Container Runtime Security

User Guide

August 29, 2022

Qualys, Inc.
919 E Hillsdale Blvd
4th Floor
Foster City, CA 94404
1 (650) 801 6100

Table of Contents

About this Guide

Welcome to Qualys Container Runtime Security (CRS). CRS provides runtime behavior visibility & enforcement capabilities for running containers. We'll help you get started.

About Qualys

Qualys, Inc. (NASDAQ: QLYS) is a pioneer and leading provider of cloud-based security and compliance solutions. The Qualys Cloud Platform and its integrated apps help businesses simplify security operations and lower the cost of compliance by delivering critical security intelligence on demand and automating the full spectrum of auditing, compliance and protection for IT systems and web applications.

Founded in 1999, Qualys has established strategic partnerships with leading managed service providers and consulting organizations including Accenture, BT, Cognizant Technology Solutions, Deutsche Telekom, Fujitsu, HCL, HP Enterprise, IBM, Infosys, NTT, Optiv, SecureWorks, Tata Communications, Verizon and Wipro. The company is also founding member of the Cloud Security Alliance (CSA). For more information, please visit www.qualys.com

Qualys Support

Qualys is committed to providing you with the most thorough support. Through online documentation, telephone help, and direct email support, Qualys ensures that your questions will be answered in the fastest time possible. We support you 7 days a week, 24 hours a day. Access online support information at www.qualys.com/support/.

About Container Runtime Security

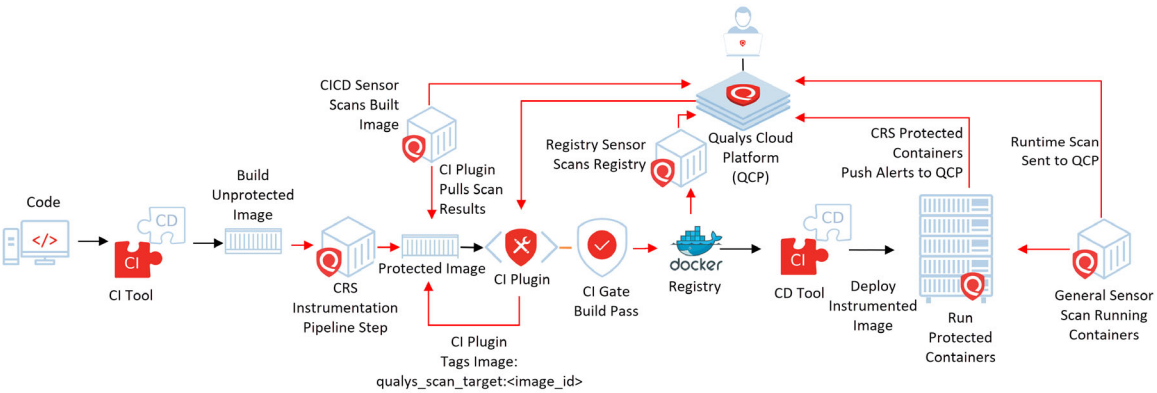
Container Runtime Security (CRS) provides runtime behavior visibility & enforcement capabilities for running containers. This allows customers to address various use cases for running containers around security best practice enforcement, file access monitoring, network access control.

CRS requires instrumentation of container images with the Qualys Container Runtime Instrumentation, which injects probes into the container image. Customers can configure instrumented images, containers with granular policies which govern container behavior, visibility. Based on these runtime enforcement policies - runtime events, telemetry can be viewed obtained from the backend via UI, API.

CRS is currently supported for Linux OS based containers only.

CRS Architecture

The diagram below provides a recommended container security workflow leveraging Qualys Container Security (Scanning + Container Runtime Security).



The workflow for Container Runtime Security starts with instrumentation of the target container image. Qualys provides a customer premise Instrumenter that can be leveraged in a customer environment to instrument application containers with Qualys' security probes. It can be run locally in CLI mode or it can be provisioned as an always running microservice backend.

Our instrumentation approach layers in an enhanced version of the glibc linux library which provides container behavior visibility and enforcement. Application containers spun up from instrumented application container images register with the Qualys Cloud Platform and obtain runtime policies. These runtime policies and the Qualys instrumentation autonomously drive container behavior visibility & enforcement.

CRS Instrumentation

Protecting containers with Qualys CRS requires instrumentation of a container image with the Qualys Instrumentation. You have 2 options for instrumenting container images - instrument images on your local host using CLI mode, or run our Instrumenter service in the backend to instrument images that have been scanned by a registry scan job.

Instrumentation using CLI mode - This approach is used for instrumenting individual images on your local host. You'll run the instrumenter.sh script with CLI mode enabled (CLI mode is enabled by default) and identify the image to instrument. The image must be present locally where you're running the CLI command. You can optionally specify the runtime policy to apply to the instrumented image. When you instrument an image using this method, we'll immediately add in our solution and create the instrumented image (appended with -layered) at the same location. One command will instrument one image only, and then it will exit as soon as instrumentation is done.

Instrumentation using the Instrumenter service - This approach is used for instrumenting images that have been scanned by a registry scan job (registry sensor). The Instrumenter service is a lightweight microservice that runs in the customer premise. The Instrumenter service is packaged and distributed to customers as a container image. This instrumenter container is meant to be run on a container host. It requires connectivity back to the Qualys backend. The backend federates instrumentation requests to this microservice. Once an image is submitted for instrumentation (via UI, API), the instrumenter inspects the image, injects the Qualys instrumentation, and provides as output a new "instrumented" version of the image. This new image is then uploaded back to the destination container registry with "-layered" appended to the tag. This workflow is coupled tightly with a registry.

Requirements

The Instrumenter service requires the following:

1) Docker engine/server and a DOCKER_HOST socket connection

2) Docker V2 registry:

Public registries: Docker Hub

Private registries: v2-private registry: JFrog Artifactory (secure: auth + https)

Compatibility

- The Instrumenter service is able to request Qualys Container Security user credentials from Vault secret engine types: kv-v1 and kv-v2. Although supported, it is not recommended to pass credentials in plain text, unencrypted to the Instrumenter service. More details further in this document.

- The Instrumenter container requires a Docker engine greater than 1.12.

Limitations

Please note the following limitations:

- Only certain container images are supported for instrumentation (see details below)

- Multiple Instrumenters per subscription are supported. Currently there is no visibility of Instrumenters via the UI or API.

- One Instrumenter service per docker engine/server host is supported

- Instrumentation jobs are delivered to any authenticated Instrumenter when using the Instrumenter service to instrument images

Images supported for instrumentation

Instrumentation is supported for container images with certain glibc versions. The table below shows the top images supported per operating system.

Want to know if your image is supported? Use the following script to check: https://github.com/Qualys/qualys_crs_instrumenter/blob/master/check_if_image_instrumentable.sh

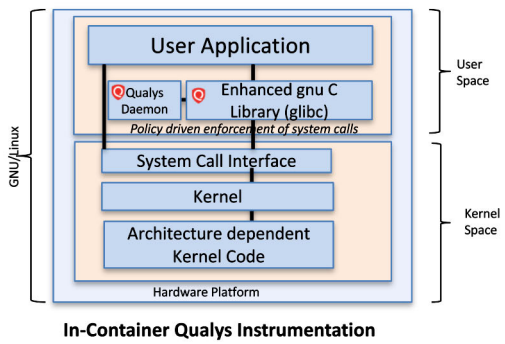
OS version	libc/glibc version	Docker Name: Tag	Docker Image SHA (Repo Digest)
Alpine			
3.16.1	musl-1.2.3-r0	alpine:3.16.1	alpine@sha256:7580ece7963bfa863801466c0a488f11c86f85d9988051a9f9c68cb27f6b7872
3.13.5	musl-1.2.2-r0	alpine:3.13.5	alpine@sha256:1d30d1ba3cb90962067e9b29491fbd56997979d54376f23f01448b5c5cd8b462
3.12.7	musl-1.1.24-r10	alpine:3.12.7	alpine@sha256:de25c7fc6c4f3a27c7f0c2dff454e4671823a34d88abd533f210848d527e0fbb
3.12.1	musl-1.1.24-r9.apk	alpine:3.12.1	alpine@sha256:c0e9560cda118f9ec63ddefb4a173a2b2a0347082d7dff7dc14272e7841a5b5a
3.11	musl-1.1.24-r3	alpine:3.11	alpine@sha256:6cf3d8abc08cf3792d590152d7a4628ec827621f55b1d3150383f5f39335d6eb
3.10.5	musl-1.1.22-r3	alpine:3.10	alpine@sha256:f0e9534a598e501320957059cb2a23774b4d4072e37c7b2cf7e95b241f019e35
3.9.4	musl-1.1.20-r4	alpine:3.9.4	alpine@sha256:7746df395af22f04212cd25a92c1d6dbc5a06a0ca9579a229ef43008d4d1302a
3.9.2	musl-1.1.20-r3	alpine:3.9.2	alpine@sha256:644fcb1a676b5165371437feaa922943aaf7afcfa8bfee4472f6860aad1ef2a0
3.9	musl-1.1.20-r5	alpine:3.9	alpine@sha256:414e0518bb9228d35e4cd5165567fb91d26c6a214e9c95899e1e056fcd349011

OS version	libc/glibc version	Docker Name: Tag	Docker Image SHA (Repo Digest)
Amazon Linux			
2	glibc-common-2.26-33.amzn2.0.2.x86_64	amazonlinux:2.0.20191217.0	amazonlinux@sha256:58d05c596a29f2cfb81543dddd01ca5613bc33e2a65a5567dc875d50e7225f9c
2	glibc-common-2.26-33.amzn2.0.1.x86_64	amazonlinux:2.0.20191016.0	amazonlinux@sha256:5aa0460abffafc6a76590f0070e1b243a93b7bbe7c8035f98c1dee2f9b46f44c
Centos			
8	glibc-2.28-151.el8.x86_64	centos:centos8	centos@sha256:a27fd8080b517143cbbbab9dfb7c8571c40d67d534bbdee55bd6c473f432b177
8	glibc-2.28-72.el8.x86_64	centos:8.1.1911	centos@sha256:fe8d824220415eed5477b63addf40fb06c3b049404242b31982106ac204f6700
7	glibc-2.17-323.el7_9.x86_64	centos:7	centos@sha256:0f4ec88e21daf75124b8a9e5ca03c37a5e937e0e108a255d890492430789b60e
7	glibc-2.17-307.el7.1.x86_64	centos:centos7	centos@sha256:19a79828ca2e505eaee0ff38c2f3fd9901f4826737295157cc5212b7a372cd2b
7	glibc-2.17-292.el7.x86_64	centos:7.7.1908	centos@sha256:50752af5182c6cd5518e3e91d48f7ff0cba93d5d760a67ac140e2d63c4dd9efc
Debian			
10	2.28-10	debian:10	debian@sha256:e2fe52e17d649812bddcac07faf16f33542129a59b2c1c59b39a436754b7f146
9 (stretch)	glibc_2.24-11+deb9u4	debian:9.13	debian@sha256:26d14aa81aa59de744d6ec9509000341f3f8e0160d78f3659f1d25a2b252d28e
9 (stretch)	glibc_2.24-11+deb9u3	debian:9.4	debian@sha256:6ee341d1cf3da8e6ea059f8bc3af9940613c4287205cd71d7c6f9e1718fdcb9b
9 (stretch)	glibc_2.24-11+deb9u1	debian:9.1	debian@sha256:5fafd38cdee6c7e6b97356092b97389faa0aa069595f1c3cc3344428b5fd2339
Ubuntu			
bionic-20201119	2.27-3ubuntu1.3	ubuntu:bionic-20201119	ubuntu@sha256:fd25e706f3dea2a5ff705dbc3353cf37f08307798f3e360a13e9385840f73fb3
bionic-20200807	2.27-3ubuntu1.2	ubuntu:bionic-20200807	ubuntu@sha256:05a58ded9a2c792598e8f4aa8ffe300318eac6f294bf4f49a7abae7544918592

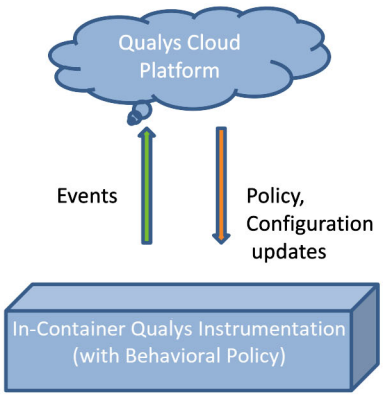
OS version	libc/glibc version	Docker Name: Tag	Docker Image SHA (Repo Digest)
18.04	glibcVersion:libc6_2.27-3ubuntu1.4	ubuntu:18.04	ubuntu@sha256:7bd7a9ca99f868bf69c4b6212f64f2af8e243f97ba13abb3e641e03a7ceb59e8
Google Distroless Images			
gcr.io/distroless/java:11	N/A	gcr.io/distroless/java:11	gcr.io/distroless/java@sha256:97c7eae86c65819664fcb7f36e8dee54bbbbc09c2cb6b448cbee06e1b42df81b
gcr.io/distroless/java:8	N/A	gcr.io/distroless/java:8	gcr.io/distroless/java@sha256:34c3598d83f0dba27820323044ebe79e63ad4f137b405676da75a3905a408adf
gcr.io/distroless/java:8-debug	N/A	gcr.io/distroless/java:8-debug	gcr.io/distroless/java@sha256:9662489f8d67e17ad371537a7b76c70c2e54ba64681b174003692e3a0200e9a5

In-Container Instrumentation

The Qualys instrumentation consists of glibc based hooks to intercept system calls being made. CRS policies, configurations for in-container instrumentation are obtained from the Qualys Cloud backend. The CRS policies are translated into syscall firewall rules and the in-container instrumentation provides visibility into and enforces container behavior. CRS policy events and CRS telemetry is regularly sent back to the Qualys backend where it can be viewed by API, UI.



In-Container Qualys Instrumentation



Qualys Backend

The Qualys backend manages the end-to-end workflow of CRS. From instrumenting images to managing the policy workflow to viewing CRS telemetry and policy hits.

CRS Deployment Workflow

Here's a look at the deployment workflow for Container Runtime Security.

Step 1: Instrument container images with Qualys instrumentation

You have 2 options for instrumenting images - you can instrument any image on your local host using CLI mode (see 1a), or you can run our Instrumenter service in the backend to instrument images that have been scanned by a registry scan job (see 1b). Choose the approach you want to take and follow the steps.

1a) Instrument image using CLI mode

Instrument an image on your local host. We'll immediately add in our runtime security solution and create the instrumented image (appended with -layered) at the same location. One command will instrument one image only, and then it will exit as soon as the instrumentation is done. Tip - If you have a runtime policy ready to go, you can immediately apply the policy to the instrumented image when running the CLI command.

Instrument images using CLI mode

1b) Instrument image using the Instrumenter service

To use the Instrumenter service, you'll need to complete the following steps:

Build image, Push image to registry, and Scan with registry sensor

You'll build the image and push it to the registry. Then scan each image you want to instrument with the registry sensor. This is required for using the Instrumenter service.

Deploy the Instrumenter service in your environment

The Instrumenter service will be used to pull down the unprotected image, package our solution into it, and then push it back to the registry as a protected image.

Deploy the Instrumenter Service

Instrument container image from the UI

When using the Instrumenter service, you'll kick off instrumentation from the UI. Identify the image you want to instrument on the Images list, and choose the Instrument option. The UI sends an instrumentation job to the deployed Instrumenter. We'll package in our solution, and push the protected image back to the registry. Once you have the protected image, you can run the image in your runtime environment as a running container.

Instrument images from the UI

Step 2: Configure policies and instrumentation

Create policies, and assign a policy to an instrumented image. You'll also want to set the policy enforcement level (determines whether policy rules are enforced) and select the log mode (determines which policy hits get logged).

Configure and Apply Policies

Set policy enforcement

Apply policy to instrumented image

Configure Instrumentation

Step 3: Run container from instrumented image

When ready, you can spawn containers from the instrumented image. The policy applied to the instrumented image gets enforced on the container and activities are logged as per the selected log mode.

Run containers from instrumented image

Step 4: View your events

Runtime events will be listed on the Events tab. Here you can search events and drill-down into event details.

View Your Events

View event details on dashboard

Deploy the Instrumenter Service

You can run the Instrumenter service using any of these options:

Option 1: Run instrumenter using docker CLI based command

This option lets you run the instrumenter in CLI mode (the default) for instrumenting images locally or in Daemon mode to use the instrumenter microservice to instrument images from the registry. You can run the instrumenter with or without a vault.

Prerequisites

- Request access to the Docker Hub private repo for qualys/crs-cli-instrumenter. To request access, reach out to Qualys Support from qualys.com/support and be sure to include your Docker Hub ID in your message.

- Run docker login with the provided Docker Hub ID on the instance where you will run instrumenter.sh in CLI mode.

Using CLI mode

1) Pull the docker CLI files from github. You can download them from https://github.com/Qualys/qualys_crs_instrumenter

2) Edit **instrumenter.sh** to configure specific details for proxy and vault usage. See File parameters for guidance on inputs.

3) Run the docker CLI script.

By default, the script will run in CLI mode and for this mode you must specify the endpoint and image. Policy ID is optional. Use this command to run the script:

```
sh instrumenter.sh --endpoint
<qualys_username>:<qualys_password>@<api_gateway_url>/crs/v1.2
--image <image> [--policyid <policy id>]
```

To use the instrumenter microservice to instrument images from the registry, you must run the script in Daemon mode. Specify --daemon-mode and specify the endpoint. In this case, you do not specify the image or policy. Use this command to run the script:

```
sh instrumenter.sh --endpoint
<qualys_username>:<qualys_password>@<api_gateway_url>/crs/v1.2
--daemon-mode
```

Usage Examples

Default Example - CLI mode:

```
./instrumenter.sh --endpoint <endpoint> --image <image> [--
policyid <policy id>]
```

Default Example - Daemon mode:

```
./instrumenter.sh --endpoint <endpoint> --daemon-mode
```

Vault Example - CLI mode:

```
./instrumenter.sh --endpoint <endpoint> --vault-token <token>
--vault-engine <engine version> [--vault-base64] --vault-path
<vault-path> --vault-address <vault-address> --image <image> [--
policyid <policy id>]
```

Vault Example - Daemon mode:

```
./instrumenter.sh --endpoint <endpoint> --vault-token <token>
--vault-engine <engine version> [--vault-base64] --vault-path
<vault-path> --vault-address <vault-address> --daemon-mode
```

Proxy Example - CLI mode:

```
./instrumenter.sh --endpoint <endpoint> --proxy <proxy> --image
<image> [--policyid <policy id>]
```

Proxy Example - Daemon mode:

```
./instrumenter.sh --endpoint <endpoint> --proxy <proxy> --daemon-
mode
```

Where:

<endpoint> is in the format of username:password@url if you are not using a vault. Only url is needed when you are using a vault.

<image> is the image Id (e.g. “6d9ae1a5c970”) or repository name:tag (e.g. “library/centos:centos72” or “java:latest”) for the container image you want to instrument using CLI mode. The image must be present locally where you’re running the CLI command.

<policy id> is the policy Id (e.g. “5fd20b4321dabf0001fdc464”) for the policy you want to immediately apply to the image being instrumented using CLI mode.

Option 2: Run docker compose file

This option is for using the instrumenter microservice to instrument images from the registry. Passing QUALYS_GATEWAY_ENDPOINT is required.

```
QUALYS_GATEWAY_ENDPOINT="<qualys_username>:<qualys_password>@<api_
gateway_url>/crs/v1.2" docker-compose up
```

Note: Use this command at the directory level where the docker compose file is present.

Please edit the fields in the docker compose file and remove # to uncomment and declare the constant you would like to use. See File parameters for guidance.

```
LI_MQURL: qas://${QUALYS_GATEWAY_ENDPOINT} # set the username
password and qualys endpoint for instrumenter in env or directly to
this file

# VAULT CONFIG (Change these settings if you have your own vault)
# LI_VAULT_SECRET_ENGINE: "kv-v2"
# LI_VAULT_DATA_VALUES_BASE64: "false"
# LI_VAULTPATH: "${USER_VAULT_PATH}"
# LI_VAULT_TOKEN: "${VAULT_TOKEN}"
# LI_VAULT_ADDRESS: "http://vault:8200"

# PROXY SETTINGS (Uncomment and fill required values for proxy)
# LI_ALLOWHTTPPROXY: true
# https_proxy: http://squid:3128
# LI_MQSKIPVERIFYTLS: true
```

Option 3: Run kubernetes instrumenter.yml

This option is for using the instrumenter microservice to instrument images from the registry.

Edit the required field QUALYS_GATEWAY_ENDPOINT in the kubernetes file. Replace QUALYS_GATEWAY_ENDPOINT with the following:

```
<qualys_username>:<qualys_password>@<api_gateway_url>/crs/v1.2
```

Edit the vault and proxy fields, as required. See File parameters for guidance.

```
- name: LI_MQURL
value: qas://{{QUALYS_GATEWAY_ENDPOINT}} # Enter the username
password of crs and qualys instrumenter pod endpoint

# VAULT CONFIG Change these settings if you have your own vault
# - name: LI_VAULTPATH
#   value: /secret/data/qgsuser # Enter path where the vault
credentials reside
```

```
# - name: LI_VAULT_ADDRESS
#   value: http://vault:8200 # Change if you have your own vault
# - name: LI_VAULT_DATA_VALUES_BASE64
#   value: "false" # Change if you store base64 version of
credentials in vault
# - name: LI_VAULT_SECRET_ENGINE
#   value: kv-v2 # Set the version of vault engine you use
# - name: LI_VAULT_TOKEN
#   value: {{VAULT_TOKEN}} # Set the vault token that you use


# proxy settings (Uncomment this if you have a proxy in your docker
host)
# - name: LI_ALLOWHTTPPROXY
#   value: true
# - name: https_proxy
#   value: http://proxy:3128
# - name: LI_MQSKIPVERIFYTLS
#   value: true
```

Then launch instrumenter using the following command:

```
kubectl apply -f instrumenter.yml
```

File parameters

Regardless of the option you picked for deploying the Instrumenter service, there are certain user/platform specific parameters you'll need to provide. See the table below.

General	Description
Username	Your Qualys username.
Password	Your Qualys password.
API Gateway URL	The Qualys API Gateway URL where your Qualys account resides. To identify your Qualys platform and get the API URL, visit: https://www.qualys.com/platform-identification/
Docker URL	The default docker URL is: tcp://qualys-docker-proxy.dockersock.jail:2375
Endpoint	The endpoint should be formatted as: <qualys_username>:<qualys_password>@<api_gateway_url>/crs/v1.2 Example: qualys_joe:abc12345@gateway.qg1.apps.qualys.com/crs/v1.2

Proxy	
Is Proxy/Allow Proxy	Set to “true” to define proxy settings if you have a proxy in your docker host.
Proxy	Enter the proxy address. Sample: http://squid:3128
Skip TLS	Set to “true” to skip TLS verification.
Vault	
Engine	Enter the version of vault engine. Sample: kv-v2.
Base64	Set to “false” by default. Change to “true” if you store base64 version of credentials in the vault.
Path	Enter the path where the vault credentials reside. Sample: /secret/data/qgsuser
Token	Enter the vault token that you use.
Address	Enter the vault address. Sample: http://vault:8200

After the Instrumenter service has been deployed

Check the instrumenter logs to verify the instrumenter is online and functional.

```
docker logs instrumenter | grep "Awaiting InstrumentRequests"
```

The output should print something similar to:

```
"[2020-05-26T21:37:52Z] DEBUG instrumenter: Awaiting
InstrumentRequests"
```

Troubleshooting the Instrumenter service

Credentials issues when deploying without a vault service

If you are not using a vault service, your Qualys credentials are being passed in plain text in a URL. If you are using special characters in your password (recommended), you will need to encode the special characters using HTML encoding.

HTML encoding site for reference: https://www.w3schools.com/tags/ref_urlencode.ASP

Logging

To view logs for the CRS instrumenter, run “docker logs instrumenter”

To view logs for the Docker socket proxy, run “docker logs proxy”

Instrument Container Images with Qualys Instrumentation

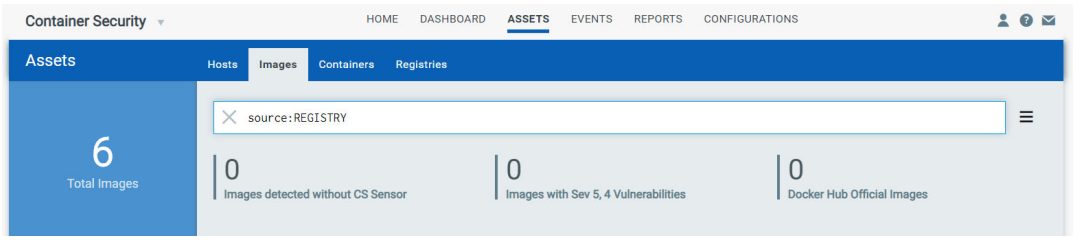
You have two options for instrumenting images:

Instrument images from the UI

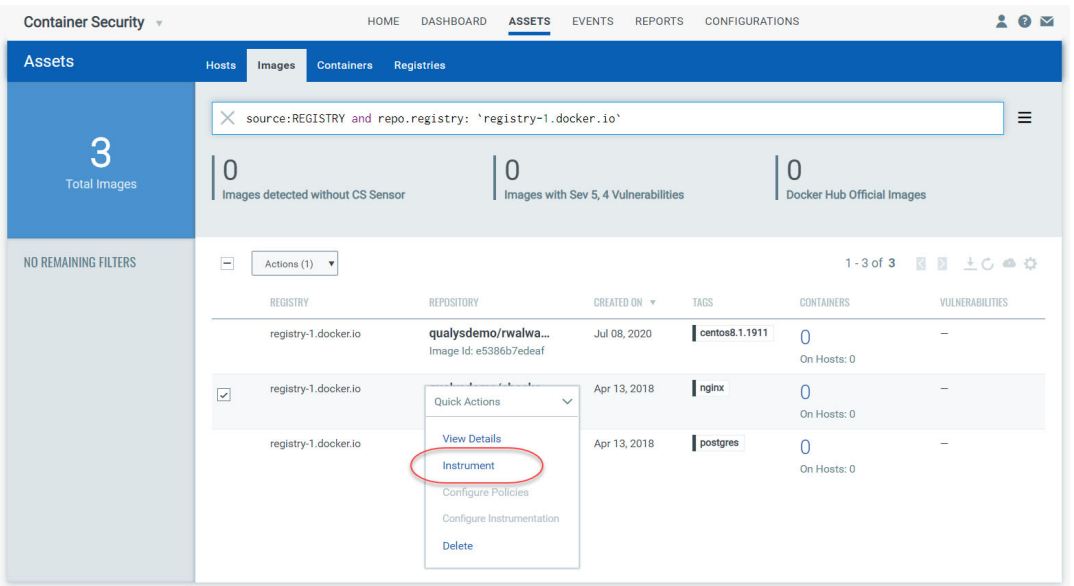
Instrument images using CLI mode

Instrument images from the UI

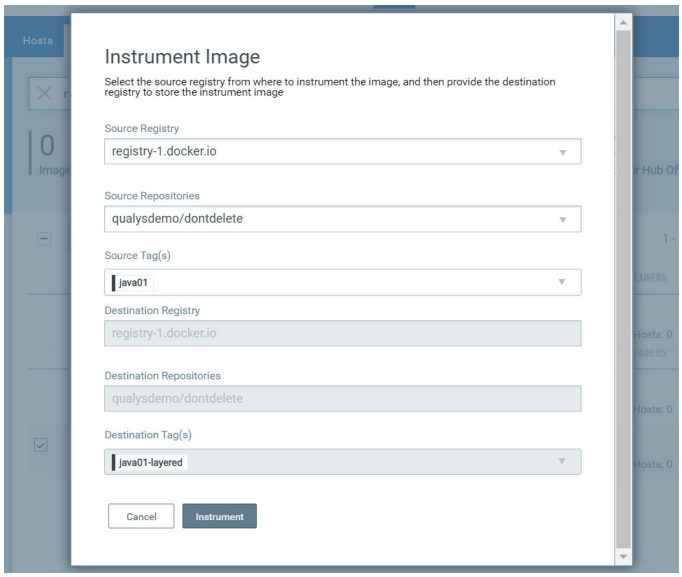
This option uses the Instrumenter service. Once the Instrumenter service is up and running in your environment, you can instrument your images. Only images that have been scanned by a registry scan job (registry sensor) will have the Instrument option in the UI. To find the images you can instrument from the UI, go to **Assets** > **Images** and perform a search using this search query: source: REGISTRY



You can add additional search fields to help narrow down the list further. Then, in the search results, identify the image you want to instrument and pick **Instrument** from the Quick Actions menu.



On the Instrument Image page, choose the source registry. You'll notice that the destination registry has the same value as the source registry. Click **Instrument** again.



What happens next?

The Instrumenter service will pull the image down, add in our solution and push the image back to the destination registry.

Note the destination tags

Take note of the destination tag(s) assigned to the instrumented image. We take the source tag and append -layered to create the destination tag. For example, in the example above, you'll see that the source tag is java01 and the destination tag is java01-layered. You'll be able to search for instrumented images by the destination tag.

Instrument images using CLI mode

The Instrument option in the UI lets you instrument container images that have been scanned by a registry scan job (registry sensor). Use the CLI mode option to instrument any image on your local host directly without the need for a registry scan. The image is not pushed to any repository because the instrumentation happens locally. The new -layered instrumented image will appear on the local machine and in the Container Security UI.

How it works

When you instrument an image using CLI mode, we'll immediately add in our solution and create the instrumented image (appended with -layered) at the same location. One command will instrument one image only, and then it will exit as soon as the instrumentation is done. The instrumented image will appear in the Container Security UI where you can view details about it.

Prerequisites

- Request access to the Docker Hub private repo for qualys/crs-cli-instrumenter. To request access, reach out to Qualys Support from qualys.com/support and be sure to include your Docker Hub ID in your message.

- Run docker login with the provided Docker Hub ID on the instance where you will run instrumenter.sh in CLI mode.

Using CLI mode

1) Pull the docker CLI files from github. You can download them from https://github.com/Qualys/qualys_crs_instrumenter

2) Edit **instrumenter.sh** to configure user specific details for proxy and vault usage.

3) Run the docker CLI script with the minimum required parameters. The script will run with CLI mode enabled by default. Required fields are endpoint and image. Policy ID is optional. (See Deploy the Instrumenter Service to learn about additional options.)

```
./instrumenter.sh --endpoint <endpoint> --image <image> [--
policyid <policy id>]
```

For example:

```
./instrumenter.sh --endpoint "qualys_joe:my-
password@gateway.qg1.apps.qualys.com/crs/v1.3" --image
"6d9ae1a5c970" [--policyid "5fd20b4321dabf0001fdc464"]
```

Where:

<endpoint> is in the format of username:password@url if you are not using a vault. Only url is needed for the endpoint when you are using a vault.

<image> is the image Id (e.g. “6d9ae1a5c970”) or repository name:tag (e.g. “library/centos:centos72”or “java:latest”) for the container image you want to instrument. The image must be present locally where you’re running the CLI command.

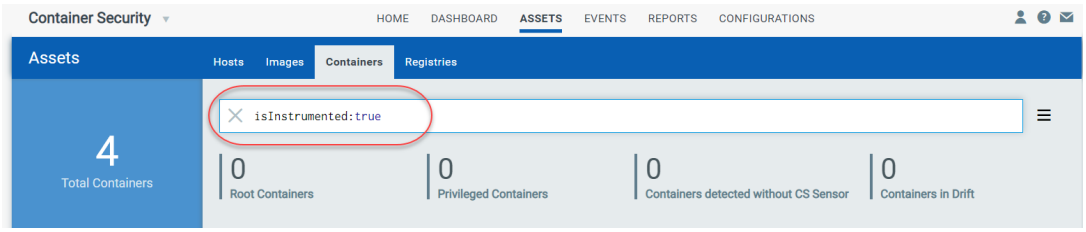
<policy> is the policy Id (e.g. “5fd20b4321dabf0001fdc464”) for the policy you want to immediately apply to the instrumented image.

Instrumented image appears in the UI

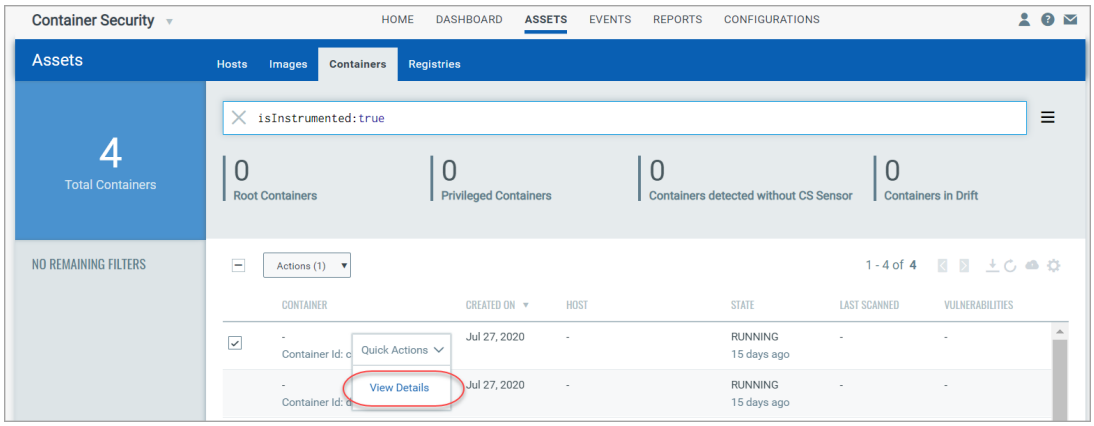
You’ll see instrumented images on the **Assets** > **Images** list. Note that for these images there is no value shown in the Registry column since these were instrumented on the local host using the CLI mode (not pulled from the registry). Also, these images have not been scanned yet so there are no vulnerabilities shown.

View details for instrumented container

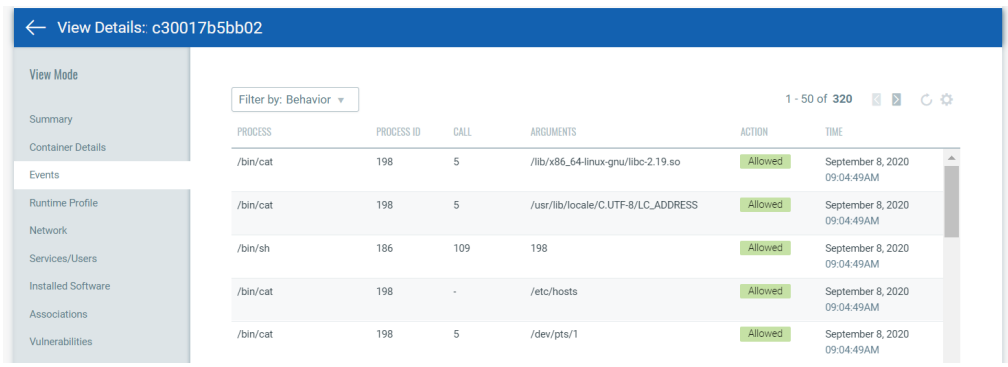
To find the instrumented container, go to **Assets** > **Containers** and perform a search using this search query: isInstrumented: true



Then choose **View Details** from the Quick Actions menu for any container listed as a result of your search.



Go to the **Events** tab to view Standard and Behavior logs (pick the type of logs you want to view from the **Filter by** menu). You can use the details you find here to configure policies.



The system call number is shown in the CALL column. Please refer to Appendix A - System Calls to look up any system call number.

Configure and Apply Policies

Create runtime policies with the rules you want to enforce, and then assign policies to instrumented images. Apply a policy to an instrumented image in order to enforce certain behavioral restrictions and secure the container spawned from the image.

About Policies

A runtime policy contains one or more rules of different types along with the mode the policy operates on, the default action for various rule types, whether to enable or disable the behavioral learning mode and more. New policies can be created from scratch or by auto-generating a behavioral profile policy from a running container (see Behavioral Learning). When defining a policy, you can change the default action for each rule type.

The core technology behind most policy rules is the idea of a “function-level” (syscall) firewall; a policy rule can specify whether a program should or should not be able to execute a particular function (syscall), given a specific set of arguments. Each rule specifies a program that the rule matches, along with a rule type, whether the rule is enabled or disabled, and then the arguments that are required for that type of rule.

Policy rule types

There are three basic policy rule types: network, file, and application (syscall/function).

Network rules

A network rule at first glance can provide “standard” firewall capabilities – allowing or denying inbound or outbound IP connectivity between the container and a given IP address and port to block lateral or external communication. The network rule has these types: Network Outbound and Network Inbound.

File rules

File rules control what files can be accessed by a specific program.The file rule has these types: Read and Write.

Application rules

Application rules are in a way a superset of the other rule types. With application rules, one can directly specify the system call that should be filtered. With the other types, Container Runtime Security (CRS) translates the rule to the appropriate one or more system calls. For example, a file rule to deny a file translates to syscall 0 (sys_read) and 2 (sys_open).The application rule has one type: Syscall. See Appendix A - System Calls.

Behavioral Learning

Note: This feature is available via the CRS API for advanced users. Please refer to the Qualys Container Runtime Security API Guide for complete details.

Container Runtime Security can automatically learn the behavior of the application in your environment by recording the activities being performed in the container. It captures all the network communication whether it is lateral or external, all the files read by any programs, program/processes and the system calls called in the container to create a baseline security policy template that can be a new policy or merged with the existing policy applied to the container.

Customers can start with enabling behavioral learning for their images in the test environment to understand the basic expected behavior of the container and how it differs from the build image. You can use CRS to create a policy template based on the learned behavior and get alerted if any violations occur.

Note: Behavioral logging, logging mode and policy mode can be changed for a specific container.

About Configurations

Configurations can be applied only to images with instrumentation in them (via UI). When you apply a configuration to the container image, all the containers spawned from that image are secured and adhere to your configuration. A change to the configuration assigned to the image will be applied to all the running containers.

You can also apply a specific configuration to containers directly in the absence of an image assigned policy (only via API). .

	Image	**Container**
Config Application	Applied via UI General; applies to all containers spawned by the instrumented image	Applied via API Specific; applies to the specific containers only
Config Usage	Day to Day Operational usage	Troubleshooting, Incident Response, Forensic

Configuration via UI consists of two objects – Policy and LogMode. More parameters are available via the API. Once a configuration is created, it can be assigned to images and containers via the API. From the UI it's only possible to update the policy and logmode for a given instrumented image (source:INSTRUMENTATION).

Configuration components

Configuration components (via UI) include: Policy and LogMode.

Policy

The policy is a component that contains one or more rules of different types along with the mode the policy operates on, the default action for various rule types, whether to enable or disable the behavioral learning mode and more.

LogMode

You can choose from the following LogMode options to determine which policy hit events should be logged:

None - No events get logged.

PolicyMonitor - Only policy hit events with Monitor action.

PolicyDeny - Only policy hit events with Deny action.

PolicyMonitorDeny - Only policy hit events with Monitor or Deny action.

PolicyAllow - Only policy hit events with Allow action.

PolicyAll - Only policy hit events with Monitor, Deny or Allow action.

Behavior - Only detailed behavioral events.

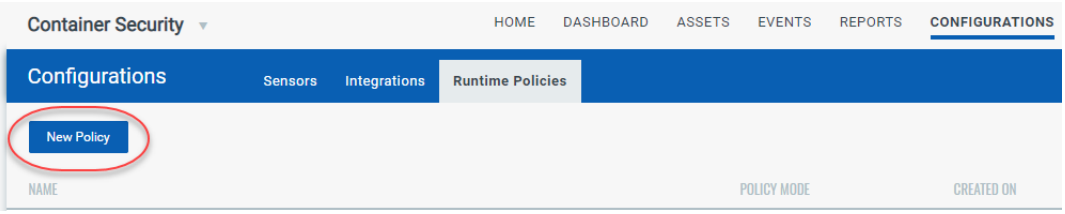
All - Includes events that match PolicyAll plus events that match Behavior.

See Configure Instrumentation to learn how to choose the LogMode for an image.

Create new policies

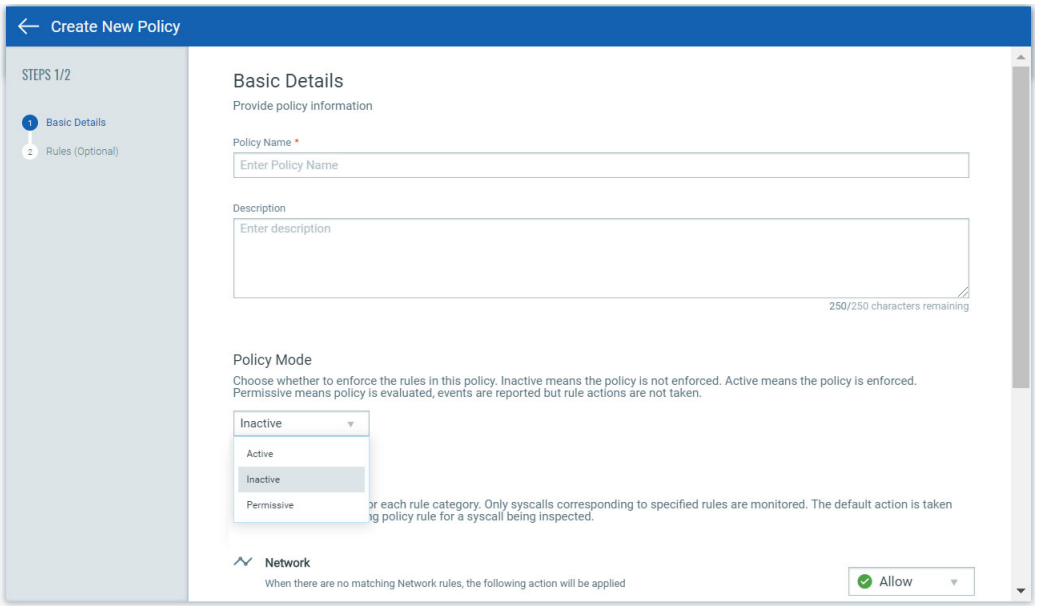
Define runtime policies with rules for monitoring and securing running containers.

Go to **Configurations** > **Runtime Policies**, and then click the **New Policy** button.



Basic Details

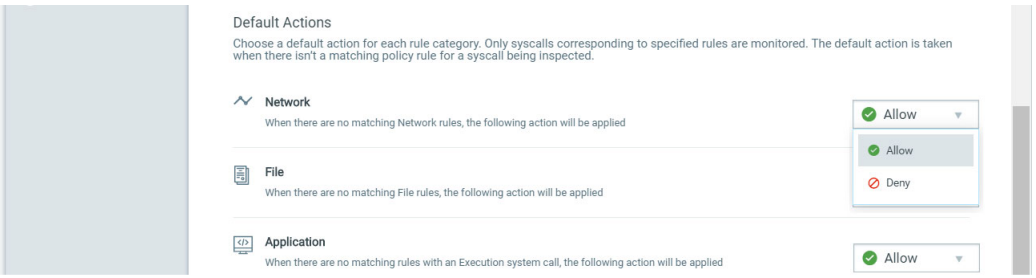
Under Basic Details, you'll provide a policy name and description, and choose a policy enforcement mode (Active, Inactive, Permissive). The option you pick determines whether or not the policy rules will be enforced on the containers that are spawned from the image. The policy is enforced only when Active is selected. When Permissive is selected, the events are reported but actions are not enforced. Note that you can change this at any time after the policy is saved. See Set policy enforcement to learn more.



Next, choose default actions for Network, File and Application rule types. This is the default action that will be taken unless there is a policy rule that overwrites this action.

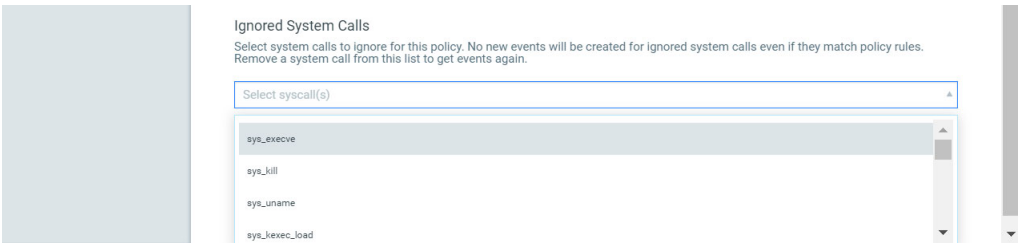
For example, you can choose Allow as the default action for Network rules to allow all inbound and outbound traffic to/from the instrumented container and then set up specific Network rules to deny traffic to a particular IP address and port.

For Application rules, the default action only applies to rules with an Execution system call selected.



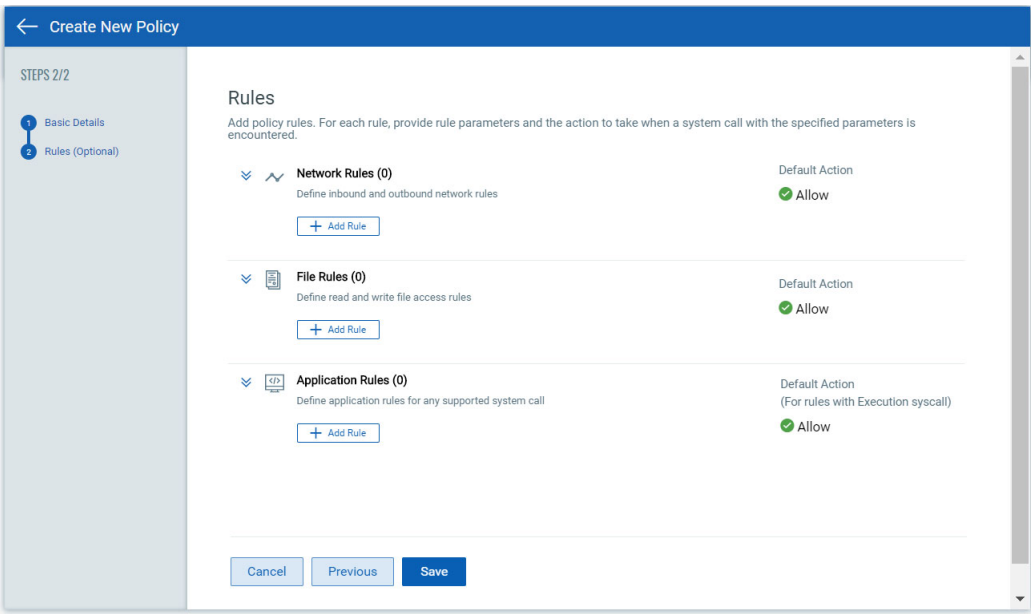
Scroll down further to define a list of system calls that you want to ignore for the policy. Add one or more system calls from the drop-down list. When a system call is ignored, no new events will be created for the system call even if it matches one of the policy rules. This will save you from having to modify all the rules that include a particular system call you want to ignore.

If you want to start getting events for an ignored system call in the future, simply edit the policy to remove the system call from the ignored system calls list. You'll be able to remove individual system calls or clear the entire list.



Rules

Go to the **Rules** tab to add policy rules. You can add as many rules as you like. Simply click the **Add Rule** button for Network Rule, File Rule or Application Rule. See Rule Types below to understand the parameters you'll set for each rule type.



For each new rule, give the rule a name, choose the rule type, set a rule action, and choose whether the rule is enabled or disabled. When you're done, click **Add Rule** to save it to your policy. Optionally, click **Save and Add another** to save the rule and create another rule of the same type.

When you're done adding rules, click **Save**. Your new policy will appear on the Runtime Policies list where you can manage it.

Rule Types

Here's a look at the types of rules you can add to your policies and the parameters you'll need to provide for each rule type. For Network and File rules, we watch particular system calls by default. For Application rules, you'll pick the system call the rule applies to.

Rule Category	Rule Type	Default System Call	Rule Parameters	Description
Network	Network Outbound	sys_connect	IP Address & Port	Allow, deny or monitor outbound traffic. The IP & port refers to the destination IP and port to which the process in the instrumented container is either to be allowed, blocked or monitored. When port is left blank, it acts as a wildcard (*).
Network	Network Inbound	sys_accept, sys_accept4	IP Address & Port	Allow, deny or monitor inbound traffic. The IP refers to the source IP from where the request is made to the instrumented container. Port refers to the bind port or container port. When port is left blank, it acts as a wildcard (*).
File	Read	sys_open	Program & File	Allow, deny or monitor read access to a particular file by a particular program
File	Write	sys_write	Program & File	Allow, deny or monitor write access to a particular file by a particular program
Application	Syscall	user selected system call	Program, Argument 1, Argument 2, Argument 3	This is an advanced rule type. You must be familiar with the selected system call to know the arguments, if any, that must be defined for the system call. Note that a rule with an Execution syscall only applies to the parent program defined in the rule and not child programs spun up from the parent program. In other words, the child program may be allowed to execute a file that the parent program is prevented from executing. Use * to prevent all programs from executing a certain file.

Using the API?

You can also create and update policies using the Container Runtime Security API. Once saved, your policies will appear in the Container Security UI.

When using the API, you have the option to auto-generate a policy based on what's been observed for your instrumented container. You'll use the following API endpoint to build a policy based on a container's behavior:

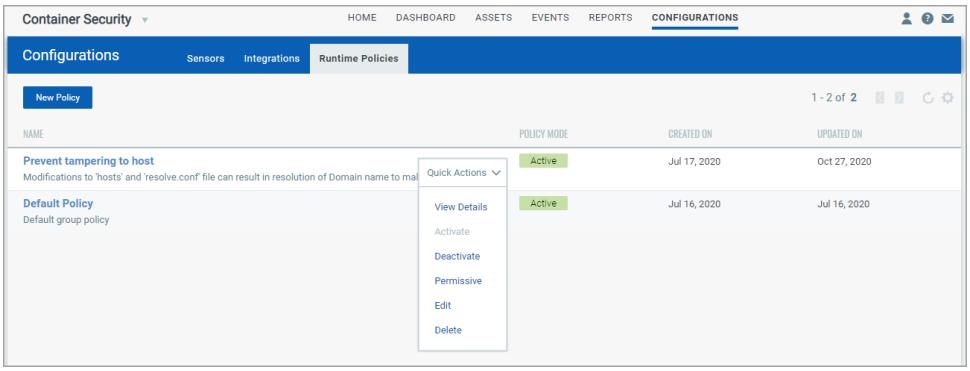
```
/csapi/v1.2/runtime/containers/{containerSha}/template
```

Please refer to the Qualys Container Runtime Security API Guide for complete details on API endpoints, input parameters and samples.

Manage your policies

You can view, update and delete policies from the Runtime Policies list. You can also change the policy enforcement mode.

Go to **Configurations** > **Runtime Policies** to get started. You'll see a list of the saved policies in your account. Choose an option from the Quick Actions menu.



View Details

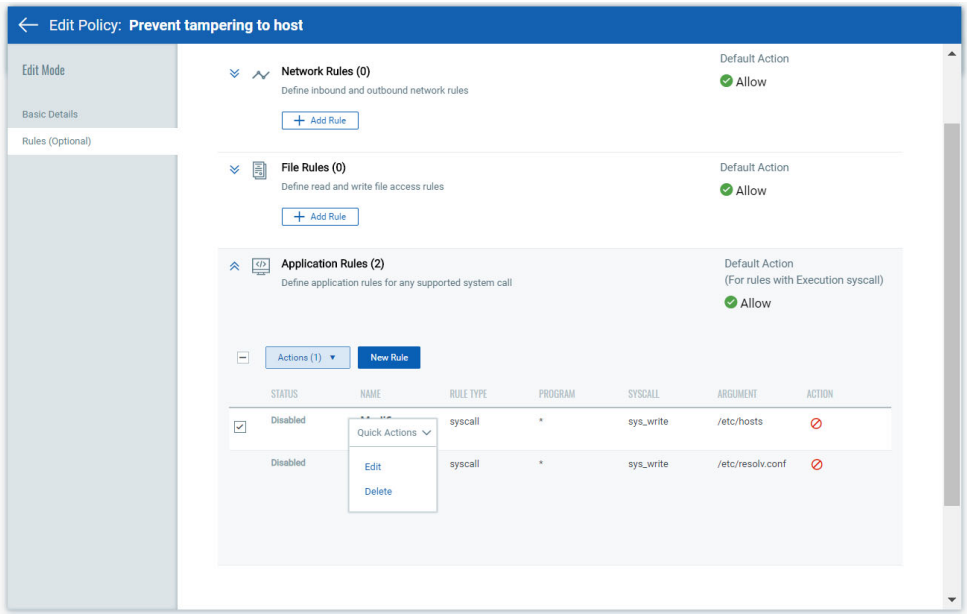
Select **View Details** for any policy in the list to see more details about the policy. You'll see basic details like policy name, description and creation date, plus default actions for the different rule types. You'll also see the rules that make up the policy.

Activate, Deactivate, Permissive

Choose one of these options to change the enforcement mode for the policy: Activate, Deactivate, Permissive. See Set policy enforcement to learn about these options.

Edit Policy

Choose **Edit** from the Quick Actions menu to make changes to a policy. You can make changes to any of the policy settings and policy rules. On the **Rules** tab, expand a rule type to see all the rules for that type. Edit and delete individual rules, and add new rules. Click **Save** when you're done making changes to the policy.



Delete Policy

Choose **Delete** from the Quick Actions menu for the policy you want to remove. Note that you can only delete policies that are not associated with instrumented images/containers. You'll see an Error if the policy is associated with an image/container. In this case, you must disassociate the policy and then try again.

Tip - To find instrumented images/containers, go to **Assets** > **Images** or **Assets** > **Containers** and use the following query.

Search query:
isInstrumented:true

Set policy enforcement

We provide three policy enforcement options, which determine whether or not the policy rules will be enforced on the containers that are spawned from the image. When testing new policies, we recommend you set the policy to Permissive mode, which allows you to see the rule hits without actually enforcing the rules.

Identify a policy in the list and choose from these policy enforcement options on the Quick Actions menu:

Activate - Activate the policy on all images that have the policy applied. The policy gets enforced on all containers spawned from that image.

Deactivate - Deactivate the policy on all images/containers where its been applied. This may be needed if you are troubleshooting an issue and want to stop policy enforcement.

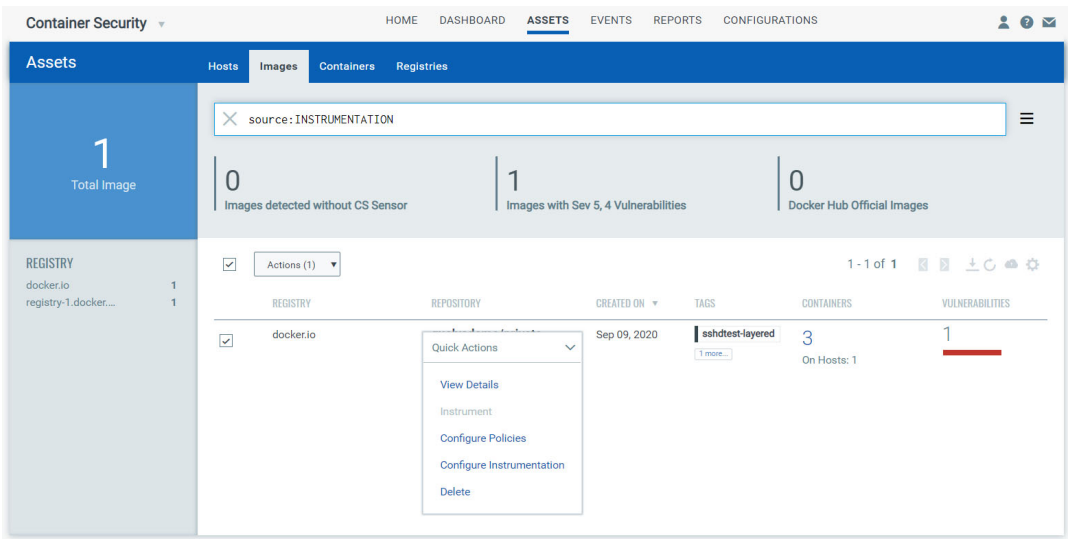
Permissive - Put the policy in permissive mode. When in permissive mode, the rules in the policy will not be enforced but all activity is logged for rule hits. This is recommended when starting out with a new policy so you can get an idea of the rule hits which will allow you to go back and fine tune the policy to make sure it's working as you expected.

Apply policy to instrumented image

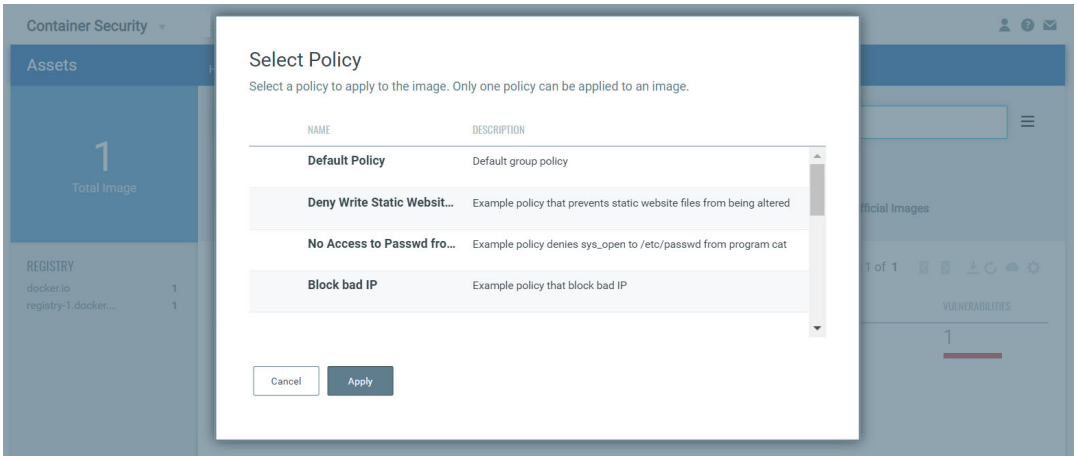
Apply a security policy to an instrumented image to enforce certain behavioral restrictions and secure the container spawned from that image. The first thing you'll want to do is find your instrumented image.

Go to **Assets** > **Images** and perform a search using this search query:
source: INSTRUMENTATION.

Then choose **Configure Policies** to select the policy you want to apply to the image.



You'll see a list of policies defined in your subscription. Select the policy you want to assign to the image. You can choose only one policy. Then click **Apply**.



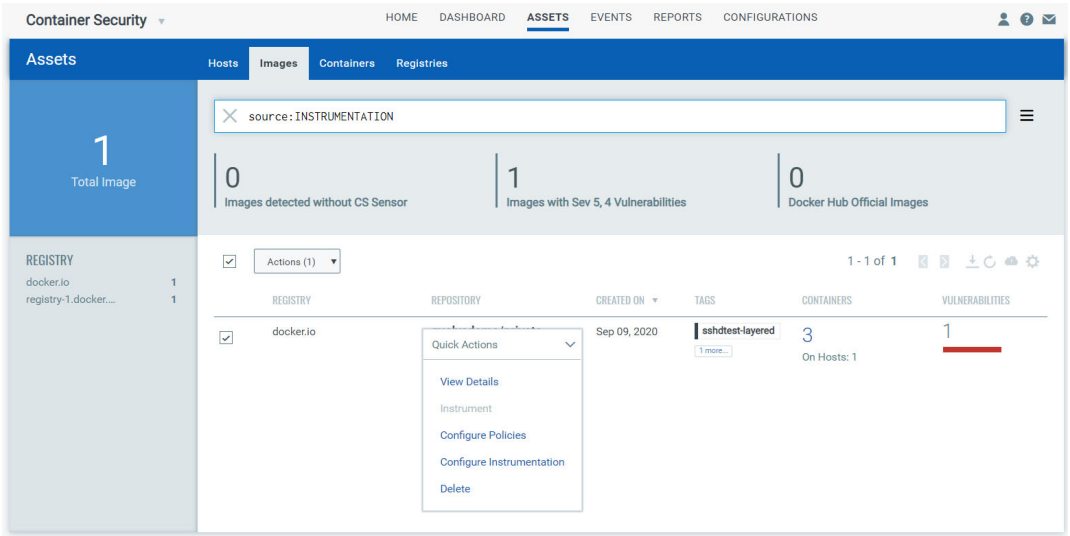
Configure Instrumentation

Once a policy is applied to an image you can choose a LogMode to determine what is logged in a container for policy hits (rule matches) and behavior.

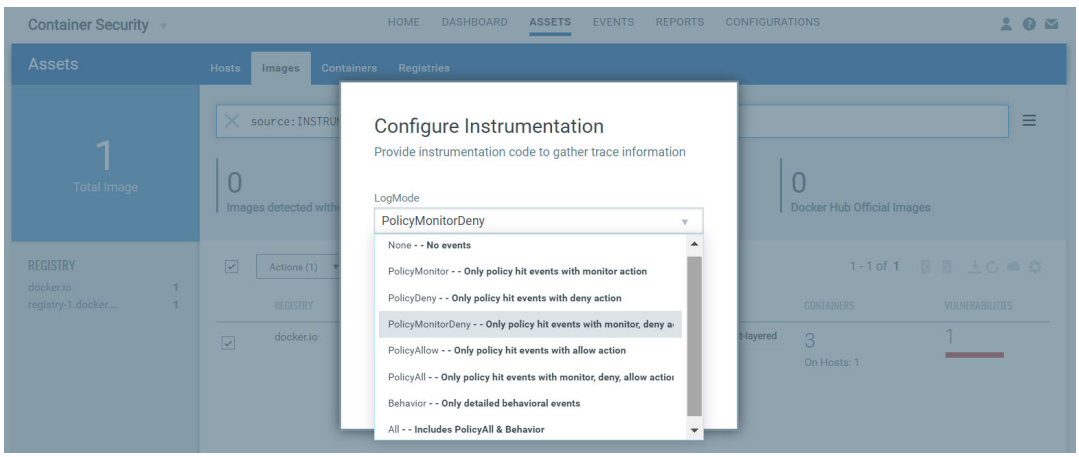
Select the LogMode

Go to **Assets** > **Images** and perform a search using this search query: source: INSTRUMENTATION.

Then choose **Configure Instrumentation** from the Quick Actions menu of an instrumented image to select the LogMode.



Choose an option from the **LogMode** menu, and then click **Apply**. Your selection will determine which policy hits get logged in the container security UI.



Run containers from instrumented image

You can now spawn a container from the instrumented image.

```
docker run -itd -e LI_MQURL=https://<cmsqagpublic VIP>/crs/v1.2 -e
LI_MQSKIPVERIFYTLS=true <your registry/repo:tag>
```

The policy applied to the instrumented image gets enforced on the container and activities are logged as per the selected log mode.

Proxy Settings

You'll need to provide proxy details if the instrumented container is running behind a proxy to allow the CRS instrumenter to talk to the Qualys backend. The instrumented container can be launched with any of following proxy environment variables. If multiple proxy environment variables are used, then they will be honored in the order shown below.

```
-e LI_HTTPS_PROXY=<proxy>
-e LI_HTTP_PROXY=<proxy>
-e HTTPS_PROXY=<proxy>
-e HTTP_PROXY=<proxy>
```

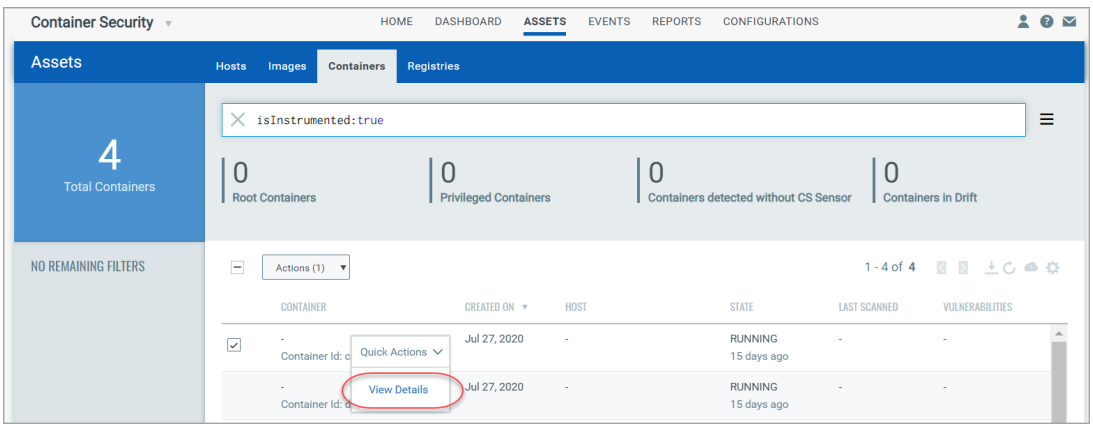
The following example uses the LI_HTTPS_PROXY environment variable:

```
docker run -itd -e LI_MQURL=https://<cmsqagpublic VIP>/crs/v1.2 -e
LI_MQSKIPVERIFYTLS=true -e LI_HTTPS_PROXY=<proxy> <your
registry/repo:tag>
```

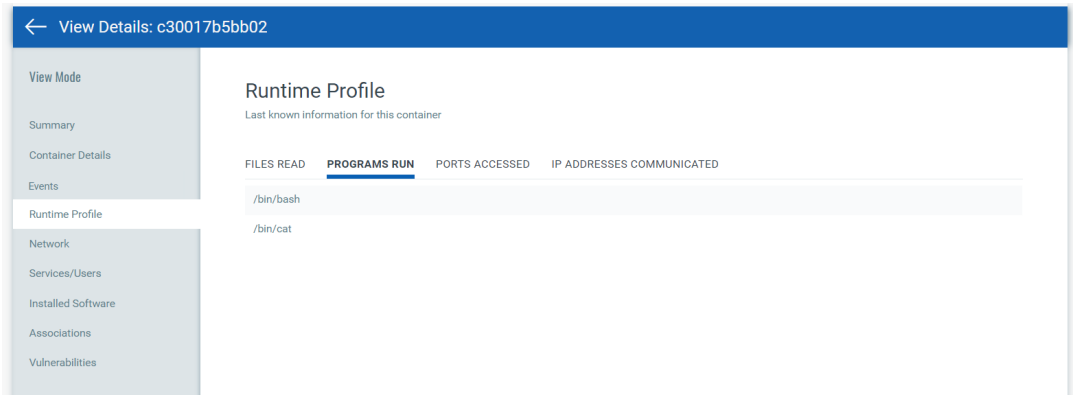
View details for instrumented container image

Go to **Assets** > **Containers** and perform a search using this search query:
isInstrumented: true

Then choose **View Details** from the Quick Actions menu for any container listed.

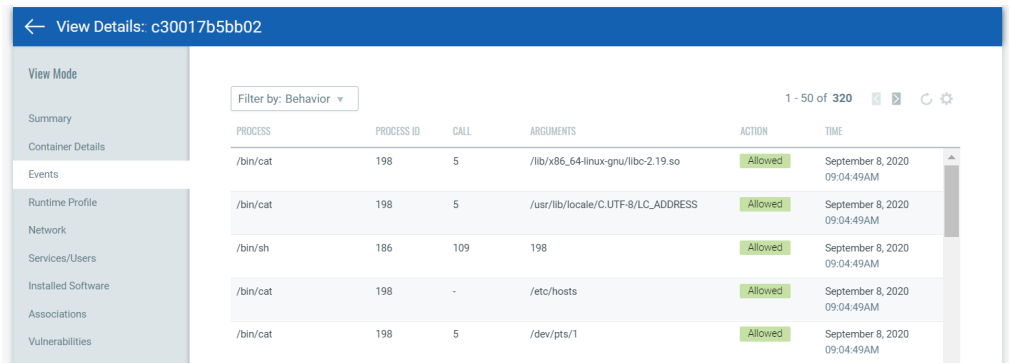


The **Runtime Profile** tab shows the resources that are tracked to gather trace information. It shows the files that are being read on the container, programs being run, ports accessed, and IP address information.



The **Events** tab shows a log of when each resource being tracked is accessed, and whether the access was allowed, monitored or denied depending on the applied policy. You can use the filter option to view standard logs or behavior logs. Standard logs show policy hits. Behavior logs show system calls. The system call number is shown in the CALL column. Please refer to Appendix A - System Calls to look up any system call number.

Tip - Use the details you find here to create new runtime policies.



Enable Additional Daemon logging (Optional)

After you've successfully instrumented an image, if you need to enable logging for troubleshooting the daemon you have two options. Option 1 covers spawning a container from the instrumented image with specific logging config environment variables. Option 2 is to edit the daemon.toml file in an already instantiated container and provide the logging config, then restarting the daemon process. The logging config will enable additional daemon log levels.

Log levels

The following log levels are supported. Please note that log levels have a certain hierarchy as listed below. When you choose a log level, all levels below it are also included. For example, a level of "trace" includes all other levels since it's at the top of the hierarchy. A level of "error" includes fatal and panic but not warn, info, debug or trace.

Log levels:
- trace
-- debug
--- info
---- warn
----- error
------ fatal
------- panic

How to enable daemon logging

You can enable daemon logging using either of the options described below.

Option 1: Use environment variables

Use LI_LOGLEVEL to specify the log level you want, and LI_DAEMONLOG to specify the log file and path where the daemon should write logs. Run the following command:

```
docker run -itd -e LI_MQSKIPVERIFYTLS=true -e LI_LOGLEVEL="<log-
level>" -e LI_DAEMONLOG="<path/filename>" <repo:tag>
```

Example:

```
docker run -itd -e LI_MQSKIPVERIFYTLS=true -e LI_LOGLEVEL="debug"
-e LI_DAEMONLOG="/tmp/daemonlogs_new" my-repo:my-tag
```

Option 2: Edit the toml file

Go to **/etc/layint** and edit the **daemon.toml** configuration file. Append the following config options to specify the log level and file path:

```
logLevel = "<log-level>"
daemonLog = "<path/filename>"
```

Example:

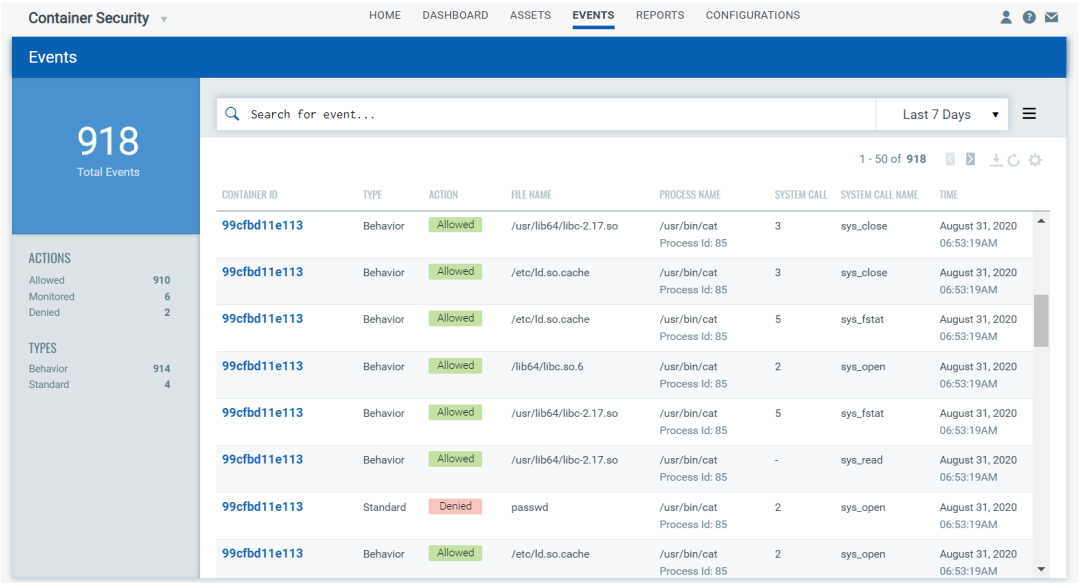
```
logLevel = "debug"
daemonLog = "/tmp/daemonlogs_new"
```

Note: You will need to restart the daemon process for this change to take effect.
Note: A valid directory path must be present inside the container.

View Your Events

Runtime events will be listed on the **Events** tab. Here you can search events and drill-down into event details. Use options on the left side bar to quickly find events by the action taken (Allowed, Monitored, Denied) and the event type (Behavior, Standard).

Use the search field above the list to find events by event details like the container SHA the event is associated with, system call, process, and more.

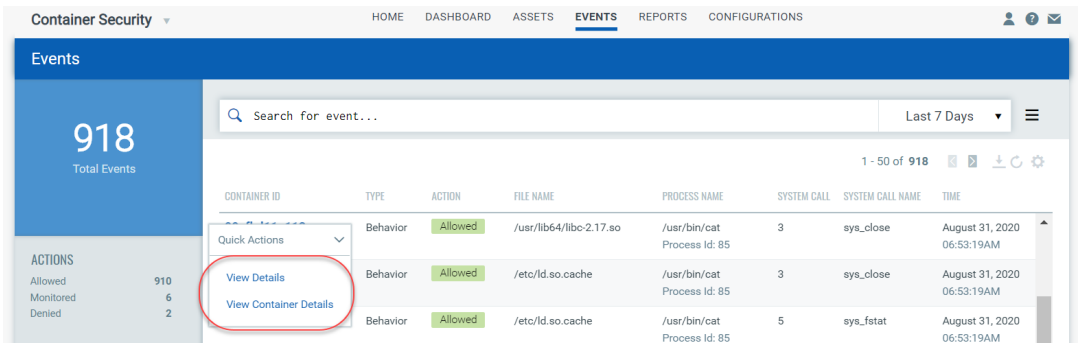


Drill-down into event details

You can choose from the following Quick Action options for any event in the list:

View Details - Select this option to get event details like the process, system call, file name and action.

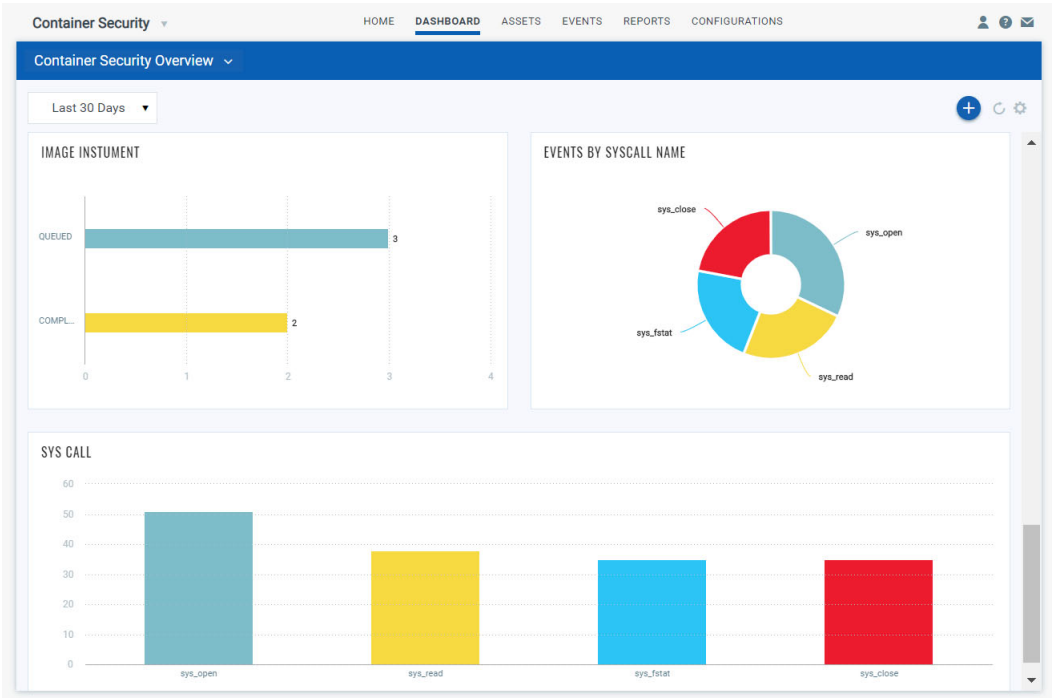
View Container Details - Select this option to see container details, including events, runtime profile, container information, associations and vulnerabilities.



View event details on dashboard

Go to **Dashboard** and you'll see widgets with info about events like the number of events by action, event type and system call name. You'll also see the number of images that have been successfully instrumented and the number of images currently queued for instrumentation.

Here's a sample dashboard. Check out the dashboard in your own account to see all widgets.



Appendix A - System Calls

See the table below for supported system calls in numerical order along with the system call names and required arguments, if available. You can use this information when configuring runtime policies with rules targeting specific system calls.

SYSCALL	SYSCALL Name	ARG1	ARG2	ARG3
0	sys_read	filename		
1	sys_write	filename		
2	sys_open	filename		
3	sys_close	filename		
4	sys_stat	filename		
5	sys_fstat	filename		
6	sys_lstat	filename		
7	sys_poll			
8	sys_lseek			
9	sys_mmap			
10	sys_mprotect			
11	sys_munmap			
12	sys_brk			
13	sys_rt_sigaction			
14	sys_rt_sigprocmask			
15	sys_rt_sigreturn			
16	sys_ioctl			
19	sys_readv	filename		
20	sys_writev	filename		
21	sys_access			
22	sys_pipe			
23	sys_select			
24	sys_sched_yield			
25	sys_mremap			
26	sys_msync			
27	sys_mincore			
28	sys_madvise			
29	sys_shmget			
30	sys_shmat			
31	sys_shmctl			

SYSCALL	SYSCALL Name	ARG1	ARG2	ARG3
32	sys_dup			
33	sys_dup2			
34	sys_pause			
35	sys_nanosleep			
36	sys_getitimer			
37	sys_alarm			
38	sys_setitimer			
39	sys_getpid			
40	sys_sendfile			
41	sys_socket	domain	type	socket
42	sys_connect	port	address	
43	sys_accept	port	address	
44	sys_sendto	port	address	
45	sys_recvfrom	port	address	
46	sys_sendmsg			
47	sys_recvmsg			
48	sys_shutdown			
49	sys_bind	port	address	
50	sys_listen			
51	sys_getsockname			
52	sys_getpeername			
53	sys_socketpair			
55	sys_setsockopt			
56	sys_clone			
57	sys_fork			
58	sys_vfork			
59	sys_execve	filename		
60	sys_exit			
61	sys_wait4			
62	sys_kill			
63	sys_uname			
64	sys_semget			
65	sys_semop			
66	sys_semctl			
67	sys_shmdt			

SYSCALL	SYSCALL Name	ARG1	ARG2	ARG3
68	sys_msgget			
69	sys_msgsnd			
70	sys_msgrcv			
71	sys_msgctl			
72	sys_fcntl			
73	sys_flock			
74	sys_fsync			
75	sys_fdatasync			
76	sys_truncate			
77	sys_ftruncate			
78	sys_getdents			
79	sys_getcwd			
80	sys_chdir			
81	sys_fchdir			
82	sys_rename			
83	sys_mkdir			
84	sys_rmdir			
85	sys_creat			
86	sys_link			
87	sys_unlink			
88	sys_symlink			
89	sys_readlink			
90	sys_chmod			
91	sys_fchmod			
92	sys_chown			
93	sys_fchown			
94	sys_lchown			
95	sys_umask			
96	sys_gettimeofday			
97	sys_getrlimit			
98	sys_getrusage			
99	sys_sysinfo			
100	sys_times			
101	sys_ptrace			
102	sys_getuid			

SYSCALL	SYSCALL Name	ARG1	ARG2	ARG3
103	sys_syslog			
104	sys_getgid			
105	sys_setuid			
106	sys_setgid			
107	sys_geteuid			
108	sys_getegid			
109	sys_setpgid			
110	sys_getppid			
111	sys_getpgrp			
112	sys_setsid			
113	sys_setreuid			
114	sys_setregid			
115	sys_getgroups			
116	sys_setgroups			
117	sys_setresuid			
118	sys_getresuid			
119	sys_setresgid			
120	sys_getresgid			
121	sys_getpgid			
122	sys_setfsuid			
123	sys_setfsgid			
124	sys_getsid			
125	sys_capget			
126	sys_capset			
127	sys_rt_sigpending			
128	sys_rt_sigtimedwait			
129	sys_rt_sigqueueinfo			
130	sys_rt_sigsuspend			
131	sys_sigaltstack			
132	sys_utime			
133	sys_mknod			
134	sys_uselib			
135	sys_personality			
136	sys_ustat			
137	sys_statfs			

SYSCALL	SYSCALL Name	ARG1	ARG2	ARG3
138	sys_fstatfs			
139	sys_sysfs			
140	sys_getpriority			
141	sys_setpriority			
142	sys_sched_setparam			
143	sys_sched_getparam			
144	sys_sched_setscheduler			
145	sys_sched_getscheduler			
146	sys_sched_get_priority_max			
147	sys_sched_get_priority_min			
148	sys_sched_rr_get_interval			
149	sys_mlock			
150	sys_munlock			
151	sys_mlockall			
152	sys_munlockall			
153	sys_vhangup			
154	sys_modify_ldt			
155	sys_pivot_root			
156	sys_sysctl			
157	sys_prctl			
158	sys_arch_prctl			
159	sys_adjtimex			
160	sys_setrlimit			
161	sys_chroot			
162	sys_sync			
163	sys_acct			
164	sys_settimeofday			
165	sys_mount			
166	sys_umount2			
167	sys_swapon			
168	sys_swapoff			
169	sys_reboot			
170	sys_sethostname			
171	sys_setdomainname			
172	sys_iopl			

SYSCALL	SYSCALL Name	ARG1	ARG2	ARG3
173	sys_ioperm			
174	sys_create_module			
175	sys_init_module			
176	sys_delete_module			
177	sys_get_kernel_syms			
178	sys_query_module			
179	sys_quotactl			
180	sys_nfsservctl			
181	sys_getpmsg			
182	sys_putpmsg			
183	sys_afs_syscall			
184	sys_tuxcall			
185	sys_security			
186	sys_gettid			
187	sys_readahead			
188	sys_setxattr			
189	sys_lsetxattr			
190	sys_fsetxattr			
191	sys_getxattr			
192	sys_lgetxattr			
193	sys_fgetxattr			
194	sys_listxattr			
195	sys_llistxattr			
196	sys_flistxattr			
197	sys_removexattr			
198	sys_lremovexattr			
199	sys_fremovexattr			
200	sys_tkill			
201	sys_time			
202	sys_futex			
203	sys_sched_setaffinity			
204	sys_sched_getaffinity			
205	sys_set_thread_area			
206	sys_io_setup			
207	sys_io_destroy			

SYSCALL	SYSCALL Name	ARG1	ARG2	ARG3
208	sys_io_getevents			
209	sys_io_submit			
210	sys_io_cancel			
211	sys_get_thread_area			
212	sys_lookup_dcookie			
213	sys_epoll_create			
214	sys_epoll_ctl_old			
215	sys_epoll_wait_old			
216	sys_remap_file_pages			
217	sys_getdents64			
218	sys_set_tid_address			
219	sys_restart_syscall			
220	sys_semtimedop			
221	sys_fadvise64			
222	sys_timer_create			
223	sys_timer_settime			
224	sys_timer_gettime			
225	sys_timer_getoverrun			
226	sys_timer_delete			
227	sys_clock_settime			
228	sys_clock_gettime			
229	sys_clock_getres			
230	sys_clock_nanosleep			
231	sys_exit_group			
232	sys_epoll_wait			
233	sys_epoll_ctl			
234	sys_tgkill			
235	sys_utimes			
236	sys_vserver			
237	sys_mbind			
238	sys_set_mempolicy			
239	sys_get_mempolicy			
240	sys_mq_open			
241	sys_mq_unlink			
242	sys_mq_timedsend			

SYSCALL	SYSCALL Name	ARG1	ARG2	ARG3
243	sys_mq_timedreceive			
244	sys_mq_notify			
245	sys_mq_getsetattr			
246	syskexec_load			
247	sys_waitid			
248	sys_add_key			
249	sys_request_key			
250	sys_keyctl			
251	sys_ioprio_set			
252	sys_ioprio_get			
253	sys_inotify_init			
254	sys_inotify_add_watch			
255	sys_inotify_rm_watch			
256	sys_migrate_pages			
257	sys_openat			
258	sys_mkdirat			
259	sys_mknodat			
260	sys_fchownat			
261	sys_futimesat			
262	sys_newfstatat			
263	sys_unlinkat			
264	sys_renameat			
265	sys_linkat			
266	sys_symlinkat			
267	sys_readlinkat			
268	sys_fchmodat			
269	sys_faccessat			
270	sys_pselect6			
271	sys_ppoll			
272	sys_unshare			
273	sys_set_robust_list			
274	sys_get_robust_list			
275	sys_splice			
276	sys_tee			
277	sys_sync_file_range			

SYSCALL	SYSCALL Name	ARG1	ARG2	ARG3
278	sys_vmsplice			
279	sys_move_pages			
280	sys_utimensat			
281	sys_epoll_pwait			
282	sys_signalfd			
283	sys_timerfd_create			
284	sys_eventfd			
285	sys_fallocate			
286	sys_timerfd_settime			
287	sys_timerfd_gettime			
288	sys_accept4			
289	sys_signalfd4			
290	sys_eventfd2			
291	sys_epoll_create1			
292	sys_dup3			
293	sys_pipe2			
294	sys_inotify_init1			
295	sys_preadv			
296	sys_pwritev			
297	sys_rt_tgsigqueueinfo			
298	sys_perf_event_open			
299	sys_recvmmsg			
300	sys_fanotify_init			
301	sys_fanotify_mark			
302	sys_prlimit64			
303	sys_name_to_handle_at			
304	sys_open_by_handle_at			
305	sys_clock_adjtime			
306	sys_syncfs			
307	sys_sendmmsg			
308	sys_setns			
309	sys_getcpu			
310	sys_process_vm_readv			
311	sys_process_vm_writev			
312	sys_kcmp			

SYSCALL	SYSCALL Name	ARG1	ARG2	ARG3
313	sys_finit_module			
314	sys_sched_setattr			
315	sys_sched_getattr			
316	sys_renameat2			
317	sys_seccomp			
318	sys_getrandom			
319	sys_memfd_create			
320	sys_kexec_file_load			
321	sys_bpf			
322	stub_execveat			
323	userfaultfd			
324	membarrier			
325	mlock2			
326	copy_file_range			
327	preadv2			
328	pwritev2			
499	li_getaddrinfo			
500	li_getnameinfo			