of resource groups.

   If unspecified, all resource groups will be used to retrieve the audits.
6. (Optional) Specify Advanced filter to reduce the amount of data transferred.

   Use this reference for help with the query syntax.
7. Click **Add**.

STEP 5 | Click **Save**.

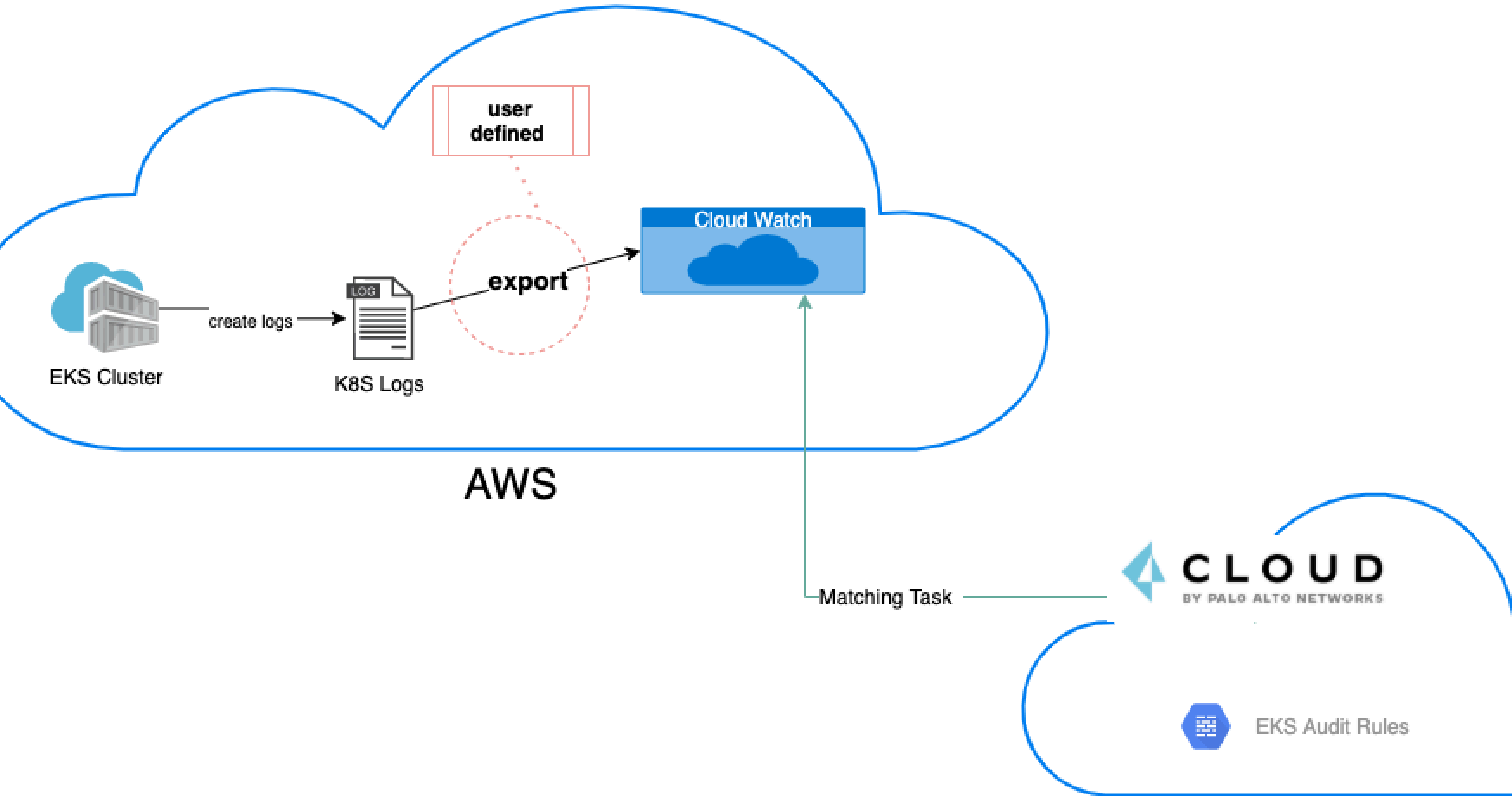
Integrating with Elastic Kubernetes Service (EKS)

On EKS, Prisma Cloud retrieves audits from AWS "Cloud watch", polling it every 10-15 minutes for new data.

You will have to enable exporting EKS logs into AWS Cloud Watch, and Prisma Cloud will extract the logs from there. You only need to enable exporting Kubernetes audits (logs of type audit), see here. Also, there can be some delay between the time an event occurs in the cluster and when it appears in CloudWatch.

Due to Prisma Cloud's polling mechanism, there's another delay between the time an audit arrives in the CloudWatch and it appears in Prisma Cloud.

Prisma Cloud supports only EKS cluster versions that allow log exporting.



STEP 1 | Open Console.

STEP 2 | Go to **Defend > Access > Kubernetes**.

STEP 3 | Set **Kubernetes auditing** to **Enabled**.

STEP 4 | Click **Add settings** to configure how Prisma Cloud connects to your cloud provider's managed Kubernetes service.

1. Set **Provider** to **EKS**.
2. Select your EKS credential.

   If there are no accounts to select, add one to the credentials store.
3. Specify the cluster region.
4. (Optional) Specify Advanced filter to reduce the amount of data transferred.

   Use AWS Log Insights syntax.
5. Click **Add**.

STEP 5 | Click **Save**.

Custom rules

A custom rule is made up of one or more conditions. Configure custom rules policy in order to trigger audits and match them. Prisma Cloud supports GKE, EKS, and AKS clusters.

Write a Kubernetes custom rule

Expression syntax is validated when you save a custom rule.

STEP 1 | Open Console, and go to **Defend > Access > Kubernetes**.

STEP 2 | Click **Add rule**.

STEP 3 | Enter a name for the rule.

STEP 4 | In **Message**, enter an audit message to be emitted when an event matches the condition logic in this custom rule.

STEP 5 | Enter your expression logic.

You can filter by cluster name (applies to all cloud providers), project ID (GCP), account ID (AWS), resource group (only capital letters, GCP), and subscription ID (Azure)

STEP 6 | Click **Add**.

Your expression logic is validated before it's saved to the Console's database.

Tools

Edit on GitHub

Prisma Cloud ships a command-line configuration and control tool called twistcli. It lets you deploy Prisma Cloud components, run scans, and more. It is supported on Linux, macOS, and Windows.

- twistcli
- Scan images with twistcli
- Scan code repos with twistcli
- Install Console with twistcli
- Update the Intelligence Stream in offline environments

twistcli

Edit on GitHub

Prisma Cloud ships a command-line configuration and control tool known as *twistcli*. It is supported on Linux, macOS, and Windows.

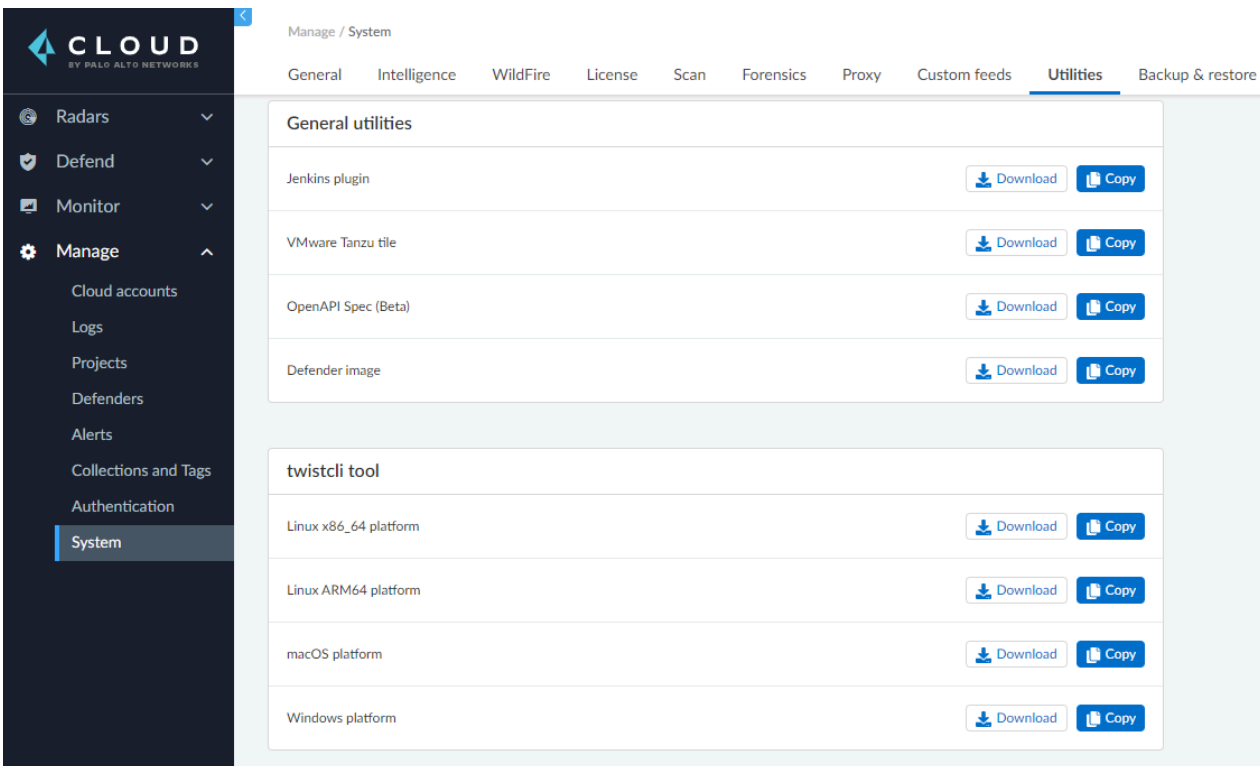
When users from a tenant project run *twistcli*, they must set the *--project* option to specify the proper context for the command.

Installing twistcli

The *twistcli* tool is delivered with every Prisma Cloud release. It is statically compiled, so it does not have any external dependencies, and it can run on any Linux host. No special installation is required. To run it, simply copy it to a host, and give it executable permissions.

The *twistcli* tool is available from the following sources.

- You find it in the release tarball.
- You can download *twistcli* from the Prisma Cloud Console UI. Go to **Manage > System > Utilities**.



Chose the correct architecture and OS when downloading the *twistcli* command-line utility.

- You can download it from the API, which is typical use case for automated workflows. For more information, see the */api/v1/util* endpoint.

The requirements for running *twistcli* are:

- The host running *twistcli* must be able to connect to the Prisma Cloud Console over the network.
- For image scanning, Docker Engine must be installed on the executing machine.

Connectivity to Console

Most *twistcli* functions require connectivity to Console. All example commands specify a variable called COMPUTE_CONSOLE, which represents the address for your Console.

The address for your Console depends on how you installed it.

For Onebox installs, where you install Console on a stand-alone host, the value for COMPUTE_CONSOLE is the IP address or DNS name of the host. HTTPS access to Console is servered on port 8083, so the full address would be:

https://<IPADDR>:8083

For the default Kubernetes installation procedure, the Console service is exposed by a LoadBalancer, and so the address for COMPUTE_CONSOLE is

https://<LOAD_BALANCER>:8083

Functions

The *twistcli* tool supports the following functions:

- *console* — Installs and uninstalls Console into a cluster. Kubernetes and OpenShift are supported. You can also export Kubernetes or OpenShift deployment files in YAML format.
- *defender* — Installs and uninstalls Defender into a cluster. Kubernetes and OpenShift are supported. Defender is installed as a daemon set (Kubernetes, OpenShift) which means one Defender is always automatically deployed to each node in the cluster. You can also export a Kubernetes or OpenShift deployment file in YAML format.
- *hosts* — Scans hosts for vulnerabilities and compliance issues.
- *images* — Scans container images for vulnerabilities and compliance issues. Because it runs from the command line, you can easily integrate Prisma Cloud's scanning capabilities into your CI/CD pipeline.
- *intelligence* — Retrieves the latest threat data from the Prisma Cloud Intelligence Stream, and push those updates to a Prisma Cloud installation running in an air-gapped environment.
- *tas* — Scans VMware Tanzu droplets.
- *app-embedded* — Embed the App Embedded Defender into a Dockerfile.
- *restore* — Restore Console to the state stored in the specified backup file. An automatated backup system (enabled by default) creates and maintains daily, weekly, and monthly backups. Additional backups can made at any point in time from the Console UI.
- *serverless* — Scans serverless functions for vulnerabilities.
- *support* — Streamlines the process of collecting and sending debug information to Prisma Cloud's support team. Collects log data from a node and uploads it to Prisma Cloud's support area.

Capabilities

The *twistcli* tool offers feature parity across all supported operating systems, with a few exceptions. The following table highlights where functions are disabled, or work differently, on a given platform.

twistcli		Platform		
Command	Subcommand	Linux	macOS	Windows
console	*export*	Yes	Yes	Yes
	install	Yes	No	No
	uninstall	Yes	No	No
defender	*export*	Yes	Yes	Yes
	install	Yes	No	No
	uninstall	Yes	No	No
hosts	*scan*	Yes	No[1]	No
images	*scan*	Yes	Yes[2]	Yes[3]
intelligence	*upload*	Yes	Yes	Yes
	download	Yes	Yes	Yes
pcf	*scan*	Yes	No	No
app-embedded	*embed*	Yes	Yes	Yes
restore		Yes	No	No
serverless	*scan*	Yes	Yes	Yes
support	*dump*	Yes	No[4]	No[4]
	upload	Yes	Yes	Yes

[1] Prisma Cloud doesn't support deployment to macOS hosts, so there is no support for scanning macOS hosts.

[2] Scans Linux images on macOS hosts. Docker for Mac must be installed.

[3] Twistcli can scan Windows images on Windows Server 2016 and Windows Server 2019 hosts. To scan Linux images on Windows, install Docker Machine on Windows with the Microsoft

Hyper-V driver. Twistcli does not support scanning Linux images on Windows hosts with Docker for Windows.

[4] The *support dump* function collects Console's logs when Console malfunctions. Copy *twistcli* to host where Console runs, then execute *twistcli support dump*. Defender logs can be retrieved directly from the Console UI under **Manage > Defenders > Manage**.

For a comprehensive list of supported options for each subcommand, run:

```
$ twistcli <COMMAND> --help
```

Install support

Support for installing Console and Defender via *twistcli* is supported on several cluster types. The following table highlights the available support:

twistcli		Platform				
>Command	>Subcommand	>Stand-alone[1]	>Kubernetes	>OpenShift	>Amazon ECS	>Windows
console	*export*	No	Yes	Yes	No	No
	install	No	Yes	Yes	No	No
	uninstall	No	Yes	Yes	No	No
defender	*export*	No	Yes	Yes	No	No
	install	Yes	Yes	Yes	No	No
	uninstall	No	Yes	Yes	No	No

[1] Stand-alone refers to installing an instance of Console or Defender onto a single host that isn't part of a cluster. For stand-alone installations of Console, use the *twistlock.sh* script to install Onebox.

The *twistcli console install* command for Kubernetes and OpenShift combines two steps into a single command to simplify how Console is deployed. This command internally generates a YAML configuration file and then creates Console's resources with *kubectl create* in a single shot. This command is only supported on Linux. Use it when you don't need a copy of the YAML configuration file. Otherwise, use *twistcli console export*.

Scan images with twistcli

Edit on GitHub

Prisma Cloud ships a command-line scanner for scanning container images and serverless functions. It is supported on Linux, macOS, and Windows.

Command reference

The *twistcli* command has several subcommands. Use the *twistcli images scan* subcommand to invoke the scanner.

Command

twistcli images scan — Scan an image for vulnerabilities and compliance issues. The image must reside on the system where twistcli runs. If not, retrieve the image with *docker pull* before scanning it. Twistcli does not pull images for you.

Synopsis

twistcli images scan [OPTIONS] [IMAGE]

When invoking twistcli, the last parameter should always be the image or tarball to scan. If you specify options after the image or tarball, they will be ignored. If scanning a tarball, be sure to specify the --tarball option.

Description

The *twistcli images scan* function collects information about the packages and binaries in the container image, and then sends it to Console for analysis.

Data collected by twistcli includes:

- Packages in the image.
- Files installed by each package.
- Hashes for files in the image.

After Console analyzes the image for vulnerabilities, *twistcli*:

- Outputs a summary report.
- Exits with a pass or fail return value.

To specify an image to scan, use either the image ID, or repository name and tag. The image should be present on the system, having either been built or pulled there. If a repository is specified without a tag, *twistcli* looks for an image tagged *latest*.

Options

- ***--address URL*** --

  Complete URL for Console, including the protocol and port. Only the HTTPS protocol is supported. By default, Console listens to HTTPS on port 8083, although your administrator can configure Console to listen on a different port. Defaults to https://127.0.0.1:8083.

  Example: --address https://console.example.com:8083
- ***-u, --user USERNAME*** --

  Username to access Console. If not provided, the *TWISTLOCK_USER* environment variable will be used if defined, or "admin" is used as the default.
- ***-p, --password PASSWORD*** --

  Password for the user specified with *-u*, *--user*. If not specified on the command-line, the *TWISTLOCK_PASSWORD* environment variable will be used if defined, or otherwise will prompt for the user's password before the scan runs.
- ***--project PROJECT NAME*** --

  Interface with a specific supervisor Console to retrieve policy and publish results.

  Example: *--project "Tenant Console"*
- ***--output-file FILENAME*** --

  Write the results of the scan to a file in JSON format.

  Example: *--output-file scan-results.json*
- ***--details*** --

  Show all vulnerability details.
- ***--containerized*** --

  Run the scan from inside the container.
- ***--custom-labels*** --

  Include the image custom labels in the results.
- ***--docker-address DOCKER_CLIENT_ADDRESS*** --

  Docker daemon listening address (default: *unix:///var/run/docker.sock*). Can be specified with the *DOCKER_CLIENT_ADDRESS* environment variable.
- ***--docker-tlscacert PATH*** --

  Path to Docker client CA certificate.
- ***--docker-tlscert PATH*** --

  Path to Docker client Client certificate.
- ***--docker-tlskey PATH*** --

  Path to Docker client Client private key.
- ***--exit-on-error TRUE/FALSE*** --

  Immediately exit the scan if an error is encountered (not supported with the --containerized flag).

- ***--tlscacert PATH*** --

  Path to Prisma Cloud CA certificate file. If no CA certificate is specified, the connection to Console is insecure.

- ***--podman-path PATH*** --

  Forces twistcli to use Podman. To use the default installation path, set as *podman*. Otherwise, provide the appropriate path.

- ***--include-js-dependencies*** --

  Evaluates packages listed only in manifests.

- ***--token TOKEN*** --

  Token to use for Prisma Cloud Console authentication. Tokens can be retrieved from the API endpoint *api/v1/authenticate* or from the **Manage > System > Utilities** page in Console.

- ***--publish TRUE/FALSE*** --

  Publishes scan results to the Console (default: --publish=true)

- ***--tarball*** --

  Boolean flag that specifies the image to scan is a tar archive. The tarball scan requires enhanced privileges, and must be executed as sudo or as a root user. Prisma Cloud supports tar archives in the Docker Image Specification format, v1.1 and later.

The *tarball* option is supported on Linux only; macOS and Windows versions of twistcli do not support it.

The last parameter in the twistcli command should always be the path to the tarball. The --*tarball* option is simply a boolean flag. It doesn't accept a corresponding value (e.g. a path to a tarball). For clarity, see the following examples:

Correct usage:

```
./twistcli --tarball --user ted image.tar
```

Incorrect usage:

```
./twistcli --tarball image.tar --user ted
```

Return value

The exit code is 0 if *twistcli images scan* finds no vulnerabilities or compliance issues. Otherwise, the exit code is 1.

The criteria for passing or failing a scan is determined by the CI vulnerability and compliance policies set in Console. The default CI vulnerability policy alerts on all CVEs detected. The default CI compliance policy alerts on all critical and high compliance issues.

There are a couple of reasons why twistcli images scan might return an exit code of 1.

- *The scan failed because the scanner found issues that violate your CI policy.*
- *Twistcli failed to run due to an error.*

Although the return value is ambiguous — you cannot determine the exact reason for the failure by just examining the return value — this setup supports automation. From an automation process perspective, you expect that the entire flow will work. If you scan an image, with or without a threshold, either it works or it does not work. If it fails, for whatever reason, you want to fail everything because there is a problem.

Scan results

To view scan reports in Console, go to **Monitor > Vulnerabilities > Images > CI** or **Monitor > Compliance > Images > CI**.

The scan reports includes the image vulnerabilities, compliance issues, layers, process info, package info, and labels.

When scanning images in the CI pipeline with twistcli or the Jenkins plugin, Prisma Cloud collects the environment variable *JOB_NAME* from the machine the scan ran on, and adds it as a label to the scan report.

You can also retrieve scan reports in JSON format using the Prisma Cloud API, see the API section.

Output

The twistcli tool can output scan results to several places:

- stdout.
- JSON file.
- Console. Scan results can be viewed under **Monitor > Vulnerabilities > Images > CI** and **Monitor > Compliance > Images > CI**.

By passing certain flags, you can adjust how the twistcli scan output looks and where it goes. By default, twistcli writes scan results to stdout and sends the results to Console.

To write scan results to stdout in tabular format, pass the *--details* flag to twistcli. This does not affect where the results are sent.

To write scan results to a file in JSON format, pass the *--output-file* flag to twistcli. The file schema is being kept for backwards compatibility.

Following is the output file schema:

```
{
  "results": [
      {
        "id": "image id",
        "name" : "image name",
        "distro": "image OS distro",
        "distroRelease": "image OS release",
        "digest": "image digest",
        "collections": [
```



```
            "collectionA",
            "collectionB"
          ],
          "packages": [
                 {
                    "type": "package type",
                    "name": "package name",
                    "version": "package version",
                    "path": "package path, if exists",
                    "licenses": [
                       "licenseA",
                       "licenseB"
                    ]
                 },
                 {
                  ...
                 }
          ],
          "applications": [
                 {
                    "name": "app name",
                    "version": "app version",
                    "path": "app path, if exists"
                 },
                 {
                  ...
                 }
          ],
          "compliances": [
                 {
                    "id": "compliance issue ID",
                    "title": "compliance issue title",
                    "severity": "compliance issue severity",
                    "description": "compliance issue description",
                    "cause": "compliance issue cause, if exists",
                    "layerTime": "layer time of the image layer to
 which the compliance issue belongs",
                    "category": "compliance category",
                    "pass": "true/false"
                 },
                 {
                   ...
                 }
          ],
          "complianceDistribution": {
                  "critical": 0,
                  "high": 1,
                  "medium": 0,
                  "low": 0,
                  "total": 1
          },
          "complianceScanPassed": true/false,
          "vulnerabilities": [
                 {
                    "id": "CVE ID",
                    "status": "CVE fix status",
```



```
                    "cvss": CVSS,
                    "vector": "CVSS vector",
                    "description": "CVE description",
                    "severity": "CVE severity",
                    "packageName": "package name",
                    "packageVersion": "package version",
                    "link": "link to the CVE as provided in the Console
 UI",
                    "riskFactors": [
                          "Attack vector: network",
                          "High severity",
                          "Has fix"
                     ],
                    "tags": [
                          "ignored",
                          "in review"
                    ],
                    "impactedVersions": [
                          "impacted versions phrase1",
                          "impacted versions phrase2"
                    ],
                    "publishedDate": "publish date",
                    "discoveredDate": "discovered date",
                    "graceDays": "grace days",
                    "fixedDate": "vendor fixed date, if exists",
                    "layerTime": "layer time of the image layer to
 which the vulnerability belongs"
                 },
                 {
                   ...
                 }
          ],
          "vulnerabilityDistribution": {
                  "critical": 0,
                  "high": 1,
                  "medium": 0,
                  "low": 19,
                  "total": 20
          },
          "vulnerabilitiesScanPassed": true/false,
          "history": [
                 {
                    "created": "time when the image layer was created",
                    "instruction": "Dockerfile instruction and
 arguments used to create the layer"
                 },
                 {
                   ...
                 }
          ],
          "scanTime": "the image scan time",
          "scanID": "the image scan ID"
       }
    ],
    "consoleURL": "url of the scan results in the Console UI"
}
```

Projects

When users from a tenant project run twistcli, they must set the *--project* option to specify the proper context for the command.

twistcli images scan --project "<project_name>"

API

You can retrieve scan reports in JSON format using the Prisma Cloud Compute API. The API returns comprehensive information for each scan report, including the full list of packages, files, and vulnerabilities.

The following example *curl* command calls the API with Basic authentication. You'll need to apply some filtering with tools like *jq* to extract specific items from the response. For more information on accessing the API, see the API reference.

```
$ curl \
  -u <COMPUTE_CONSOLE_USER> \
  -o scan_results.json \
  'https://<COMPUTE_CONSOLE>/api/v1/scans?type=ciImage'
```

If you are using assigned collections, then specify the collection in a query parameter:

```
$ curl \
  -u <COMPUTE_CONSOLE_USER> \
  -o scan_results.json \
  'https://<COMPUTE_CONSOLE>/api/v1/scans?
type=ciImage&collections=<COLLECTION_NAME>'
```

Dockerless scan

By default, twistcli is run from outside the container image.

Podman Twistcli scans

Twistcli can run scans on Podman hosts. Use *--podman-path PATH* to specify the path to podman and force the twistcli scanner to use podman. For additional information, see the Podman section.

Running from inside of the container

In some cases, you might need to copy twistcli to the container's file system, and then run the scanner from inside the container.

One reason you might want to run the scanner this way is when your build platform doesn't give you access to the Docker socket. CodeFresh is an example of such a platform.

There are some shortcomings with scanning from inside a container, so you should only use this approach when no other approach is viable. The shortcomings are:

- Automating the scan in your continuous integration pipeline is more difficult.
- Image metadata, such as registry, repository, and tag aren't available in the scan report. When twistcli is run from outside the container, this information is retrieved from the Docker API.

- The image ID isn't available in the scan report because it cannot be determined when the scan is run from inside a container.
- The scan report won't show a layer-by-layer analysis of the image.

Usage

When running the scanner from inside a container, you need to properly orient it by passing it the *--containerized* flag. There are a couple of ways to run twistcli with the *--containerized* flag: build-time and run-time.

For security reasons, Prisma Cloud recommends that you create a user with the *CI User* role for running scans.

Build-time invocation

After building an image, run it. Mount the host directory that holds the twistcli binary, pass the Prisma Cloud Console user credentials to the container with environment variables, then run the scanner inside the container. The *<REPORT_ID>* is a user defined string that uniquely identifies the scan report in the Console UI.

```
$ docker run \
  -v /PATH/TO/TWISTCLIDIR:/tools \
  -e TW_USER=<COMPUTE_CONSOLE_USER> \
  -e TW_PASS=<COMPUTE_CONSOLE_PASSWD> \
  -e TW_CONSOLE=<COMPUTE_CONSOLE> \
  --entrypoint="" \
  <IMAGE_NAME> \
  /tools/twistcli images scan \
    --containerized \
    --details \
    --address $TW_CONSOLE \
    --user $TW_USER \
    --password $TW_PASS \
    <REPORT_ID>
```

Rather than username and password, twistcli can also authenticate to Console with a token. Your API token can be found in Console under **Manage > Authentication > User Certificates > API token**. For security reasons, API tokens expire.

```
$ docker run \
  -v /PATH/TO/TWISTCLI_DIR:/tools \
  -e TW_TOKEN=<API_TOKEN> \
  -e TW_CONSOLE=<COMPUTE_CONSOLE> \
  --entrypoint="" \
  <IMAGE_NAME> \
  /tools/twistcli images scan \
    --containerized \
    --details \
    --address $TW_CONSOLE \
    --token $TW_TOKEN \
    <REPORT_ID>
```


Run-time invocation

If you have access to the orchestrator, you can exec into the running container to run the twistcli scanner. Alternatively, you could SSH to the container. Once you have a shell on the running container, invoke the scanner:

```
$ ./twistcli images scan \
  --address <COMPUTE_CONSOLE> \
  --user <COMPUTE_CONSOLE_USER> \
  --password <COMPUTE_CONSOLE_PASSWD> \
  --containerized \
  <REPORT_ID>
```

To invoke the scanner with an API token:

```
$ ./twistcli images scan \
  --address <COMPUTE_CONSOLE> \
  --token <API_TOKEN> \
  --containerized \
  <REPORT_ID>
```

Simple scan

Scan an image with twistcli and print the summary report to stdout.

Scan an image named *myimage:latest*.

```
$ twistcli images scan \
  --address <COMPUTE_CONSOLE> \
  --user <COMPUTE_CONSOLE_USER> \
  --password <COMPUTE_CONSOLE_PASSWD> \
  myimage:latest
```

Command output:

```
myimage:latest sha256:2073e0bcb60ee98548d313ead5eacbfe16d9054f8


for image myimage:latest: total - 35, critical - 0, high - 2, m
d check results: PASS


mage myimage:latest: total - 1, critical - 0, high - 1, medium
heck results: PASS
```

Scan with detailed report

You can have twistcli generate a detailed report for each scan. The following procedure shows you how to scan an image with twistcli, and then retrieve the results from Console.

STEP 1 | Scan an image named *myimage:latest*.

```
$ twistcli images scan \
  --address <COMPUTE_CONSOLE> \
  --user <COMPUTE_CONSOLE_USER> \
  --password <COMPUTE_CONSOLE_PASSWD> \
  --details \
```

```
  myimage:latest
```

Sample command output (results have been truncated):


```
d313ead5eacbfe16d9054f8800a32bedd859922a99a6e1


-----------------+----------------------------+--------------+---
VERSION          |           STATUS           |  PUBLISHED   | DI
-----------------+----------------------------+--------------+---
                 | open                       | > 9 months   | >
                 |                            |              |
                 |                            |              |
                 |                            |              |
-----------------+----------------------------+--------------+---
                 | fixed in 2.0.5-1+deb10u1   | > 4 months   | >
                 |                            |              |
                 |                            |              |
-----------------+----------------------------+--------------+---
deb10u2          | fixed in 1.44.5-1+deb10u3  | 47 days      | 47
                 |                            |              |
                 |                            |              |
                 |                            |              |
-----------------+----------------------------+--------------+---
                 | open                       | > 3 months   | >
                 |                            |              |
                 |                            |              |
                 |                            |              |
-----------------+----------------------------+--------------+---
191117-2~deb10u1 | open                       | > 2 years    | >
                 |                            |              |
                 |                            |              |
-----------------+----------------------------+--------------+---
10u2             | open                       | > 10 months  | >
                 |                            |              |
                 |                            |              |
                 |                            |              |
-----------------+----------------------------+--------------+---
10u2             | open                       | > 10 months  | >
```

STEP 2 | This outputs a tabular representation of your scan results to stdout. If you need to retrieve the results of your scan in JSON format, this can be done using the API. For more information on the API, see the API reference.

1. Call the API with authentication (demonstrated here using Basic authentication) to fetch the results of the scan.

```
$ curl \
  -o scan_results.json \
  -H 'Authorization: Basic YXBpOmFwaQ==' \
  'https://<COMPUTE_CONSOLE>/api/v1/scans?
search=myimage&limit=1&reverse=true&type=ciImage'
```

2. Format the scan results into human-readable format.

```
$ python -m json.tool scan_results.json > scan_results_pp.json
```

3. Inspect the results.

   Open *scan_results_pp.json* to view the results. Vulnerability information can be found in the *vulnerabilities* array, and compliance results can be found in the *complianceIssues* array.

```
[
  {
    "entityInfo": {
      "_id": "",
      "type": "ciImage",
      ...
      "complianceIssues": [
        {
          "text": "",
          "id": 41,
          "severity": "high",
          "cvss": 0,
          "status": "",
          "cve": "",
          "cause": "",
          "description": "It is a good practice to run the
 container as a non-root user, if possible. Though user
\nnamespace mapping is now available, if a user is already
 defined in the container image, the\ncontainer is run as that
 user by default and specific user namespace remapping is not
\nrequired",
          "title": "(CIS_Docker_CE_v1.1.0 - 4.1) Image should
 be created with a non-root user",
          "vecStr": "",
          "exploit": "",
          "riskFactors": null,
          "link": "",
          "type": "image",
          "packageName": "",
          "packageVersion": "",
          "layerTime": 0,
          "templates": [],
          "twistlock": false,
```

```
          "published": 0,
          "discovered": "0001-01-01T00:00:00Z"
        }
      ],
      ...
      "vulnerabilities": [
        {
          "text": "",
          "id": 46,
          "severity": "medium",
          "cvss": 9.8,
          "status": "deferred",
          "cve": "CVE-2018-20839",
          "cause": "",
          "description": "systemd 242 changes the VT1 mode
 upon a logout, which allows attackers to read cleartext
 passwords in certain circumstances, such as watching a
 shutdown, or using Ctrl-Alt-F1 and Ctrl-Alt-F2. This occurs
 because the KDGKBMODE (aka current keyboard mode) check is
 mishandled.",
          "title": "",
          "vecStr": "CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/
A:H",
          "exploit": "",
          "riskFactors": {
            "Attack complexity: low": {},
            "Attack vector: network": {},
            "Medium severity": {}
          },
          "link": "https://people.canonical.com/~ubuntu-
security/cve/2018/CVE-2018-20839",
          "type": "image",
          "packageName": "systemd",
          "packageVersion": "237-3ubuntu10.39",
          "layerTime": 1587690420,
          "templates": [],
          "twistlock": false,
          "published": 1558067340,
          "discovered": "0001-01-01T00:00:00Z",
          "binaryPkgs": [
            "libnss-systemd",
            "libsystemd0",
            "libpam-systemd",
            "udev",
            "systemd-sysv",
            "libudev1",
            "systemd"
          ]
        },
        ...
      ],
      ...
    },
    ...
  }
]
```


Scan images built with Jenkins in an OpenShift environment

If you are building and deploying images on OpenShift Container Platform (OCP), and you are utilizing their Jenkins infrastructure, then invoke a scan with the *twistcli hosts scan* command, not the *twistcli images scan* command.

You can scan images generated by Jenkins with the OpenShift plugin by invoking twistcli from a build hook. Build hooks let you inject custom logic into the build process. They run your commands inside a temporary container instantiated from build output image. Build hooks are called when the last layer of the image has been committed, but before the image is pushed to a registry. An non-zero exit code fails the build. A zero exit code passes the build, and allows it to proceed to the next step.

To call twistcli from a build hook:

STEP 1 | Download twistcli into your build environment. Depending on your build strategy, one option is to download it as an external artifact using a *save-artifacts* S2I script.

STEP 2 | In your *BuildConfig*, call twistcli as a *script* from the *postCommit* hook.

```
$ twistcli hosts scan \
  --address <COMPUTE_CONSOLE> \
  --user <COMPUTE_CONSOLE_USER> \
  --password <COMPUTE_CONSOLE_PASSWD> \
  --skip-docker \
  --include-3rd-party
```

Where the *--skip-docker* option skips all Docker compliance checks such as the Docker daemon configuration and the *--include-3rd-party* option scans application-specific files such as JARs.

Scan images when the Docker socket isn't in the default location

The twistcli scanner uses the Docker API, so it must be able to access the socket where the Docker daemon listens. If your Docker socket isn't in the default location, use the *--docker-address* option to tell twistcli where to find it:

- ***--docker-address PATH*** --

  Path to the Docker socket. By default, twistcli looks for the Docker socket *unix:///var/run/docker.sock*.

```
$ ./twistcli images scan \
  --address <COMPUTE_CONSOLE> \
  --user <COMPUTE_CONSOLE_USER> \
  --password <COMPUTE_CONSOLE_PASSWD> \
  --docker-address unix:///<PATH/TO>/docker.sock \
  <IMAGE_NAME>
```

Scan Podman/CRI images

Podman is a daemon-less container engine for developing, managing, and running OCI containers on Linux. The twistcli tool can use the preinstalled Podman binary to scan CRI images.

- ***--podman-path PATH*** --

  Forces twistcli to use Podman. To call podman from its default install path, specify *podman*. Otherwise, specify an explicit path.

  ```
  $ ./twistcli images scan \
    --address <COMPUTE_CONSOLE> \
    --user <COMPUTE_CONSOLE_USER> \
    --password <COMPUTE_CONSOLE_PASSWD> \
    --podman-path podman \
    <IMAGE_NAME>
  ```

CI/CD Automation

Twistcli images scan can be used to shift-left security scans inside of your build pipeline. Plugins are available for Jenkins and other CI/CD tools, but twistcli can also be used from a CI pipeline in order to initiate vulnerability and compliance scans on images.

The exit status code can be verified inside of your pipeline to determine pass and fail status of the image scan. A zero exit code signals the scan passes, and any non-zero exit code signals a failure.

In order to automate the download and version sync of twistcli, reference the sample Jenkins code below:

```
stage('Check twistcli version') {

  def TCLI_VERSION = sh(script: "./twistcli | grep -A1 VERSION | sed
 1d", returnStdout:true).trim()
  def CONSOLE_VERSION = sh(script: "curl -k -u \"$TL_USER:
$TL_PASS\" https://$TL_CONSOLE/api/v1/version | tr -d \'\"'",
 returnStdout:true).trim()

  println "TCLI_VERSION = $TCLI_VERSION"
  println "CONSOLE_VERSION = $CONSOLE_VERSION"

  if ("$TCLI_VERSION" != "$CONSOLE_VERSION") {
    println "downloading twistcli"
    sh 'curl -k -u $TL_USER:$TL_PASS --output ./twistcli https://
$TL_CONSOLE/api/v1/util/twistcli'
    sh 'sudo chmod a+x ./twistcli'
  }
}

stage('Scan with Twistcli') {
  sh './twistcli images scan --address https://$TL_CONSOLE -u
 $TL_USER -p $TL_PASS --details $IMAGE'
}
```

Scan image tarballs

twistcli can scan image tarballs. This capability is designed to support the following workflows:

- Integration with Kaniko. Kaniko is a tool that builds images in a Kubernetes cluster from a Dockerfile without access to a Docker daemon.

- Vendors deliver container images as tar files, not through a registry.

twistcli supports the Docker Image Specification v1.1 and later. Currently, twistcli doesn't support the Open Container Initiative (OCI) Image Format Specification.

Both Kaniko and the *docker save* command output tarballs using the Docker Image Specification.

To scan an image tarball, specify the --*tarball* option:

```
twistcli images scan --tarball <image_tarball>
```

For example:

```
docker pull vulnerables/web-dvwa:1.9
docker save vulnerables/web-dvwa:1.9 | gzip >
 vulnerables_web_dvwa19.tar.gz
twistcli images scan --tarball vulnerables_web_dvwa19.tar.gz
```

Scan Windows images on Windows hosts with containerd

You can use twistcli to scan Windows images on Windows hosts with containerd installed.

```
.\twistcli.exe images scan \
--address <CONSOLE_URL> \
-u <USER> \
--containerd \
--containerd-namespace <NAMESPACE>
<IMAGE_ID | IMAGE_NAME>
```

The image ID passed to twistcli must be the full length image ID. Short IDs aren't supported. Get full length image IDs using the following command:

```
ctr -n <namespace> images ls
```

The ctr utility can be downloaded from here.

Windows requires the host OS version to match the container OS version. If you want to run a container based on a newer Windows build, make sure you have an equivalent host build. Otherwise, you can use Hyper-V isolation to run older containers on new host builds. For more information, see Windows containers version compitability.

Limitations

Due to a bug in Kaniko, twistcli can't map vulnerabilities to layers when scanning image tarballs built by Kaniko.

Scan code repos with twistcli

Edit on GitHub

Prisma Cloud ships a command-line scanner for scanning code repos. It is supported on Linux, macOS, and Windows.

twistcli scans repositories locally and sends a bill of materials to Console for evaluation. Console assesses the components in the BoM against the latest threat data, and replies back to twistcli with the scan results.

The policy Console uses to assess a code repo is set in **Defend > Vulnerabilities > Code repositories > CI** and **Defend > Compliance > Code repositories > CI**.

By default, twistcli publishes scan results to Console. Scan results can viewed in Console under **Monitor > Vulnerabilities > Code repositories > CI**.

Many developers don't have access to Prisma Cloud directly, but may want to run twistcli to evaluate a repo before code is submitted and a build job is initiated. twistcli has can print detailed results locally and optionally suppress publishing scan results to Console (which they might not be able to access anyway).

Basic command line format and options

The basic command format is as follows:

```
twistcli coderepo scan <REPO_PATH> --repository <REPO_NAME>
```

Where:

- REPO_PATH can be an absolute or relative path
- REPO_NAME is the unique key that Console to identify the repo. If --*repository* isn't specified, the repository's path is used as the repository name.

Print detailed scan results

Detailed result contain all dependencies files and a vulnerability distribution summary. For each dependency file that has any vulnerabilities, a table with information about the vulnerability is printed.

```
twistcli coderepo scan <REPO_PATH> --repository rowan --details
```

Excluding files from a scan

To exclude files from the scan, use the --*excluded-paths* argument. Attach only a single value to the argument. The value should be a relative path from the root of the repository.

To exclude a file:

```
twistcli coderepo scan <REPO_PATH> --repository rowan --excluded-
paths <PATH1>
```


To exclude multiple files, specify the --*excluded-paths* argument multiple times:

```
twistcli coderepo scan <REPO_PATH> --repository rowan --excluded-
paths <PATH1> --excluded-paths <PATH2>
```

Scanning specific files

To explicitly scan files scan use the --*explicit-files* argument. Specify one file per argument. For multiple files, specify multiple --*explicit-files* arguments.

Suppress publishing results

Using—publish=false to avoid publishing the result to the console.

```
twistcli coderepo scan <REPO_PATH> --repository rowan --publish=false
```

Install Console with twistcli

Edit on GitHub

When twistcli installs Console into a Kubernetes or OpenShift cluster, it executes a series of steps. To help you troubleshoot issues when twistcli fails, the steps in the install flow are described here:

When you run *twistcli console install*, it:

1. Loads the Console image on localhost, and tags it with the registry address.
2. Deletes the old Console replication controller, if it exists, and waits for Console deletion.
3. Deletes the config map, if it exists.
4. Creates Prisma Cloud namespace, if it does not exist.
5. If the service does not exist, twistcli resolves the service template to a file and creates a new service.
6. If persistent volume claim (PVC) does not exist, twistcli resolves the PVC template to a file and creates a new PVC.
7. Waits to the PVC to bind to a persistent volume resource. twistcli expects that the persistent volume has already been created by the user. Note that the PVC is not deleted and recreated because once the PVC is be deleted, it cannot bind again to the persistent volume without recreating the persistent volume.
8. Retrieves the service IPs (Cluster IPs, and adds them to the SAN.
9. Creates a config map.
10. Resolves Console template to a file, and creates a Console replication controller.
11. Deletes the working directory.

Update the Intelligence Stream in offline environments

Edit on GitHub

Prisma Cloud lets you update Console's vulnerability and threat data even if it runs in an offline environment.

The Prisma Cloud Intelligence Stream (IS) is a real-time feed that contains vulnerability data and threat intelligence from commercial providers, Prisma Cloud Labs, and the open source community.

When you install Prisma Cloud, Console is automatically configured to connect to intelligence.twistlock.com to download updates. The IS is updated several times per day, and Console continuously checks for updates.

If you run Prisma Cloud in an offline environment, where Console does not have access to the Internet to download updates from the IS, then you can manually download and install IS updates.

Update strategies for offline environments

There are a number of update strategies. The right strategy for you depends on the size of your deployment, and in particular, the number of air-gapped Consoles in your environment.

Basic strategy

Use the basic strategy when you've got one or two air-gapped Consoles. The basic strategy for updating the threat data for an isolated, air-gapped Console is:

- Download the IS data from an Internet-connected machine.
- Move the archived data to a location accessible by the air-gapped environment.
- Load the IS data into the offline Console.

Both the download and upload operations use twistcli, so the process can be automated.

If you've got a large number of air-gapped Consoles, individually updating each one can be challenging and brittle, especially in dynamic environments. As such, Prisma Cloud lets you scale the basic strategy to any number of Consoles. Each deployed Console can be configured to look for the latest threat data in a central location. From there, each Console will update itself every 24 hours. Your job is to ensure that the central location always serves the latest threat data.

For example, consider how the U.S. Navy would keep a fleet of submarines up-to-date with the latest threat data. When a submarine surfaces and establishes brief connection to its command's network, the submarine's Console needs to pull the latest Intelligence Stream updates. For this type of setup, see Scale approach 1 and Scale approach 2.

Scale approach 1

Distribute the latest Intelligence Stream data from an HTTP/S server. Use the basic strategy to keep the data at the endpoint up-to-date. To configure your Console for this approach, see Download the IS from an HTTP server.

Scale approach 2

Distribute the latest Intelligence Stream data from a so-called "relay" Console. Downstream Consoles connect to the relay Console to pull the latest threat data. To keep the relay Console up-to-date:

- Use the basic strategy when the relay Console is also isolated in an air-gapped environment.
- Let the relay Console update itself by connecting to the Intelligence Stream over the Internet.

To configure your Console for this approach, see Download the IS from another Console.

Projects

By default, projects utilize the distribution mechanism described in Scale approach 2. Central Console connects to https://intelligence.twistlock.com to retrieve the latest theat data. All tenant projects connect to Central Console to get the latest threat data. Central Console itself can be configured for manual threat feed updates, Scale approach 1, or Scale approach 2.

To force Central Console to push Intelligence Stream updates down to all tenants, go to **Manage > System > Intelligence** in the Central Console and click **Update Now**.

Download the IS data with twistcli

Before starting, ensure the Internet-connected host to where you will initially download the data can access the Intelligence Stream. The most reliable way to test connectivity is to ping the Intelligence Stream. This following curl command verifies that name resolution and any intermediary HTTP proxies are functioning properly.

```
$ curl -k \
  --silent \
  --output /dev/null \
  --write-out "%{http_code}\n" \
  https://intelligence.twistlock.com/api/v1/_ping
```

If you’ve got connectivity, you’ll get back a 200 (Successful) response code.

```
200
```

STEP 1 | Open Console.

STEP 2 | Go to **Manage > System > Intelligence**.

STEP 3 | Copy the access token.

STEP 4 | Download twistcli. You have several options:

- Download twistcli from the Console UI. Go to **Manage > System > Utilities**.
- Download twistcli from the API. Use */api/v1/util/twistcli* for the Linux binary or */api/v1/util/osx/twistcli* for the macOS binary..
- Get a copy from the release tarball.

STEP 5 | Download the the Intelligence Stream data.

Open a shell window, and run the following command:

```
$ ./linux/twistcli intelligence download --token <ACCESS-TOKEN>
```

All data is downloaded and saved in a file named *twistlock_feed_<random_string>.tar.gz*

Upload IS data to Console with twistcli

Use the *twistcli* tool to upload the Intelligence Stream archive to your Prisma Cloud Console.

Prerequisite: You've disabled over-the-Internet updates for your air-gapped Console. Go to **Manage > System > Intelligence** and set **Update the Intelligence Stream from Prisma Cloud over the Internet** to **Off**.

STEP 1 | Download twistcli. You have several options:

- Download twistcli from the Console UI. Go to **Manage > System > Utilities**.
- Download twistcli from the API. Use */api/v1/util/twistcli* for the Linux binary or */api/v1/util/osx/twistcli* for the macOS binary..
- Get a copy from the release tarball.

STEP 2 | Run the following command:

```
$ ./linux/twistcli intelligence upload \
  --address \https://<COMPUTE-CONSOLE>:8083 \
  --user <USER> \
  --password <PASSWORD> \
  --tlscacert <PATH-TO-CERT> \
  <FEED-ARCHIVE>
```

Where:

- ***<COMPUTE-CONSOLE>*** --

  URL for the air-gapped Console.

- ***<USER>***, ***<PASSWD>*** --

  Credentials for a user with a minumum role of Vulnerability Manager.

- ***<PATH-TO-CERT>*** --

  (Optional) Path to to Prisma Cloud's CA certificate file. With the CA cert, a secure connection is used to upload the intelligence data to Console. For example, */var/lib/twistlock/certificates/console-cert.pem*.

- ***<FEED-ARCHIVE>*** --

  File generated from downloading an archive of the IS with twistcli. For example, *twistlock_feed_1524655717.tar.gz*.

Sometimes after Console is restarted, you might see an error on the login page that says "failed to query license". This is by design, and it's not a bug. It happens because a Console restart triggers a user auth token renewal. For more information, see long-lived tokens.

Download the IS from an HTTP server

Configure Console to download the IS archive file from a custom HTTPS location.

When enabled, Console downloads the file from this location every 24 hours. If the download fails, Console retries every 1 hour until it's successful, then waits for 24 hours until the next download.

In this strategy, you must get the latest IS data with twistcli and copy the archive file to the HTTP/S server, where the air-gapped Console(s) will retrieve it.

STEP 1 | Open Console.

STEP 2 | Go to **Manage > System > Intelligence**.

STEP 3 | Set **Update the Intelligence Stream from a custom location** to **On**.

STEP 4 | In **Address**, specify the full URL to the HTTP/S endpoint where the archive is served.

STEP 5 | If credentials are required to access this endpoint, create them.

STEP 6 | (Optional) Configure a certificate chain for trusting the HTTPS endpoint.

STEP 7 | Click **Save**.

Console immediately attempts to load the IS data from the specified endpoint. Assuming, Console is successful, it schedules subsequent updates every 24 hours. Click **Update Now** to force an immediate update.

Download the IS from another Console

You can configure a Console to retrieve the latest Intelligence Stream data from another Console. In this configuration, you have a single relay Console, and all other deployed Consoles connect to it to retrieve the latest Intelligence Stream data.

When enabled, Console downloads the file from this location every 24 hours. If the download fails, Console retries every 1 hour until it's successful, then waits for 24 hours until the next download.

In this strategy, you must implement a method for the relay Console to get a copy of the latest Intelligence Stream data.

STEP 1 | Open Console.

STEP 2 | Go to **Manage > System > Intelligence**.

STEP 3 | Set **Update the Intelligence Stream from a custom location** to **On**.

STEP 4 | In **Address**, specify the full URL to the relay Console.

https://<COMPUTE-CONSOLE>:8083/api/v1/feeds/bundle

Where:

- ***<COMPUTE-CONSOLE>*** --

  URL for the relay Console.

STEP 5 | In **Credential**, create basic auth credentials for a user that has a minimum role of Vulnerability Manager.

STEP 6 | Enter a certificate to trust the HTTPS endpoint.

1. Copy the relay Console's certificate from */var/lib/twistlock/certificates/ca.pem*, and paste it here.

STEP 7 | Click **Save**.

Console immediately attempts to load the IS data from the specified endpoint. Assuming, Console is successful, it schedules subsequent updates every 24 hours. Click **Update Now** to force an immediate update.

Deployment patterns

Edit on GitHub

As you prepare to deploy Prisma Cloud, consider how to tailor it fit into your environment. Prisma Cloud supports multitenancy, which gives you a way to manage all your deployments from a single interface, and control which data each team can see.

- Projects
- Migration options for scale projects
- Best practices for DNS and certificate management
- Storage limits for audits and reports
- Migrating to a SaaS Console
- Performance planning
- Automated deployment
- High Availability and Disaster Recovery guidelines

Projects

Edit on GitHub

Some deployments must be compartmentalized for regulatory or operational reasons. Projects solve the problem of multi-tenancy. Each project, or tenant, consists of a Console and its Defenders. Each project is a separate, compartmentalized environment which operates independently with its own rules and configurations.

Projects are federated behind a single master Console with a single URL. For example, https://console.customer.com might be the URL for accessing the master Console UI and API. Tenant projects are deployed, accessed, and managed from the single master Console. You could deploy a tenant Console for each business unit, giving each team their own segregated environment. Each team accesses their tenant through the master Console's URL.

Role-based access control (RBAC) rules manage who can access which project. When users log onto Prisma Cloud Central Console, they are shown a list of projects to which they have access and can switch between them.

Scale projects have been deprecated. If you've deployed a scale project, see migration options for scale projects for more information about how to transition to a supported configuration.

Terminology

The following terms are used throughout this article:

- **Central Console** --

  Also known as the master Console or just master. This is the interface from which administrators manage (create, access, and delete) their projects.

- **Supervisor** --

  Secondary, slave Console responsible for the operation of a project. Supervisor Consoles are headless. Their UI and API are not directly accessible. Instead, users interact with a project from Central Console's UI and API.

- **Project (also tenant project, or just tenant)** --

  Deployment unit that consists of a supervisor Console and it's connected Defenders. Tenant projects are like silos. Each tenant maintains its own rules and settings, separate from Central Console and any other tenant.

When to use projects

Carefully assess whether you need projects. Provisioning projects when they are not required will needlessly complicate the operation and administration of your environment.

1. Do you have multiple segregated environments, where each environment must be configured with its own rules and policies?

If yes, then deploy a tenant project for each environment.

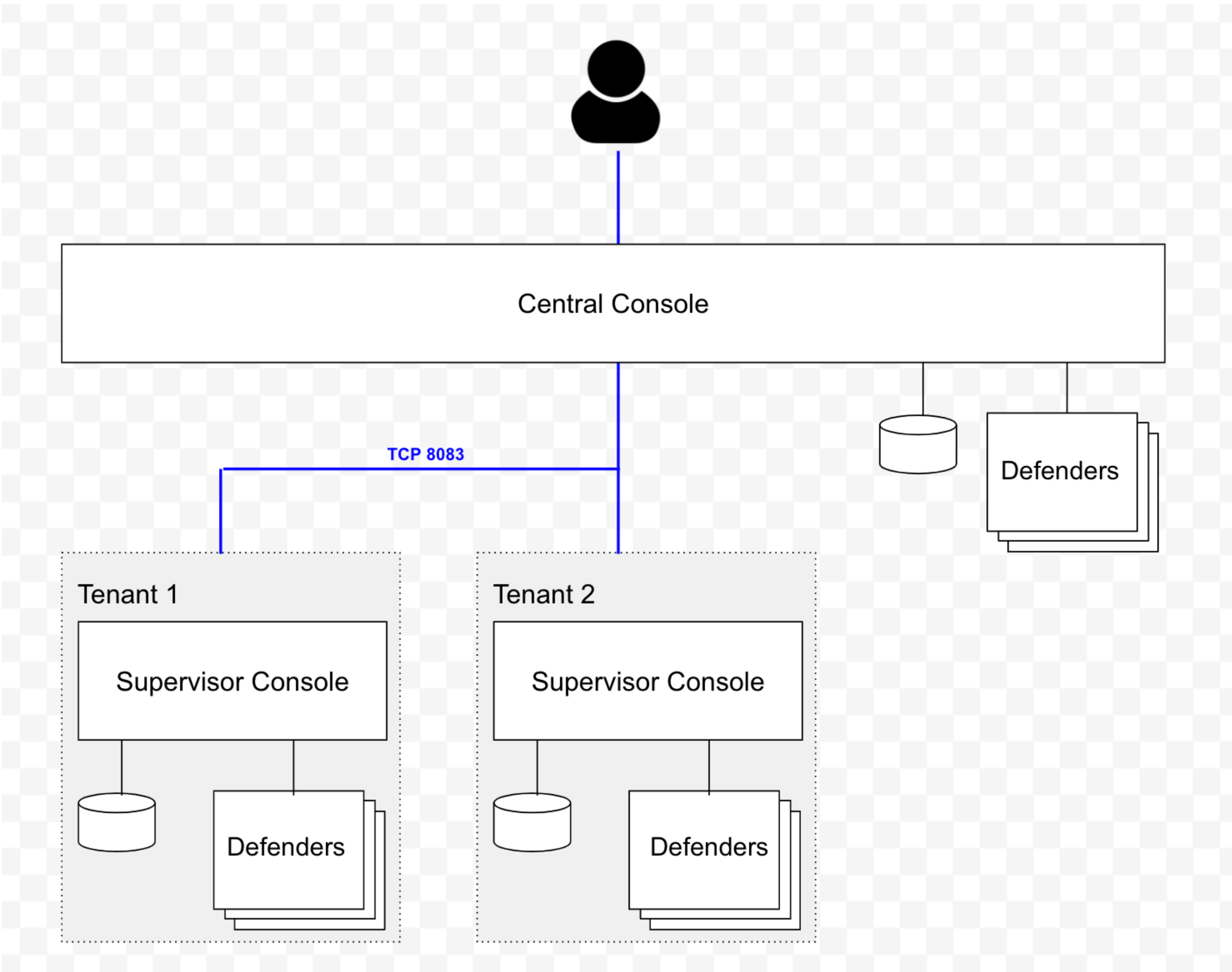
2. If you choose not to use projects now, can you migrate to projects at a later time?

Yes. Even if you choose not to use projects now, you're not locked into that decision. You can always migrate to projects at a later time. For more information, see Migration strategies.

Architecture

Projects federate the UI and API for multiple Consoles.

For example, if you have three separate instances of Consoles for development, test, and production environments, projects let you manage all of them from a single Central Console. With projects, one Console is designated as the master and all others are designated as supervisors. Thereafter, all UI and API requests for a project are proxied through the master and routed to the relevant supervisor. Supervisors do not serve a UI or API.



Connectivity

By default, the master and its supervisor Consoles communicate over port 8083. You can configure a different port by setting MANAGEMENT_PORT_HTTPS in *twistlock.cfg* at install time. All Consoles must use the same value for MANAGEMENT_PORT_HTTPS. Communication between the master and supervisor Consoles must be direct, and cannot be routed through a proxy.

Defenders communicate with their respective supervisor Consoles. Project Defenders never communicate directly with the Central Console.

Prisma Cloud CA signed certs are used for establishing the Central Console to supervisor Console communication link. Since no user interacts with the supervisor Console directly, the link is an internal architecte detail, and we use our own CA. This setup reduces the risk of outages due to expired certs.

When configuring Central and supervisor Consoles, you must configure the supervisor Console to include the Subject Alternative Name (SAN) *for the Central Console.*

When configuring access to the Consoles via Ingress Network Routes in Kubernetes, you must add the Central Console to the supervisor Console Ingress configuration.

Central Console can have its own set of Defenders. In this case, these Defenders do communicate directly with Central Console. However, no project Defenders ever communicate directly with Central Console.

Access control

When users log into Prisma Cloud Console, they are presented with a list of projects to which they have access, and they can chose the project they want to work in. Access to projects is controlled by role-based access control rules.

You can grant access to specific projects for any 'local' users created in Console under **Manage > Authentication > Users**. If you have integrated Console with an OpenLDAP, Active Directory, or SAML provider, you can grant access to projects by group. Users and groups can be granted access to multiple projects.

A user's role is applied globally across all projects. That is, a user will have the same role for each project for which he has been granted access.

Project access control rules at the user level takes precedence over access control granted at the group level. For example, if a 'local' user has been granted access to project1, but also belongs to group1, which has been granted access to project2, he will only have permissions to access project1.

Secrets

Prisma Cloud fully supports secrets management for tenant projects. Secrets management can be independently configured and managed for each tenant project.

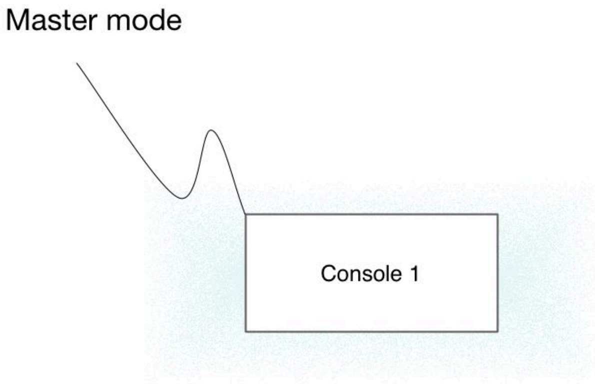
Limitations

Moving Defenders between projects is not supported. To "move" a Defender, decommission it from one project and deploy it to another.

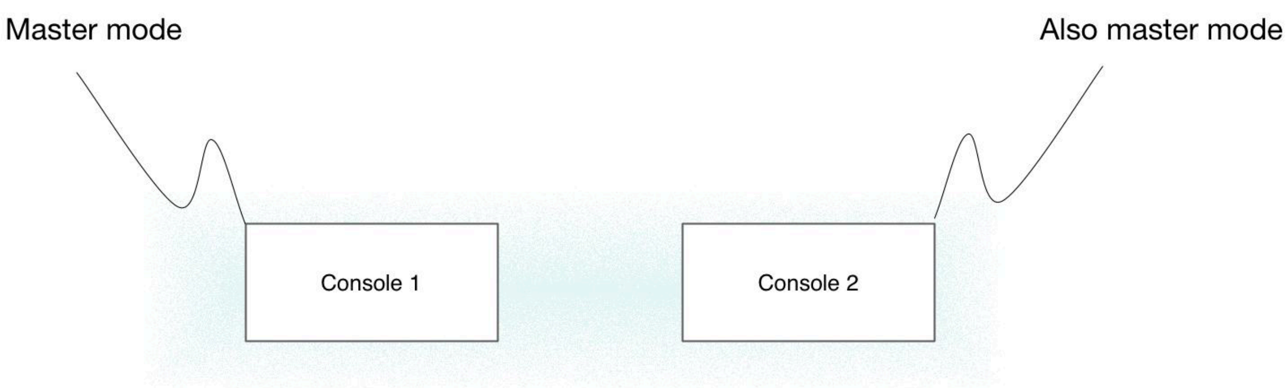
Provisioning flow

Let's look at how projects are provisioned.

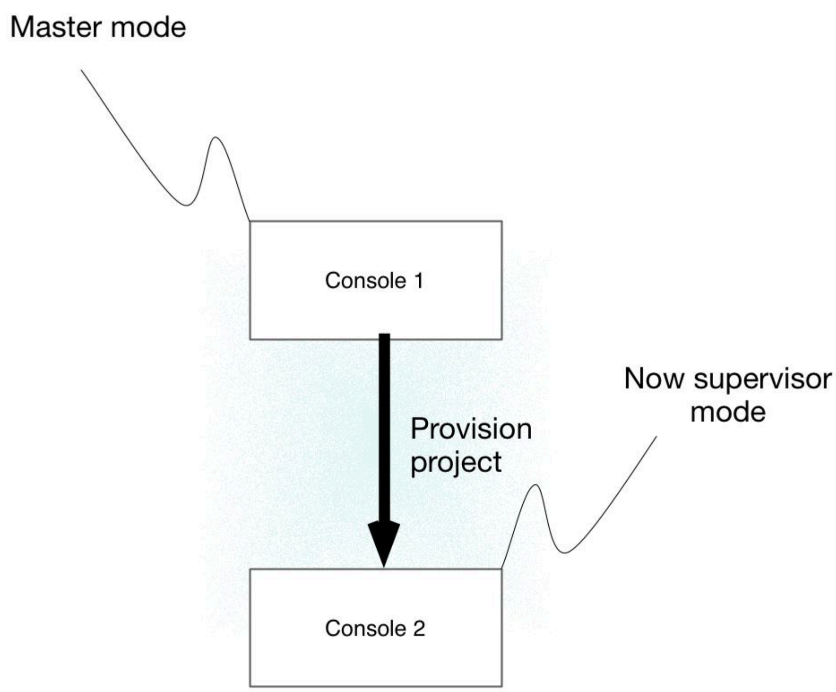
Step 1: Install Console using any installation method. For example, you could install Console (onebox) with the *twistlock.sh* script or as a service in a Kubernetes cluster. When Console is installed, it runs in master mode by default.



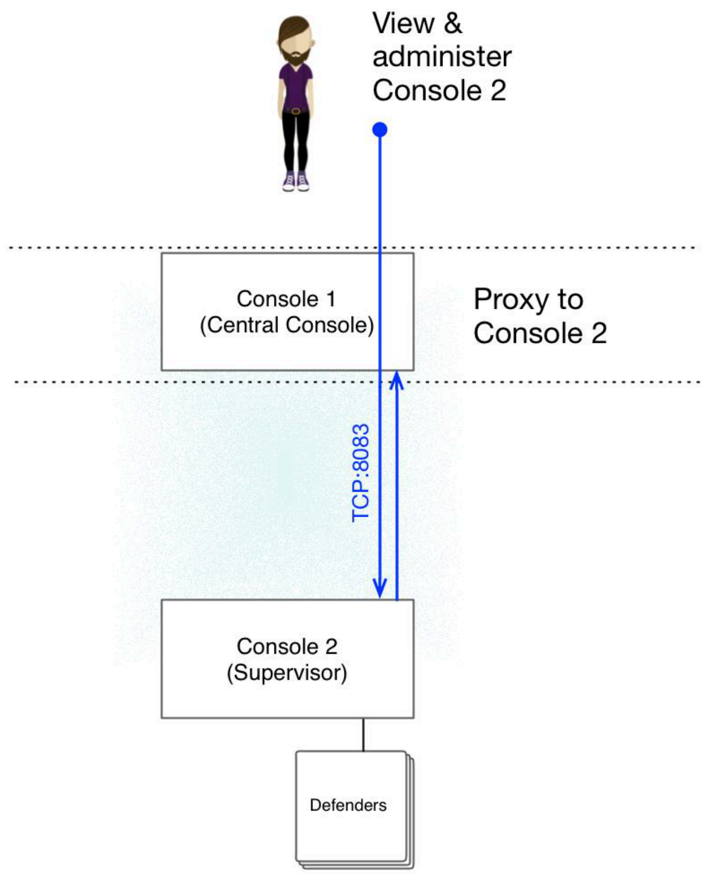
Step 2: Install a second Console on a different host. By default, it also runs in master mode.



Step 3: In the UI for Console 1, provision a new project. Specify the URL to Console 2. The provisioning process automatically changes the operating mode for Console 2 to supervisor. The UI and API for Console 2 are now no longer directly accessible.



Step 4: The only difference between a master Console and a supervisor Console is whether its UI and API can be accessed directly, or whether it is proxied through the master. To view your tenant project (managed by Console 2), open Console 1 and select the project. All your rules and settings for your project are loaded and displayed in Console 1.



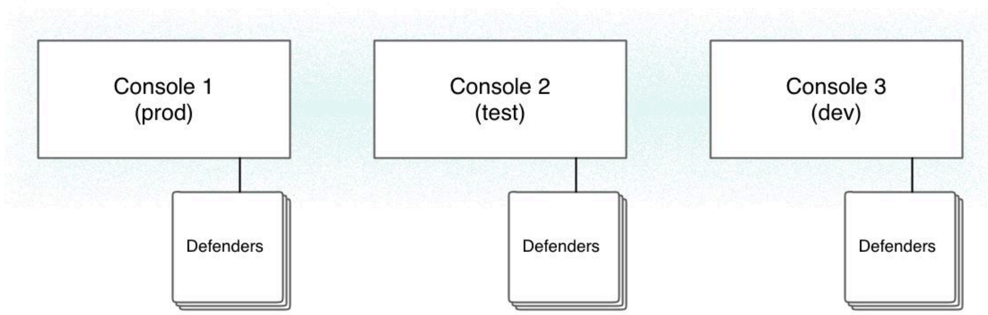
You can release a supervisor, and return it to its original state, by deleting the project. The supervisor Console reverts back to master mode.

Migration strategies

If you have already deployed one or more stand-alone Consoles, and you want to adopt a project-based structure, then the migration is easy. Designate one Console as master, then designate each remaining Console as a supervisor by provisioning projects for them.

Adding an existing Console to a project is not a destructive operation. All data is preserved, and the process can be reversed. The only thing that changes is the way you access Console when it's mode changes to supervisor. Supervisor Consoles cannot be accessed directly. They can only be accessed through the master Console, by selecting a project from the **Selected project** drop-down list.

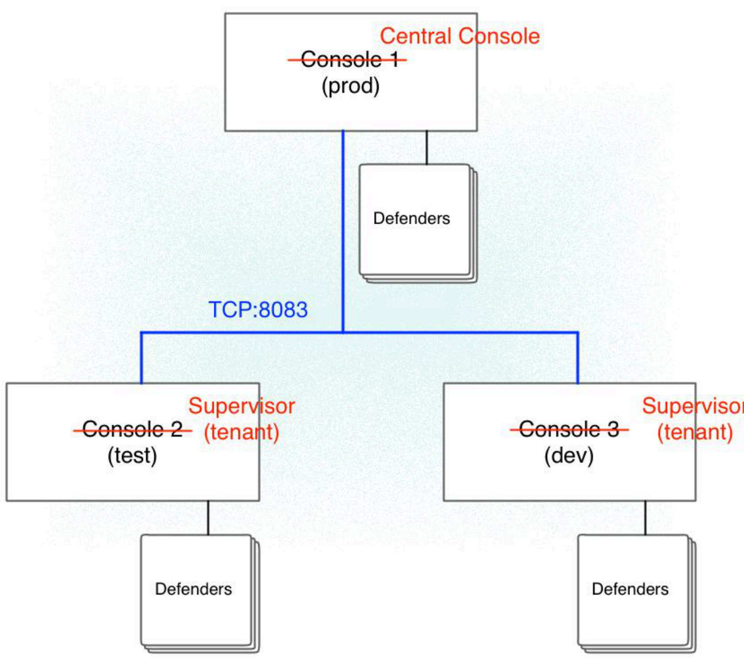
For example, assume you've deployed three separate stand-alone Consoles: one for your production environment, one for your test environment, and one for your development environment.



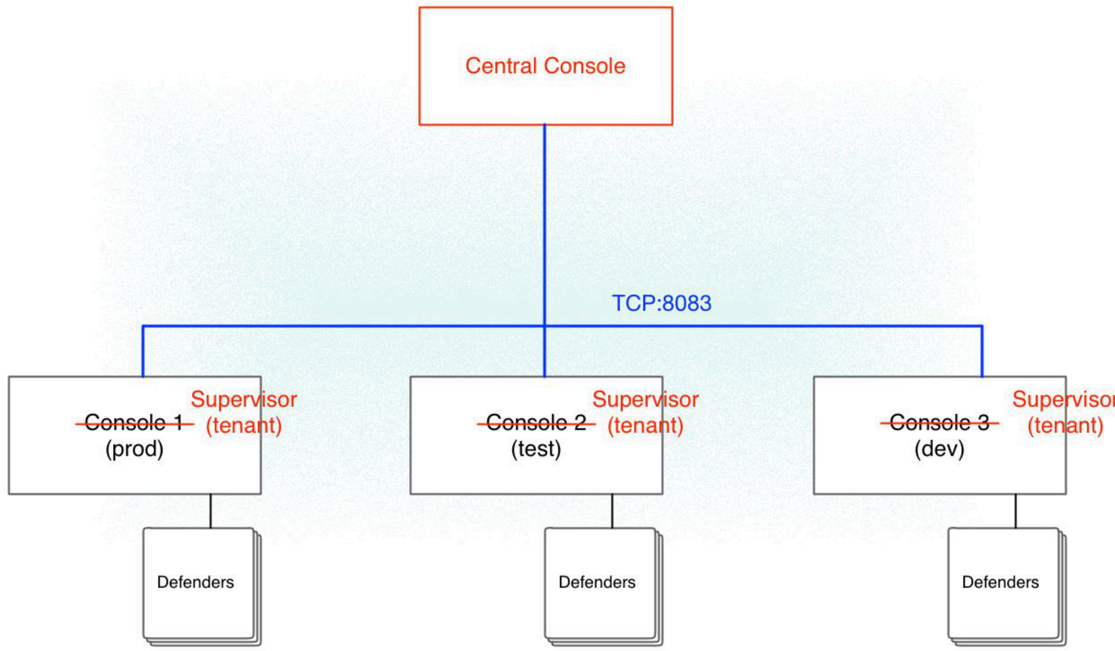
When migrating to projects, you have the following options:

Option 1: Promote one Console to master, and designate the others as supervisors. In this example, you pick the prod Console to be master, then create tenant projects for the test and development Consoles.

By default, Consoles run in master mode when they are installed, so you don't need to do anything to "promote" prod to master. To relegate test and dev to supervisor, provision a project for each one.



Option 2: Install a new Console on a dedicated host and designate it as master. Provision a tenant project for each of the prod, test, and dev Consoles.



Accessing the API

All API requests should be routed to Central Console only. Central Console checks if the client has the correct permissions to access the given project, and then Central Console redirects the request to right supervisor, and then returns to supervisor's response to the client.

For API requests that create, modify, or delete data, Central Console responds to the client with a success return code, and then updates the supervisor asynchronously.

To target an API request to a specific project, append the *project=* query parameter to your request. For example, to get a list of Defenders deployed in the *prod* project:

```
GET http://<CENTRAL-CONSOLE>:8083/api/v1/defenders?project=prod
```

Central Console reroutes the request to the appropriate supervisor. Not all requests need to be rerouted. For example, the endpoints for getting a list of users, groups, or projects are handled by Central Console directly. Some endpoints require no special permissions to access them, such as getting a list projects to which a user has been granted access.

Provisioning a project

Provision new projects from the Central Console UI.

Communication between the master and supervisor Consoles must be direct, and cannot be routed through a proxy.

STEP 1 | Install a Console on a host in your environment using any install procedure.

There is no need to create an admin user or enter your license. Those details will be handled for you in the provisioning phase of this procedure.

STEP 2 | Register the newly installed Console with the Central Console and create a project.

STEP 3 | Go to **Manage > Projects > Manage**

STEP 4 | Set **Use Projects** to **On**.

STEP 5 | Click **Provision project**.

STEP 6 | In **Project name**, give your project a name.

STEP 7 | In **Supervisor address**, enter the URL for accessing Console Include both the protocol (https://) and port.

STEP 8 | For a fresh Console install, there is no need to enter any credentials. They will be created for you automatically.

If you are migrating an existing Console to a project, specify the admin credentials.

Decommissioning a project

Decommissioning a project simply reverts the supervisor Console back to a stand-alone master Console. The link between Central Console and the former supervisor Console is severed. All project data (rules, audits, scan reports) is left in tact.

When a project is created, the Console is configured with an admin user. When you delete the project, the admin credentials are shown to you so that you can continue to access and administer it. The credentials are shown only one time, so copy them, and set them aside in a safe place.

STEP 1 | Open Central Console.

STEP 2 | Go to **Manage > Projects > Manage**.

STEP 3 | In the **Provisioned Projects** table, click delete on the project you want to delete.

Decommissioning disconnected projects

Central Console lets you delete projects, even if the supervisor Console is disconnected. The project is deleted from the master's database, but it leaves the supervisor Console in the wrong state.

When you delete a disconnected project, Prisma Cloud tells you that the supervisor cannot be reached. To manually revert the supervisor Console back to a stand-alone master Console, call the supervisor's REST API to change its settings.

STEP 1 | Decide how you want to access the supervisor's REST API. You can use basic auth or an auth token.

STEP 2 | Update the supervisor's project settings. The following example command uses basic auth. Only admin users are permitted to change project settings.

```
$ curl -k \
  -u <USER> \
  -X POST \
  -H 'Content-Type:application/json' \
```

```
  -d '{"master":false, "redirectURL":""}' \
  https://<SUPERVISOR-CONSOLE>:8083/api/v1/settings/projects
```

Deploying Defender DaemonSets for projects (Console UI)

When creating a DaemonSet for a project, you can use the Console UI, twistcli, or API. This section shows you how to use the Console UI.

STEP 1 | In Console, use the drop-down menu at the top right of the UI to select the project where you want to deploy your DaemonSet.

STEP 2 | Go to **Manage > Defenders > Deploy Daemon Set**.

STEP 3 | Configure the deployment parameters, then copy and run the resulting install script.

Deploying Defender DaemonSets for projects (twistcli)

Create a DaemonSet deployment file with twistcli. Specify both the project name and the DNS name or IP address of the supervisor Console to which the DaemonSet Defenders will connect. The DNS name or IP address must be a Subject Alternative Name in the supervisor Console's certificate.

```
$ <PLATFORM>/twistcli defender export kubernetes \
  --address https://<CENTRAL-CONSOLE>:8083 \
  --project <PROJECT-NAME>
  --user <USER> \
  --cluster-address <SUPERVISOR-CONSOLE-SAN>
```

Deploying Defender DaemonSets for projects (API)

A DaemonSet deployment file can also be created with the API. Specify both the project name and the DNS name or IP address of the supervisor Console to which the DaemonSet Defenders will connect. The DNS name or IP address must be a Subject Alternative Name in the supervisor Console's certificate.

```
$ curl -k \
  -u <USER>
  -X GET \
  'https://<CENTRAL-CONSOLE>:8083/api/v1/defenders/daemonset.yaml?
consoleaddr=<SUPERVISOR_CONSOLE_SAN>&listener=none&namespace=twistlock&orchestra
```

Migration options for scale projects

Edit on GitHub

Starting in 20.12, Console has substantially increased the number of simultaneous Defenders it can support. Each instance of Console can support 10K Defenders. With this new capability, scale projects have been deprecated.

Scale projects gave security teams full control over policies for all application teams. If you're currently using scale projects, we offer the following migration paths when upgrading to 20.12

Migration paths

If you're using scale projects, there are two ways you can migrate to a supported configuration.

1. Convert existing scale projects to tenant projects --

When upgrading to 20.12, all existing scale projects will automatically be converted into tenant projects. Scale project policy rules will be converted to tenant project policy rules. From that point, any changes to the tenant project policies will only apply to the project itself, without any sync with Central Console.

If you choose this migration option, reevaluate the roles assigned to your users. After upgrading, users with the Admin, Operator, or Vulnerability Manager roles on the converted projects (now tenant projects) will have the ability to edit policy rules, so you might need to lower their privileges.

2. Unify the scale projects into Central Console --

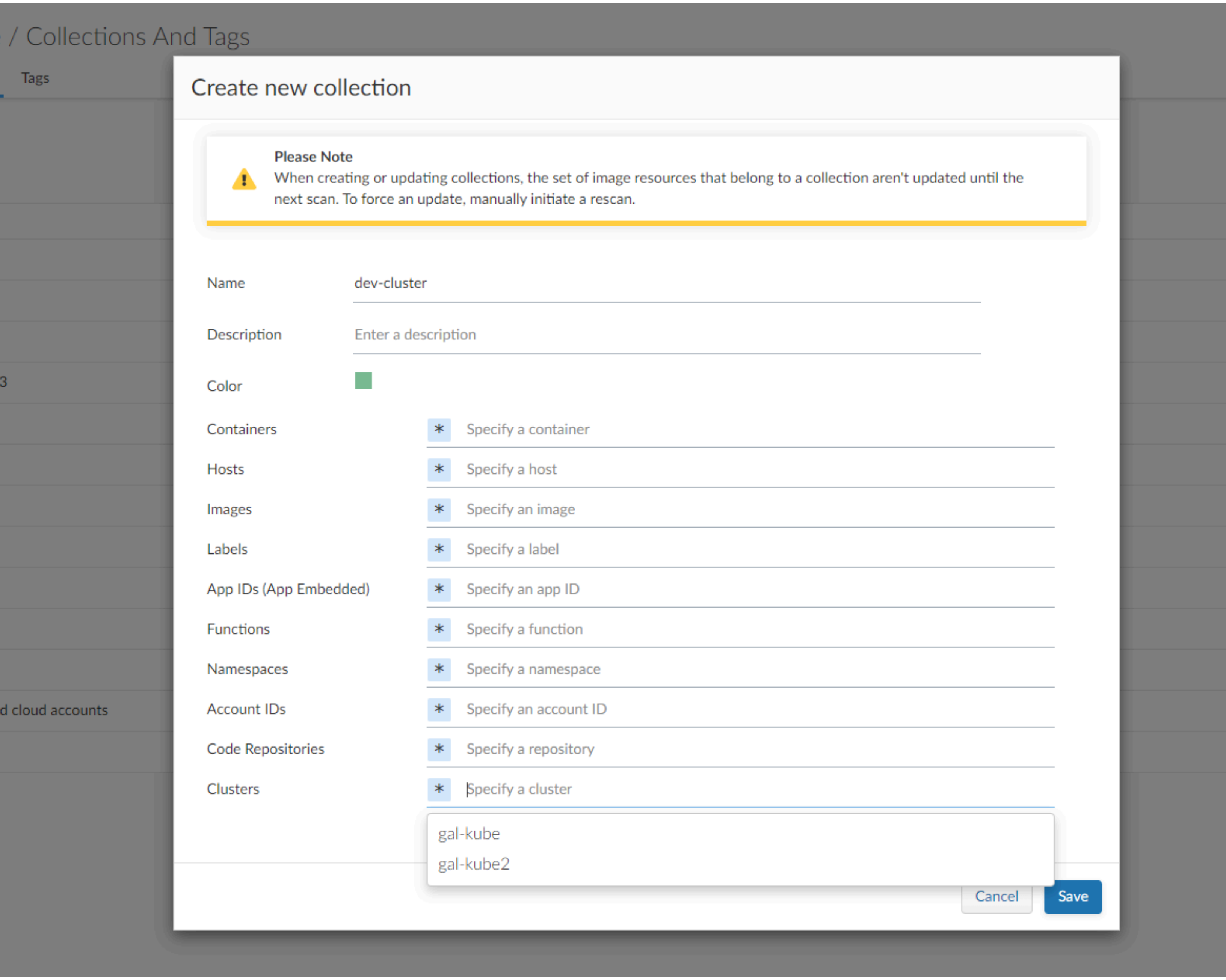
Before upgrading to 20.12, redeploy all scale project Defenders and connect them directly to Central Console. Use collections and RBAC to control which resources can be viewed and managed by different users (see example below).

If you have more than 10K Defenders, consider deploying more than one tenant project. To share policies between tenants, develop an automated process on top of the API to push policies from one Console to the other.

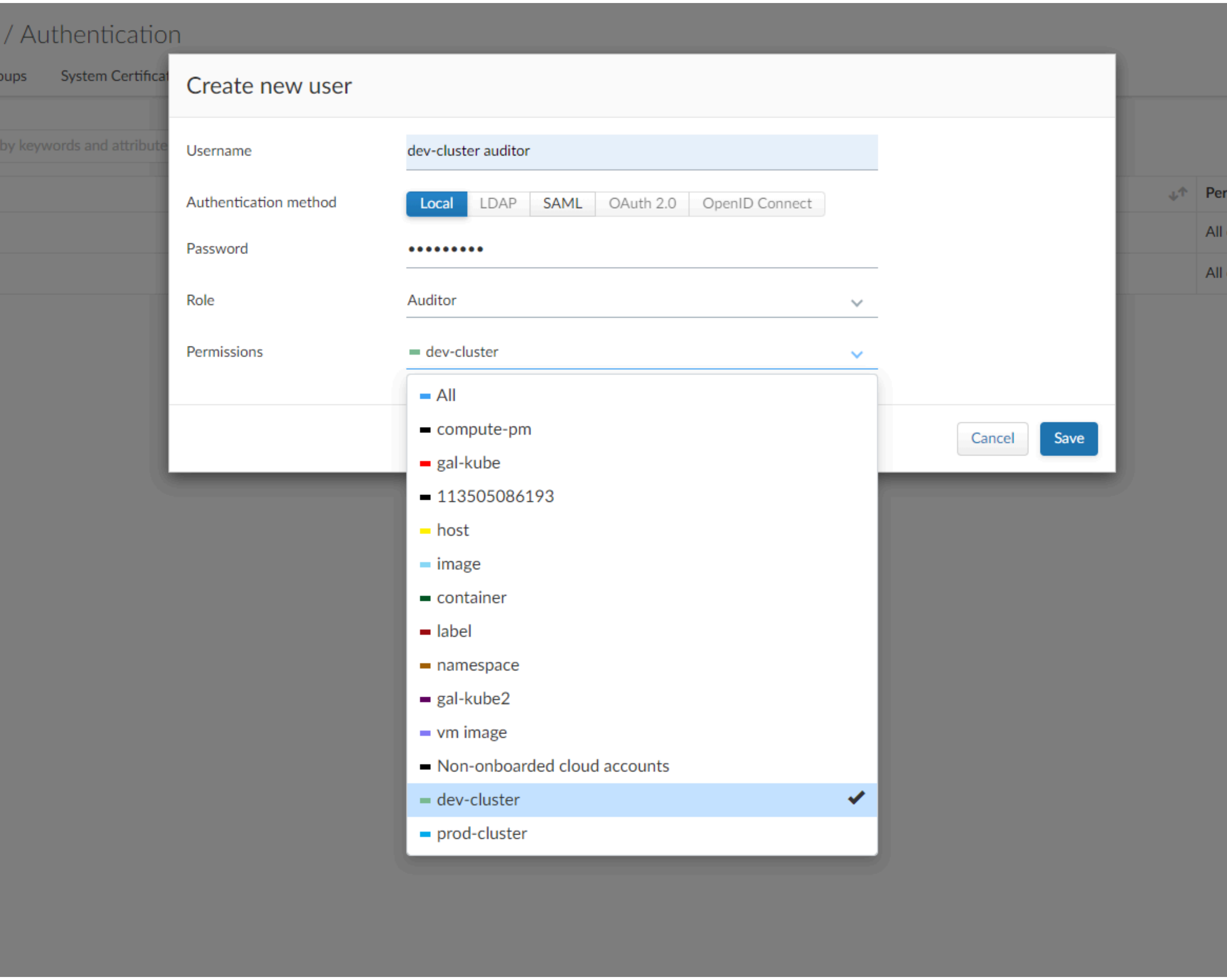
Using collections

Examples of how to use collections in your migration.

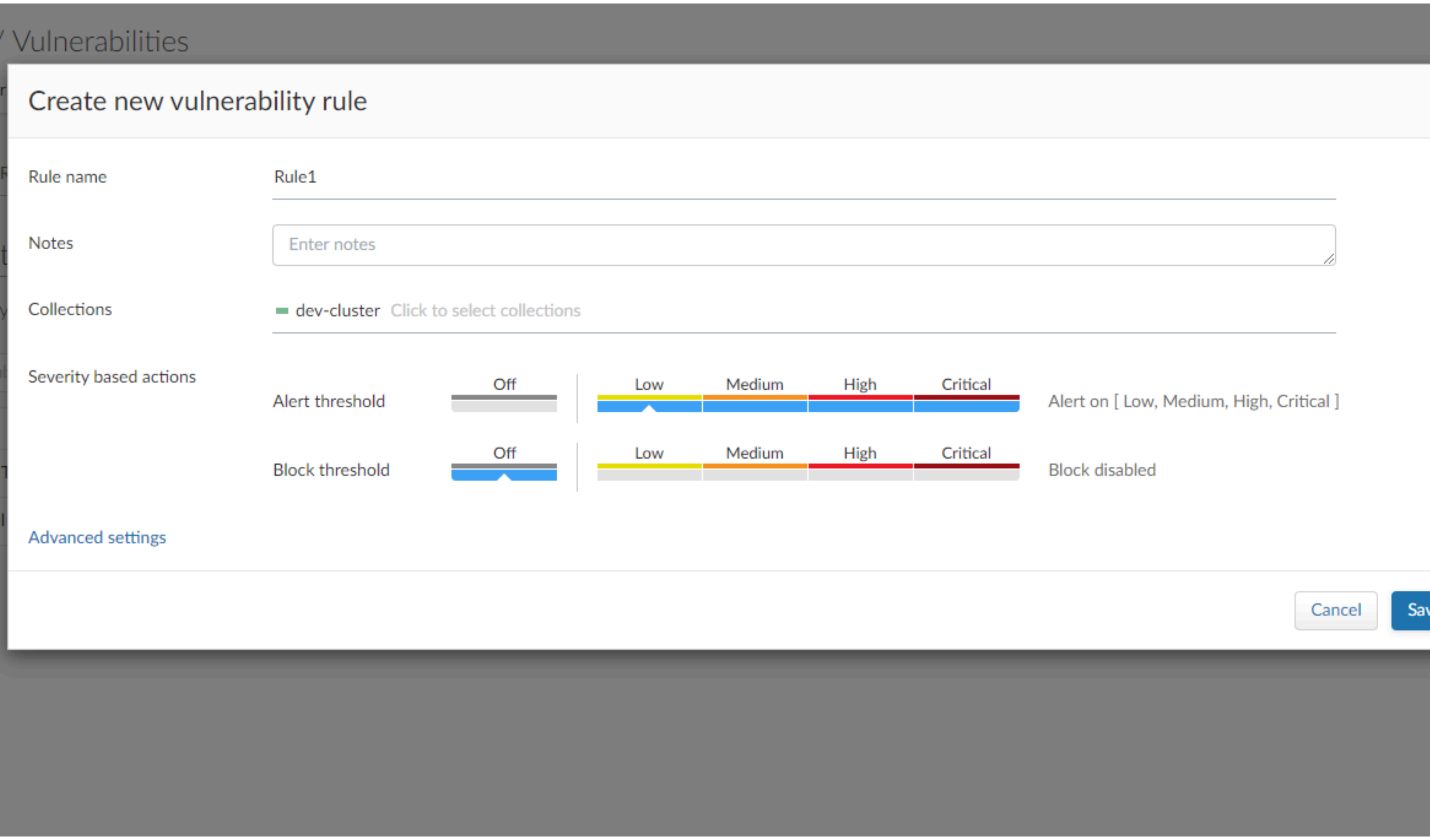
Create a collection for a specific cluster:



Assign the collection to a user:



Collections can also be used when defining policies:


Vulnerabilities
Create new vulnerability rule
Rule name
Rule1
Notes
Enter notes
Collections
dev-cluster
Click to select collections
Severity based actions
Alert threshold
Off
Low
Medium
High
Critical
Alert on [Low, Medium, High, Critical]
Block threshold
Block disabled
Advanced settings
Cancel

Best practices for DNS and certificate management

Edit on GitHub

As with most cloud-native software, Prisma Cloud relies on core infrastructure services, such as x509 cryptography and DNS name resolution. Defenders use these services to find and securely connect back to Console, and administrators use them to connect to Console and the API endpoints. When Console's name can't be resolved, or its certificate doesn't include the name that Defenders use to connect to it, set up might fail and/or Defenders might not be able to successfully connect to Console.

In relatively simple environments, such as an on-premises environment with a flat network, Prisma Cloud can automatically discover and configure the appropriate network configuration during setup. However, in more complex environments, auto-discovery is difficult and administrators typically have to manually configure the appropriate settings.

Consider a deployment where Console exists in one cloud service, but protects hosts distributed across other cloud services in different regions. In this model, Console's hostname is probably not resolvable by remote Defenders. And since Defenders probably do not connect directly to Console, but through some reverse NAT or a load balancer, the details of the underlying connectivity are probably obscured.

Map out your topology

Mapping out your topology is a fairly obvious step that is often overlooked, but it is the single best way to avoid connectivity problems.

First, document Console's local hostname and IP. Try to determine whether this name is the actual name that Defenders will use to connect, or if there is an another entity in between, such as a load balancer or reverse NAT service.

Then, map out all the potential connection paths from Defenders to Console. For example, there might be some Defenders deployed in the same cloud service as Console. They can connect to Console directly. Other Defenders might connect from another routed network or over the Internet using different names.

Documenting all of these paths and names at the beginning of the planning process saves significant time later when you're troubleshooting. Use the following sample worksheet as a starting point:

```
Console IP address:
Console local host name:
Console management port:
Console / Defender communication port:
```

```
Load balancer / NAT IP address Load balancer / NAT name:
Load balancer / NAT management port:
Load balancer / NAT Console / Defender communication port:
```

```
Defender to Console connection paths:
  Direct?
  From other cloud services in same deployment?
```



```
Over the Internet?
```

Because naming is so critical to connectivity, you should use durable, Prisma Cloud-specific names for accessing Console. For example, although the default host name might be ip-10-1-27-12, it would be a poor choice because it's tied to a specific hostname, which could change if you redeploy Console to a new host.

Instead, create a CNAME with a short TTL to reference this hostname, and use the CNAME for all name resolution. This way, if your hostname changes in the future, you simply need to remap the CNAME to the new hostname. Using CNAMEs is preferable to directly mapping an A record because many cloud services automate DNS resolution within their fabric and offer limited options for overriding this behavior. In a complex, multi-network environment, the CNAME can be used to reference Console both from the local network and from other networks, including the Internet, through simple and well established DNS configurations.

Consider the following example scenario:

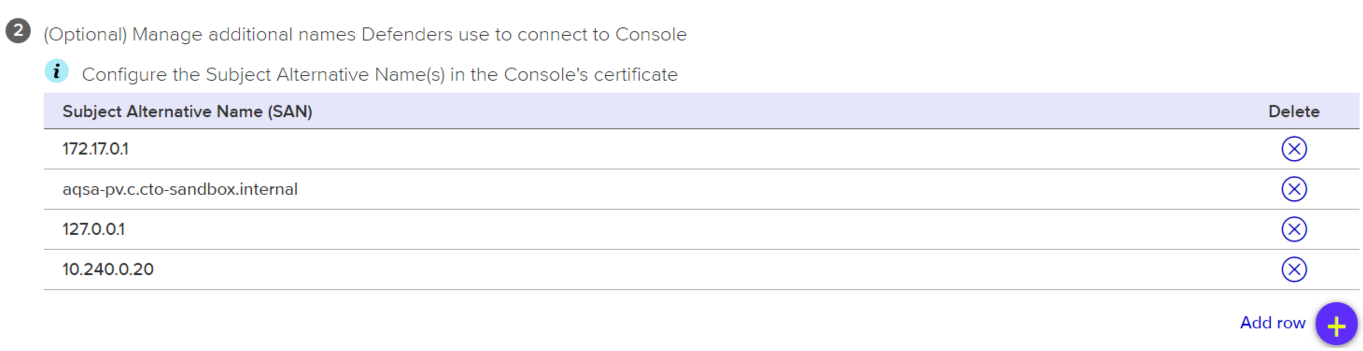
- Console runs in cloud network 1, with an IP of 10.1.27.12, and local hostname of ip-10-1-27-12.
- This IP can be accessed over the Internet through a load balancer.
- The load balancer's IP is 100.4.1.8, with a name of lb1.cloudprovider.com.
- Some Defenders also run in cloud network 1.
- Other Defenders run in a data center in another region, and connect to Console over the Internet.

In this scenario, a good approach would be to create a CNAME, such as console.customer.com. Internet facing DNS servers would answer queries for Console with lb1.cloudprovider.com. Internal facing DNS servers would answer queries for Console with ip-10-1-27-12.

Implement the topology

After your naming scheme has been planned, the final step is implementing the names in Prisma Cloud.

When you deploy a Defender, you must specify how it connects to Console, with either an IP address or, preferably, a DNS name. The Prisma Cloud dashboard lets you specify these names, and provides some preconfigured names, in the **Subject Alternative Names** table on the **Manage > Defenders > Deploy** page. Any name in the table is added to Console's certificate and becomes available as a configuration parameter in the Defender deployment pages.


2 (Optional) Manage additional names Defenders use to connect to Console

Configure the Subject Alternative Name(s) in the Console's certificate

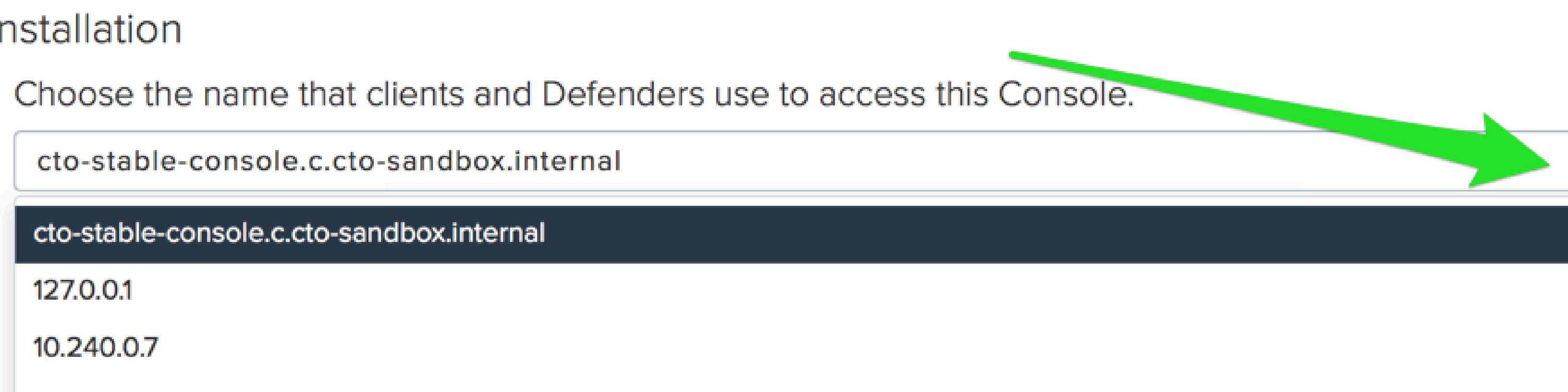
Subject Alternative Name (SAN)	Delete
172.17.0.1	
aqsa-pv.c.cto-sandbox.internal	
127.0.0.1	
10.240.0.20	

Add row

Using our example scenario described in the previous section, the Subject Alternative Name table should contain the CNAME we chose (console.customer.com). If you have multiple names that you want to use to address Console, add them to the Subject Alternative Name table. For example, if Defenders in the same cloud network should access Console using cs1-console, you should have the following entries:

- console.customer.com
- cs1-console

After Prisma Cloud is set up with these values, you will see them in the drop down menu in all of the Defender deployment pages as a configuration parameter. When you set up a new Defender, select how it should connect to Console from the same list of names in the Subject Alternative Names table.



When you're installing Defender, always ensure that the name you select from the drop down list can be resolved from the host where Defender will run. Using our example scenario, this means that you would select cs1-console for 'local' hosts that run in the same cloud service as the Console, and that you would select console.customer.com for 'remote' hosts. If the name you select cannot be resolved from the host where you install Defender, Defender set up will fail.

Updating the list of resolvable names for Console

Define additional names Defenders can use to connect to Console. After adding a name to the Subject Alternative Name table, the name is added to Console's certificate and it is available in the drop down list in the Defender deployment pages.

The values for CONSOLE_CN and DEFENDER_CN in twistlock.cfg should never be modified unless you are directed to do so by Prisma Cloud Support. These values are needed to work around distribution-specific abnormalities in the hostname command, which we use to create certificates during set up. Your custom names should always go in the Subject Alternative Name table, and never be hard-coded into CONSOLE_CN or DEFENDER_CN.

STEP 1 | In Console, go to **Manage > Defenders > Deploy**.

STEP 2 | In the **Subject Alternative Name** table, click **Add row**.

STEP 3 | Specify an IP address or fully qualified domain name.

STEP 4 | Redeploy any Defenders that require the new name to connect to Console.

If the old names are still accessible, this step can be skipped.

Storage limits for audits and reports

Edit on GitHub

Prisma Cloud restricts the size of some data collections to prevent misconfigured or noisy systems from consuming all available disk space and compromising the availability of the Console service.

Registry scanning

Prisma Cloud scans a maximum of 100,000 registry images, ordered by most recently published. Publish date is the time an image is pushed to the registry.

Data collections limits

The following limits are currently enforced in Console's database.

For audits: if you must retain all audits, consider configuring Console to send audits to syslog, and then forward the audits to a log management system for long term storage.

Collection	Cap
Jenkins plugin and twistcli scan reports	5000 scan reports or 500 MB (whichever limit is reached first)
Container runtime audits	25K audits or 50 MB (whichever limit is reached first)
Container network firewall audits	25K audits or 50 MB (whichever limit is reached first)
Image sandbox analysis reports	5000 scan reports or 500 MB (whichever limit is reached first)
Access audits	100K audits or 50 MB (whichever limit is reached first)
Kubernetes audits	100K audits or 50 MB (whichever limit is reached first)
Admission audits	100K audits or 50 MB (whichever limit is reached first)
Log inspection events	100K audits or 50 MB (whichever limit is reached first)
File integrity events	100K audits or 50 MB (whichever limit is reached first)
Host activities	100K audits or 50 MB (whichever limit is hit first)
Host history	100K audits or 50 MB (whichever limit is reached first)
Host runtime audits	25K audits or 50 MB (whichever limit is reached first)

Collection	Cap
Host network firewall audits	25K audits or 50 MB (whichever limit is reached first)
Serverless runtime audits	25K audits or 50 MB (whichever limit is reached first)
App-Embedded runtime audits	25K audits or 50 MB (whichever limit is reached first)
Trust audits	25K audits or 50 MB (whichever limit is reached first)
WAAS for containers events	200K audits or 200 MB (whichever limit is reached first)
WAAS for hosts events	200K audits or 200 MB (whichever limit is reached first)
WAAS for serverless events	200K audits or 200 MB (whichever limit is reached first)
WAAS for app-embedded events	200K audits or 200 MB (whichever limit is reached first)
Incidents	25K incidents or 50 MB (which limit is reached first)

Migrating to a SaaS Console

Edit on GitHub

If you are interested in moving from Prisma Cloud Compute Edition (self-hosted) to Prisma Cloud Enterprise Edition (SaaS), contact Palo Alto Network Customer Support or your Customer Success Team to discuss the migration process in detail.

Points to consider:

- This is a one-time migration.
- The direction is Prisma Cloud Compute Edition to Prisma Cloud Enterprise Edition. You cannot migrate from Prisma Cloud Enterprise Edition to Prisma Cloud Compute Edition.
- If you have projects enabled with Prisma Cloud Compute Edition, you will need to break them apart and pick a single Console to migrate.
- Your Prisma Cloud Compute Edition Console version must exactly match the Prisma Cloud Enterprise Edition Console, which is always the latest version of Prisma Cloud Compute that is available.

Performance planning

Edit on GitHub

This section details the run-time characteristics of a typical Prisma Cloud deployment. The information provided is for planning and estimation purposes.

System performance depends on many factors outside of our control. For example, heavily loaded hosts have fewer available resources than hosts with balanced workloads.

Scale

Prisma Cloud has been tested and optimized to support up to 10,000 Defenders per Console.

Higher numbers of Defenders per Console can be supported, as long as the required resources are allocated to Console.

Storage

Using a network based storage is not recommended because it affects the database performance. if you choose to use a network based storage, such as NFS, make sure to review the Mongodb documentation for NFS storage requirements.

Scanning performance

This section describes the resources consumed by Prisma Cloud Defender during a scan. Measurements were taken on a test system with 1GB RAM, 8GB storage, and 1 CPU core.

Host scans

Host scans consume the following resources:

Resource	Measured consumption
Memory	10-15%
CPU	1%
Time to complete a host scan	1 second

Container scans

Container scans consume the following resources:

Resource	Measured consumption
Memory	10-15%
CPU	1%

Resource	Measured consumption
Time to complete a container scan	1-5 seconds per container

Image scans

When an image is first scanned, Prisma Cloud caches its contents so that subsequent scans run more quickly. The first image scan, when there is no cache, consumes the following resources:

Resource	Measured consumption
Memory	10-15%
CPU	2%
Time to complete an image scan.	1-10 seconds per image. (Images are estimated to be 400-800 MB in size.)

Scans of cached images consume the following resources:

Resource	Measured consumption
Memory	10-15%
CPU	2%
Time to complete an image scan	1-5 seconds per image. (Images are estimated to be 400-800 MB in size.)

Real-world system performance

Each release, Prisma Cloud tests performance in a scaled out environment that replicates a real-world workload and configuration. The test environment is built on Kubernetes clusters, and has the following properties:

- **Hosts:** 20,000
- **Hardware:**
  - **Console:** 16 vCPUs, 50 GB memory
  - **Defenders:** 2 vCPUs, 8 GB memory
- **Operating system:** Container-Optimized OS
- **Images:** 323
- **Containers:** 192,087 (density of 9.6 containers per host)

The results are collected over the course of 24 hours. The default vulnerability policy (alert on everything) and compliance policy (alert on critical and high issues) are left in place. CNNS is enabled.

Resource consumption:

The following table shows normal resource consumption.

Component	Memory (RAM)	CPU (single core)
Console	1,474 MiB	8.0%
Defender	82 MiB	1.0%

WAAS performance benchmark

Minimum requirements

Results detailed in this document assume a Defender instance complying with these minimum requirements.

Methodology

Benchmark target servers

Benchmark target servers were run on AWS EC2 instances running Ubuntu Server 18.04 LTS

Instance type	Environment	Compared servers	Versions
t2.large	Docker	Nginx vs WAAS-protected Nginx	Nginx/1.19.0
t2.large	Host	Nginx vs WAAS-protected Nginx	Nginx/1.14.0
t2.large	Kubernetes	Nginx vs WAAS-protected Nginx	Nginx/1.17.10

Benchmarking client

Benchmarking was performed using the hey load generating tool deployed on a ‘t2.large’ instance running Ubuntu Server 18.04 LTS

Benchmark scenarios

Test scenarios were run using hey against each server:

Scenario	HTTP Requests	Concurrent Connections
HTTP GET request	5,000	10, 100, 250, 500, 1,000
HTTP GET request with query parameters	5,000	10, 100, 250, 500, 1,000

Scenario	HTTP Requests	Concurrent Connections
HTTP GET request with an attack payload in a query parameter	5,000	10, 100, 250, 500, 1,000
HTTP GET with 1 MB response body	1,000	10, 100, 250, 500, 1,000
HTTP GET with 5 MB response body	1,000	10, 100, 250, 500, 1,000
HTTP POST request with body payload size of 100 bytes	5,000	10, 100, 250, 500, 1,000
HTTP POST request with body payload size of 1 KB	5,000	10, 100, 250, 500, 1,000
HTTP POST request with body payload size of 5 KB	5,000	10, 100, 250, 500, 1,000

In order to support 1,000 concurrent connections in large file scenarios, WAAS HTTP body inspection size limit needs to be set to 104,857 bytes

Results

HTTP transaction overhead

The following table details request average **overhead** (in milliseconds):

>Environment		>Concurrent Connections				
		>10	>100	>250	>500	>1,000
Docker	HTTP GET request	3	30	70	99	185
	HTTP GET request with query parameters	4	34	70	100	151
	GET w/ attack payload	1	6	6	26	96
	GET - 1MB Response	1	-268	-1314	-3211	-5152
	GET - 5MB Response	15	-1,641	-6,983	-9,262	-18,231
	POST w/ 100B body	5	42	84	119	194
	POST w/ 1KB body	12	106	245	430	800

	POST w/ 5KB body	42	402	970	1,853	3,189
Host	HTTP GET request	2	22	53	82	217
	HTTP GET request with query parameters	3	27	63	93	212
	GET w/ attack payload	0	6	17	78	104
	GET - 1MB Response	-1	-6	32	131	-681
	GET - 5MB Response	7	-45	-638	-2,677	-9,099
	POST w/ 100B body	3	29	66	114	300
	POST w/ 1KB body	10	97	234	436	774
	POST w/ 5KB body	39	407	940	1,831	3,196
Kubernetes	HTTP GET request	3	29	58	78	155
	HTTP GET request with query parameters	4	33	79	114	288
	GET w/ attack payload	0	5	15	63	177
	GET - 1MB Response	-4	-252	-981	-2827	-5754
	GET - 5MB Response	15	-1,653	-5,254	-14,966	-23,828
	POST w/ 100B body	5	39	92	130	280
	POST w/ 1KB body	11	109	252	498	907
	POST w/ 5KB body	43	421	1,013	2,005	3,557

Negative numbers indicate a performance improvement. WAAS response time can be faster than origin-server response time when attacks are blocked and not forwarded to the origin server.

Load testing

The following table details average request time (in milliseconds) of 1,000,000 request benchmarking load (includes response time for both WAAS and underlying origin):

>Environment	>Concurrent Connections

		>10	>100	>250	>500	>1,000
Docker	HTTP GET request	4	36	90	177	358
	HTTP POST request, 100 Byte body	5	47	116	232	472
Host	HTTP GET request	3	28	70	140	298
	HTTP POST request, 100 Byte body	4	40	99	197	397
Kubernetes	HTTP GET request	4	38	92	181	363
	HTTP POST request, 100 Byte body	5	49	119	236	460

Automated deployment

Edit on GitHub

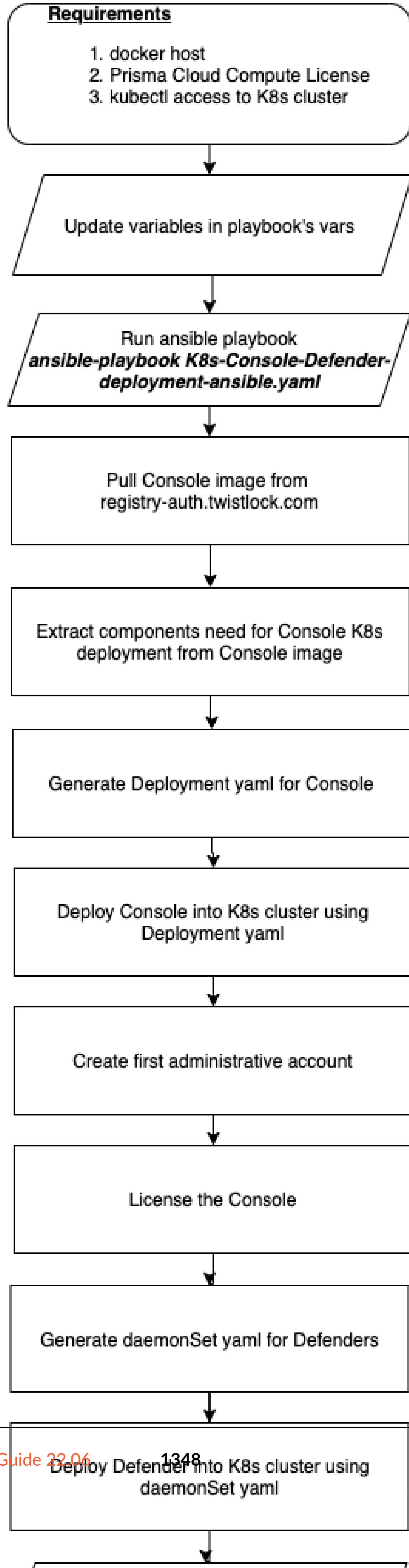
The following is an example of Infrastructure as Code (IaC) for the automated deployment of a Console and Defenders within a Kubernetes cluster using an Ansible playbook. This requires a docker host, Prisma Cloud Compute license and kubectl administrative access to the Kubernetes cluster. The Ansible playbook must run on a host that is able to route to the Console service's ClusterIP address to perform the required API calls to configure the Console. Use of this Ansible playbook does not imply any rights to Palo Alto Networks products and/or services.

Requirements

This sample IaC deployment runs on a unix based host with the following requirements:

- docker
- Ansible
- Prisma Cloud Compute license
- kubectl access to Kubernetes cluster with permissions to deploy Prisma Cloud Compute.
- Ability to pull images from registry-auth.twistlock.com
- K8s-Console-Defender-deployment-ansible.yaml Ansible playbook

Process


Requirements
1. docker host
2. Prisma Cloud Compute License
3. kubectl access to K8s cluster
Update variables in playbook's vars
Run ansible playbook
ansible-playbook K8s-Console-Defender-deployment-ansible.yaml
Pull Console image from registry-auth.twistlock.com
Extract components need for Console K8s deployment from Console image
Generate Deployment yaml for Console
Deploy Console into K8s cluster using Deployment yaml
Create first administrative account
License the Console
Generate daemonSet yaml for Defenders
Deploy Defender into K8s cluster using daemonSet yaml

Ansible playbook

Pull the Ansible playbook from here. Update the variables in the *vars:* section in *K8s-Console-Defender-deployment-ansible.yaml.*

- twistlock_registry_token: <license_token>
- twistlock_license: <license>
- twistlock_install_version: <version_to_deploy, e.g. "21_04_421">
- user: <first_admin_username>
- password: <first_admin_password>
- storage_class: <k8s_storage_class_for_dynamic_persistent_volume>
- namespace: <namespace>

Execution

On the unix host, sudo to root and run the command **ansible-playbook K8s-Console-Defender-deployment-ansible.yaml**

The supporting files will be written to the /root/twistlock directory.

Post execution

Once the playbook has successfully completed, establish communications to the twistlock-console service's management-port-https port (default 8083/TCP) using a Kubernetes LoadBalancer or your organization's approved cluster ingress technology.

High Availability and Disaster Recovery guidelines

Edit on GitHub

The following article describes the key guidelines for keeping your Prisma Cloud Compute deployment highly available, and creating a disaster recovery process.

Prisma Cloud Compute deployment consists of two components - Console and Defenders.

- Console is the management interface. It lets you define policy and monitor your environment.
- Defenders are spread across your environment and protect its workloads according to the policies set in the Console.

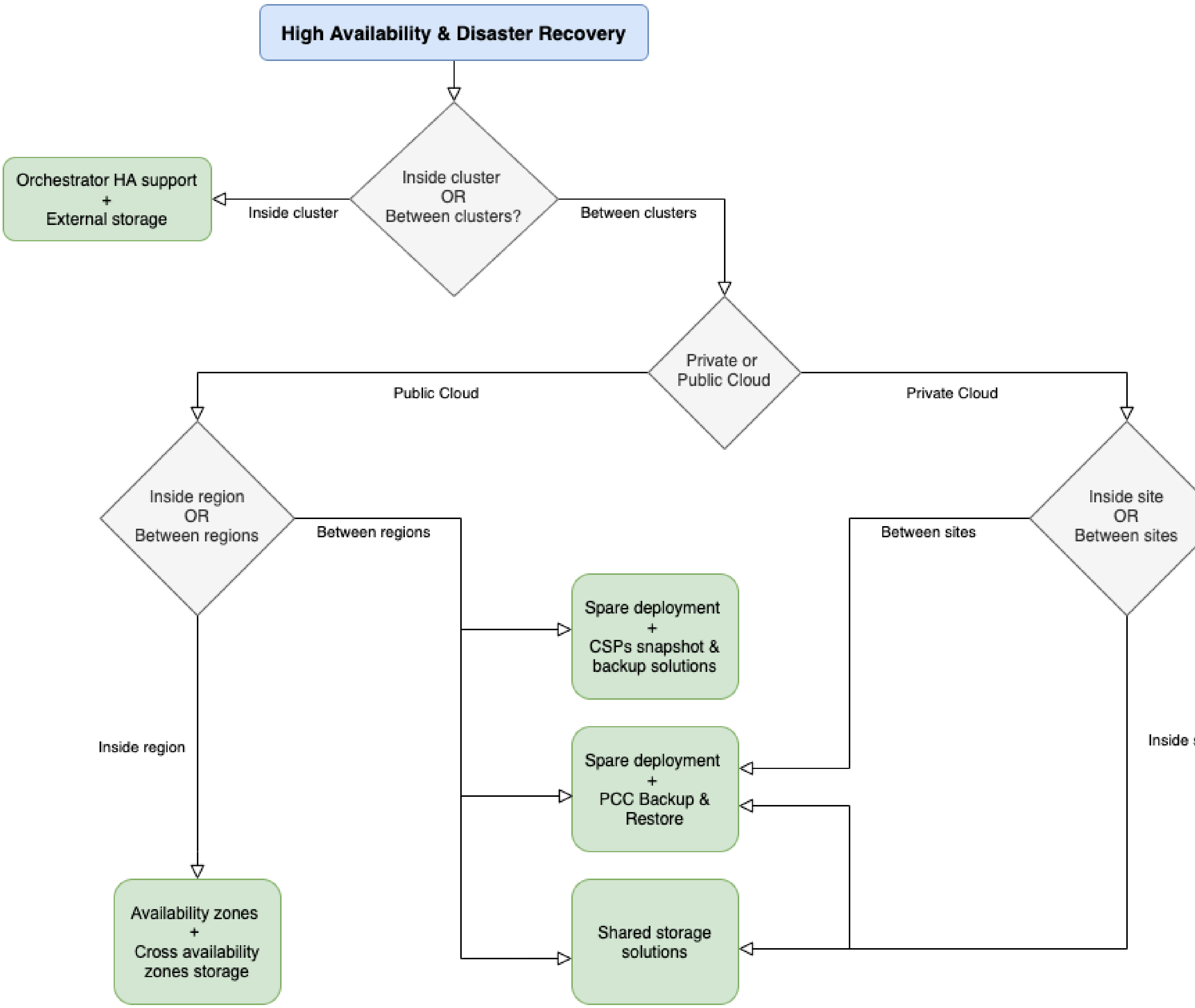
When the Console fails or stops working, your environment still has active runtime protection done by the Defenders. Each Defender holds the updated policies, and keeps protecting your workloads according to them.

This article mainly focuses on Prisma Cloud Compute Edition deployment (self-hosted Console). When leveraging Prisma Cloud Enterprise Edition (SaaS Console), high availability for the console is automatically provided by Palo Alto Networks.

Guidelines

Use the guidelines in this section to create high availability and disaster recovery processes for your deployment.

The following flowchart depicts the guidelines:



Inside each cluster

Whether your deployment is in the cloud or on-premises, orchestrators, such as Kubernetes, OpenShift, and AWS ECS, automatically support HA of the cluster and the containers running on it.

- **Console** — Set your storage to be external to the Console container/node. In case the Console container/node fails, the orchestrator brings Console back up, where it connects to the external storage to get its latest state.

- **Defenders** — Defenders are deployed as a DaemonSet. In case of a node failure, the orchestrator automatically brings up another node and deploys a Defender container on it, as a part of the DaemonSet definition.

Between clusters

While not explicitly tested or supported by Palo Alto Networks, in general, solutions that replicate storage between clusters to provide disaster recovery work transparently to Prisma Cloud Compute Edition. Note that ingress into the Console (DNS mapping and IP routing) may require additional steps during the activation of the secondary sites to ensure the Console is reachable over the network.

Public cloud

- **Inside each region** — CSPs provide high availability using availability zones inside each region. In case of an AZ failure, most cloud providers bring the cluster back up in another AZ.

  Use cross availability zones storage solutions, so when the cluster is up in another AZ, it connects to the shared storage and keeps functioning as before. For example, in AWS, EFS can be used as a shared storage between availability zones.
- **Between regions** — CSPs provide solutions such as snapshots and backups that can be moved between regions, shared storage between regions, etc. You can also use Compute's backup and restore capabilities for moving the data between regions.

Private cloud (on-premises)

- **Inside each site/data center** (between clusters on the same site)
  - Use shared storage between the clusters.
  - Create a disaster recovery process using Compute's backup and restore capabilities:
    - Create a spare cluster (warm or cold) with a Prisma Cloud Compute (PCC) deployment.
    - Backup PCC's data periodically to a location outside of the active cluster.
    - If the active cluster fails, bring the spare cluster up, and restore PCC's data to it.
- **Between sites/data centers**
  - Create a disaster recovery process for cases where one site goes down, using Compute's backup and restore capabilities:
    - Create a spare site (warm or cold) with a PCC deployment.
    - Backup PCC's data periodically to a location outside of the active site.
    - If the entire active site fails, bring the spare site up, and restore PCC's data from the external location to it.

Projects

Projects solve the problem of multi-tenancy. Each project consists of a Console and its Defenders. Each project is a separate, compartmentalized environment which operates independently with its own rules and configurations.

High availability and disaster recovery processes should be created for each tenant project, similar to the way you would handle a single Console deployment. If using Compute's backup and restore capabilities, backups should be created and restored separately for each project.

API

Edit on GitHub

All information for the CWPP API has now moved to prisma.pan.dev, our home for developer docs.

API reference:

https://prisma.pan.dev/api/cloud/cwpp

API-related documentation, including the porting guide:

https://prisma.pan.dev/docs/cloud/cwpp/cwpp-gs

Howto

Edit on GitHub

This section contains guides for deploying various advanced setups.

- Configure an AWS Classic Load Balancer for ECS
- Configure Prisma Cloud Console’s listening ports
- Provision tenant projects in OpenShift
- Disable automatic learning
- Debug data

Configure an AWS Classic Load Balancer for ECS

Edit on GitHub

Configure an AWS Classic Load Balancer for accessing Prisma Cloud Console. Console serves its UI and API over HTTPS on port 8083, and Defender communicates with Console over a websocket on port 8084. You'll set up a single load balancer to forward requests for both port 8083 and 8084 to Console, with the load balancer checking Console's health using the */api/v1/_ping* endpoint on port 8083.

For the complete install procedure for Prisma Cloud on Amazon ECS, see here.

STEP 1 | Log into the AWS Management Console.

STEP 2 | Go to **Services > Compute > EC2**.

STEP 3 | In the left menu, go to **Load Balancing > Load Balancers**.

STEP 4 | Create a load balancer.

1. Click **Create Load Balancer**.
2. In **Classic Load Balancer**, click **Create**.
3. Give your load balancer a name, such as **pc-ecs-lb**.
4. Leave default **VPC**.
5. Create the following listener configuration:
   - **Load Balancer Protocol**: TCP
   - **Load Balancer Port**: 8083
   - **Instance Protocol**: TCP
   - **Instance Port**: 8083
6. Click **Add** to add another listener using following listener configuration:
   - **Load Balancer Protocol**: TCP
   - **Load Balancer Port**: 8084
   - **Instance Protocol**: TCP
   - **Instance Port**: 8084
7. Click **Next: Assign Security Groups**.
   - Select the **pc-security-group**
8. Click **Next Configure Security Settings**.
   - Ignore the warning and click **Next: Configure Health Check**
9. Use the following health check configuration:
   - **Ping Protocol**: HTTPS
   - **Ping Port**: 8083
   - **Ping Path**: /api/v1/_ping
   - For **Advanced Details**, accept the default settings.
10. Click **Next: Add EC2 Instances**
    - Do not select any instances.
11. Click **Next: Add Tags**.
    - Under **Key**, enter **Name**.
    - Under **Value**, enter **pc-ecs-lb**.
12. Click **Review and Create**.
13. Review your settings and select **Create**.
14. Review the load balancer that was created and record its **DNS Name**.

Configure Prisma Cloud Console's listening ports

Edit on GitHub

This guide shows you how to configure Prisma Cloud to listen on different ports. Typically this type of configuration is made at the load balancer layer, but it can be done directly with Prisma Cloud.

By default Prisma Cloud listens on:

- *8083* HTTPS management port for access to Console.
- *8084* WSS port for Defender **to** Console communication.

If you are setting the port *below 1024* then Prisma Cloud needs permission to access this privileged port. You must also set *RUN_CONSOLE_AS_ROOT=${RUN_CONSOLE_AS_ROOT:-false}* to true.

STEP 1 | Download and unpack the Prisma Cloud software.

STEP 2 | Go to the directory where you unpacked the bits.

STEP 3 | Open *twistlock.cfg* for editing.

- *MANAGEMENT_PORT_HTTP* sets the HTTP access port, leaving this blank disables HTTP access.

  Example: *MANAGEMENT_PORT_HTTP=${MANAGEMENT_PORT_HTTP-80}* configures Console to listen on port *80*.
- *MANAGEMENT_PORT_HTTPS* sets the HTTPS access port.

  Example: *MANAGEMENT_PORT_HTTPS=443* configures Console to to listen on port *443*.
- *COMMUNICATION_PORT* sets the WSS port used for Defender to Console communication.

  Example: *COMMUNICATION_PORT=9090* configures Console to listen on port *9090*.

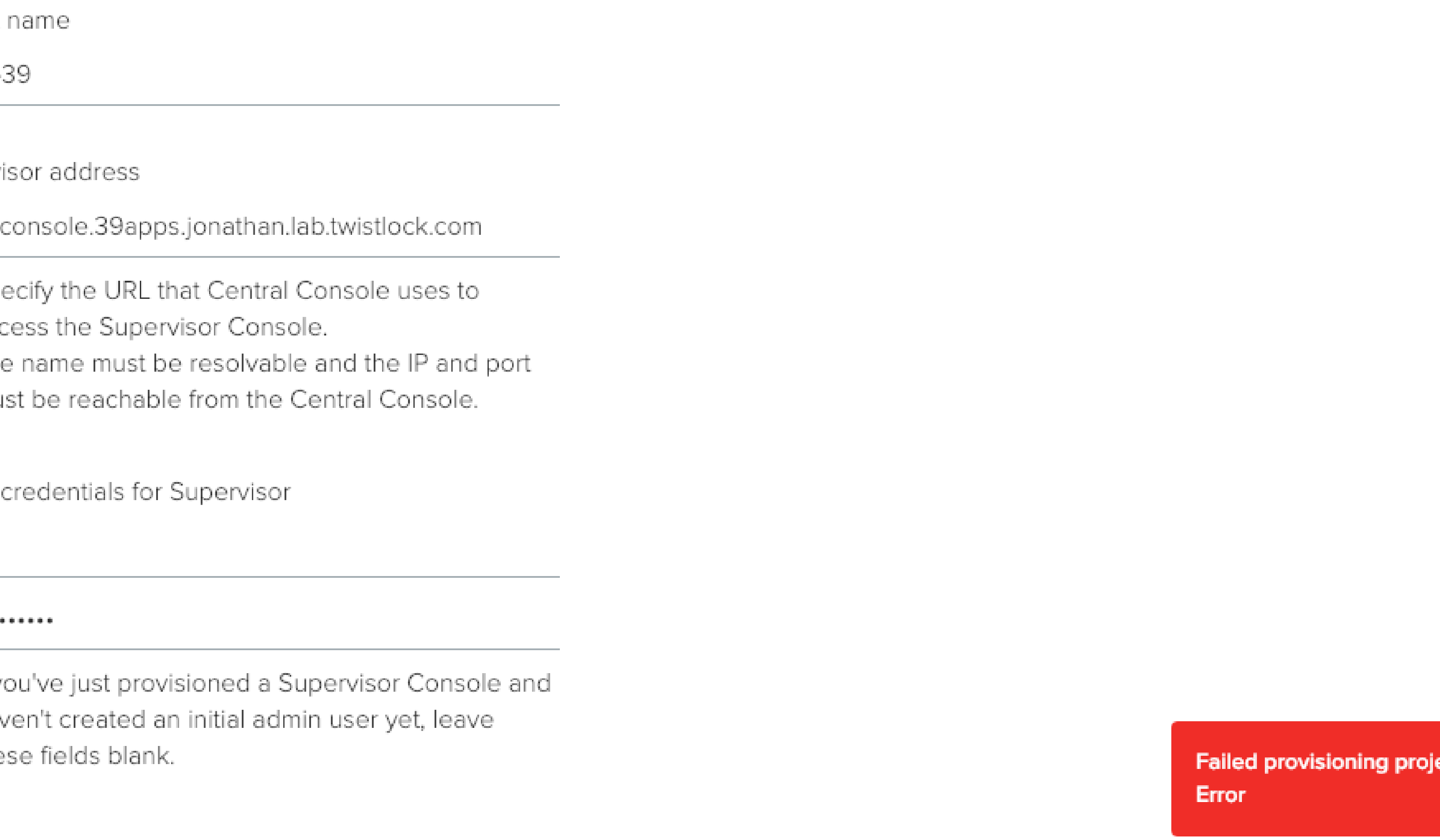
STEP 4 | Run *twistlock.sh* to install Prisma Cloud Console with your settings.

If you are setting the port *below 1024* then Prisma Cloud needs permission to access this privileged port. You must also set *RUN_CONSOLE_AS_ROOT=${RUN_CONSOLE_AS_ROOT:-false}* to true.

Provision tenant projects in OpenShift

Edit on GitHub

This guide shows you how to set up tenant projects on Openshift clusters. If you try to provision tenant projects using the normal provisioning flow, Central Console cannot reach the host where Supervisor Console runs. Failing to follow these steps can lead an 'Internal Server Error', even when everything seems to be set up properly.



In this example provisioning flow, the DNS names for Central Console and Supervisor Console are:

- Central Console — https://console.apps.jonathan.lab.twistlock.com
- Supervisor Console to be provisioned — https://console.39apps.jonathan.lab.twistlock.com

Prerequisites:

- Two fully operational Prisma Cloud Consoles are already deployed. For more information, see the OpenShift 4 deployment.
- OpenShift external routes to both Consoles' TCP port 8083 (Prisma Cloud UI and API), with the TLS termination type set to passthrough, already exist.

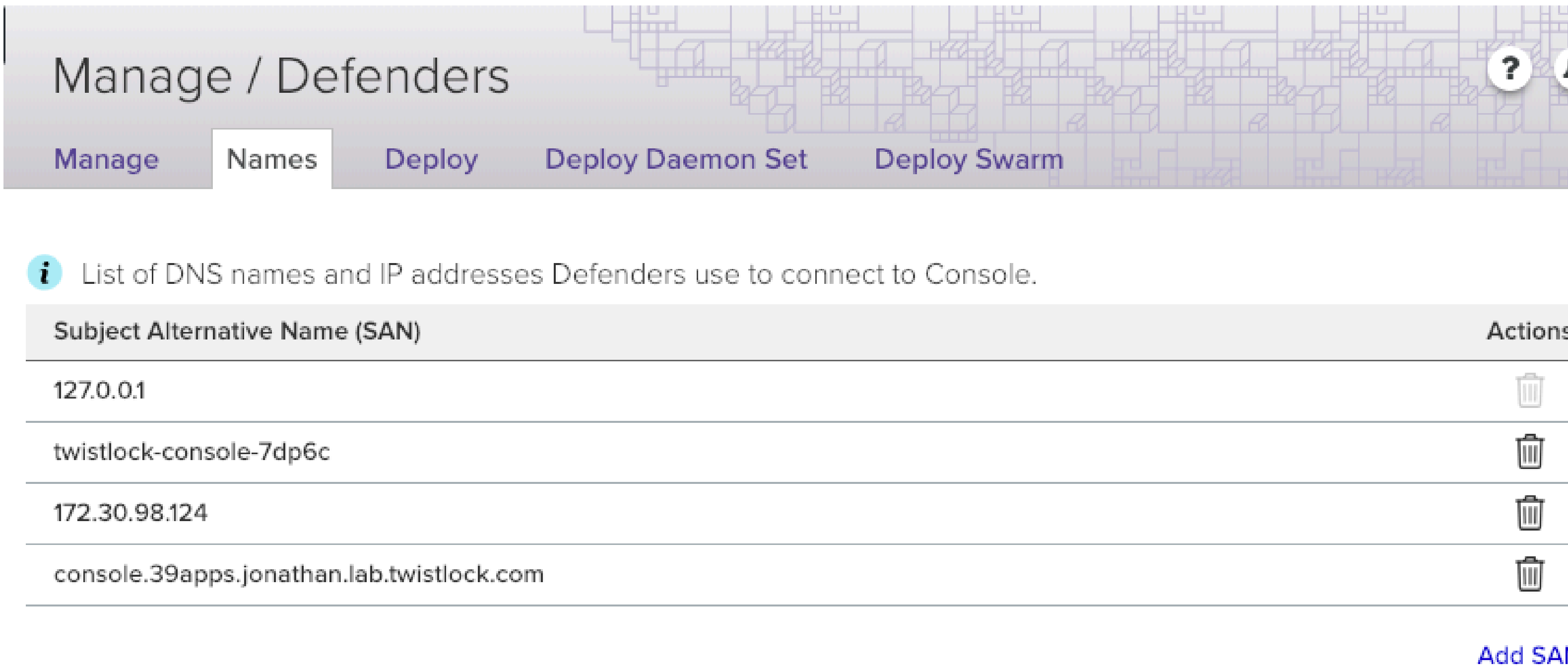
- The to-be Central and Supervisor Consoles are already licensed and you've created initial admin users.

STEP 1 | Designate one Console to be Supervisor and the other to be Central.

STEP 2 | Log into the Supervisor Console with your admin user.

STEP 3 | Add the FQDN of the Supervisor Console to the Subject Alternative Name field of the Supervisor Console's certificate.

1. In the Supervisor Console, go to **Manage > Defenders > Names**.
2. Click **Add SAN**.
3. Add the Supervisor Console's FQDN. In this example, it is **console.39apps.jonathan.lab.twistlock.com**.
4. Click **Add**.



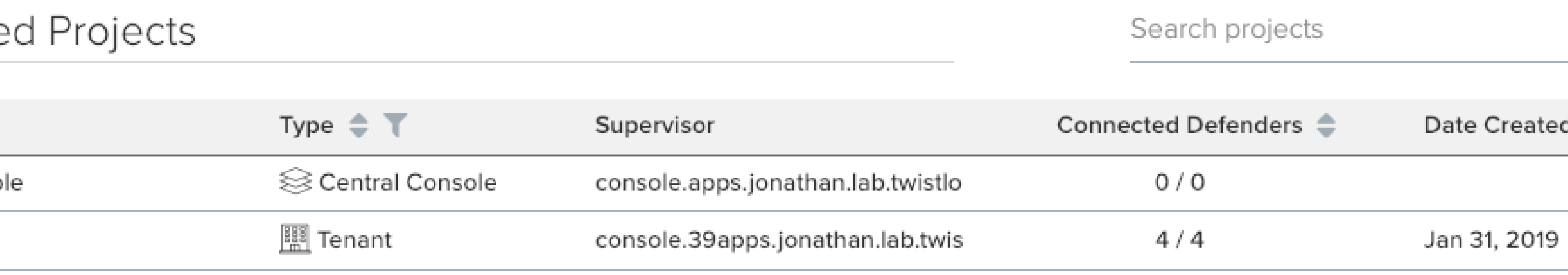
STEP 4 | Log into the Central Console with your admin user.

STEP 5 | Enable Projects by going to **Manage > Projects > Manage** and setting **Use Projects** to **On**.

STEP 6 | Click the **Provision** tab and to provision a tenant Console.

1. Under **Select Project type**, choose **Tenant**.
2. In **Project name**, give your project a name.
3. In **Supervisor address**, add the FQDN of the Supervisor. In this example, it is https://console.39apps.jonathan.lab.twistlock.com.
4. Add the **Admin credentials for Supervisor**.
5. Click **Provision**.

   Your Supervisor Console should be successfully provisioned.



Disable automatic learning

Edit on GitHub

Prisma Cloud lets you disable automatic learning to give you full control over creating, managing, and maintaining runtime rules for your apps.

Disabling automatic runtime learning is strongly discouraged. Prisma Cloud has been architected and optimized to automatically learn known good runtime behaviors, then create models that explicitly allow those behaviors. Disabling learning requires creating manual rules for all of these behaviors and greatly increases the likelihood of encountering false positive events.

If you have a regimented deployment process that must guarantee consistency between your test environment and your production environment, then you might want to disable automatic runtime learning, and manually create runtime rules instead. With this approach, the full range of runtime behaviors is locked down in production, and cannot be extended without manually adding new rules.

Models and learning

When a model is created for an entity, it's initially empty. Empty models don't allow any runtime behaviors. In a default installation, Prisma Cloud uses machine learning to compose models that encapsulate all known good behaviors. Models are sets of rules that allow process, network, and file system activity.

When learning is disabled, newly created models are empty. Since empty models don't allow any behaviors, you must manually create rules that explicitly allow process, network, and file system activity. Remember that rules come from two places: models (automatically created) and runtime rules (manually created). Manually created rules are designed to augment models when learning does not capture the full range of known good behaviors. When automatic learning is disabled, they must fully specify the full range of known good behaviors.

Deploying Prisma Cloud

Models created before automatic learning is disabled might still contain learned content. To guarantee all models are empty, disable automatic learning before deploying Defenders to your environment.

1. Disable automatic learning.
    1. On the Prisma Cloud Console, select **Defend > Runtime > Containers**.
    2. Set the toggle off for **Enable automatic runtime learning**.
2. Deploy Defenders.

Workflow

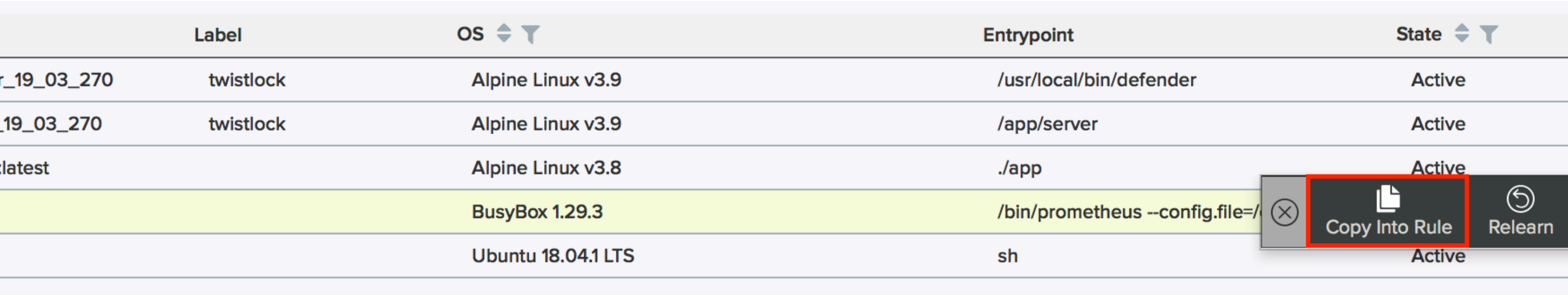
You should have two environments: test and production. Deploy Prisma Cloud Console to each environment. In the test environment, enable automatic learning. You'll use automatic learning to assist with the creation of rules. In the production environment, disable automatic learning. You'll port the rules from the test environment to the production environment.

The recommended workflow is:

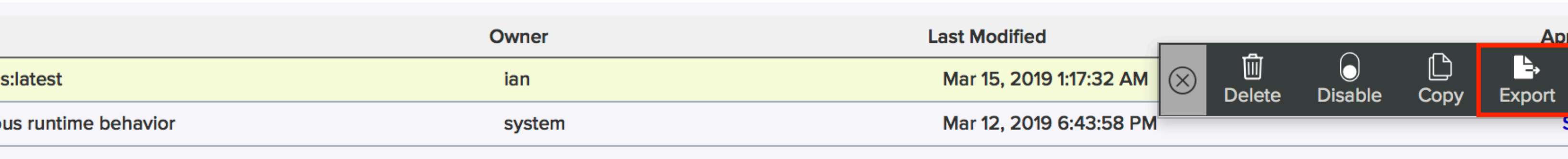
1. Deploy your app to the test environment, and fully exercise it.
2. Validate models that were automatically created.
3. Export models from the test environment as rules.
4. Optionally store the rules in a source control system.
5. Import the rules into your production environment, where automatic learning is disabled.

Exporting and importing rules from the Console UI

After your app has been fully exercised in the test environment, create a rule from the runtime model. In **Monitor > Runtime > Container Models**, find your model, click **Actions**, then click **Copy Into Rule**.



Next, download the rule in JSON format. Go to **Defend > Runtime > Container Policy**, find your rule, and in the **Actions** menu, click **Export**.



Finally, import your rule into Console in your production environment. Go to **Defend > Runtime > Container Policy**, and click **Import rule**.

Exporting and importing rules programmatically

After your app has been fully exercised in the test environment, retrieve the model as a runtime rule. Use the *GET /profiles/container/{id}/rule* endpoint, where *{id}* is the profile ID.

A list of profiles (models) can be retrieved from GET /api/v1/profiles/container. Profile IDs can be found in the _id field. Profile ID is simply the concatenation of the image ID and an underscore.

```
$ curl -k \
  -u ian \
  -H 'Content-Type: application/json' \
  -X GET \
  https://<TEST-CONSOLE>:8083/api/v1/profiles/container/{id}/rule \
  | jq '.' > model_rules.json
```

Then push the rule to Console in your production environment. When a rule is pushed with this endpoint, it is ordered first in the policy. Rule order is important, so be sure you're pushing rules in the right order. The version of Console where the rule was exported must match the version of Console where it's imported.

```
$ curl -k \
  -u <USER> \
  -X POST \
  -H "Content-Type:application/json" \
  https://<PROD-CONSOLE>:8083/api/v1/policies/runtime/container \
  --data-binary "@model_rules.json"
```

The POST /api/v1/policies/runtime/container endpoint pushes one rule at a time. The PUT /api/v1/policies/runtime/container endpoint pushes the entire policy (i.e. all rules) in a single shot.

Debug data

Edit on GitHub

Console and Defender generate logs as they run. These logs, also known as debug data, are designed to help troubleshoot operational issues. They're different from audits, which are designed to report significant security events.

If you contact Prisma Cloud Support with an issue, you'll be asked to collect debug data from your setup and send it to us. Debug data helps us find the root cause of problems, and provide timely resolutions.

Collect Console debug logs

The simplest way to view Console's debug logs is from within the UI itself. Go to **Manage > Logs > Console**.

Collect Defender debug logs

To view Defender's debug logs, go to **Manage > Defenders > Manage > Defenders**. Select the Defender from the table and then click **Actions > Logs**.

