



# Container Security

Sensor Deployment Guide  
Version 1.32.0

April 8, 2024

Copyright 2018-2024 by Qualys, Inc. All Rights Reserved.

Qualys and the Qualys logo are registered trademarks of Qualys, Inc. All other trademarks are the property of their respective owners.

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



# Table of Contents

<b>About this Guide .....</b>	<b>5</b>
About Qualys .....	5
Qualys Support .....	5
About Container Security Documentation .....	5
<b>Container Security Overview .....</b>	<b>6</b>
Qualys Container Sensor .....	6
Sensor Modes .....	7
What data does Container Security collect? .....	8
<b>Get Started .....</b>	<b>9</b>
Qualys Subscription and Modules required .....	9
System support .....	9
Deploying Container Sensor .....	10
Installsensor.sh script command line parameters .....	13
Proxy Support .....	18
Qualys Platform (POD URL) your hosts need to access .....	19
Sensor network configuration .....	19
Static scanning of container images .....	20
Log4j vulnerability scanning .....	20
Static log4j detection .....	20
SCA scanning .....	21
Secrets Detection .....	22
Malware Detection .....	23
Events that lead to Docker asset scanning .....	23
Storage Requirements for Sensor Scans .....	24
<b>Installing the sensor on MacOS .....</b>	<b>26</b>
<b>Installing the sensor on Linux .....</b>	<b>28</b>
<b>Installing the sensor on CoreOS.....</b>	<b>29</b>
<b>Installing the sensor from Docker Hub.....</b>	<b>30</b>
Deploying the sensor on standalone docker host using docker compose .....	30
Deploying the sensor on standalone docker host using docker run .....	36
Deploying the sensor using Docker Hub on Kubernetes .....	43
<b>Installing the CI/CD Sensor in Docker-in-Docker Environment .....</b>	<b>55</b>
Step 1: Have the CS Sensor image inside a Docker-in-Docker Container .....	55
Step 2: Launch the Container Security Sensor .....	56

<b>Deploying sensor in Kubernetes .....</b>	<b>58</b>
How to Detect the Container Runtime in your Kubernetes Cluster Environment .....	59
Obtain the Container Sensor Image .....	59
Deploy in Azure Kubernetes Service (AKS) .....	62
Deploy in Kubernetes - Docker Runtime .....	62
Deploy in Kubernetes - Containerd Runtime .....	80
Deploy in Kubernetes - CRI-O Runtime .....	91
Deploy in Kubernetes - OpenShift .....	101
Deploy in Kubernetes - OpenShift4.4+ with CRI-O Runtime .....	105
Deploy in Kubernetes with TKGI - Docker Runtime .....	115
Deploy in Kubernetes with TKGI - Containerd Runtime .....	126
Deploy in Kubernetes with RKE1 - Docker Runtime .....	139
Deploy in Kubernetes with RKE2 - Containerd Runtime .....	146
Deploy in Google Kubernetes Engine (GKE) with multi-node clusters .....	150
Deploy in Kubernetes using Helm Charts .....	151
Collection of Kubernetes Cluster Attributes .....	158
Update the sensor deployed in Kubernetes .....	158
<b>Deploying sensor in Docker Swarm .....</b>	<b>163</b>
<b>Deploying sensor in AWS ECS Cluster .....</b>	<b>167</b>
<b>Scan Container Images in AWS Fargate (ECS) .....</b>	<b>172</b>
<b>Compliance with CIS Benchmark for Docker .....</b>	<b>180</b>
<b>Administration .....</b>	<b>186</b>
Sensor updates .....	186
How to uninstall the sensor .....	186
<b>Troubleshooting .....</b>	<b>188</b>
Check sensor logs .....	188
Sensor health status .....	188
Diagnostic script .....	188
Sensor crashes during upgrade .....	189
What if sensor restarts? .....	189
Duplicate Kubernetes containers .....	191
Get container runtime details .....	191

# About this Guide

Welcome to Qualys Container Security! We'll help you get acquainted with the Qualys solutions for securing your Container environments like Images, Containers and Docker Hosts using the Qualys Cloud Security Platform.

## 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](http://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/](http://www.qualys.com/support/).

## About Container Security Documentation

This document provides information on deploying the sensor on MAC, CoreOS, and various orchestrators and cloud environments.

For information on using the Container Security UI and API, refer to:

[Qualys Container Security User Guide](#)

[Qualys Container Runtime Security User Guide](#)

[Qualys Container Security API Guide](#)

[Qualys Container Runtime Security API Guide](#)

For information on deploying the sensor in CI/CD environments, refer to:

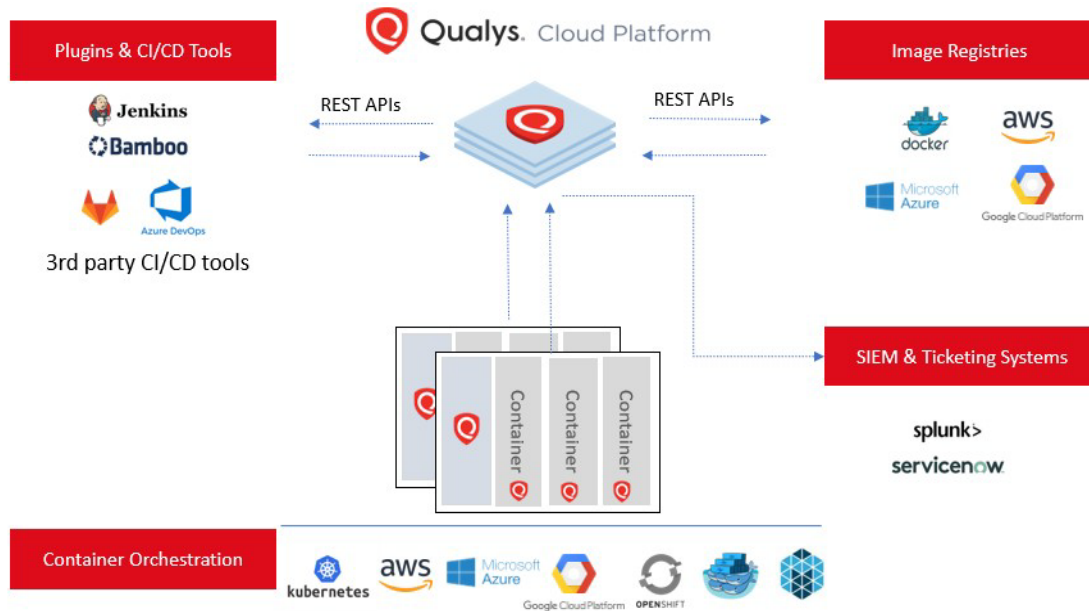
[Qualys Container Scanning Connector for Jenkins](#)

[Qualys Container Scanning Connector for Bamboo](#)

[Qualys Container Scanning Connector for Azure DevOps](#)

# Container Security Overview

Qualys Container Security provides discovery, tracking, and continuously protecting container environments. This addresses vulnerability management and policy compliance for images and containers in their DevOps pipeline and deployments across cloud and on-premise environments.



With this version, Qualys Container Security supports

- Discovery, inventory, and near-real time tracking of container environments
- Vulnerability analysis for images and containers
- Vulnerability analysis for registries
- Compliance assessment for images and containers
- Integration with CI/CD pipeline using APIs (DevOps flow)
- Uses 'Container Sensor' - providing native container support, distributed as docker image

## Qualys Container Sensor

The sensor from Qualys is designed for native support of Docker environments. Sensor is packaged and delivered as a Docker Image. Download the image and deploy it as a Container alongside with other application containers on the host.

The sensor is docker based, can be deployed on hosts in your data center or cloud environments like AWS ECS. Sensor currently is only supported on Linux Operating systems and requires docker daemon of version 1.12 and higher to be available.

Since they are docker based, the sensor can be deployed into orchestration tool environments like Kubernetes, Mesos or Docker Swarm just like any other application container.

Upon installation, the sensor does automatic discovery of Images and Containers on the deployed host, provides a vulnerability analysis of them, and additionally it monitors and reports on the docker related events on the host. The sensor lists and scans registries for vulnerable images. The sensor also performs compliance assessments. The sensor container runs in non-privileged mode. It requires a persistent storage for storing and caching files.

Currently, the sensor only scans Images and Containers. To get a vulnerability posture on the Host, you would require Qualys Cloud Agents or a scan through Qualys Virtual Scanner Appliance.

## Sensor Modes

A sensor can only be deployed in a single mode on a single container's host/cluster node.

### General

The General mode sensor is installed on your container nodes/hosts. It provides vulnerability and compliance assessments for your running containers and locally cached images. The general sensor performs demand driven assessments based on container events like containers instantiated and images pulled. There is no on demand scan or scheduled scan assessments; the sensor reacts to the container environment changes in real time. The general mode sensor must be deployed separately from the Registry or CICD sensor.

### Registry

Registry mode provides inventory and vulnerability assessment for images stored in registries. The sensor, in registry mode, will not inventory or perform vulnerability assessments of the images or containers on the host where the sensor is deployed. The sensor in registry mode must have network access to the registry URL. The registry mode sensor will not discover registries automatically. The images inventoried and assessed are scoped by the registry connector scan jobs. These scan jobs are either automatic (scheduled) or on demand. Log into the Container Security UI to configure a registry connector and scan job. Refer to the online help for guidance. The registry mode sensor must be deployed separately from the General or CICD sensor.

### CICD

CICD mode is for sensors running on CI Pipeline workers. It is demand driven assessment based on specific events. The sensor in CICD mode does not inventory or assess other images or containers running on the host/node. The sensor in CICD mode performs vulnerability assessments on specifically tagged images and the assessment results are put into a priority processing queue with a faster SLA specifically for CI Pipeline assessments. The CICD sensor must be deployed separately from the General or Registry sensor.

## What data does Container Security collect?

The Qualys Container Security sensor fetches the following information about Images and Containers in your environment:

**Inventory of Images and Containers** in your environment from commands, such as **docker ps** that lists all containers.

**Metadata information** about Images and Containers from commands, such as **docker inspect** and **docker info** that fetches low level information on docker objects.

**Event information** about Images and Containers from the docker host for docker events like created, started, killed, push, pull, etc.

**Vulnerabilities** found on Images and Containers. This is the output of the vulnerability management manifests run for identifying vulnerability information in Images and Containers. This is primarily software package listing, services running, ports, etc. For example, package manager outputs like **rpm -qa**, **npm**. This is supported across various Linux distributions (CentOS, Ubuntu, CoreOS, etc) and across images like Python, NodeJS, Ruby, and so on.

**Compliance configurations** for OCI compliant images, running containers. We are supporting a subset of controls from CIS Docker benchmarks, which are applicable to running containers and container images. Customers can assess configuration risks in their running containers and images and remediate them accordingly based on the Qualys finding. The compliance scans of containers, images will be transparent to customers and will function in a similar real-time cloud native manner like the vulnerability scanning feature.



# Get Started

## Qualys Subscription and Modules required

You need the “Container Security” (CS) module enabled for your account. Additionally, in order to get vulnerabilities for the hosts that run the containers, you would need to enable Vulnerability Management (VM), either via Scanner Appliance or Cloud Agent.

## System support

The Container Security Sensor can be run on any Operating System that has Docker version 1.12 or later. We’ve verified the Sensor on the following systems:

- CentOS Linux 7.3, 7.4, 7.5, 7.6, 7.7
- CoreOS 1855.4.0
- Debian Linux 8 (Jessie)
- Fedora Release 28
- Kali Linux
- Mac OS 10.13
- Red Hat Enterprise Linux 7.4
- Red Hat Enterprise Linux Atomic Host 7.5, 7.7
- Ubuntu 14.04, 16.04, 18.04, 20.04

The Container Security Sensor can scan container images based on the following Operating Systems:

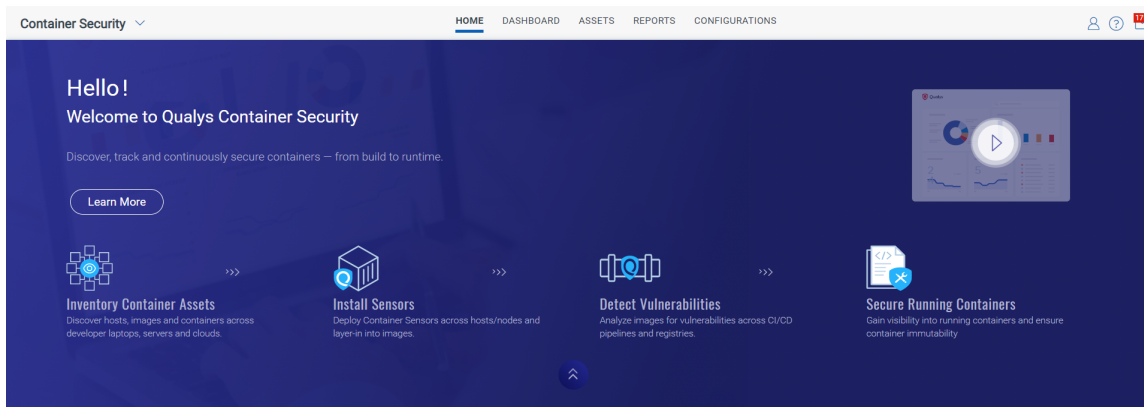
- AlmaLinux
- Alpine Linux
- Amazon Linux
- CBL-Mariner Linux
- CentOS
- Debian
- Fedora
- Google Distroless (Debian Linux)
- OpenSUSE
- Oracle Enterprise Linux
- Red Hat Enterprise Linux Server
- Rocky Linux
- SUSE Linux Enterprise Server
- Ubuntu

## Deploying Container Sensor

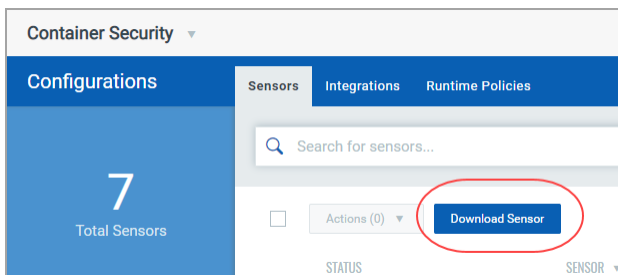
The Container Security Sensor can be installed in either of the following ways:

- download the sensor tar file from Qualys Cloud Platform and then install it on the host.
- install the sensor from Docker Hub. See [Installing the sensor from Docker Hub](#).

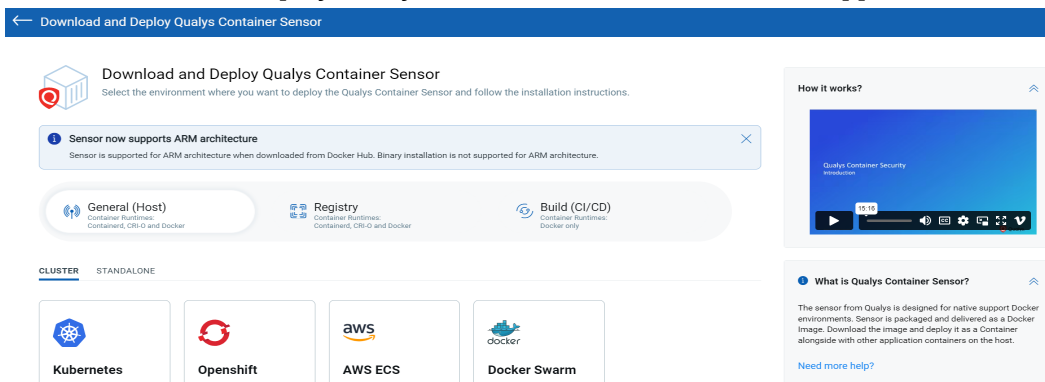
To download the sensor from Qualys Cloud Platform, log into your Qualys portal with your user credentials. Select Container Security from the module picker. As a first time user, you'll land directly into the Home page.



Go to Configurations > Sensors, and click Download Sensor.



The Download and Deploy Qualys Container Sensor window will appear, as shown below.



Pick the type of sensor you want to deploy.

**IMPORTANT:** Sensor deployment is one sensor in one mode on one host/node. Deploying more than one sensor or more than one sensor in another mode is not supported.

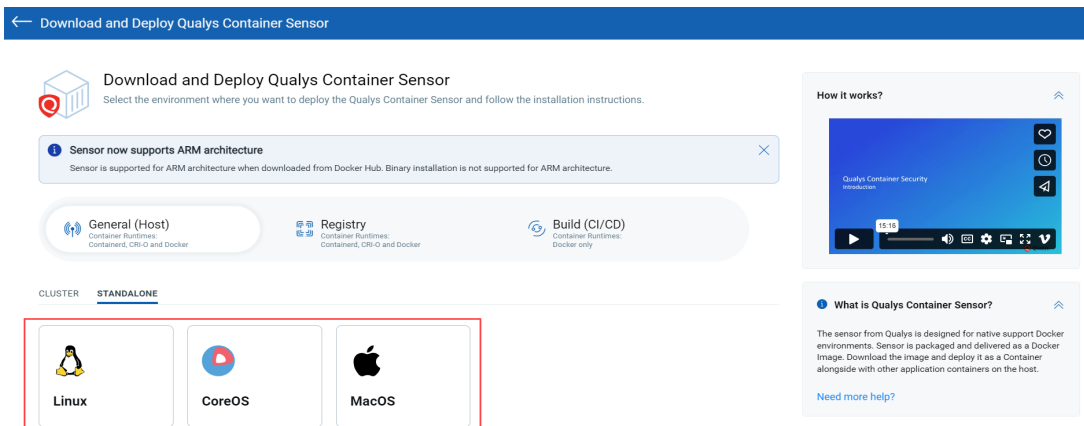
Sensor modes:

**General (Host) Sensor:** Scan any image and container on the host where sensor is running. General Sensor is installed by default if parameters for Registry or CI/CD are not provided.

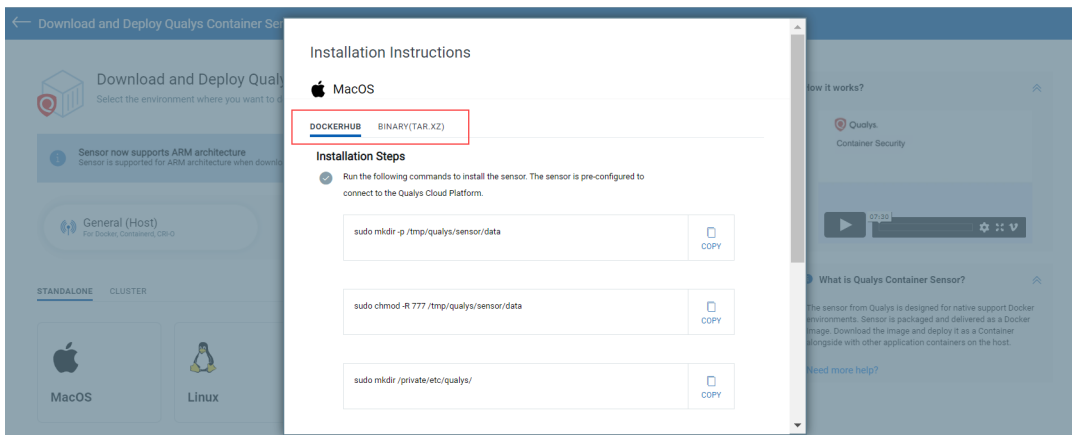
**Registry Sensor:** Scan images in a registry (public / private). For Registry you need to append the install command with **--registry-sensor** or **-r**

**Build (CI/CD) Sensor:** Scan images on CI/CD pipeline (Jenkins / Bamboo). For CI/CD you need to append the install command with **--cicd-deployed-sensor** or **-c**

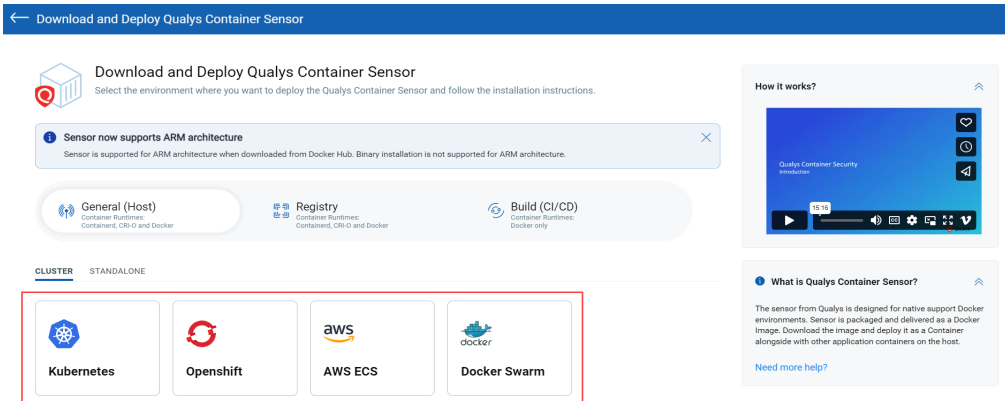
To deploy on a standalone host, pick the host's operating system: MacOS, Linux or CoreOS.



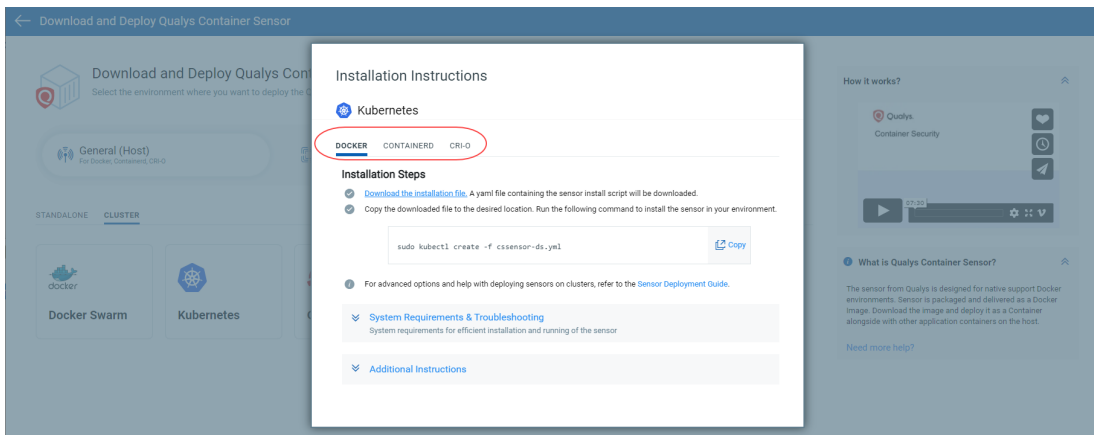
In the window that appears, choose DOCKERHUB or BINARY(TAR.XZ) for how you want to install the sensor. Then simply follow the steps on the screen. For Tar, you'll download the tar file and run the install commands on the screen. For Docker Hub, you'll run the docker commands on the screen.



To deploy to a cluster, first pick from the cluster options: Kubernetes, Openshift, AWS ECS, or Docker Swarm.



In the window that appears, choose the runtime. In the example below for General sensor being deployed in Kubernetes, you'll see DOCKER, CONTAINERD and CRI-O runtime options. After making your selection, follow the steps on the screen. The installation yaml file will already be pre-filled with your Activation ID, Customer ID and POD URL.



Be sure to note the System Requirements for installing the sensor. The sensor needs a minimum of 1 GB persistent storage on the host.

## Installsensor.sh script command line parameters

Here’s a quick overview of the “installsensor.sh” script command line parameters.

**Note:** Only a few of the parameters have default values. Default values can be changed during sensor installation. However, the default values (e.g., LogLevel) once set may get overridden by a config update. If you want to change any default value post sensor installation, you must rerun the “installsensor.sh” script with new values.

Parameter	Description
ActivationId	(Required) The Activation Id for the container sensor, auto-generated based on your subscription.
CustomerId	(Required) The Qualys subscription’s customer Id, auto-generated based on your subscription.
--cicd-deployed-sensor or -c	(Optional) Run the sensor in a CI/CD environment. This allows you to scan images on CI/CD pipeline (Jenkins, Bamboo).
--registry-sensor or -r	(Optional) Run the sensor to list and scan registry assets. This allows you to scan images in a public or private registry.
Storage	(Optional) Directory where the sensor would store the files.  Default value: /usr/local/qualys/sensor/data.  Create this directory if not already available or specify a custom directory location.
--sensor-without-persistent-storage	(Optional) Use this option to run the sensor without using persistent storage on the host.  Note: The sensor should be run either with the “--sensor-without-persistent-storage” option OR with the “--read-only” option and not with both options enabled together.  To install the sensor without persistent storage, exclude the “Storage” option, and include the “--sensor-without-persistent-storage” option in the installer script. We recommend you use the “--enable-console-logs” option along with “--sensor-without-persistent-storage” to preserve the logs as data is not available on host but stored at the /usr/local/qualys/qpq/data folder relative to the Sensor.  As the sensor is running with “--sensor-without-persistent-storage”, upon auto-update the updated sensor is a completely new instance of sensor container hence data from the old sensor is not available in the new sensor. Thus, the new sensor rescans existing already scanned assets.  If the sensor is installed without any persistent storage, the Container Summary page may not display any sensor details, and instead, it may show the error "There is no sensor activity recorded".

Parameter	Description
<code>--read-only</code>	(Optional) Use this option to run the sensor in read-only mode. In this mode the sensor uses persistent storage on the host.  Note: The sensor should be run either with the “ <code>--sensor-without-persistent-storage</code> ” option OR with the “ <code>--read-only</code> ” option and not with both options enabled together.
<code>ConcurrentScan</code>	(Optional) Number of docker/registry asset scans to run in parallel. A valid range is between 1-20. Default value is 4.
<code>CpuShares</code>	(Optional) Define CPU shares for the sensor container. A valid value is a non-zero, positive integer other than 1024.
<code>CpuUsageLimit</code>	(Optional) CPU usage limit in percentage for sensor. A valid range is between 0-100. Default is 0.2, i.e. 20% per core on the host.  The <code>installsensor</code> script has intelligence to find the number of CPU cores present on the host and apply the CPU limit based on the <code>CpuUsageLimit</code> input value and number of CPU cores available. For example, when <code>CpuUsageLimit=30</code> , it's considered as 30% CPU of overall CPU capacity of the host. If the host has 8 CPU cores, the total CPU limit applied to sensor container would be $0.30 * 8 = 2.4$ CPU cores.
<code>--disable-auto-update</code>	(Optional) Do not let sensor update itself automatically.
<code>--disableImageScan</code>	(Optional) This parameter should be passed if you want to disable image scans for General Sensor. Images will not be scanned by sensors deployed with this option. This is available for General sensor type only, and is available for all Runtimes (Docker, CRI-O and Containerd).
<code>ScanningPolicy</code>	(Optional) Specifies the scanning policy, which allows you to select the suitable scan type as per your requirement. The available values are: - <code>DynamicScanningOnly</code> : performs only dynamic scanning. - <code>StaticScanningOnly</code> : performs only static scanning. - <code>DynamicWithStaticScanningAsFallback</code> : performs static scanning as a fallback to dynamic scanning for images without shell.
<code>--disable-log4j-scanning</code>	(Optional) This parameter should be passed if you want to disable log4j vulnerability scanning for container images. See <a href="#">Log4j vulnerability scanning</a> .
<code>--disable-log4j-static-detection</code>	(Optional) This parameter should be passed if you want to disable log4j static detection for dynamic/static image scans. See <a href="#">Static log4j detection</a> .
<code>DockerHost</code>	(Optional) The address on which the docker daemon is configured to listen. This option is mandatory if <code>DOCKER_TLS_VERIFY=1</code> is defined.  DockerHost format: <Docker daemon host's IPv4 address, or FQDN, or hostname>:<port#>

Parameter	Description
DockerSocketDirectory	(Optional) Docker socket directory path. The default value is Default: /var/run
DOCKER_TLS_VERIFY	(Optional) This parameter enables the TLS authentication. The value should be 0 or 1.  Note: If DOCKER_TLS_VERIFY=1 is defined, then ensure that the provided IPv4 address or FQDN or hostname in DockerHost matches either the CN or the Alternative Subject Name in the docker server certificate.  Note: By enabling sensor communication with docker daemon over TLS, customer can restrict the sensor's access to docker socket by using docker authorization plugin.
TLS_CERT_PATH	(Optional) Provide client certificate directory path. This is mandatory if DOCKER_TLS_VERIFY=1 is defined.  tlscert=<Name of CA (default "ca.pem")> tlskey=<Name of TLS certificate file (default "cert.pem")> tlskey=<Name of TLS key file (default "key.pem")>  Note: If any of the CA certificate, client certificate, or client private key have default file names such as ca.pem, cert.pem, key.pem respectively they can be omitted.
--enable-console-logs	(Optional) Print logs on console. These logs can be retrieved using the docker logs command.
HostIdSearchDir	(Optional) Directory to map the marker file created by Qualys Agent or Scanner appliance on the host, update if modified. Default value is /etc/qualys
ImageFile	(Optional) Location of the Sensor ImageFile. This defaults to the local directory.
LogFilePurgeCount	(Optional) Integer value that specifies the maximum number of archived log files. Default value is 5.
LogFileSize	(Optional) Configuration to set the maximum size per log file for sensor in bytes. Accepts "<digit><K/M/>" where K is kilobytes and M is megabytes. For example, specify "10" for 10 bytes, "10K" for 10 kilobytes, "10M" for 10 megabytes. Default value is "10M".
LogLevel	(Optional) Configuration to set the logging level for sensor, accepts 0 to 5. Default value is 3 (Information).
--mask-env-variable	(Optional) Use this parameter to mask environment variables for images and containers. The environment variables will be masked/removed in sensor logs and in the Container Security UI.

Parameter	Description
MemoryUsageLimit	(Optional) Define the memory usage limit for the sensor container. The value should be formatted as <digit><unit> where unit can be any of the following: b (bytes), k (kilobytes), m (megabytes), g (gigabytes). The recommended value is 500m for 500 megabytes.
--optimize-image-scans	<p>(Optional) This parameter should be passed if you want to optimize Image scans for General Sensor. This is available for General sensor type only.</p> <p>By default, the sensor scans every image that it detects on the host. This results in redundant scanning of images. When you install the General sensor with "--optimize-image-scans", the sensor will communicate with the Qualys Cloud Platform and perform informed scans to avoid redundant image scans. The sensor will determine if the images present on the host are already scanned by other sensors for the same manifest and version and will not scan those images again.</p>
PidLimit	(Optional) Define the PID limit for the sensor container. The value provided must be a positive integer.
Proxy	(Optional) IPv4/IPv6 address or FQDN of the proxy server.
ProxyCertFile	<p>(Optional) Proxy certificate file path. ProxyCertFile is applicable only if Proxy has valid certificate file. If this option is not provided, then Sensor will try to connect to the server with given https Proxy settings only.</p> <p>If only ProxyCertFile is provided without Proxy then Sensor would simply ignore the ProxyCertFile and it would try to connect to the server without any https proxy settings.</p>
--silent or -s	(Optional) Run installsensor.sh in non-interactive mode.
--perform-secret-detection	<p>(Optional) Use this parameter to enable secret detection for container images. You can specify a timeout for this command using the SCAScanTimeoutInSeconds={value} parameter.</p> <p><b>Note:</b> Secret detection is supported only on:</p> <ul style="list-style-type: none"><li>- CICD and registry sensors</li><li>- Linux operating system</li><li>- Docker, Containerd, and CRI-O runtimes</li></ul>
--limit-resource-usage	(Optional) Use this parameter to limit usage of resources for SCA/Secret/Malware Scan



## Optional parameters for SCA scanning

The following parameters are optional when the SCA scanning feature is enabled for your subscription. See [SCA scanning](#) to learn more.

Parameter	Description
<code>--perform-sca-scan</code>	(Optional when SCA scanning is enabled for your subscription) By default, SCA scanning is not performed. Use this parameter to enable SCA scanning for container images. When specified, the SCA scan will be performed after a standard vulnerability scan (Static or Dynamic). The SCA scan is attempted even when the vulnerability scan is not successful.
<code>--disallow-internet-access-for-sca</code>	(Optional when <code>--perform-sca-scan</code> is specified) By default, Internet access is enabled for the SCA scan and the SCA scan is performed in online mode because in online mode, the package collection is more accurate than compared to scans performed in offline mode. Use this parameter to disable Internet access for the SCA scan and run the scan in offline mode instead.  <b>Note:</b> We recommend you run the SCA scan in online mode. Quality of software package enumeration for Java substantially degrades when the SCA scan is run in offline mode. The remote maven repository may need to be consulted for an accurate package detection. This can affect accuracy of the vulnerability posture of the image. The sensor must be able to reach the URL "http://search.maven.org" and "https://ghcr.io".
<code>SCAScanTimeoutInSeconds={value}</code>	(Optional when <code>--perform-sca-scan</code> is specified) The default SCA scan command timeout is 5 minutes (300 seconds). Use this parameter to overwrite the default timeout with a new value specified in seconds. For example, you may need to increase the SCA scan timeout when scanning large container images to ensure the SCA scan has time to finish.  <b>Note:</b> This parameter is also used for specifying a timeout for secret detection.
<code>--limit-resource-usage</code>	(Optional) Use this parameter to limit usage of resources for SCA/Secret/Malware Scan

## Docker Hub

For information on installing the sensor from Docker Hub, see:

[Installing the sensor from Docker Hub](#)

## CI/CD Environments

For information on deploying the sensor in CI/CD environments, refer to:

[Qualys Container Scanning Connector for Jenkins](#)

[Qualys Container Scanning Connector for Bamboo](#)

[Qualys Container Scanning Connector for Azure DevOps](#)

**Note:** Your hosts must be able to reach your Qualys Cloud Platform (or the Qualys Private Cloud Platform) over HTTPS port 443. See [Qualys Platform \(POD URL\) your hosts need to access](#).

## How to Comply with CIS Benchmark for Docker using Installsensor.sh Commands

Qualys Container Security adheres to the CIS Benchmark for Docker for our Sensor image. Refer to [Compliance with CIS Benchmark for Docker](#) for guidance on how to use the Sensor image in a way that complies with the CIS Benchmark for Docker. We've provided instructions for a number of controls so you can operate the Sensor in a compliant manner.

## Proxy Support

The install script asks for proxy configuration. You need to provide the IP Address/FQDN and port number along with the proxy certificate file path. For example:

```
Do you want connection via Proxy [y/N]: y
Enter Https Proxy settings [<IP Address>:<Port #>]: 10.xxx.xx.xx:3xxx
Enter Https Proxy certificate file path: /etc/qualys/cloud-agent/cert/ca-bundle.crt
```

Your proxy server must provide access to the Qualys Cloud Platform (or the Qualys Private Cloud Platform) over HTTPS port 443. See [Qualys Platform \(POD URL\) your hosts need to access](#).

## Qualys Platform (POD URL) your hosts need to access

The Qualys URL you use depends on the Qualys platform where your account is located.

[Click here to identify your Qualys platform and get the Container Security Server URL](#)

### POD URL value

The “Container Security Server URL” for your platform is the URL you’ll need to provide for the POD\_URL variable in Container Security Sensor commands and in configuration yaml files when deploying the sensor.

## Sensor network configuration

The sensor is pre-configured with the Qualys URL and subscription details it needs to communicate to Qualys. In order for the sensor to communicate to Qualys, the network configuration and firewall need to provide accessibility to Qualys domain over port 443.

After successful installation of the sensor, the sensor is listed in the Container Security UI under Configurations > Sensors where you can see its version, status, etc, and access details. Additionally, you can Download the sensor from the link under Configurations > Sensors.

## Static scanning of container images

Static scanning is supported for deployments on standalone Docker hosts and deployments in Kubernetes with Docker Runtime and Containerd Runtime.

The sensor will perform static scanning for container images as a fallback mechanism to current dynamic scanning in case container image does not have a shell. Static scanning will also be performed for Google distroless images without shell. Static scanning will not be performed on container or container images having a shell.

Static scanning collects the list of installed software from the container image file system to find vulnerabilities in the container images. The installed software list is retrieved from the Package manager metadata files. Package managers supported are RPM, DPKG and Alpine.

If you have large images without shell on the host where sensor is running, the requirement for disk space may exceed the minimum requirement of 1GB.

## Log4j vulnerability scanning

The sensor can detect Log4j Remote Code Execution (RCE) Vulnerability QIDs. It can detect the presence of vulnerable log4j packages on your container images and running containers. The sensor will automatically perform a file system search to detect log4j vulnerabilities on your container images, and this can have a performance impact. To disable log4j vulnerability scanning, specify `--disable-log4j-scanning` as a command line parameter for “`installsensor.sh`” script or provide it as a command or args parameter when deploying a sensor.

## Static log4j detection

Static log4j detection is enabled by default for dynamic/static image scans. Static log4j detection is implemented by executing the log4j detection command for each image layer and then merging the results. You have the option to disable static log4j detection for dynamic/static image scans using the parameter `--disable-log4j-static-detection`.

For static scans, static log4j detection will always be invoked unless the `--disable-log4j-static-detection` parameter is specified. The log4j commands are read from the VM manifest.

For dynamic scans, static log4j detection will only be invoked when the following is true:

- The parameter `--disable-log4j-scanning` is not used.
- The parameter `--disable-log4j-static-detection` is not used.
- The primary log4j detection command was unsuccessful in collecting log4j data points.

To disable the static log4j detection, specify `--disable-log4j-static-detection` as a command line parameter for “`installsensor.sh`” script or provide it as a command or args parameter when deploying a sensor.

## SCA scanning

Qualys Container Security Sensor supports Software Composition Analysis (SCA) scanning of container images. An SCA scan discovers installed open source software and libraries, as well as associated vulnerabilities, present in your container images.

While evaluating security posture of container images it is important to identify all software packages present in the image. The SCA scan can be used to identify programming language-based software packages inside the image. In addition, metadata information for each image layer is also provided. The SCA scan detects packages for these programming languages: Java, Python, Go, Node.js, .NET, PHP, Ruby, and Rust.

SCA scanning is supported for all sensor types – General, Registry and CI/CD. It's supported for Docker, ContainerD, and CRI-O runtimes. SCA scanning is only supported when scanning container images. SCA scanning is not supported for Mac OS.

**Note:** For CRI-O runtime to support SCA, it is required to launch the sensor with privilege rights. To do that, in the 'cssensor-crio-ds.yml' file, the following parameter must be set to true.

```
securityContext:  
  privileged: true
```

### Prerequisites

- The SCA Scanning feature must be enabled for your subscription. Contact Qualys Support to have this feature enabled.
- Sensor version 1.19 or later.
- Relaunch your sensors with the parameter `--perform-sca-scan` to perform SCA scanning.
- Additional storage on the host to store SCA scan metadata. Refer to [Storage Requirements for Sensor Scans](#).

### How it Works

SCA scanning is not performed by default. Users must enable SCA scanning using the parameter `--perform-sca-scan`. When enabled, an SCA scan is performed after a standard vulnerability scan (Static or Dynamic) on your container images. When the SCA scan completes, the sensor uploads the metadata information collected by the scan to the Qualys backend where posture evaluation is performed. You can view SCA scan data findings in the Container Security UI and API as part of image details. Vulnerability detections found by the SCA scan are presented as QIDs. Filters are provided so you can identify the type of scan (SCA, Dynamic or Static) used to detect a particular vulnerability.

Internet access is enabled for the SCA scan and the SCA scan is performed in online mode by default. Make sure the sensor can reach the URL "http://search.maven.org" and "https://ghcr.io".

During an SCA scan, the following files are scanned for the language-specific software packages:

Language	Files
Python	egg package wheel package
Node.js	package.json
.NET	packages.lock.json packages.config *.deps.json
Java	JAR/WAR/PAR/EAR
Go	Binaries built by Go
PHP	Composer.lock
Ruby	gemspec
Rust	Cargo.lock and Binaries built with cargo-auditable

## Secrets Detection

Container secrets are digital credentials providing identity authentication and authorizing access to privileged accounts, applications, and services. They can include passwords, API keys, and other credentials that are needed for applications to function properly.

If these secrets are not properly secured, they can be accessed by unauthorized users, leading to malicious attacks. Therefore, discovering secrets is one of the important aspects of container security that organizations must prioritize to protect their sensitive data, meet compliance requirements, and reduce the risk of security incidents.

Container Security Sensor can detect secrets for container images enabling you to mitigate potential security risks associated with the accidental or intentional exposure of secrets within containers.

To enable secret detection, you need to use the `--perform-secret-detection` parameter.

Secret detection involves scanning the filesystem. It does not detect secrets that are stored as environment variables or passed as arguments within the image. Therefore, the performance of secret detection depends on the number of files present in the image.

For optimal performance in secret detection, it is recommended to allocate a higher CPU count to the sensor container. Ensure that at least two CPUs of the host are specifically utilized for the sensor container.

For instance:

- When using the `InstallSensor.sh` script, by default only 20% of the host's CPUs are utilized by the sensor container.
- When using `dockerrun`, by default all CPUs of the host are fully utilized for the sensor container

**Note:** Secret detection is supported only on:

- Sensors: CICD and registry
- OS: Linux
- Runtimes: Docker, Containerd, and CRI-O

For more information about secret detection, see Online Help: Detect Container Secrets.

## Malware Detection

You can scan your container images for the presence of malware or any malicious files.

Malware detection ensures that malicious container images are not deployed into the production environment. This prevents potential breaches, data theft, or unauthorized access that could result from running malicious containers.

To enable malware detection, you need to use the `--perform-malware-detection` parameter while installing the sensor.

For efficient malware scanning, it is recommended to allocate 1 CPU core for the sensor. For instance:

- When using the `InstallSensor.sh` script, by default 20% of the host's CPUs are utilized by the sensor container. If the host has 8 CPU cores, the total CPU limit applied to the sensor container would be  $0.2 * 8 = 1.6$  CPU cores.
- When using `dockerrun`, by default all CPUs of the host are fully utilized for the sensor container.
- In Kubernetes, to allocate 1 CPU core for the sensor container, regardless of the number of cores available on the host system, set the CPU limit value to 1.

```
Example:
resources:
  limits:
    cpu: "1"
```

**Note:** Malware detection is supported only on:

- Sensors: Registry sensor (x86\_64 architecture only)
- OS: Linux
- Runtimes: Docker, Containerd, and CRI-O

For more information about malware detection, see Online Help: Malware Detection.

## Events that lead to Docker asset scanning

A new asset scan is launched when any of the following events occur.

- Events on images: load, pull, import, tag
- Events on containers: start, create, unpause
- Scan is launched when there's a new manifest.
- Scan is launched every 48 hours on containers (i.e. 48 hours after the last successful scan).

## Storage Requirements for Sensor Scans

If you want to enable all types of scans, Qualys recommends the following server requirements.

- CPU Cores: 6
- RAM: 5GB

See the sections below to understand storage requirements for different scan types.

### Dynamic Scan

Applicable to Registry Sensor only.

The Registry sensor pulls the Docker image on the host for scanning. Storage required on the partition where Docker is installed is based on the size of the image. The dynamic scan is performed on the cached image.

For an average image size of 4GB, the maximum storage requirement would be 16GB:

$4\text{GB image} * 4 \text{ scan threads} = 16\text{GB}$

### Static Scan

Applicable to General (Host) Sensor, Registry Sensor and Build (CI/CD) Sensor.

Additional storage is required on persistent storage to scan the image if the image does not have a shell. The static scan is performed on the container image. The storage requirement is approximately 3 times the size of the image.

For an average image size of 4GB where 4 scan threads are performing the image scan on images with no shell, the maximum storage requirement would be 48GB:

$(4\text{GB image} * 3) * 4 \text{ scan threads} = 48\text{GB}$

### SCA Scan

Applicable to General (Host) Sensor, Registry Sensor and Build (CI/CD) Sensor.

When the CS Sensor is running with `--perform-sca-scan`, it will require additional storage on the host to accommodate the image tar, which is usually the size of the image plus 100MB additional disk space used to store SCA scan metadata. The storage required is the image size plus 100MB times the number of threads performing the Docker image scan.

For an average image size of 4GB where 4 scan threads are performing the image scan, the maximum storage requirement is approximately 16.4GB:



$(4\text{GB image} + 100\text{MB}) * 4 \text{ scan threads} = 16.4\text{GB}$

## Static Log4j Detection

If static detection is triggered for images having shell, then additional space is required. This needs 3 times the size of the image.

For an average image size of 4GB, the additional storage requirement would be 12GB:

$4\text{GB image} * 3 = 12\text{GB}$

# Installing the sensor on MacOS

**Note:** If you are running the sensor on MacOS Catalina version 10.15 or above, please use sensor version 1.8.0 or above.

You can install the Qualys Container Sensor on MacOS. Download the QualysContainerSensor.tar.xz file using the “Download Container Sensor” button on the Home page or from the Configurations > Sensors tab on Qualys Cloud Platform.

Copy the file to the target MAC host. Then run the following commands in sequence.

This command extracts the tar file:

```
sudo tar -xvf QualysContainerSensor.tar.xz
```

This command creates the directory where the sensor data like configuration, manifest, logs, and setup is stored:

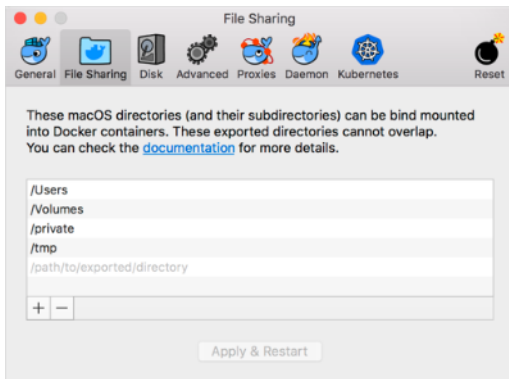
```
sudo mkdir -p /tmp/qualys/sensor/data
```

This command provides required permissions to the directory to run the installer script:

```
sudo chmod -R 777 /tmp/qualys/sensor/data
```

If you want to specify a custom location for storage, ensure that the Docker's File Sharing is enabled for the same. On your MAC host, go to Docker > Preferences > File Sharing, add the custom path e.g. /usr/local/qualys/sensor/data, then click Apply & Restart.

Enabling file sharing is required only if the custom location is NOT from /Users, /Volumes, /private or /tmp.



To avoid this step, we recommend using Storage=/tmp/qualys/sensor/data and HostIdSearchDir=/private/etc/qualys during sensor install. That way you can leverage the existing shared location with docker, without the need of additional configuration to launch the CS Sensor. If you are using a custom location, provide permissions to the directory to run the installer script. For example:

```
sudo chmod -R 777 /usr/local/qualys/sensor/data
```

Make sure CS Sensor has permissions to write to `hostid` file in `/private/etc/qualys/` directory. To provide the sufficient permissions, execute:

```
sudo mkdir /private/etc/qualys/
sudo chmod 777 /private/etc/qualys
```

The following commands install the sensor. Notice that the command includes the Activation ID and your Customer ID, both generated based on your subscription. The Storage parameter specifies where to install the sensor. Ensure that the `HostIdSearchDir` exists, otherwise the installer script will throw an error.

Use the following command to install a General Sensor:

```
sudo ./installsensor.sh ActivationId=d5814d5f-5fd2-44ec-8969-
e03cc58a4ef5 CustomerId=6f35826e-4430-d75e-8356-c444a0abbb31
HostIdSearchDir=/private/etc/qualys Storage=/tmp/qualys/sensor/data -s
```

Use the following command to install a Registry Sensor:

```
sudo ./installsensor.sh ActivationId=d5814d5f-5fd2-44ec-8969-
e03cc58a4ef5 CustomerId=6f35826e-4430-d75e-8356-c444a0abbb31
HostIdSearchDir=/private/etc/qualys Storage=/tmp/qualys/sensor/data -s -
-registry-sensor
```

Use the following command to install a CI/CD Sensor:

```
sudo ./installsensor.sh ActivationId=d5814d5f-5fd2-44ec-8969-
e03cc58a4ef5 CustomerId=6f35826e-4430-d75e-8356-c444a0abbb31
HostIdSearchDir=/private/etc/qualys Storage=/tmp/qualys/sensor/data -s -
-cicd-deployed-sensor
```

### Notes:

- If you want to install the Sensor without persistent storage, exclude the “Storage” option, and include the “--sensor-without-persistent-storage” option in the installer script. It is recommended to use the “--enable-console-logs” option along with “--sensor-without-persistent-storage” to preserve the logs as data is not available on host but stored at the `/usr/local/qualys/qpa/data` folder relative to the Sensor.
- Secrets and malware detection is not supported on MacOS.

## Installing the sensor on Linux

You can install the Qualys Container Sensor on Linux. Download the QualysContainerSensor.tar.xz file using the “Download Container Sensor” button on the Home page or from the Configurations > Sensors tab on Qualys Cloud Platform.

Copy the file to the target host. Then run the following commands in sequence.

This command extracts the tar file:

```
sudo tar -xvf QualysContainerSensor.tar.xz
```

This command creates the directory where the sensor data like configuration, manifest, logs, and setup is stored:

```
sudo mkdir -p /usr/local/qualys/sensor/data
```

This command adds the required permissions to storage:

```
sudo chmod 777 /usr/local/qualys/sensor/data
```

The following commands install the sensor. Notice that the command includes the Activation ID and your Customer ID, both generated based on your subscription. The Storage parameter specifies where to install the sensor.

Use the following command to install a General Sensor:

```
sudo ./installsensor.sh ActivationId=d5814d5f-5fd2-44ec-8969-
e03cc58a4ef5 CustomerId=6f35826e-4430-d75e-8356-c444a0abbb31
Storage=/usr/local/qualys/sensor/data -s
```

Use the following command to install a Registry Sensor:

```
sudo ./installsensor.sh ActivationId=d5814d5f-5fd2-44ec-8969-
e03cc58a4ef5 CustomerId=6f35826e-4430-d75e-8356-c444a0abbb31
Storage=/usr/local/qualys/sensor/data -s -r
```

Use the following command to install a CI/CD Sensor:

```
sudo ./installsensor.sh ActivationId=d5814d5f-5fd2-44ec-8969-
e03cc58a4ef5 CustomerId=6f35826e-4430-d75e-8356-c444a0abbb31
Storage=/usr/local/qualys/sensor/data -s -c
```

**Note:** To install the Sensor without persistent storage, exclude the “Storage” option, and include the “--sensor-without-persistent-storage” option in the installer script. It is recommended to use the “--enable-console-logs” option along with “--sensor-without-persistent-storage” to preserve the logs as data is not available on host but stored at the /usr/local/qualys/qpa/data folder relative to the Sensor.

# Installing the sensor on CoreOS

You can install the Qualys Container Sensor on CoreOS. Download the `QualysContainerSensor.tar.xz` file using the “Download Container Sensor” button on the Home page or from the Configurations > Sensors tab on Qualys Cloud Platform.

Copy the file to the target host. Then run the following commands in sequence.

This command extracts the tar file:

```
sudo tar -xvf QualysContainerSensor.tar.xz
```

This command creates the directory where the sensor data like configuration, manifest, logs, and setup is stored:

```
sudo mkdir -p /var/opt/qualys/sensor/data
```

**Note:** You need to set the directory path `/var/opt/qualys/sensor/data` to Storage which is writable on CoreOS.

This command provides required permissions to the directory to run the installer script:

```
sudo chmod -R 777 /var/opt/qualys/sensor/data
```

The following commands install the sensor. Notice that the command includes the Activation ID and your Customer ID, both generated based on your subscription. The Storage parameter specifies where to install the sensor.

Use the following command to install a General Sensor:

```
Sudo ./installsensor.sh ActivationId=d5814d5f-5fd2-44ec-8969-
e03cc58a4ef5 CustomerId=6f35826e-4430-d75e-8356-c444a0abbb31
Storage=/var/opt/qualys/sensor/data/ -s
```

Use the following command to install a Registry Sensor:

```
Sudo ./installsensor.sh ActivationId=d5814d5f-5fd2-44ec-8969-
e03cc58a4ef5 CustomerId=6f35826e-4430-d75e-8356-c444a0abbb31
Storage=/var/opt/qualys/sensor/data/ -s --registry-sensor
```

Use the following command to install a CI/CD Sensor:

```
Sudo ./installsensor.sh ActivationId=d5814d5f-5fd2-44ec-8969-
e03cc58a4ef5 CustomerId=6f35826e-4430-d75e-8356-c444a0abbb31
Storage=/var/opt/qualys/sensor/data/ -s --cicd-deployed-sensor
```

**Note:** To install the Sensor without persistent storage, exclude the “Storage” option, and include the “`--sensor-without-persistent-storage`” option in the installer script. It is recommended to use the “`--enable-console-logs`” option along with “`--sensor-without-persistent-storage`” to preserve the logs as data is not available on host but stored at the `/usr/local/qualys/qpa/data` folder relative to the Sensor.

## Installing the sensor from Docker Hub

This section provides information on deploying the sensor through Docker Hub. Please note that the Docker Hub sensor image is not supported for customers on a Private Cloud Platform (PCP).

On standalone docker host:

[Deploying the sensor on standalone docker host using docker compose](#)

[Deploying the sensor on standalone docker host using docker run](#)

On Kubernetes:

[Deploying the sensor using Docker Hub on Kubernetes](#)

The Container Security Sensor on Docker Hub is available as:

```
qualys/qcs-sensor:<tag>
```

```
qualys/qcs-sensor:latest
```

Look up the most recent tag in Docker Hub.

## Deploying the sensor on standalone docker host using docker compose

Prerequisites:

- Docker engine version: 1.13.0+
- Docker-compose file format version: 2.2
- Docker host should be able to communicate with the Docker Hub

Create a new yml file containing the following information. You can name the file **qualys\_cs\_sensor\_docker\_compose.yml**.

**Note:** The field alignment in the yml file is very important. Please make sure to honor the formatting provided in the below template.

```
version: '2.2'
services:
  cs_sensor:
    container_name: qualys-container-sensor
    image: qualys/qcs-sensor:latest
    restart: on-failure
# Uncomment the below security option if SELinux is enabled with
# enforcing mode on docker host
#   security_opt:
#     - label:disable

# Enable the flag if you want to launch CS sensor in read-only mode.
```

```
#   read_only: true
network_mode: host
cpus: 0.2
command: ["--scan-thread-pool-size", "4"]
environment:
  - ACTIVATIONID=<Activation id>
  - CUSTOMERID=<Customer id>
  - POD_URL=<POD URL>
# Define TCP socket if sensor will be communicating with docker daemon
listening on TCP port
#   - DOCKER_HOST=<IPv4 address or FQDN>:<port#>
# Enable TLS authentication if sensor will be communicating with docker
daemon over TCP TLS socket
#   - DOCKER_TLS_VERIFY=1
# Define the proxy if required
#   - qualys_https_proxy=<IP address or FQDN>:<Port#>

volumes:
# Provide host Id search directory path
  - /etc/qualys:/usr/local/qualys/qpa/data/conf/agent-data
# Mount volume for docker socket
  - /var/run/docker.sock:/var/run/docker.sock:ro
# Mount volume for persistent storage
  - /usr/local/qualys/sensor/data:/usr/local/qualys/qpa/data
# Mount volume proxy certificate if required
#   - <Proxy certificate path on host>:/etc/qualys/qpa/cert/custom-
ca.crt
# Mount volume for docker client cert directory path
#   - <Client certificate directory on the docker host>:/root/.docker
```

## Parameters used in the yml file

### **container\_name**

set to qualys-container-sensor

### **image\_name**

set to qualys/qcs-sensor:<tag>

OR

set to qualys/qcs-sensor:latest

The image will get pulled from the Docker Hub by docker-compose.

### **restart**

Defines the sensor restart policy and should be set to on-failure.

### security\_opt

This parameter should be used only when SELinux is enabled with enforcing mode on the docker host.

```
security_opt:  
  - label:disable
```

### read-only

Set to true when launching the sensor in read-only mode.

### network\_mode

Set to host specifying that the sensor needs to be launched with host's network stack.

### cpus

Restrict the cpu usage to a certain value.

cpus: 0.2 # Default CPU usage limit (20% of one core/processor on the host).

For example, for limiting the cpu usage to 5%, set cpus: 0.05. This limits the cpu usage to 5% of one core/processor on the host.

If there are multiple processors on a node, setting the cpus value applies the CPU limit to one core/processor only. For example, if you have 4 CPUs on the system and you want to set CPU limit as 20% of overall CPU capacity, then the CPU limit should be set to 0.8 i.e., 80% of one core only which becomes 20% of total CPU capacity.

To disable any CPU usage limit, set cpus value to 0 or remove/comment it out.

**Note:** If docker host's kernel does not support setting the CPU limit on running containers, disable CPU usage limit, otherwise the sensor won't get launched.

### command

If you want to deploy the sensor for CI/CD environment provide the command value as:

```
command: ["--cicd-deployed-sensor"]
```

If you want to deploy a Registry Sensor provide the command value as:

```
command: ["--registry-sensor"]
```

**Note:** The General Sensor gets installed by default if the parameters for Registry or CI/CD are not provided.

Additional values you can provide in the command parameter:

"--enable-console-logs" to print logs on console. These logs can be retrieved using the docker logs command.

"--log-level" to set the logging level for sensor, accepts 0 to 5. Default is 3 (Information).

"--log-filesize" to set the maximum size per log file for sensor in bytes. Accepts "<digit><K/M/>" where K is kilobytes and M is megabytes. For example, specify "10" for 10 bytes, "10K" for 10 kilobytes, "10M" for 10 megabytes. Default is "10M".



"--log-filepurgecount" to define the number of archived qpa.log files to be generated. Default is 5.

"--scan-thread-pool-size" to launch the sensor with scan thread value. Default is 4.

"--sensor-without-persistent-storage" to run the sensor without using persistent storage on host. In this case do not provide persistent storage mapping under volumes. It is recommended to use the "--enable-console-logs" option along with "--sensor-without-persistent-storage" to preserve the logs as data is not available on host but stored at the /usr/local/qualys/qpa/data folder relative to the Sensor.

Example,

```
command: ["--cicd-deployed-sensor", "--sensor-without-persistent-storage", "--enable-console-logs"]
  volumes:
    # mount volume for persistent storage
    # -/usr/local/qualys/qpa/data
```

"--tls-cacert", "<file name of the CA certificate used to sign docker server certificate>", "--tls-cert", "<docker client certificate file name>", "--tls-key", "<docker client private key file name>" if the sensor will be communicating with the docker daemon over TLS. If any of the three files have a default name such as ca.pem, cert.pem, key.pem respectively the corresponding argument can be omitted.

"--mask-env-variable" to mask environment variables for images and containers. The environment variables will be masked/removed in sensor logs and in the Container Security UI.

"--disableImageScan" to disable image scans for General Sensor. Images will not be scanned by sensors deployed with this option. This is available for General sensor type only, and is available for all Runtimes (Docker, CRI-O and Containerd).

"--disable-log4j-scanning" to disable log4j vulnerability scanning for container images. See [Log4j vulnerability scanning](#).

"--disable-log4j-static-detection" to disable log4j static detection for dynamic/static image scans. See [Static log4j detection](#).

"--optimize-image-scans" to optimize image scans for General Sensor. By default, the sensor scans every image that it detects on the host. This results in redundant scanning of images. When you install the General sensor with "--optimize-image-scans", the sensor will communicate with the Qualys Cloud Platform and perform informed scans to avoid redundant image scans. The sensor will determine if the images present on the host are already scanned by other sensors for the same manifest and version and will not scan those images again.

"--perform-secret-detection" to enable secrets detection for container images. Note that secret detection is supported only on CICD and registry sensors.

"--perform-malware-detection" to enable malware detection for container images. Note that malware detection is supported only on registry sensor.

## environment

Provide the ACTIVATIONID, CUSTOMERID, and POD\_URL from your subscription. To get the Activation ID and Customer ID, login to the Container Security UI, go to Configurations > Sensors, click Download, and then click any sensor type. The installation command on the Installation Instructions screen contains your Activation ID and Customer ID. Activation ID is like a password, do not share it.

**Note:** Your hosts must be able to reach your Qualys Cloud Platform (or the Qualys Private Cloud Platform) over HTTPS port 443. See [Qualys Platform \(POD URL\) your hosts need to access](#).

Specify DOCKER\_HOST if sensor will be communicating with docker daemon listening on TCP port either with or without TLS enabled.

```
DOCKER_HOST=<IPv4 address, or FQDN, or hostname>:<Port#>
```

If TLS is enabled for the TCP socket specified please make sure that the provided IP, FQDN or hostname matches either the CN or Alternative Subject Name in the docker server certificate.

If sensor is listening on TCP socket without TLS do not provide unix domain socket directory mapping. Under 'volumes' comment out the following part:

```
volumes:  
# mount volume for docker socket  
# - /var/run/docker.sock:/var/run/docker.sock:ro
```

Specify DOCKER\_TLS\_VERIFY=1 to enable TLS authentication.

**Note:** By enabling sensor communication with docker daemon over TLS customer can restrict the sensor's access to docker socket by using docker authorization plugin.

Specify qualys\_https\_proxy if a proxy is required for the sensor to communicate with the Qualys Cloud Platform.

```
- qualys_https_proxy=<IP/ address or FQDN>:<Port#>
```

## volumes

Specify the persistent storage mapping to launch the sensor with persistent storage. The persistent storage directory is automatically created if doesn't exist.

```
volumes:  
# mount volume for persistent storage  
-/usr/local/qualys/sensor/data:/usr/local/qualys/qpq/data
```

Specify hostid directory location if you want to use the same hostid used in the previous installation.

```
# Provide host Id search directory path  
- /etc/qualys:/usr/local/qualys/qpq/data/conf/agent-data
```

Map the Unix socket file to sensor file system if the Docker daemon on the Docker host is communicating over Unix socket.

```
# mount volume for docker socket
- /var/run/docker.sock:/var/run/docker.sock:ro
```

**Note:** If the Docker daemon is communicating over TCP port specify the DOCKER\_HOST parameter under environment and DO NOT provide mapping for docker unix socket file under volumes.

Specify the proxy certificate (if required):

```
- <Proxy certificate path on host>:/etc/qualys/qpa/cert/custom-ca.crt
```

Specify docker client certificate directory mapping if the sensor will be communicating with docker daemon over TLS:

```
# mount volume for docker client certificate directory
- <docker client certificate directory on the docker daemon
host>:/root/.docker
```

## Launching the sensor

Once the yml file is created, use the following command to launch the sensor:

```
docker-compose -f <path to qualys_cs_sensor_docker_compose.yml file> up
-d
```

## Upgrading the sensor

The Qualys Container Sensor image hosted on Docker Hub does not support auto update. Perform the following steps to update the sensor installed from Docker Hub:

1. Update the image name in the yml file:

Set to qualys/qcs-sensor:<tag>

OR

Set to qualys/qcs-sensor:latest

2. Run the command to recreate the sensor:

```
docker-compose -f <path to qualys_cs_sensor_docker_compose.yml file> up
-d
```

## Removing the sensor

Run the following command to remove the sensor:

```
docker-compose -f <path to qualys_cs_sensor_docker_compose.yml file> rm
-s
```

**Note:** The docker-compose does not provide an option to delete the persistent storage. You must delete the persistent storage files manually.

## Deploying the sensor on standalone docker host using docker run

Prerequisites: Docker engine version: 1.13.0+

Run the following commands to install the sensor. Provide the ACTIVATIONID, CUSTOMERID, and POD\_URL from your subscription. To get the Activation ID and Customer ID, login to the Container Security UI, go to Configurations > Sensors and click Download Sensor. Choose the sensor type (General, Registry, CI/CD) and then the Standalone technology: MacOS, Linux or CoreOS. The Installation Instructions page appears. Pick the DOCKERHUB tab to see installation steps.

The installation command on the Installation Instructions screen contains your Activation ID and Customer ID. Activation ID is like a password, do not share it.

Note: To meet compliance with CIS Benchmark 5.9, you must remove `--net=host` from the installation command. Please note, however, that when the sensor is launched without `--net=host`, it won't be able to detect its host IP address. Refer to [Compliance with CIS Benchmark for Docker](#) for guidance and recommendations.

### General Sensor

#### Linux:

```
sudo docker run -d --restart on-failure -v
/var/run/docker.sock:/var/run/docker.sock:ro -v
/usr/local/qualys/sensor/data:/usr/local/qualys/qa/data -e
ACTIVATIONID=<Activation id> -e CUSTOMERID=<Customer id> -e POD_URL=<POD
URL> --net=host --name qualys-container-sensor qualys/qcs-sensor:latest
```

#### MacOS:

```
sudo mkdir -p /tmp/qualys/sensor/data
sudo chmod -R 777 /tmp/qualys/sensor/data
sudo mkdir /private/etc/qualys/
sudo chmod 777 /private/etc/qualys

sudo docker run -d --restart on-failure -v
/var/run/docker.sock:/var/run/docker.sock:ro -v
/private/etc/qualys:/usr/local/qualys/qa/data/conf/agent-data -v
/tmp/qualys/sensor/data:/usr/local/qualys/qa/data -e
ACTIVATIONID=<Activation id> -e CUSTOMERID=<Customer id> -e POD_URL=<POD
URL> --net=host --name qualys-container-sensor qualys/qcs-sensor:latest
```

#### CoreOS:

```
sudo mkdir -p /var/opt/qualys/sensor/data
sudo chmod -R 777 /var/opt/qualys/sensor/data

sudo docker run -d --restart on-failure -v
/var/run/docker.sock:/var/run/docker.sock:ro -v
/var/opt/qualys/sensor/data:/usr/local/qualys/qa/data -e
```

```
ACTIVATIONID=<Activation id> -e CUSTOMERID=<Customer id> -e POD_URL=<POD URL> --net=host --name qualys-container-sensor qualys/qcs-sensor:latest
```

## Registry Sensor

### Linux:

```
sudo docker run -d --restart on-failure -v  
/var/run/docker.sock:/var/run/docker.sock:ro -v  
/usr/local/qualys/sensor/data:/usr/local/qualys/qpq/data -e  
ACTIVATIONID=<Activation id> -e CUSTOMERID=<Customer id> -e POD_URL=<POD URL>  
--net=host --name qualys-container-sensor qualys/qcs-sensor:latest  
--registry-sensor
```

### MacOS:

```
sudo mkdir -p /tmp/qualys/sensor/data  
sudo chmod -R 777 /tmp/qualys/sensor/data  
sudo mkdir /private/etc/qualys/  
sudo chmod 777 /private/etc/qualys
```

```
sudo docker run -d --restart on-failure -v  
/var/run/docker.sock:/var/run/docker.sock:ro -v  
/private/etc/qualys:/usr/local/qualys/qpq/data/conf/agent-data -v  
/tmp/qualys/sensor/data:/usr/local/qualys/qpq/data -e  
ACTIVATIONID=<Activation id> -e CUSTOMERID=<Customer id> -e POD_URL=<POD URL>  
--net=host --name qualys-container-sensor qualys/qcs-sensor:latest  
--registry-sensor
```

### CoreOS:

```
sudo mkdir -p /var/opt/qualys/sensor/data  
sudo chmod -R 777 /var/opt/qualys/sensor/data
```

```
sudo docker run -d --restart on-failure -v  
/var/run/docker.sock:/var/run/docker.sock:ro -v  
/var/opt/qualys/sensor/data:/usr/local/qualys/qpq/data -e  
ACTIVATIONID=<Activation id> -e CUSTOMERID=<Customer id> -e POD_URL=<POD URL>  
--net=host --name qualys-container-sensor qualys/qcs-sensor:latest  
--registry-sensor
```

## CI/CD Sensor

### Linux:

```
sudo docker run -d --restart on-failure -v  
/var/run/docker.sock:/var/run/docker.sock:ro -v  
/usr/local/qualys/sensor/data:/usr/local/qualys/qpq/data -e  
ACTIVATIONID=<Activation id> -e CUSTOMERID=<Customer id> -e POD_URL=<POD URL>  
--net=host --name qualys-container-sensor qualys/qcs-sensor:latest  
--cicd-deployed-sensor
```

### MacOS:

```
sudo mkdir -p /tmp/qualys/sensor/data
sudo chmod -R 777 /tmp/qualys/sensor/data
sudo mkdir /private/etc/qualys/
sudo chmod 777 /private/etc/qualys

sudo docker run -d --restart on-failure -v
/var/run/docker.sock:/var/run/docker.sock:ro -v
/private/etc/qualys:/usr/local/qualys/qpa/data/conf/agent-data -v
/tmp/qualys/sensor/data:/usr/local/qualys/qpa/data -e
ACTIVATIONID=<Activation id> -e CUSTOMERID=<Customer id> -e POD_URL=<POD
URL> --net=host --name qualys-container-sensor qualys/qcs-sensor:latest
--cicd-deployed-sensor
```

### CoreOS:

```
sudo mkdir -p /var/opt/qualys/sensor/data
sudo chmod -R 777 /var/opt/qualys/sensor/data

sudo docker run -d --restart on-failure -v
/var/run/docker.sock:/var/run/docker.sock:ro -v
/var/opt/qualys/sensor/data:/usr/local/qualys/qpa/data -e
ACTIVATIONID=<Activation id> -e CUSTOMERID=<Customer id> -e POD_URL=<POD
URL> --net=host --name qualys-container-sensor qualys/qcs-sensor:latest
--cicd-deployed-sensor
```

### Volumes used in the above commands:

**/var/run/docker.sock:/var/run/docker.sock:ro** - mounts the Docker socket to the sensor file system. This is mandatory unless user specifies the DOCKER\_HOST environment variable if docker daemon is running on TCP port.

**/usr/local/qualys/sensor/data:/usr/local/qualys/qpa/data** - provides persistent storage for the sensor container. This mapping is mandatory unless “--sensor-without-persistent-storage” option is used. You may change the storage directory. The directory is automatically created if doesn't exist.

Additional environment variables/volumes can be provided:

1) If proxy is used to communicate with Qualys Cloud Platform, specify:

```
-e qualys_https_proxy=<IP/ address or FQDN>:<Port#>
```

2) If the proxy cert is required, mount volume for proxy certificate by adding:

```
-v <Proxy_File_Path>:/etc/qualys/qpa/cert/custom-ca.crt
```

3) If the Docker daemon is running on TCP port, specify -e DOCKER\_HOST=<IPv4 address or FQDN>:<port#>. Ensure that you remove the docker Unix domain socket volume mount (-v /var/run/docker.sock:/var/run/docker.sock:ro) in this case.

4) /etc/qualys:/usr/local/qualys/qpa/data/conf/agent-data - HostID search directory to map the marker file created by Qualys Agent or Scanner appliance on the host.

5) If you want the sensor to communicate with docker daemon over TLS socket, specify the following mandatory environment variables and the volume mount.

Specify TLS docker socket to connect to by setting DOCKER\_HOST environment variable:

```
-e DOCKER_HOST=<docker daemon host's IPv4 address, or FQDN, or  
hostname>:<port#>
```

where provided IPv4 address, or FQDN or hostname matches either the CN or the Alternative Subject Name in the docker server certificate.

To enable TLS authentication set:

```
-e DOCKER_TLS_VERIFY=1
```

**Note:** By enabling sensor communication with docker daemon over TLS customer can restrict the sensor's access to docker socket by using docker authorization plugin.

Volume mount the directory on the docker daemon host where docker client certificate, client private key and CA certificate files are available:

```
-v <docker client certificate directory on the docker daemon  
host>:/root/.docker
```

Specify docker client certificate, client private key and CA certificate file names as arguments to sensor:

```
--tls-cacert <file name of the CA certificate used to sign docker server  
certificate> --tls-cert <docker client certificate file name> --tls-key  
<docker client private key file name>
```

If any of the CA certificate, client certificate or client private key have default file names such as ca.pem, cert.pem, key.pem respectively they can be omitted. For example, if docker daemon is listening on both unix domain socket and TCP TLS sockets you can launch the sensor like this:

```
docker run -d --restart on-failure --cpus=0.2 -v  
/var/run/docker.sock:/var/run/docker.sock:ro -v <client cert directory  
on the docker host>:/root/.docker -v  
/usr/local/qualys/sensor/data:/usr/local/qualys/qpa/data -e  
ACTIVATIONID=<Activation id> -e CUSTOMERID=<Customer id> -e POD_URL=<POD  
URL> -e DOCKER_TLS_VERIFY=1 -e DOCKER_HOST=<IPv4 or FQDN>:<port#> --  
net=host --name qualys-container-sensor qualys/qcs-sensor:latest --log-  
level 5 --tls-cacert <file name of the CA certificate used to sign docker  
server certificate> --tls-cert <docker client certificate file name> --  
tls-key <docker client private key file name>
```

## Optional Parameters

### **--cpus**

Restrict the cpu usage to a certain value.

--cpus=0.2 # Default CPU usage limit (20% of one core/processor on the host).

For example, for limiting the cpu usage to 5%, set --cpus=0.05. This limits the cpu usage to 5% of one core/processor on the host.

If there are multiple processors on a node, setting the cpus value applies the CPU limit to one core/processor only. For example, if you have 4 CPUs on the system and you want to set CPU limit as 20% of overall CPU capacity, then the CPU limit should be set to 0.8 i.e., 80% of one core only which becomes 20% of total CPU capacity.

To disable any CPU usage limit, do not specify the option.

**Note:** If docker host's kernel does not support setting the CPU limit on running containers, disable CPU usage limit, otherwise the sensor won't get launched.

### **--enable-console-logs**

Print logs on console. These logs can be retrieved using the docker logs command.

### **--sensor-without-persistent-storage**

Run the sensor without using persistent storage on host. In this case do not provide persistent storage mapping under volumes. It is recommended to use the "--enable-console-logs" option along with "--sensor-without-persistent-storage" to preserve the logs as data is not available on host but stored at the /usr/local/qualys/qpa/data folder relative to the Sensor.

### **--log-level**

Set the logging level for sensor, accepts 0 to 5. Default is 3 (Information).

### **--log-filesize**

Set the maximum size per log file for sensor in bytes. For example, specify "10" for 10 bytes, "10K" for 10 kilobytes, "10M" for 10 megabytes. Default is "10M".

### **--log-filepurgecount**

Define the number of archived qpa.log files to be generated. Default is 5.

### **--scan-thread-pool-size**

Launch the sensor with scan thread value. Default is 4.

### **--read-only**

Run sensor in read-only mode. In this mode the sensor uses persistent storage on host.

**Note:** The sensor should be run either with "--sensor-without-persistent-storage" option or with "--read-only" option and not with both options enabled together.



### **--mask-env-variable**

Mask environment variables for images and containers. The environment variables will be masked/removed in sensor logs and in the Container Security UI.

### **--disableImageScan**

This parameter should be passed if you want to disable image scans for General Sensor. Images will not be scanned by sensors deployed with this option. This is available for General sensor type only, and is available for all Runtimes (Docker, CRI-O and Containerd).

### **--disable-log4j-scanning**

This parameter should be passed if you want to disable log4j vulnerability scanning for container images. See [Log4j vulnerability scanning](#).

### **--disable-log4j-static-detection**

This parameter should be passed if you want to disable log4j static detection for dynamic/static image scans. See [Static log4j detection](#).

### **--optimize-image-scans**

This parameter should be passed if you want to optimize image scans for General Sensor. By default, the sensor scans every image that it detects on the host. This results in redundant scanning of images. When you install the General sensor with “--optimize-image-scans”, the sensor will communicate with the Qualys Cloud Platform and perform informed scans to avoid redundant image scans. The sensor will determine if the images present on the host are already scanned by other sensors for the same manifest and version and will not scan those images again.

### **--scanning-policy**

This parameter should be passed if you want to specify a scanning policy. The scanning policy allows you to select the suitable scan type as per your requirement. The available values are:

- `DynamicScanningOnly`: performs only dynamic scanning.
- `StaticScanningOnly`: performs only static scanning.
- `DynamicWithStaticScanningAsFallback`: performs static scanning as a fallback to dynamic scanning for images without shell.

### **--perform-secret-detection**

This parameter should be passed if you want to perform secret detection for your container images. You can specify a timeout for secret detection using `--sca-scan-timeout-in-seconds={value}` parameter. For information about secret detection, see Online Help: Detect Container Secrets.

### **--perform-malware-detection**

This parameter should be passed if you want to perform malware detection for your container images. You can specify a timeout for malware detection using `--sca-scan-timeout-in-seconds={value}` parameter. For information about malware detection, see Online Help: Malware Detection.

### **--limit-resource-usage**

This parameter can be used to limit usage of resources for SCA or Secret or Malware Scan.

## **Optional parameters for SCA scanning**

The following parameters are optional when the SCA scanning feature is enabled for your subscription. See [SCA scanning](#) to learn more.

### **--perform-sca-scan**

(Optional) By default, SCA scanning is not performed. Use this parameter to enable SCA scanning for container images. When specified, the SCA scan will be performed after a standard vulnerability scan (Static or Dynamic). The SCA scan is attempted even when the vulnerability scan is not successful.

### **--disallow-internet-access-for-sca**

(Optional when `--perform-sca-scan` is specified) By default, SCA scans run in online mode. Use this parameter to disable Internet access for the SCA scan and run the scan in offline mode.

Note: We recommend you run the SCA scan in online mode. Quality of software package enumeration for Java substantially degrades when the SCA scan is run in offline mode. The remote maven repository may need to be consulted for an accurate package detection. This can affect accuracy of the vulnerability posture of the image.

### **--sca-scan-timeout-in-seconds={value}**

(Optional when `--perform-sca-scan` is specified) The default SCA scan command timeout is 5 minutes (300 seconds). Use this parameter to overwrite the default timeout with a new value specified in seconds. For example, you may need to increase the SCA scan timeout when scanning large container images to ensure the SCA scan has time to finish.

**Note:** The `--sca-scan-timeout-in-seconds` parameter is also used for specifying a timeout for secret and malware detection.

### **--limit-resource-usage**

This parameter can be used to limit usage of resources for SCA or Secret or Malware Scan.

## **How to Comply with CIS Benchmark for Docker using Docker Run Commands**

Qualys Container Security adheres to the CIS Benchmark for Docker for our Sensor image. Refer to [Compliance with CIS Benchmark for Docker](#) for guidance on how to use the Sensor image in a way that complies with the CIS Benchmark for Docker. We've provided instructions for a number of controls so you can operate the Sensor in a compliant manner.

## Deploying the sensor using Docker Hub on Kubernetes

Prerequisites:

- Kubernetes setup should be up and running
- K8S nodes should be able to communicate with the Docker hub/private registry
- The container sensor image should be available in the private registry if you are installing from there.

### Modify the `cssensor-ds.yml` file

Create a new yml file containing the following information and name it `cssensor-ds.yml` or download the yml file directly from [https://github.com/Qualys/cs\\_sensor](https://github.com/Qualys/cs_sensor)

**Note:** The field alignment in the yml file is very important. Please make sure to honor the formatting provided in the below template.

```
kind: List
apiVersion: v1
items:
  - kind: Namespace
    apiVersion: v1
    metadata:
      name: qualys
  # Service Account
  - kind: ServiceAccount
    apiVersion: v1
    metadata:
      name: qualys-service-account
      namespace: qualys
  # Role for read/write/delete permission to qualys namespace
  - kind: Role
    # if k8s version is 1.17 and earlier then change apiVersion to
"rbac.authorization.k8s.io/v1beta1"
    apiVersion: rbac.authorization.k8s.io/v1
    metadata:
      name: qualys-reader-role
      namespace: qualys
    rules:
      - apiGroups: [ "", "batch" ]
        resources: [ "pods", "jobs" ]
        verbs: [ "get", "list", "watch", "create", "delete",
"deletecollection" ]
      - apiGroups: [ "" ]
        resources: [ "pods/status" ]
        verbs: [ "get" ]
      - apiGroups: [ "" ]
        resources: [ "pods/attach", "pods/exec" ]
```

```
    verbs: ["create"]
- kind: ClusterRole
  # if k8s version is 1.17 and earlier then change apiVersion to
"rbac.authorization.k8s.io/v1beta1"
  apiVersion: rbac.authorization.k8s.io/v1
  metadata:
    name: qualys-cluster-reader-role
  rules:
- apiGroups: [""]
  resources: ["nodes", "pods/status",
"replicationcontrollers/status", "nodes/status"]
  verbs: ["get"]
- apiGroups: ["apps"]
  resources: ["replicasets/status", "daemonsets/status",
"deployments/status", "statefulsets/status"]
  verbs: ["get"]
- apiGroups: ["batch"]
  resources: ["jobs/status", "cronjobs/status"]
  verbs: ["get"]
  # RoleBinding to assign permissions in qualys-reader-role to qualys-
service-account
- kind: RoleBinding
  # if k8s version is 1.17 and earlier then change apiVersion to
"rbac.authorization.k8s.io/v1beta1"
  apiVersion: rbac.authorization.k8s.io/v1
  metadata:
    name: qualys-reader-role-rb
    namespace: qualys
  subjects:
- kind: ServiceAccount
  name: qualys-service-account
  namespace: qualys
  roleRef:
    kind: Role
    name: qualys-reader-role
    apiGroup: rbac.authorization.k8s.io
- kind: ClusterRoleBinding
  # if k8s version is 1.17 and earlier then change apiVersion to
"rbac.authorization.k8s.io/v1beta1"
  apiVersion: rbac.authorization.k8s.io/v1
  metadata:
    name: qualys-cluster-reader-rb
  subjects:
- kind: ServiceAccount
  name: qualys-service-account
  namespace: qualys
```

```
roleRef:
  kind: ClusterRole
  name: qualys-cluster-reader-role
  apiGroup: rbac.authorization.k8s.io
# Qualys Container Sensor pod with
- apiVersion: apps/v1
  kind: DaemonSet
  metadata:
    name: qualys-container-sensor
    namespace: qualys
    labels:
      k8s-app: qualys-cs-sensor
  spec:
    selector:
      matchLabels:
        name: qualys-container-sensor
    updateStrategy:
      type: RollingUpdate
    template:
      metadata:
        labels:
          name: qualys-container-sensor
      spec:
        #tolerations:
        # this toleration is to have the daemonset runnable on master
nodes
  # remove it if want your masters to run sensor pod
  #- key: node-role.kubernetes.io/master
  # effect: NoSchedule
  serviceAccountName: qualys-service-account
  containers:
  - name: qualys-container-sensor
    image: qualys/qcs-sensor:latest
    imagePullPolicy : IfNotPresent
    resources:
      limits:
        cpu: "0.2" # Default CPU usage limit on each node for
sensor.
    args: ["--k8s-mode"]
    env:
      - name: CUSTOMERID
        value: __customerId
      - name: ACTIVATIONID
        value: __activationId
      - name: POD_URL
        value:
```

```
    - name: QUALYS_SCANNING_CONTAINER_LAUNCH_TIMEOUT
      value: "10"
# uncomment (and indent properly) below section if using Docker HTTP
socket with TLS
    #- name: DOCKER_TLS_VERIFY
    # value: "1"
# uncomment (and indent properly) below section if proxy is required to
connect Qualys Cloud
    #- name: qualys_https_proxy
    # value: <proxy FQDN or Ip address>:<port#>
    - name: QUALYS_POD_NAME
      valueFrom:
        fieldRef:
          fieldPath: metadata.name
    - name: QUALYS_POD_NAMESPACE
      valueFrom:
        fieldRef:
          fieldPath: metadata.namespace
volumeMounts:
  - mountPath: /var/run/docker.sock
    name: socket-volume
    readOnly: true
  - mountPath: /usr/local/qualys/qpq/data
    name: persistent-volume
  - mountPath: /usr/local/qualys/qpq/data/conf/agent-data
    name: agent-volume
# uncomment (and indent properly) below section if proxy (with CA cert)
required to connect Qualys Cloud
    #- mountPath: /etc/qualys/qpq/cert/custom-ca.crt
    # name: proxy-cert-path
# uncomment (and indent properly) below section if using Docker HTTP
socket with TLS
    #- mountPath: /root/.docker
    # name: tls-cert-path
securityContext:
  allowPrivilegeEscalation: false
volumes:
  - name: socket-volume
    hostPath:
      path: /var/run/docker.sock
      type: Socket
  - name: persistent-volume
    hostPath:
      path: /usr/local/qualys/sensor/data
      type: DirectoryOrCreate
  - name: agent-volume
```

```
        hostPath:
          path: /etc/qualys
          type: DirectoryOrCreate
# uncomment (and indent properly) below section if proxy (with CA cert)
# required to connect Qualys Cloud
#- name: proxy-cert-path
#   hostPath:
#     path: <proxy certificate path>
#     type: File
# uncomment (and indent properly) below section if using Docker HTTP
# socket with TLS
#- name: tls-cert-path
#   hostPath:
#     path: <Path of directory of client certificates>
#     type: Directory
hostNetwork: true
```

Qualys Container Sensor DaemonSet should be deployed in 'qualys' namespace as a part of ServiceAccount with adequate permission to communicate with Kubernetes API Server. The Role, ClusterRole, RoleBinding and ClusterRoleBinding are used to assign the necessary permissions to the ServiceAccount. If you already have Qualys Container Sensor running in a different namespace other than 'qualys', you'll need to first uninstall Qualys Container Sensor from the other namespace and then deploy it fresh in 'qualys' namespace.

You'll need these permissions:

get, list, watch - to monitor the resources in 'qualys' namespace

create, delete, deletecollection - to spawn containers for image vulnerability assessment in 'qualys' namespace then clean up after itself

## Modify parameters in the yaml file

Copy the **cssensor-ds.yaml** file to Kubernetes cluster's master node then modify it by providing values for the following parameters. In order for the yaml file to work properly, ensure you only update the parameters/sections specified below. Note that you can download the yaml file directly from [https://github.com/Qualys/cs\\_sensor](https://github.com/Qualys/cs_sensor)

Uncomment the **tolerations** section under **spec** if you want Sensor daemonset to be deployed on master nodes.

```
spec:
#tolerations:
# this toleration is to have the daemonset runnable on master nodes
# remove it if want your masters to run sensor pod
#- key: node-role.kubernetes.io/master
# effect: NoSchedule
```

If you want to prioritize Qualys Sensor PODs:

Locate and Uncomment the below lines of code in the .yaml file. And change the PriorityClass value mentioned under kind: PriorityClass as per your requirement.

```
#- kind: PriorityClass
#  apiVersion: scheduling.k8s.io/v1
#  metadata:
#    name: qualys-priority-class
#  value: 0
#  preemptionPolicy: PreemptLowerPriority
#  description: Priority class for daemonset
```

Also, locate the PriorityClass name present below and Uncomment it.

```
#priorityClassName: qualys-priority-class
```

In order for the yaml file to work properly, ensure that you do not remove/comment the respective sections mentioned below. Ensure that from all Kubernetes nodes the Docker hub/private registry (where the CS Sensor image is published) is accessible.

```
containers:
  - name: qualys-container-sensor
    image: <CS Sensor image name in the private/docker hub registry>
    args: ["--k8s-mode"]
```

**Note:** Make sure all nodes that will be running sensor pod have access to private or docker hub registry where sensor image is stored.

If you want to deploy the sensor for CI/CD environment provide the args value as:

```
args: ["--k8s-mode", "--cicd-deployed-sensor"]
```

If you want to deploy a Registry Sensor provide the args value as: args:

```
["--k8s-mode", "--registry-sensor"]
```

If you want print logs on the console, provide "--enable-console-logs" as an additional value in args.

If you want to change the log level, provide "--log-level", "<a number between 0 and 5>" as an additional value in args, e.g if you want logs in trace provide:

```
args: ["--k8s-mode", "--log-level", "5"]
```

If you want to launch the sensor with scan thread value other than default 4, provide "-- scan-thread-pool-size", "<number of threads>" as an additional value in args.

```
args: ["--k8s-mode", "--scan-thread-pool-size", "6"]
```



If you want to define the number of archived qpa.log files to be generated provide "--log-filepurgecount", "" as an additional value in args. The default, "--logfilepurgecount", "5" is applied via config. Note that there will always be current qpa.log file in log/ directory.

If you want to define the number of archived qpa.log files to be generated and size per log file, provide "--log-filesize", "" where "K" means kilobyte and "M" means megabyte, and "--log-filepurgecount", "" as an additional value in args. Default is "--log-filesize": "10M" and "--log-filepurgecount": "5" applied via config.

```
args: ["--k8s-mode", "--log-filesize", "5M", "--logfilepurgecount", "4"]
```

If you want image scanning pods to be instantiated using kubernetes native `kubectl run` command provide "--use-kubectl" as an additional value in args. In this case sensor uses native kubernetes facilities to launch image scans. When this argument is omitted image containers are launched using docker run.

```
args: ["--k8s-mode", "--use-kubectl"]
```

If TLS authentication is enabled specify docker client certificate, client private key and CA certificate names in the args,

```
args: ["--k8s-mode", "--tls-cacert", "<file name of the CA certificate  
that was used to sign docker server certificate>", "--tls-cert",  
<docker client certificate file name>", "--tls-key", "<docker client  
private key file name>"]
```

**Note:** If any of the three files have a default name such as ca.pem, cert.pem, key.pem respectively the corresponding argument can be omitted.

If you want to mask environment variables for images and containers in sensor logs and in the Container Security UI, add the "--mask-env-variable" parameter to args:

```
args: ["--k8s-mode", "--mask-env-variable"]
```

If you want to disable image scans for General Sensor (supported for all Runtimes), add the "--disableImageScan" parameter to args:

```
args: ["--k8s-mode", "--disableImageScan"]
```

If you want to specify a scanning policy, add the "--scanning-policy" parameter to args. The available values for the scanning policy are: "DynamicScanningOnly", "StaticScanningOnly", and "DynamicWithStaticScanningAsFallback".

```
args: ["--k8s-mode", "--scanning-policy", "StaticScanningOnly"]
```

If you want to disable log4j vulnerability scanning on your container images, add the "--disable-log4j-scanning" parameter to args:

```
args: ["--k8s-mode", "--disable-log4j-scanning"]
```

If you want to disable log4j static detection for dynamic/static image scans, add the "--disable-log4j-static-detection" parameter to args:

```
args: ["--k8s-mode", "--disable-log4j-static-detection"]
```

If you want to optimize image scans for General Sensor, add the "--optimize-image-scans" parameter to args:

```
args: ["--k8s-mode", "--optimize-image-scans"]
```

If you want to enable secret detection for container images, add the "--perform-secret-detection" parameter to args:

```
args: ["--k8s-mode", "--container-runtime", "containerd", "--perform-secret-detection"]
```

If you want to enable malware detection for container images, add the "--perform-malware-detection" parameter to args:

```
args: ["--k8s-mode", "--registry-sensor", "--perform-malware-detection"]
```

If you want to limit the memory usage of sensor container, add the "--limit-resource-usage" parameter to args.

```
args: ["--k8s-mode", "--limit-resource-usage"]
```

Under **resources** specify the following:

```
resources:  
limits:  
cpu: "0.2" # Default CPU usage limit on each node for sensor.
```

For example, for limiting the CPU usage to 5%, set resources:limits:cpu: "0.05". This limits the CPU usage to 5% of one core on the host.

If there are multiple processors on a node, setting the resources:limits:cpu value applies the CPU limit to one core only. For example, if you have 4 CPUs on the system and you want to set CPU limit as 20% of overall CPU capacity, then the CPU limit should be set to 0.8 i.e., 80% of one core only which becomes 20% of total CPU capacity.

To disable any CPU usage limit, set resources:limits:cpu value to 0.

Optionally, if you want to specify the memory resources for Container Sensor, you can specify it under **resources**. Recommended values for the Container Sensor's memory requests and memory limits are:

```
resources:  
  limits:  
    cpu: "0.2" # Default CPU usage limit on each node for sensor  
    memory: "500Mi"  
  requests:
```

```
memory: "300Mi"
```

If you want to specify the memory resources for SCA Scan, or Secret Scan, or Malware Scan you can specify it under **resources**. It is recommended to use "--limit-resource-usage" argument while installing the sensor.

Recommended values for the Container Sensor's memory requests and memory limits for above mentioned scan types are:

```
resources:
  limits:
    cpu: "0.2" # Default CPU usage limit on each node for sensor
    memory: "3.5Gi"
  requests:
    memory: "1.5Gi"
```

Disclaimer: The above recommendation is based on the average image size of 1.5GB, if the image sizes are more, please increase the memory limit accordingly.

When either of the memory resource values (limits or requests) is specified for Container Sensor and "--use-kubect!" is supplied in args, we automatically apply both memory requests and memory limits to image scanning containers. Default values are 200Mi and 700Mi, respectively.

Additionally, you could overwrite one or both values by specifying the following variables under **env**. In this example, the values were changed to 300Mi and 800Mi.

```
- name: QUALYS_SCANNING_CONTAINER_MEMORYREQUESTMB
  value: "300Mi"
- name: QUALYS_SCANNING_CONTAINER_MEMORYLIMITMB
  value: "800Mi"
```

Under **env** specify the following:

Activation ID (Required)

```
- name: ACTIVATIONID
  value: XXXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXXXXXX
```

Customer ID (Required)

```
- name: CUSTOMERID
  value: XXXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXXXXXX
```

Specify **POD\_URL** when using docker hub image. Otherwise, remove it.

```
- name: POD_URL
  value: <Specify POD URL>
```

Specify the scanning container launch timeout in minutes. If this env variable is not present, then 10 minutes is the default.

```
- name: QUALYS_SCANNING_CONTAINER_LAUNCH_TIMEOUT  
  value: "10"
```

With each scan, we check the node status to see if the node is schedulable or not, and launch the scan only if the node is schedulable. If the node status indicates that the node is unschedulable, then we retry the scan after a default interval of 15 minutes. You can increase or decrease the time the sensor waits before retrying the scan by specifying a different scan retry interval in minutes.

```
- name: UNSCHEDULABLE_NODE_SCAN_RETRY_INTERVAL  
  value: "30"
```

To enable TLS authentication uncomment the following 2 lines.

```
- name: DOCKER_TLS_VERIFY  
  value: "1"
```

**Note:** To disable TLS use DOCKER\_TLS\_VERIFY=""(empty string) or remove or keep it commented in yml file.

**Note:** By enabling sensor communication with docker daemon over TLS customer can restrict the sensor's access to docker socket by using docker authorization plugin.

**Note:** When TLS authentication is enabled and DOCKER\_HOST is not specified the sensor will automatically detect the FQDN of the worker node it is running on and set DOCKER\_HOST environment variable inside the sensor to <worker node's FQDN>:<2376> where 2376 is the default TLS TCP port for docker daemon

**Note:** You can set DOCKER\_HOST yourself to 127.0.0.1:<port#> or localhost:<port#> by adding:

```
- name: DOCKER_HOST  
  value: "<loopback IPv4 address or hostname>:<port#>"
```

**Note:** Please make sure that FQDN, or hostname, or IPv4 address set in the DOCKER\_HOST matches the CN or Subject Alternative Name in the docker server certificate on each worker node.

Uncomment and indent properly the proxy information, or keep it as is if not required:

- For the communication between the sensor and the backend (Container Management Service):

```
#- name: qualys_https_proxy  
#   value: <proxy FQDN or Ip address>:<port#>
```

- For a registry sensor version 1.21.0 or later used for a public registry:

```
#- name: https_proxy  
#   value: <proxy FQDN or Ip address>:<port#>
```

Uncomment `tls-cert-path` under **volumes** if TLS authentication needs to be enabled and provide directory path for client certificates, or keep it as is if not required:

```
#- name: tls-cert-path
# hostPath:
# path: <Path of directory of client certificates>
# type: Directory
```

Uncomment `proxy-cert-path` under **volumes**, or keep it as is if not required:

```
#- name: proxy-cert-path
# hostPath:
# path: /root/cert/proxy-certificate.crt
# type: File
```

Activation ID and Customer ID are required. Use the Activation ID and Customer ID from your subscription. To get the Activation ID and Customer ID, login to the Container Security UI, go to Configurations > Sensors, click Download, and then click any sensor type. The installation command on the Installation Instructions screen contains your Activation ID and Customer ID. Activation ID is like a password, do not share it.

If you are using an https proxy, ensure that all Kubernetes nodes have a valid certificate file for the sensor to communicate with the Container Management Server.

If you are not using a proxy and you have kept the above-mentioned parts commented, you can keep the following part commented from **volumeMounts** as well:

```
#- mountPath: /etc/qualys/qpa/cert/custom-ca.crt
# name: proxy-cert-path
```

If you are not using TLS and you have kept the above-mentioned parts commented, you can keep the following part commented from **volumeMounts** as well:

```
#- mountPath: /root/.docker
# name: tls-cert-path
```

## Remove `hostNetwork: true` when IP identification is not necessary

Remove/comment out the **hostNetwork: true** option to follow the security best practice of least privilege, where host IP address identification is not necessary.

```
hostNetwork: true
```

The `hostNetwork: true` option provides the sensor the ability to detect the Docker host's IP address. The user can remove/comment out the `hostNetwork: true` line in the yaml file, however a drawback is that the UI will show the docker bridge network IP address as the host IP address. To aid with host identification, the hostname appears in the UI so customers can identify the Docker host by hostname. We also recommend that the hostname be unique within the environment.

## Deploying the Container Sensor DaemonSet

Once you have created the **cssensor-ds.yml** file, run the following command on Kubernetes master to create a DaemonSet:

```
kubectl create -f cssensor-ds.yml
```

## Removing the Container Sensor DaemonSet

If you need to uninstall Qualys Container Sensor, run the following command on Kubernetes master:

```
kubectl delete -f cssensor-ds.yml
```

## Upgrading the Container Sensor DaemonSet

Perform the following steps on Kubernetes master for updating the Container Sensor DaemonSet to a new version.

**Note:** Ensure that the Container Sensor DaemonSet is running in the Kubernetes environment.

1) Get the name of the Container Sensor DaemonSet

```
kubectl get ds -n kube-system
```

2) Update the image for the DaemonSet to use new qualys/sensor image

```
kubectl set image ds/<daemonset-name> -n kube-system <container-name>=<container-new-image>
```

3) Monitor Rollout status - this may take a while depending on the number of nodes

```
kubectl rollout status daemonset <daemonset-name> -n kube-system
```

4) If something goes wrong during rollout, rollback the DaemonSet to use the last image

```
kubectl rollout undo daemonset <daemonset-name> -n kube-system
```

# Installing the CI/CD Sensor in Docker-in-Docker Environment

In this section we'll describe how to install the CS Sensor in a CI/CD pipeline build for a Docker-in-Docker environment. This will allow you to scan images inside the Docker-in-Docker container.

## Step 1: Have the CS Sensor image inside a Docker-in-Docker Container

There are two ways to do this: 1) You can pull the CS Sensor image from the registry and launch the sensor when the container is spun up, or 2) you can bake the Docker-in-Docker container image with the CS Sensor tar in it.

### Pull the CS Sensor image from the registry and launch the sensor

#### Benefits:

- No need to have pre-baked Docker-in-Docker container image with CS Sensor image/tar.
- You can easily use CS sensor image hosted on Docker hub registry

#### Disadvantages:

- All Docker-in-Docker containers need to have access to the registry.
- An image is pulled each time a Docker-in-Docker container is spun up and that would be overhead.

### Pre-baked Docker-in-Docker container image with CS sensor tar in it

#### Benefits:

- No need to have access to the registry from Docker-in-Docker container.
- The execution of a few commands and `installsensor.sh` script is enough to launch the CS Sensor.

#### Disadvantages:

- The Docker-in-Docker container image size will be increased.
- You'll need to re-bake the Docker-in-Docker image for each new sensor release.

## Step 2: Launch the Container Security Sensor

There are two ways to do this: 1) You can launch the sensor when the Docker-in-Docker container boots up, or 2) launch the sensor from a build job. We'll describe both methods.

### Launch sensor on Docker-in-Docker container bootup

#### Benefits:

- No need to modify the build pipeline configuration to launch the CS Sensor.

#### Disadvantages:

- The credentials (Activation ID/Customer ID) need to be stored in the init script.
- Only predefined persistent storage path can be provided.

### Launch init script inside the Docker-in-Docker container

Use the init script to launch the sensor. The init script will have following command:

```
docker run -d --restart on-failure --cpus=0.2 -v
/etc/qualys:/usr/local/qualys/qpa/data/conf/agent-data -v
/var/run/docker.sock:/var/run/docker.sock:ro - v
/usr/local/qualys/sensor/data:/usr/local/qualys/qpa/data -e
ACTIVATIONID=<Activation id> -e CUSTOMERID=<Customer id> -e POD_URL=<POD
URL> --net=host --name qualys-container-sensor <Qualys CS Sensor image
name from registry> --scan-thread-pool-size 4 --cicd-deployed-sensor
```

### Use installsensor.sh script

Use the installsensor.sh script to launch the sensor on Docker-in-Docker bootup.

```
tar -xvf QualysContainerSensor.tar.xz
```

```
docker load -i qualys-sensor.tar
```

```
./installsensor.sh ActivationId=<Activation id> CustomerId=<Customer id>
HostIdSearchDir=/private/etc/qualys Storage=/tmp/qualys/sensor/data
--cicd-deployed-sensor -s
```

### Launch sensor from build job

#### Benefits:

- Credentials (AI/CI) and sensor parameters can be passed from build job configuration.
- The persistent storage can be defined during launch.
- It's easy to have a unique directory for each job (using Job ID) and using it as persistent storage.

#### Disadvantages:

- You'll need to modify the build job configuration to launch the CS Sensor.



### Launch CS Sensor using docker run command to pull image from registry

Launch the CS Sensor using the docker run command in order to pull the CS Sensor image from the registry.

```
docker run -d --restart on-failure --cpus=0.2 -v  
/etc/qualys:/usr/local/qualys/qpa/data/conf/agent-data -v  
/var/run/docker.sock:/var/run/docker.sock:ro -v  
/usr/local/qualys/sensor/data:/usr/local/qualys/qpa/data -e  
ACTIVATIONID=<Activation id> -e CUSTOMERID=<Customer id> -e POD_URL=<POD  
URL> --net=host --name qualys-container-sensor <Qualys CS Sensor image  
name from registry> --scan-thread-pool-size 4 --cicd-deployed-sensor
```

### Launch CS Sensor as part of a build job using pre-baked Docker-in-Docker image

This command will launch the CS Sensor as part of a build job using a pre-baked Docker-in-Docker container image with the CS Sensor tar in it. It will launch the CS Sensor as part of the job.

```
<path>/installsensor.sh ActivationId=<Activation id>  
CustomerId=<Customer id> HostIdSearchDir=/private/etc/qualys  
Storage=/tmp/qualys/sensor/data --cicd-deployed-sensor -s
```

### Persistent storage for CS sensor running in Docker-in-Docker build container

Please provide the appropriate persistent storage for CS sensor so that the logs can be retrieved in case of CS sensor failure or container image scan failure.

# Deploying sensor in Kubernetes

This section provides steps for deploying the container sensor in Kubernetes.

## **Jump to a section:**

[How to Detect the Container Runtime in your Kubernetes Cluster Environment](#)

[Obtain the Container Sensor Image](#)

[Deploying the sensor using Docker Hub on Kubernetes](#)

[Deploy in Azure Kubernetes Service \(AKS\)](#)

[Deploy in Kubernetes - Docker Runtime](#)

[Deploy in Kubernetes - Containerd Runtime](#)

[Deploy in Kubernetes - CRI-O Runtime](#)

[Deploy in Kubernetes - OpenShift](#)

[Deploy in Kubernetes - OpenShift4.4+ with CRI-O Runtime](#)

[Deploy in Kubernetes with TKGI - Docker Runtime](#)

[Deploy in Kubernetes with TKGI - Containerd Runtime](#)

[Deploy in Kubernetes with RKE1 - Docker Runtime](#)

[Deploy in Kubernetes with RKE2 - Containerd Runtime](#)

[Deploy in Google Kubernetes Engine \(GKE\) with multi-node clusters](#)

[Deploy in Kubernetes using Helm Charts](#)

[Collection of Kubernetes Cluster Attributes](#)

[Update the sensor deployed in Kubernetes](#)

## How to Detect the Container Runtime in your Kubernetes Cluster Environment

Before installing the sensor image, it's important to know which container runtime is installed in your Kubernetes Cluster environment. Knowing this will help you determine what kind of sensor needs to be running in your environment.

You can get details about the container runtime by executing the following command:

```
kubectl get nodes -o wide
```

Use this command to get additional information about the cluster with details for each node like name, status, roles, age, version, internal and external IP addresses, OS image, kernel version. See the example below.

```
[root@CentOS7-Server ~]# kubectl get nodes -o wide
```

NAME	STATUS	ROLES	AGE	VERSION	INTERNAL-IP	EXTERNAL-IP	OS-IMAGE	KERNEL-VERSION	CONTAINER-RUNTIME
gke	-fkkd	Ready	<none>	10d	v1.20.10-gke.1600		Container-Optimized OS from Google	5.4.120+	containerd://1.4.4
gke	-h3vc	Ready	<none>	10d	v1.20.10-gke.1600		Container-Optimized OS from Google	5.4.120+	containerd://1.4.4
gke	-ssh1	Ready	<none>	10d	v1.20.10-gke.1600		Container-Optimized OS from Google	5.4.120+	containerd://1.4.4

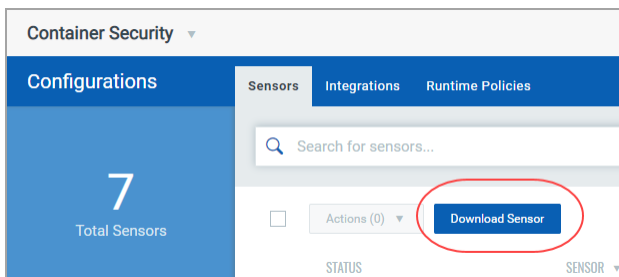
```
[root@CentOS7-Server ~]#
```

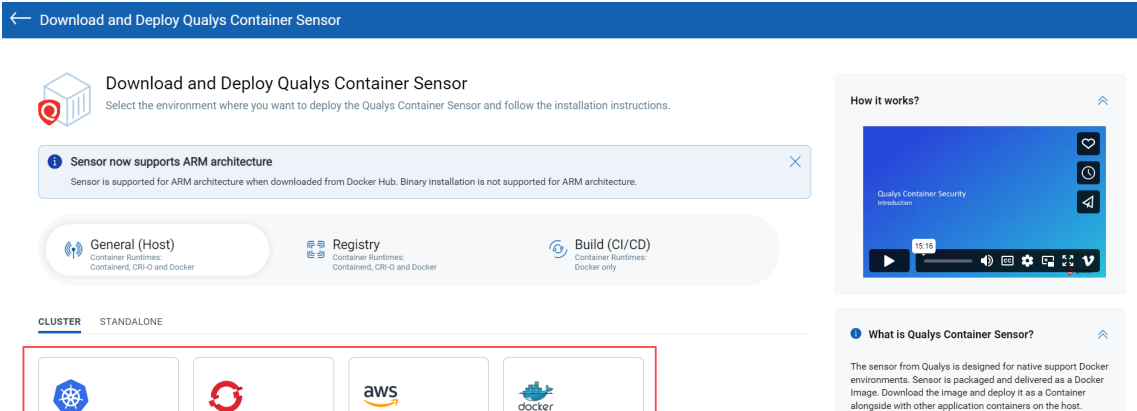
## Obtain the Container Sensor Image

The first step for any Kubernetes deployment is to obtain the sensor image. You can download QualysContainerSensor.tar.xz from the Container Security UI. Or you can use the latest Qualys Container Sensor image - qualys/qcs-sensor:latest - from Docker Hub.

### Download from UI

Download the sensor from the UI on a Linux computer with Docker installed on it.





You can download sensor deployment templates either from the UI or from GitHub directly at [https://github.com/Qualys/cs\\_sensor](https://github.com/Qualys/cs_sensor). To get details on how to use the sensor templates, follow the deployment steps outlined in the sections that follow.

Orchestration Platform	Container Runtime	Sensor Template
AWS ECS	Docker	cssensor-aws-ecs.json
Kubernetes (Cloud, On-Prem)	Docker	cssensor-ds.yml
Kubernetes (Cloud, On-Prem)	Docker	cssensor-ds_pv_pvc.yml
Kubernetes (Cloud, On-Prem)	Containerd	cssensor-containerd-ds.yml
Kubernetes (Cloud, On-Prem)	CRI-O	cssensor-crio-ds.yml
Docker Swarm	Docker	cssensor-swarm-ds.yml
Red Hat OpenShift	Docker	cssensor-openshift-ds.yml
Red Hat OpenShift	CRI-O	cssensor-openshift-crio-ds.yml

**Note:** CRI-O Runtime are supported by the General (host) sensor and Registry sensor. It is not supported by Build CI/CD sensor.

After downloading the file, untar the sensor package using this command:

```
sudo tar -xvf QualysContainerSensor.tar.xz
```

### To load the images in Docker Runtime environment:

Push the Qualys sensor image to a repository common to all nodes in the Kubernetes cluster using these commands:

```
sudo docker load -i qualys-sensor.tar
sudo docker tag <IMAGE NAME/ID> <URL to push image to the repository>
sudo docker push <URL to push image to the repository>
```

For example:

```
sudo docker load -i qualys-sensor.tar
sudo docker tag c3fa63a818df mycloudregistry.com/container-
```

```
sensor:qualys-sensor-xxx  
sudo docker push mycloudregistry.com/container-sensor:qualys-sensor-xxx
```

**Note:** Do not use these examples as is. Replace the registry/image path with your own.

**To load the images in Containerd Runtime environment:**

Push the Qualys sensor image to a repository common to all nodes in the Kubernetes cluster using these commands:

```
ctr -n=k8s.io images import qualys-sensor.tar  
ctr images tag <IMAGE NAME/ID> <URL to push image to the repository>  
ctr images push <URL to push image to the repository>
```

For example:

```
ctr -n=k8s.io images import qualys-sensor.tar  
ctr images tag c3fa63a818df mycloudregistry.com/container-sensor:qualys-  
sensor-xxx  
ctr images push mycloudregistry.com/container-sensor:qualys-sensor-xxx
```

**Note:** Do not use these examples as is. Replace the registry/image path with your own.

**To load the images in CRI-O Runtime environment:**

Push the Qualys sensor image to a repository common to all nodes in the Kubernetes cluster using these commands:

```
podman load -i qualys-sensor.tar  
podman tag <IMAGE NAME/ID> <URL to push image to the repository>  
podman push <URL to push image to the repository>
```

For example:

```
podman load -i qualys-sensor.tar  
podman tag c3fa63a818df mycloudregistry.com/container-sensor:qualys-  
sensor-xxx  
podman push mycloudregistry.com/container-sensor:qualys-sensor-xxx
```

**Note:** Do not use these examples as is. Replace the registry/image path with your own.

## Get image from Docker Hub

Use the latest Qualys Container Sensor image - `qualys/qcs-sensor:latest` - from Docker Hub. The Container Security Sensor on Docker Hub is available as:

```
qualys/qcs-sensor:<tag>
```

```
qualys/qcs-sensor:latest
```

Look up the most recent tag in Docker Hub. The Docker Hub Qualys Container Sensor image can either be pushed to your private registry or used directly. Ensure that from all Kubernetes nodes the Docker Hub/private registry (where the CS Sensor image is published) is accessible.

## Deploy in Azure Kubernetes Service (AKS)

The steps you take to deploy a sensor in Azure Kubernetes Service (AKS) clusters depends on the Kubernetes version and the container runtime.

When deploying a sensor in AKS clusters using Kubernetes version 1.18 and older, and the container runtime is Docker runtime, please see: [Deploy in Kubernetes - Docker Runtime](#).

When deploying a sensor in AKS clusters using Kubernetes version 1.19, and the container runtime is Containerd runtime, please see: [Deploy in Kubernetes - Containerd Runtime](#).

## Deploy in Kubernetes - Docker Runtime

This section assumes you have the sensor image: [Obtain the Container Sensor Image](#)

Integrate the Container Sensor into the DaemonSet like other application containers and set the replication factor to 1 to ensure there is always a sensor deployed on the Docker Host. This information is applicable for Amazon Elastic Container Service for Kubernetes (Amazon EKS), Google Kubernetes Engine (GKE), and Azure Kubernetes Service (AKS).

Perform the following steps for creating a DaemonSet for the Qualys sensor to be deployed in Kubernetes.

**Note:** Ensure that the Container Sensor has read and write access to the persistent storage and the docker daemon socket.

### Modify the `cssensor-ds.yml` file

Below is the Kubernetes DaemonSet deployment template that can be used to deploy the Qualys Container Sensor. For customers' convenience, this template is available in the `QualysContainerSensor.tar.xz` as `cssensor-ds.yml` file. Note that you can download the yaml file directly from [https://github.com/Qualys/cs\\_sensor](https://github.com/Qualys/cs_sensor)

IMPORTANT: The field alignment in the yaml file is very important. Please make sure to honor the formatting provided in the template.

```
kind: List
apiVersion: v1
```

```
items:
- kind: Namespace
  apiVersion: v1
  metadata:
    name: qualys
# Service Account
- kind: ServiceAccount
  apiVersion: v1
  metadata:
    name: qualys-service-account
    namespace: qualys
# Role for read/write/delete permission to qualys namespace
- kind: Role
  # if k8s version is 1.17 and earlier then change apiVersion to
"rbac.authorization.k8s.io/v1beta1"
  apiVersion: rbac.authorization.k8s.io/v1
  metadata:
    name: qualys-reader-role
    namespace: qualys
  rules:
  - apiGroups: [ "", "batch" ]
    resources: [ "pods", "jobs" ]
    verbs: [ "get", "list", "watch", "create", "delete",
"deletecollection" ]
  - apiGroups: [ "" ]
    resources: [ "pods/status" ]
    verbs: [ "get" ]
  - apiGroups: [ "" ]
    resources: [ "pods/attach", "pods/exec" ]
    verbs: [ "create" ]
- kind: ClusterRole
  # if k8s version is 1.17 and earlier then change apiVersion to
"rbac.authorization.k8s.io/v1beta1"
  apiVersion: rbac.authorization.k8s.io/v1
  metadata:
    name: qualys-cluster-reader-role
  rules:
  - apiGroups: [ "" ]
    resources: [ "nodes", "pods/status",
"replicationcontrollers/status", "nodes/status" ]
    verbs: [ "get" ]
  - apiGroups: [ "apps" ]
    resources: [ "replicasets/status", "daemonsets/status",
"deployments/status", "statefulsets/status" ]
    verbs: [ "get" ]
  - apiGroups: [ "batch" ]
```

```
    resources: ["jobs/status", "cronjobs/status"]
    verbs: ["get"]
# RoleBinding to assign permissions in qualys-reader-role to qualys-
service-account
- kind: RoleBinding
  # if k8s version is 1.17 and earlier then change apiVersion to
"rbac.authorization.k8s.io/v1beta1"
  apiVersion: rbac.authorization.k8s.io/v1
  metadata:
    name: qualys-reader-role-rb
    namespace: qualys
  subjects:
  - kind: ServiceAccount
    name: qualys-service-account
    namespace: qualys
  roleRef:
    kind: Role
    name: qualys-reader-role
    apiGroup: rbac.authorization.k8s.io
- kind: ClusterRoleBinding
  # if k8s version is 1.17 and earlier then change apiVersion to
"rbac.authorization.k8s.io/v1beta1"
  apiVersion: rbac.authorization.k8s.io/v1
  metadata:
    name: qualys-cluster-reader-rb
  subjects:
  - kind: ServiceAccount
    name: qualys-service-account
    namespace: qualys
  roleRef:
    kind: ClusterRole
    name: qualys-cluster-reader-role
    apiGroup: rbac.authorization.k8s.io
# Qualys Container Sensor pod with
- apiVersion: apps/v1
  kind: DaemonSet
  metadata:
    name: qualys-container-sensor
    namespace: qualys
    labels:
      k8s-app: qualys-cs-sensor
  spec:
    selector:
      matchLabels:
        name: qualys-container-sensor
    updateStrategy:
```



```
    type: RollingUpdate
template:
  metadata:
    labels:
      name: qualys-container-sensor
  spec:
    #tolerations:
    # this toleration is to have the daemonset runnable on master
nodes
    # remove it if want your masters to run sensor pod
    #- key: node-role.kubernetes.io/master
    # effect: NoSchedule
    serviceAccountName: qualys-service-account
  containers:
  - name: qualys-container-sensor
    image: qualys/qcs-sensor:latest
    imagePullPolicy : IfNotPresent
    resources:
      limits:
        cpu: "0.2" # Default CPU usage limit on each node for
sensor.
    args: ["--k8s-mode"]
    env:
      - name: CUSTOMERID
        value: __customerId
      - name: ACTIVATIONID
        value: __activationId
      - name: POD_URL
        value:
      - name: QUALYS_SCANNING_CONTAINER_LAUNCH_TIMEOUT
        value: "10"
# uncomment (and indent properly) below section if using Docker HTTP
socket with TLS
    #- name: DOCKER_TLS_VERIFY
    # value: "1"
# uncomment (and indent properly) below section if proxy is required to
connect Qualys Cloud
    #- name: qualys_https_proxy
    # value: <proxy FQDN or Ip address>:<port#>
    - name: QUALYS_POD_NAME
      valueFrom:
        fieldRef:
          fieldPath: metadata.name
    - name: QUALYS_POD_NAMESPACE
      valueFrom:
        fieldRef:
```

```
        fieldPath: metadata.namespace
volumeMounts:
- mountPath: /var/run/docker.sock
  name: socket-volume
  readOnly: true
- mountPath: /usr/local/qualys/qpaa/data
  name: persistent-volume
- mountPath: /usr/local/qualys/qpaa/data/conf/agent-data
  name: agent-volume
# uncomment (and indent properly) below section if proxy (with CA cert)
required to connect Qualys Cloud
  #- mountPath: /etc/qualys/qpaa/cert/custom-ca.crt
  # name: proxy-cert-path
# uncomment (and indent properly) below section if using Docker HTTP
socket with TLS
  #- mountPath: /root/.docker
  # name: tls-cert-path
securityContext:
  allowPrivilegeEscalation: false
volumes:
- name: socket-volume
  hostPath:
    path: /var/run/docker.sock
    type: Socket
- name: persistent-volume
  hostPath:
    path: /usr/local/qualys/sensor/data
    type: DirectoryOrCreate
- name: agent-volume
  hostPath:
    path: /etc/qualys
    type: DirectoryOrCreate
# uncomment (and indent properly) below section if proxy (with CA cert)
required to connect Qualys Cloud
  #- name: proxy-cert-path
  # hostPath:
  #   path: <proxy certificate path>
  #   type: File
# uncomment (and indent properly) below section if using Docker HTTP
socket with TLS
  #- name: tls-cert-path
  # hostPath:
  #   path: <Path of directory of client certificates>
  #   type: Directory
hostNetwork: true
```

Qualys Container Sensor DaemonSet should be deployed in 'qualys' namespace as a part of ServiceAccount with adequate permission to communicate with Kubernetes API Server. The Role, ClusterRole, RoleBinding and ClusterRoleBinding are used to assign the necessary permissions to the ServiceAccount. If you already have Qualys Container Sensor running in a different namespace other than 'qualys', you'll need to first uninstall Qualys Container Sensor from the other namespace and then deploy it fresh in 'qualys' namespace.

You'll need these permissions:

get, list, watch - to monitor the resources in 'qualys' namespace

create, delete, deletecollection - to spawn containers for image vulnerability assessment in 'qualys' namespace then clean up after itself

## Modify parameters in the yaml file

Copy the **cssensor-ds.yml** file to Kubernetes cluster's master node then modify it by providing values for the following parameters. In order for the yaml file to work properly, ensure you only update the parameters/sections specified below. Note that you can download the yaml file directly from [https://github.com/Qualys/cs\\_sensor](https://github.com/Qualys/cs_sensor)

Uncomment the **tolerations** section under **spec** if you want Sensor daemonset to be deployed on master nodes.

```
spec:
  #tolerations:
  # this toleration is to have the daemonset runnable on master nodes
  # remove it if want your masters to run sensor pod
  #- key: node-role.kubernetes.io/master
  # effect: NoSchedule

containers:
  - name: qualys-container-sensor
    image: <CS Sensor image name in the private/docker hub registry>
    args: ["--k8s-mode"]
```

**Note:** Make sure all nodes that will be running sensor pod have access to private or docker hub registry where sensor image is stored.

If you want to deploy the sensor for CI/CD environment provide the args value as:

```
args: ["--k8s-mode", "--cicd-deployed-sensor"]
```

If you want to deploy a Registry Sensor provide the args value as:

```
args: ["--k8s-mode", "--registry-sensor"]
```

If you want print logs on the console, provide "--enable-console-logs" as an additional value in args.

If you want to change the log level, provide "--log-level", "<a number between 0 and 5>" as an additional value in args, e.g if you want logs in trace provide:

```
args: ["--k8s-mode", "--log-level", "5"]
```

If you want to launch the sensor with scan thread value other than default 4, provide "--scan-thread-pool-size", "<number of threads>" as an additional value in args.

```
args: ["--k8s-mode", "--scan-thread-pool-size", "6"]
```

If you want to define the number of archived qpa.log files to be generated provide "--log-filepurgecount", "<digit>" as an additional value in args. The default, "--log-filepurgecount", "5" is applied via config. Please note that there will always be current qpa.log file in log/ directory.

If you want to define the number of archived qpa.log files to be generated and size per log file, provide "--log-filesize", "<digit><K/M/>" where "K" means kilobyte and "M" means megabyte, and "--log-filepurgecount", "<digit>" as an additional value in args. Default is "--log-filesize": "10M" and "--log-filepurgecount": "5" applied via config.

```
args: ["--k8s-mode", "--log-filesize", "5M", "--log-filepurgecount", "4"]
```

If you want image scanning pods to be instantiated using kubernetes native `kubectl run` command provide "--use-kubectl" as an additional value in args. In this case sensor uses native kubernetes facilities to launch image scans. When this argument is omitted image containers are launched using `docker run`.

```
args: ["--k8s-mode", "--use-kubectl"]
```

If TLS authentication is enabled, specify docker client certificate, client private key and CA certificate names in the args

```
args: ["--k8s-mode", "--tls-cacert", "<file name of the CA certificate that was used to sign docker server certificate>", "--tls-cert", "<docker client certificate file name>", "--tls-key", "<docker client private key file name>"]
```

**Note:** If any of the three files have a default name such as ca.pem, cert.pem, key.pem respectively the corresponding argument can be omitted.

If you want to mask environment variables for images and containers in sensor logs and in the Container Security UI, add the "--mask-env-variable" parameter to args:

```
args: ["--k8s-mode", "--mask-env-variable"]
```

If you want to disable image scans for General Sensor, add the "--disableImageScan" parameter to args:

```
args: ["--k8s-mode", "--disableImageScan"]
```

If you want to specify a scanning policy, add the "--scanning-policy" parameter to args. The available values for the scanning policy are: "DynamicScanningOnly", "StaticScanningOnly", and "DynamicWithStaticScanningAsFallback".

```
args: ["--k8s-mode", "--scanning-policy", "StaticScanningOnly"]
```

If you want to disable log4j vulnerability scanning on your container images, add the "--disable-log4j-scanning" parameter to args:

```
args: ["--k8s-mode", "--disable-log4j-scanning"]
```

If you want to disable log4j static detection for dynamic/static image scans, add the "--disable-log4j-static-detection" parameter to args:

```
args: ["--k8s-mode", "--disable-log4j-static-detection"]
```

If you want to optimize Image scans for General Sensor, add the "--optimize-image-scans" parameter to args:

```
args: ["--k8s-mode", "--optimize-image-scans"]
```

If you have the [SCA scanning](#) feature, then you can enable SCA scanning for container images by adding the "--perform-sca-scan" parameter to args:

```
args: ["--k8s-mode", "--perform-sca-scan"]
```

By default, SCA scans run in online mode. You can choose to disable Internet access for the SCA scan and run the scan in offline mode. Note - We recommend you run the SCA scan in online mode. Quality of software package enumeration for Java substantially degrades when the SCA scan is run in offline mode. The remote maven repository may need to be consulted for an accurate package detection. This can affect accuracy of the vulnerability posture of the image.

```
args: ["--k8s-mode", "--perform-sca-scan" "--disallow-internet-access-for-sca"]
```

The default SCA scan command timeout is 5 minutes (300 seconds). You can overwrite the default timeout with a new value specified in seconds. For example, you may need to increase the SCA scan timeout when scanning large container images to ensure the SCA scan has time to finish.

**Note:** The --sca-scan-timeout-in-seconds=600 parameter also applies to secret and malware detection.

```
args: ["--k8s-mode", "--perform-sca-scan", "--sca-scan-timeout-in-seconds=600"]
```

If you want to enable secret detection for container images, add the "--perform-secret-detection" parameter to args. Note that secret detection is supported only on CICD and registry sensors.

```
args: ["--k8s-mode", "--container-runtime", "containerd", "--
```

```
perform-secret-detection"]
```

If you want to enable malware detection for container images, add the "--perform-malware-detection" parameter to args. Note that malware detection is supported only on the registry sensor.

```
args: ["--k8s-mode", "--registry-sensor", "--perform-malware-  
detection"]
```

Under **resources** specify the following:

```
resources:  
limits:  
cpu: "0.2" # Default CPU usage limit on each node for sensor.
```

For example, for limiting the CPU usage to 5%, set resources:limits:cpu: "0.05". This limits the CPU usage to 5% of one core on the host.

If there are multiple processors on a node, setting the resources:limits:cpu value applies the CPU limit to one core only. For example, if you have 4 CPUs on the system and you want to set CPU limit as 20% of overall CPU capacity, then the CPU limit should be set to 0.8 i.e., 80% of one core only which becomes 20% of total CPU capacity.

To disable any CPU usage limit, set resources:limits:cpu value to 0.

Optionally, if you want to specify the memory resources for Container Sensor, you can specify it under **resources**. Recommended values for the Container Sensor's memory requests and memory limits are:

```
resources:  
limits:  
  cpu: "0.2" # Default CPU usage limit on each node for sensor  
  memory: "500Mi"  
requests:  
  memory: "300Mi"
```

When either of the memory resource values (limits or requests) is specified for Container Sensor and "--use-kubectrl" is supplied in args, we automatically apply both memory requests and memory limits to image scanning containers. Default values are 200Mi and 700Mi, respectively.

Additionally, you could overwrite one or both values by specifying the following variables under **env**. In this example, the values were changed to 300Mi and 800Mi.

```
- name: QUALYS_SCANNING_CONTAINER_MEMORYREQUESTMB  
  value: "300Mi"  
- name: QUALYS_SCANNING_CONTAINER_MEMORYLIMITMB  
  value: "800Mi"
```

Under **env** specify the following:

Activation ID (Required)

```
- name: ACTIVATIONID
  value: XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX
```

Customer ID (Required)

```
- name: CUSTOMERID
  value: XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX
```

Specify `POD_URL` when using docker hub image. Otherwise, remove it.

```
- name: POD_URL
  value: <Specify POD URL>
```

Specify the scanning container launch timeout in minutes. If this env variable is not present, then 10 minutes is the default.

```
- name: QUALYS_SCANNING_CONTAINER_LAUNCH_TIMEOUT
  value: "10"
```

To enable TLS authentication uncomment the following 2 lines.

```
- name: DOCKER_TLS_VERIFY
  value: "1"
```

**Note:** To disable TLS use `DOCKER_TLS_VERIFY=""`(empty string) or remove it or keep it commented in yml file.

**Note:** By enabling sensor communication with docker daemon over TLS customer can restrict the sensor's access to docker socket by using docker authorization plugin.

**Note:** When TLS authentication is enabled and `DOCKER_HOST` is not specified the sensor will automatically detect the FQDN of the worker node it is running on and set `DOCKER_HOST` environment variable inside the sensor to `<worker node's FQDN>:<2376>` where 2376 is the default TLS TCP port for docker daemon

**Note:** You can set `DOCKER_HOST` yourself to `127.0.0.1:<port#>` or `localhost:<port#>` by adding:

```
- name: DOCKER_HOST
  value: "<loopback IPv4 address or hostname>:<port#>"
```

**Note:** Please make sure that FQDN, or hostname, or IPv4 address set in the `DOCKER_HOST` matches the CN or Subject Alternative Name in the docker server certificate on each worker node

Uncomment `tls-cert-path` under volumes if TLS authentication needs to be enabled and provide directory path for client certificates, or keep it as is if not required:

```
#- name: tls-cert-path
#   hostPath:
```

```
# path: <Path of directory of client certificates>
# type: Directory
```

With each scan, we check the node status to see if the node is schedulable or not, and launch the scan only if the node is schedulable. If the node status indicates that the node is unschedulable, then we retry the scan after a default interval of 15 minutes. You can increase or decrease the time the sensor waits before retrying the scan by specifying a different scan retry interval in minutes.

```
- name: UNSCHEDULABLE_NODE_SCAN_RETRY_INTERVAL
  value: "30"
```

Uncomment below part under **volumeMounts** as well if you are using TLS. Otherwise, keep it commented out.

```
#- mountPath: /root/.docker
#name: tls-cert-path
```

Uncomment and indent properly the proxy information, or keep it as is if not required:

- For the communication between the sensor and the backend (Container Management Service):

```
#- name: qualys_https_proxy
# value: <proxy FQDN or Ip address>:<port#>
```

- For a registry sensor version 1.21.0 or later used for a public registry:

```
#- name: https_proxy
# value: <proxy FQDN or Ip address>:<port#>
```

Uncomment proxy-cert-path under **volumes**, or keep it as is if not required:

```
#- name: proxy-cert-path
# hostPath:
# path: /root/cert/proxy-certificate.crt
# type: File
```

Activation ID and Customer ID are required. Use the Activation ID and Customer ID from your subscription. To get the Activation ID and Customer ID, login to the Container Security UI, go to Configurations > Sensors, click Download, and then click any sensor type. The installation command on the Installation Instructions screen contains your Activation ID and Customer ID. Activation ID is like a password, do not share it.

If you are using an https proxy, ensure that all Kubernetes nodes have a valid certificate file for the sensor to communicate with the Container Management Server.



If you are not using a proxy and you have kept the above-mentioned parts commented, you can keep the following part commented from **volumeMounts** as well:

```
#- mountPath: /etc/qualys/qpa/cert/custom-ca.crt
#   name: proxy-cert-path
```

Once you have modified the **cssensor-ds.yml** file, run the following command on Kubernetes master to create a DaemonSet:

```
kubectl create -f cssensor-ds.yml
```

If you need to uninstall Qualys Container Sensor, run the following command on Kubernetes master:

```
kubectl delete -f cssensor-ds.yml
```

**Note:** The persistent storage will need to be removed manually on each worker node.

## Using Persistent Volume Claims

You can use PersistentVolumeClaim (PVC) to request for storage of specific size from the gross Persistent Volume you have specified.

## Modify the **cssensor-ds\_pv\_pvc.yml** file

Below is the **cssensor-ds\_pv\_pvc.yml**. For customers' convenience, the **cssensor-ds\_pv\_pvc.yml** file can be extracted from QualysContainerSensor.tar.xz. Note that you can download the yaml file directly from [https://github.com/Qualys/cs\\_sensor](https://github.com/Qualys/cs_sensor)

IMPORTANT: The field alignment in the yaml file is very important. Please make sure to honor the formatting provided in the template.

```
kind: List
apiVersion: v1
items:
  - kind: Namespace
    apiVersion: v1
    metadata:
      name: qualys
  - kind: PersistentVolume
    apiVersion: v1
    metadata:
      name: qualys-sensor-pv-volume
      labels:
        type: local
    spec:
      storageClassName: manual
      capacity:
        storage: 5Gi
      accessModes:
```

```
    - ReadWriteOnce
  hostPath:
    path: "/mnt/data/"
- kind: PersistentVolumeClaim
  apiVersion: v1
  metadata:
    name: qualys-sensor-pv-claim
    namespace: qualys
  spec:
    storageClassName: manual
    accessModes:
      - ReadWriteOnce
    resources:
      requests:
        storage: 1Gi
# Service Account
- kind: ServiceAccount
  apiVersion: v1
  metadata:
    name: qualys-service-account
    namespace: qualys
# Role for read/write/delete permission to qualys namespace
- kind: Role
  # if k8s version is 1.17 and earlier then change apiVersion to
  # "rbac.authorization.k8s.io/v1beta1"
  apiVersion: rbac.authorization.k8s.io/v1
  metadata:
    name: qualys-reader-role
    namespace: qualys
  rules:
    - apiGroups: ["", "batch"]
      resources: ["pods", "jobs"]
      verbs: ["get", "list", "watch", "create", "delete",
"deletecollection"]
    - apiGroups: [""]
      resources: ["pods/status"]
      verbs: ["get"]
    - apiGroups: [""]
      resources: ["pods/attach", "pods/exec"]
      verbs: ["create"]
- kind: ClusterRole
  # if k8s version is 1.17 and earlier then change apiVersion to
  # "rbac.authorization.k8s.io/v1beta1"
  apiVersion: rbac.authorization.k8s.io/v1
  metadata:
    name: qualys-cluster-reader-role
```

```
rules:
- apiGroups: [""]
  resources: ["nodes", "pods/status",
"replicationcontrollers/status", "nodes/status"]
  verbs: ["get"]
- apiGroups: ["apps"]
  resources: ["replicasets/status", "daemonsets/status",
"deployments/status", "statefulsets/status"]
  verbs: ["get"]
- apiGroups: ["batch"]
  resources: ["jobs/status", "cronjobs/status"]
  verbs: ["get"]
# RoleBinding to assign permissions in qualys-reader-role to qualys-
service-account
- kind: RoleBinding
  # if k8s version is 1.17 and earlier then change apiVersion to
"rbac.authorization.k8s.io/v1beta1"
  apiVersion: rbac.authorization.k8s.io/v1
  metadata:
    name: qualys-reader-role-rb
    namespace: qualys
  subjects:
- kind: ServiceAccount
  name: qualys-service-account
  namespace: qualys
  roleRef:
    kind: Role
    name: qualys-reader-role
    apiGroup: rbac.authorization.k8s.io
- kind: ClusterRoleBinding
  # if k8s version is 1.17 and earlier then change apiVersion to
"rbac.authorization.k8s.io/v1beta1"
  apiVersion: rbac.authorization.k8s.io/v1
  metadata:
    name: qualys-cluster-reader-rb
  subjects:
- kind: ServiceAccount
  name: qualys-service-account
  namespace: qualys
  roleRef:
    kind: ClusterRole
    name: qualys-cluster-reader-role
    apiGroup: rbac.authorization.k8s.io
# Qualys Container Sensor pod with
- apiVersion: apps/v1
  kind: DaemonSet
```

```
metadata:
  name: qualys-container-sensor
  namespace: qualys
  labels:
    k8s-app: qualys-cs-sensor
spec:
  selector:
    matchLabels:
      name: qualys-container-sensor
  updateStrategy:
    type: RollingUpdate
  template:
    metadata:
      labels:
        name: qualys-container-sensor
    spec:
      #tolerations:
      # this toleration is to have the daemonset runnable on master
nodes
  # remove it if want your masters to run sensor pod
  #- key: node-role.kubernetes.io/master
  # effect: NoSchedule
  serviceAccountName: qualys-service-account
  containers:
  - name: qualys-container-sensor
    image: qualys/qcs-sensor:latest
    imagePullPolicy : IfNotPresent
    resources:
      limits:
        cpu: "0.2" # Default CPU usage limit on each node for
sensor.
      args: ["--k8s-mode"]
      env:
      - name: CUSTOMERID
        value: __customerId
      - name: ACTIVATIONID
        value: __activationId
      - name: POD_URL
        value:
      - name: QUALYS_SCANNING_CONTAINER_LAUNCH_TIMEOUT
        value: "10"
# uncomment(and indent properly) below section if using Docker HTTP
socket with TLS
  #- name: DOCKER_TLS_VERIFY
  # value: "1"
# uncomment(and indent properly) below section if proxy is required to
```

```
connect Qualys Cloud
  #- name: qualys_https_proxy
  # value: <proxy FQDN or Ip address>:<port#>
  - name: QUALYS_POD_NAME
    valueFrom:
      fieldRef:
        fieldPath: metadata.name
  - name: QUALYS_POD_NAMESPACE
    valueFrom:
      fieldRef:
        fieldPath: metadata.namespace
volumeMounts:
  - mountPath: /var/run/docker.sock
    name: socket-volume
    readOnly: true
  - mountPath: /usr/local/qualys/qpa/data
    name: persistent-volume
  - mountPath: /usr/local/qualys/qpa/data/conf/agent-data
    name: agent-volume
# uncomment (and indent properly) below section if proxy (with CA cert)
required to connect Qualys Cloud
  #- mountPath: /etc/qualys/qpa/cert/custom-ca.crt
  # name: proxy-cert-path
# uncomment (and indent properly) below section if using Docker HTTP
socket with TLS
  #- mountPath: /root/.docker
  # name: tls-cert-path
securityContext:
  allowPrivilegeEscalation: false
volumes:
  - name: socket-volume
    hostPath:
      path: /var/run/docker.sock
      type: Socket
  - name: persistent-volume
    persistentVolumeClaim:
      claimName: qualys-sensor-pv-claim
  - name: agent-volume
    hostPath:
      path: /etc/qualys
      type: DirectoryOrCreate
# uncomment (and indent properly) below section if proxy (with CA cert)
required to connect Qualys Cloud
  #- name: proxy-cert-path
  # hostPath:
  #   path: <proxy certificate path>
```

```
        #    type: File
# uncomment (and indent properly) below section if using Docker HTTP
socket with TLS
        #- name: tls-cert-path
        #    hostPath:
        #        path: <Path of directory of client certificates>
        #    type: Directory
hostNetwork: true
```

## Modify parameters in the yaml file

Modify the `cssensor-ds_pv_pvc.yml` file to provide values for the following parameters. In order for the yml file to work properly, ensure that you do not remove/comment the respective sections mentioned below. Note that you can download the yml file directly from [https://github.com/Qualys/cs\\_sensor](https://github.com/Qualys/cs_sensor)

```
- kind: PersistentVolume
  apiVersion: v1
  metadata:
    name: qualys-sensor-pv-volume
    labels:
      type: local
  spec:
    storageClassName: manual
    capacity:
      storage: 5Gi
    accessModes:
      - ReadWriteOnce
    hostPath:
      path: "/mnt/data/"
- kind: PersistentVolumeClaim
  apiVersion: v1
  metadata:
    name: qualys-sensor-pv-claim
    namespace: qualys
  spec:
    storageClassName: manual
    accessModes:
      - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

Here a PVC of 1Gi is made on a Persistent Volume to 5Gi. [Click here](#) for a list of supported Persistent Volume Types.

Add the name of the PVC under **volumes**:

```
- name: persistent-volume
  persistentVolumeClaim:
    claimName: qualys-sensor-pv-claim
```

## Launch sensor without persistent storage

You can run the sensor without using persistent storage on host. In this case data is not stored on host but stored at the `/usr/local/qualys/qpq/data` folder relative to the Sensor.

To launch sensor without persistent storage, modify the `cssensor-ds.yml` file and provide `--sensor-without-persistent-storage` as an additional value in `args`.

```
args: ["--k8s-mode", "--sensor-without-persistent-storage"]
```

It is recommended to use the `--enable-console-logs` option along with `--sensor-without-persistent-storage` to preserve the logs.

Under `volumeMounts` remove/comment the persistent-volume section.

```
volumeMounts:
- mountPath: /usr/local/qualys/qpq/data
  name: persistent-volume
```

Under `volumes` remove/comment the persistent-volume section.

```
volumes:
- name: persistent-volume
  hostPath:
    path: /usr/local/qualys/sensor/data
    type: DirectoryOrCreate
```

## Deploy in Kubernetes - Containerd Runtime

This section assumes you have the sensor image: [Obtain the Container Sensor Image](#)

### Modify the `cssensor-containerd-ds.yml` file

Below is the Kubernetes DaemonSet deployment template that can be used to deploy the Qualys Container Sensor. For customers' convenience this template is available in the `QualysContainerSensor.tar.xz`. Note that you can download the yml file directly from [https://github.com/Qualys/cs\\_sensor](https://github.com/Qualys/cs_sensor)

IMPORTANT: The field alignment in the yml file is very important. Please make sure to honor the formatting provided in the template.

```
kind: List
apiVersion: v1
items:
  # Create custom namespace qualys
  - kind: Namespace
    apiVersion: v1
    metadata:
      name: qualys
  # Service Account
  - kind: ServiceAccount
    apiVersion: v1
    metadata:
      name: qualys-service-account
      namespace: qualys
  # Role for all permission to qualys namespace
  - kind: Role
    # if k8s version is 1.17 and earlier then change apiVersion to
    "rbac.authorization.k8s.io/v1beta1"
    apiVersion: rbac.authorization.k8s.io/v1
    metadata:
      name: qualys-reader-role
      namespace: qualys
    rules:
      - apiGroups: [ "", "batch" ]
        resources: [ "pods", "jobs" ]
        verbs: [ "get", "list", "watch", "create", "delete",
"deletecollection" ]
      - apiGroups: [ "" ]
        resources: [ "pods/attach", "pods/exec" ]
        verbs: [ "create" ]
  # ClusterRole for read permission to whole cluster
  - kind: ClusterRole
    # if k8s version is 1.17 and earlier then change apiVersion to
    "rbac.authorization.k8s.io/v1beta1"
```



```
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: qualys-cluster-reader-role
rules:
- apiGroups: [""]
  resources: ["pods/exec"]
  verbs: ["create"]
- apiGroups: [""]
  resources: ["nodes", "pods", "pods/status",
"replicationcontrollers/status", "nodes/status"]
  verbs: ["get"]
- apiGroups: ["apps"]
  resources: ["replicasets/status", "daemonsets/status",
"deployments/status", "statefulsets/status"]
  verbs: ["get"]
- apiGroups: ["batch"]
  resources: ["jobs/status", "cronjobs/status"]
  verbs: ["get"]
# RoleBinding to assign permissions in qualys-reader-role to qualys-
service-account
- kind: RoleBinding
  # if k8s version is 1.17 and earlier then change apiVersion to
"rbac.authorization.k8s.io/v1beta1"
  apiVersion: rbac.authorization.k8s.io/v1
  metadata:
    name: qualys-reader-rb
    namespace: qualys
  subjects:
- kind: ServiceAccount
  name: qualys-service-account
  namespace: qualys
  roleRef:
    kind: Role
    name: qualys-reader-role
    apiGroup: rbac.authorization.k8s.io
# ClusterRoleBinding to assign permissions in qualys-cluster-reader-
role to qualys-service-account
- kind: ClusterRoleBinding
  # if k8s version is 1.17 and earlier then change apiVersion to
"rbac.authorization.k8s.io/v1beta1"
  apiVersion: rbac.authorization.k8s.io/v1
  metadata:
    name: qualys-cluster-reader-rb
  subjects:
- kind: ServiceAccount
  name: qualys-service-account
```

```
    namespace: qualys
  roleRef:
    kind: ClusterRole
    name: qualys-cluster-reader-role
    apiGroup: rbac.authorization.k8s.io
# Qualys Container Sensor pod with
- apiVersion: apps/v1
  kind: DaemonSet
  metadata:
    name: qualys-container-sensor
    namespace: qualys
    labels:
      k8s-app: qualys-cs-sensor
  spec:
    selector:
      matchLabels:
        name: qualys-container-sensor
    updateStrategy:
      type: RollingUpdate
    template:
      metadata:
        labels:
          name: qualys-container-sensor
      spec:
        #tolerations:
        # this toleration is to have the daemonset runnable on master
nodes
  # remove it if want your masters to run sensor pod
  #- key: node-role.kubernetes.io/master
  # effect: NoSchedule
  serviceAccountName: qualys-service-account
  containers:
  - name: qualys-container-sensor
    image: qualys/qcs-sensor:latest
    imagePullPolicy : IfNotPresent
    resources:
      limits:
        cpu: "0.2" # Default CPU usage limit on each node for
sensor.
    args: ["--k8s-mode", "--container-runtime", "containerd"]
    env:
      - name: CUSTOMERID
        value: __customerId
      - name: ACTIVATIONID
        value: __activationId
      - name: POD_URL
```

```
    value:
  - name: QUALYS_SCANNING_CONTAINER_LAUNCH_TIMEOUT
    value: "10"
# uncomment(and indent properly) below section if proxy is required to
connect Qualys Cloud
  #- name: qualys_https_proxy
  # value: <proxy FQDN or Ip address>:<port#>
  - name: QUALYS_POD_NAME
    valueFrom:
      fieldRef:
        fieldPath: metadata.name
  - name: QUALYS_POD_NAMESPACE
    valueFrom:
      fieldRef:
        fieldPath: metadata.namespace
volumeMounts:
  - mountPath: /var/run/containerd/containerd.sock
    name: socket-volume
    readOnly: true
  - mountPath: /usr/local/qualys/qpa/data
    name: persistent-volume
  - mountPath: /usr/local/qualys/qpa/data/conf/agent-data
    name: agent-volume
# uncomment(and indent properly) below section if proxy(with CA cert)
required to connect Qualys Cloud
  #- mountPath: /etc/qualys/qpa/cert/custom-ca.crt
  # name: proxy-cert-path
securityContext:
  allowPrivilegeEscalation: false
volumes:
  - name: socket-volume
    hostPath:
      path: /var/run/containerd/containerd.sock
      type: Socket
  - name: persistent-volume
    hostPath:
      path: /usr/local/qualys/sensor/data
      type: DirectoryOrCreate
  - name: agent-volume
    hostPath:
      path: /etc/qualys
      type: DirectoryOrCreate
# uncomment(and indent properly) below section if proxy(with CA cert)
required to connect Qualys Cloud
  #- name: proxy-cert-path
  # hostPath:
```

```
# path: <proxy certificate path>
# type: File
hostNetwork: true
```

Qualys Container Sensor DaemonSet should be deployed in 'qualys' namespace as part of ServiceAccount with adequate permission to communicate with Kubernetes API Server. The Role, ClusterRole, RoleBinding and ClusterRoleBinding are used to assign the necessary permissions to ServiceAccount. If you already have Qualys Container Sensor running in a different namespace other than 'qualys', you'll need to uninstall Qualys Container Sensor from the other namespace and deploy it fresh in the 'qualys' namespace.

You'll need these permissions:

get, list, watch - to monitor the resources to be scanned for vulnerabilities across the cluster

create, delete, deletecollection - to spawn containers for image vulnerability assessment in 'qualys' namespace, scan pods across the cluster and then clean up after itself

**Note:** Ensure that the Container Sensor has read and write access to the persistent storage and the containerd daemon socket.

## Modify parameters in the yaml file

Copy the **cssensor-containerd-ds.yml** file to Kubernetes cluster's master node then modify it by providing values for the following parameters. In order for the yaml file to work properly, ensure you only update the parameters/sections specified below. Note that you can download the yaml file directly from [https://github.com/Qualys/cs\\_sensor](https://github.com/Qualys/cs_sensor)

General Sensor is assumed here. Uncomment the **tolerations** section under **spec** if you want Sensor daemonset to be deployed on master nodes.

```
spec:
  #tolerations:
  # this toleration is to have the daemonset runnable on master nodes
  # remove it if want your masters to run sensor pod
  #- key: node-role.kubernetes.io/master
  # effect: NoSchedule
```

If you want to prioritize Qualys Sensor PODs:

Locate and Uncomment the below lines of code in the .yaml file. And change the PriorityClass value mentioned under kind: PriorityClass as per your requirement.

```
#- kind: PriorityClass
# apiVersion: scheduling.k8s.io/v1
# metadata:
#   name: qualys-priority-class
# value: 0
# preemptionPolicy: PreemptLowerPriority
# description: Priority class for daemonset
```

Also, locate the PriorityClass name present below and Uncomment it.

```
#priorityClassName: qualys-priority-class
```

```
containers:
```

```
- name: qualys-container-sensor
  image: <CS Sensor image name in the private/docker hub registry>
  args: ["--k8s-mode", "--container-runtime", "containerd"]
```

If you want to deploy a Registry Sensor provide the args value as:

```
args: ["--k8s-mode", "--container-runtime", "containerd", "--
registry-sensor"]
```

If you want to deploy a CICD Sensor provide the args value as:

```
args: ["--k8s-mode", "--container-runtime", "containerd", "--cicd-
deployed-sensor"]
```

If you want to change the log level, provide "--log-level", "<a number between 0 and 5>" as an additional value in args, e.g if you want logs in trace provide:

```
args: ["--k8s-mode", "--container-runtime", "containerd", "--log-
level", "5"]
```

If you want to launch the sensor with scan thread value other than default 4, provide "--scan-thread-pool-size", "<number of threads>" as an additional value in args.

```
args: ["--k8s-mode", "--container-runtime", "containerd", "--scan-
thread-pool-size", "6"]
```

If you want to define the number of archived qpa.log files to be generated and size per log file, provide "--log-filesize", "<digit><K/M/>" where "K" means kilobyte and "M" means megabyte, and "--log-filepurgecount", "<digit>" as an additional value in args. Default is "--log-filesize": "10M" and "--log-filepurgecount": "5" applied via config.

"--log-filesize": can be used to define the maximum size per log file. For example, "10K" (kilobytes), "10M" (megabytes) or "10" (bytes).

"--log-filepurgecount": can be used to define the number of archived log files to be generated, please note that there will always be current qpa.log file in log/ directory.

```
args: ["--k8s-mode", "--container-runtime", "containerd", "--log-
filesize", "5M", "--log-filepurgecount", "4"]
```

If you want to mask environment variables for images and containers in sensor logs and in the Container Security UI, add the "--mask-env-variable" parameter to args:

```
args: ["--k8s-mode", "--container-runtime", "containerd", "--mask-
env-variable"]
```

If you want to disable image scans for General Sensor, add the "--disableImageScan" parameter to args:

```
args: ["--k8s-mode", "--container-runtime", "containerd", "--  
disableImageScan"]
```

If you want to specify a scanning policy, add the "--scanning-policy" parameter to args. The available values for the scanning policy are: "DynamicScanningOnly", "StaticScanningOnly", and "DynamicWithStaticScanningAsFallback".

```
args: ["--k8s-mode", "--scanning-policy", "StaticScanningOnly"]
```

If you want to disable log4j vulnerability scanning on your container images, add the "--disable-log4j-scanning" parameter to args:

```
args: ["--k8s-mode", "--container-runtime", "containerd", "--  
disable-log4j-scanning"]
```

If you want to disable log4j static detection for dynamic/static image scans, add the "--disable-log4j-static-detection" parameter to args:

```
args: ["--k8s-mode", "--container-runtime", "containerd", "--  
disable-log4j-static-detection"]
```

If you want to optimize Image scans for General Sensor, add the "--optimize-image-scans" parameter to args:

```
args: ["--k8s-mode", "--container-runtime", "containerd", "--  
optimize-image-scans"]
```

If you want to enable secret detection for container images, add the "--perform-secret-detection" parameter to args. Note that secret detection is supported only on CICD and registry sensors.

```
args: ["--k8s-mode", "--container-runtime", "containerd", "--  
perform-secret-detection"]
```

If you want to enable malware detection for container images, add the "--perform-malware-detection" parameter to args. Note that malware detection is supported only on registry sensor.

```
args: ["--k8s-mode", "--registry-sensor", "--perform-malware-  
detection"]
```

If you want to scan the secured private registry (HTTPS) having a self-signed certificate, you can either provide the certificate through the yaml file or use the "--insecure-registry" argument to ignore the certificate check.

In case you provide both - the certificate through the yaml file and you use "--insecure-registry" argument - you will be able to login to the secured private registry.

```
args: ["--k8s-mode", "--container-runtime", "containerd", "--registry-sensor", "--insecure-registry"]
```

To provide self-signed certificate through the Yaml file, follow the steps mentioned below.

1. Open the .yaml file, and search "volumeMounts:".
2. Uncomment the following lines from the code.

```
# name: registry-cert-volume  
# subPath: ca.crt  
# readOnly: true
```

3. Similarly, uncomment the following lines and save the file.

```
# - name: registry-cert-volume  
# secret:  
#   secretName: cert-config
```

If you want to limit the memory usage of sensor container, add the "--limit-resource-usage" parameter to args.

```
args: ["--k8s-mode", "--limit-resource-usage"]
```

Under **resources** specify the following:

```
resources:  
  limits:  
    cpu: 0.2 # Default CPU usage limit (20% of one core on the host).
```

For example, for limiting the cpu usage to 5%, set resources:limits:cpu: "0.05". This limits the cpu usage to 5% of one core on the host.

If there are multiple processors on a node, setting the resources:limits:cpu value applies the CPU limit to one core only. For example, if you have 4 CPUs on the system and you want to set CPU limit as 20% of overall CPU capacity, then the CPU limit should be set to 0.8 i.e., 80% of one core only which becomes 20% of total CPU capacity.

To disable any CPU usage limit, set resources:limits:cpu value to 0.

Optionally, if you want to specify the memory resources for Container Sensor, you can specify it under **resources**.

Recommended values for the Container Sensor's memory requests and memory limits for a VM Scan are:

```
resources:  
  limits:  
    cpu: "0.2" # Default CPU usage limit on each node for sensor
```

```
memory: "500Mi"  
requests:  
  memory: "300Mi"
```

If you want to specify the memory resources for SCA Scan, or Secret Scan, or Malware Scan you can specify it under **resources**. It is recommended to use "--limit-resource-usage" argument while installing the sensor.

Recommended values for the Container Sensor's memory requests and memory limits for above mentioned scan types are:

```
resources:  
  limits:  
    cpu: "0.2" # Default CPU usage limit on each node for sensor  
    memory: "3.5Gi"  
  requests:  
    memory: "1.5Gi"
```

Disclaimer: The above recommendation is based on the average image size of 1.5GB, if the image sizes are more, please increase the memory limit accordingly.

When either of the memory resource values (limits or requests) is specified for Container Sensor, we automatically apply both memory requests and memory limits to image scanning containers. Default values are 200Mi and 700Mi, respectively.

Additionally, you could overwrite one or both values by specifying the following variables under **env**. In this example, the values were changed to 300Mi and 800Mi.

```
- name: QUALYS_SCANNING_CONTAINER_MEMORYREQUESTMB  
  value: "300Mi"  
- name: QUALYS_SCANNING_CONTAINER_MEMORYLIMITMB  
  value: "800Mi"
```

Under **env** specify the following:

Activation ID (Required)

```
- name: ACTIVATIONID  
  value: XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX
```

Customer ID (Required)

```
- name: CUSTOMERID  
  value: XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX
```

Specify `POD_URL` when using docker hub image. Otherwise, remove it.

```
- name: POD_URL  
  value: <Specify POD URL>
```



Specify the scanning container launch timeout in minutes. If this env variable is not present, then 10 minutes is the default.

```
- name: QUALYS_SCANNING_CONTAINER_LAUNCH_TIMEOUT
  value: "10"
```

With each scan, we check the node status to see if the node is schedulable or not, and launch the scan only if the node is schedulable. If the node status indicates that the node is unschedulable, then we retry the scan after a default interval of 15 minutes. You can increase or decrease the time the sensor waits before retrying the scan by specifying a different scan retry interval in minutes.

```
- name: UNSCHEDULABLE_NODE_SCAN_RETRY_INTERVAL
  value: "30"
```

Uncomment and indent properly the proxy information, or keep it as is if not required:

- For the communication between the sensor and the backend (Container Management Service):

```
#- name: qualys_https_proxy
#   value: <proxy FQDN or Ip address>:<port#>
```

- For a registry sensor version 1.21.0 or later used for a public registry:

```
#- name: https_proxy
#   value: <proxy FQDN or Ip address>:<port#>
```

Uncomment proxy-cert-path under **volumes**, or keep it as is if not required:

```
#- name: proxy-cert-path
#   hostPath:
#     path: /root/cert/proxy-certificate.crt
#     type: File
```

Activation ID and Customer ID are required. Use the Activation ID and Customer ID from your subscription. To get the Activation ID and Customer ID, login to the Container Security UI, go to Configurations > Sensors, click Download, and then click any sensor type. The installation command on the Installation Instructions screen contains your Activation ID and Customer ID. Activation ID is like a password, do not share it.

If you are using an https proxy, ensure that all Kubernetes nodes have a valid certificate file for the sensor to communicate with the Container Management Server.

If you are not using a proxy and you have kept the above-mentioned parts commented, you can keep the following part commented from **volumeMounts** as well:

```
#- mountPath: /etc/qualys/qp/cert/custom-ca.crt
#   name: proxy-cert-path
```

## How to deploy the Container Sensor DaemonSet

Once you have modified the **cssensor-containerd-ds.yml** file, run the following command on Kubernetes master to create a DaemonSet:

```
kubectl create -f cssensor-containerd-ds.yml
```

## How to remove the Container Sensor DaemonSet

To uninstall Qualys Container Sensor, run the following command on Kubernetes master:

```
kubectl delete -f cssensor-containerd-ds.yml
```

**Note:** The persistent storage will need to be removed manually on each worker node.

## Launch sensor without persistent storage

You can run the sensor without using persistent storage on host. In this case data is not stored on host but stored at the `/usr/local/qualys/qpa/data` folder relative to the Sensor.

To launch sensor without persistent storage, modify the **cssensor-containerd-ds.yml** file and provide `--sensor-without-persistent-storage` as an additional value in **args**.

```
args: ["--k8s-mode", "--container-runtime", "containerd", "--sensor-without-persistent-storage"]
```

It is recommended to use the `--enable-console-logs` option along with `--sensor-without-persistent-storage` to preserve the logs.

Under **volumeMounts** remove/comment the persistent-volume section.

```
volumeMounts:  
- mountPath: /usr/local/qualys/qpa/data  
  name: persistent-volume
```

Under **volumes** remove/comment the persistent-volume section.

```
volumes:  
- name: persistent-volume  
  hostPath:  
    path: /usr/local/qualys/sensor/data  
    type: DirectoryOrCreate
```

## How to Tag Target Images for the CICD Sensor

To tag CICD images with `qualys_scan_target:<tag>` to mark them for scanning, use the **nerdctl** tool. Use `<image-repo:tag>` for tagging an image with `qualys_scan_target:<any-tag>`.

```
nerdctl -n k8s.io tag <image-repo:tag> qualys_scan_target:<any-tag>
```

For example,

```
./nerdctl -n k8s.io tag docker.io/library/known:latest  
qualys_scan_target:known
```

**Notes:**

- The nerdctl binary must be available on the host.
- You must create target images in the k8s.io namespace only.

## Deploy in Kubernetes - CRI-O Runtime

**Note:** If you have CRI-O runtime on OpenShift, please follow instructions under [Deploy in Kubernetes - OpenShift4.4+ with CRI-O Runtime](#).

This section assumes you have the sensor image: [Obtain the Container Sensor Image](#)

**Note:** CRI-O runtime is supported by the General (host) sensor and Registry sensor. It is not supported by Build CI/CD sensor.

### Modify the `cssensor-crio-ds.yml` file

Below is the Kubernetes DaemonSet deployment template that can be used to deploy the Qualys Container Sensor. For customers' convenience this template is available in the `QualysContainerSensor.tar.xz`. Note that you can download the yml file directly from [https://github.com/Qualys/cs\\_sensor](https://github.com/Qualys/cs_sensor)

IMPORTANT: The field alignment in the yml file is very important. Please make sure to honor the formatting provided in the template.

```
kind: List  
apiVersion: v1  
items:  
  # Create custom namespace qualys  
  - kind: Namespace  
    apiVersion: v1  
    metadata:  
      name: qualys  
  # Service Account  
  - kind: ServiceAccount  
    apiVersion: v1  
    metadata:  
      name: qualys-service-account  
      namespace: qualys  
  # Role for all permission to qualys namespace  
  - kind: Role  
    # if k8s version is 1.17 and earlier then change apiVersion to  
    "rbac.authorization.k8s.io/v1beta1"  
    apiVersion: rbac.authorization.k8s.io/v1
```

```
metadata:
  name: qualys-reader-role
  namespace: qualys
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["create", "delete", "deletecollection"]
- apiGroups: ["batch"]
  resources: ["jobs"]
  verbs: ["get", "create", "delete", "deletecollection"]
- apiGroups: [""]
  resources: ["pods/attach"]
  verbs: ["create"]
# ClusterRole for read permission to whole cluster
- kind: ClusterRole
  # if k8s version is 1.17 and earlier then change apiVersion to
  "rbac.authorization.k8s.io/v1beta1"
  apiVersion: rbac.authorization.k8s.io/v1
  metadata:
    name: qualys-cluster-reader-role
  rules:
  - apiGroups: [""]
    resources: ["nodes", "pods/status",
"replicationcontrollers/status", "nodes/status"]
    verbs: ["get"]
  - apiGroups: [""]
    resources: ["pods"]
    verbs: ["get", "list", "watch"]
  - apiGroups: [""]
    resources: ["pods/exec"]
    verbs: ["create"]
  - apiGroups: ["apps"]
    resources: ["replicasets/status", "daemonsets/status",
"deployments/status", "statefulsets/status"]
    verbs: ["get"]
  - apiGroups: ["batch"]
    resources: ["jobs/status", "cronjobs/status"]
    verbs: ["get"]
# RoleBinding to assign permissions in qualys-reader-role to qualys-
service-account
- kind: RoleBinding
  # if k8s version is 1.17 and earlier then change apiVersion to
  "rbac.authorization.k8s.io/v1beta1"
  apiVersion: rbac.authorization.k8s.io/v1
  metadata:
    name: qualys-reader-rb
```

```
    namespace: qualys
subjects:
- kind: ServiceAccount
  name: qualys-service-account
  namespace: qualys
roleRef:
  kind: Role
  name: qualys-reader-role
  apiGroup: rbac.authorization.k8s.io
# ClusterRoleBinding to assign permissions in qualys-cluster-reader-
role to qualys-service-account
- kind: ClusterRoleBinding
  # if k8s version is 1.17 and earlier then change apiVersion to
"rbac.authorization.k8s.io/v1beta1"
  apiVersion: rbac.authorization.k8s.io/v1
  metadata:
    name: qualys-cluster-reader-rb
  subjects:
- kind: ServiceAccount
  name: qualys-service-account
  namespace: qualys
roleRef:
  kind: ClusterRole
  name: qualys-cluster-reader-role
  apiGroup: rbac.authorization.k8s.io
# Qualys Container Sensor pod with
- apiVersion: apps/v1
  kind: DaemonSet
  metadata:
    name: qualys-container-sensor
    namespace: qualys
    labels:
      k8s-app: qualys-cs-sensor
  spec:
    selector:
      matchLabels:
        name: qualys-container-sensor
    updateStrategy:
      type: RollingUpdate
    template:
      metadata:
        labels:
          name: qualys-container-sensor
      spec:
        #tolerations:
        # this toleration is to have the daemonset runnable on master
```

```
nodes
    # remove it if want your masters to run sensor pod
    #- key: node-role.kubernetes.io/master
    # effect: NoSchedule
    serviceAccountName: qualys-service-account
    containers:
    - name: qualys-container-sensor
      image: qualys/qcs-sensor:latest
      imagePullPolicy : IfNotPresent
      resources:
        limits:
          cpu: "0.2" # Default CPU usage limit on each node for
sensor.
    args: ["--k8s-mode", "--container-runtime", "cri-o"]
    env:
    - name: CUSTOMERID
      value: __customerId
    - name: ACTIVATIONID
      value: __activationId
    - name: POD_URL
      value:
    - name: QUALYS_SCANNING_CONTAINER_LAUNCH_TIMEOUT
      value: "10"
# uncomment(and indent properly) below section if proxy is required to
connect Qualys Cloud
    #- name: qualys_https_proxy
    # value: <proxy FQDN or Ip address>:<port#>
    - name: QUALYS_POD_NAME
      valueFrom:
        fieldRef:
          fieldPath: metadata.name
    - name: QUALYS_POD_NAMESPACE
      valueFrom:
        fieldRef:
          fieldPath: metadata.namespace
    volumeMounts:
    - mountPath: /var/run/crio/crio.sock
      name: socket-volume
      readOnly: true
    - mountPath: /usr/local/qualys/qpqa/data
      name: persistent-volume
    - mountPath: /usr/local/qualys/qpqa/data/conf/agent-data
      name: agent-volume
# uncomment(and indent properly) below section if proxy(with CA cert)
required to connect Qualys Cloud
    #- mountPath: /etc/qualys/qpqa/cert/custom-ca.crt
```

```
# name: proxy-cert-path
securityContext:
  privileged: true
volumes:
- name: socket-volume
  hostPath:
    path: /var/run/crio/crio.sock
    type: Socket
- name: persistent-volume
  hostPath:
    path: /usr/local/qualys/sensor/data
    type: DirectoryOrCreate
- name: agent-volume
  hostPath:
    path: /etc/qualys
    type: DirectoryOrCreate
# uncomment (and indent properly) below section if proxy (with CA cert)
# required to connect Qualys Cloud
#- name: proxy-cert-path
#  hostPath:
#    path: <proxy certificate path>
#    type: File
hostNetwork: true
```

Qualys Container Sensor DaemonSet should be deployed in 'qualys' namespace as part of ServiceAccount with adequate permission to communicate with Kubernetes API Server. The Role, ClusterRole, RoleBinding and ClusterRoleBinding are used to assign the necessary permissions to ServiceAccount. If you already have Qualys Container Sensor running in a different namespace other than 'qualys', you'll need to uninstall Qualys Container Sensor from the other namespace and deploy it fresh in the 'qualys' namespace.

You'll need these permissions:

get, list, watch - to monitor the resources to be scanned for vulnerabilities across the cluster

create, delete, deletecollection - to spawn containers for image vulnerability assessment in 'qualys' namespace, scan pods across the cluster and then clean up after itself

**Note:** Ensure that the Container Sensor has read and write access to the persistent storage and the cri-o daemon socket.

## Modify parameters in the yaml file

Copy the cssensor-crio-ds.yaml file to Kubernetes cluster's master node then modify it by providing values for the following parameters. In order for the yaml file to work properly, ensure you only update the parameters/sections specified below. Note that you can download the yaml file directly from [https://github.com/Qualys/cs\\_sensor](https://github.com/Qualys/cs_sensor)

General Sensor is assumed here. As mentioned earlier, CRI-O is supported by the General (host) sensor and Registry sensor. It is not supported by Build CI/CD sensor.

Uncomment the **tolerations** section under **spec** if you want Sensor daemonset to be deployed on master nodes.

```
spec:
  #tolerations:
  # this toleration is to have the daemonset runnable on master nodes
  # remove it if want your masters to run sensor pod
  #- key: node-role.kubernetes.io/master
  # effect: NoSchedule
```

If you want to prioritize Qualys Sensor PODs:

Locate and Uncomment the below lines of code in the .yaml file. And change the PriorityClass value mentioned under kind: PriorityClass as per your requirement.

```
#- kind: PriorityClass
#  apiVersion: scheduling.k8s.io/v1
#  metadata:
#    name: qualys-priority-class
#  value: 0
#  preemptionPolicy: PreemptLowerPriority
#  description: Priority class for daemonset
```

Also, locate the PriorityClass name as shown below and Uncomment it.

```
#priorityClassName: qualys-priority-class
```

containers:

```
- name: qualys-container-sensor
  image: qualys/qcs-sensor:latest
  imagePullPolicy: IfNotPresent
  args: ["--k8s-mode", "--container-runtime", "cri-o"]
```

If you want to deploy a Registry Sensor provide the args value as:

```
args: ["--k8s-mode", "--container-runtime", "cri-o", "--registry-
sensor"]
```

If you want to change the log level, provide "--log-level", "<a number between 0 and 5>" as an additional value in args, e.g if you want logs in trace provide:

```
args: ["--k8s-mode", "--container-runtime", "cri-o", "--log-
level", "5"]
```



If you want to launch the sensor with scan thread value other than default 4, provide "--scan-thread-pool-size", "<number of threads>" as an additional value in args.

```
args: ["--k8s-mode", "--container-runtime", "cri-o", "--scan-thread-pool-size", "6"]
```

If you want to define the number of archived qpa.log files to be generated and size per log file, provide "--log-filesize", "<digit><K/M/>" where "K" means kilobyte and "M" means megabyte, and "--log-filepurgecount", "<digit>" as an additional value in args. Default is "--log-filesize": "10M" and "--log-filepurgecount": "5" applied via config.

"--log-filesize": can be used to define the maximum size per log file. For example, "10K" (kilobytes), "10M" (megabytes) or "10" (bytes).

"--log-filepurgecount": can be used to define the number of archived log files to be generated, please note that there will always be current qpa.log file in log/ directory.

```
args: ["--k8s-mode", "--container-runtime", "cri-o", "--log-filesize", "5M", "--log-filepurgecount", "4"]
```

If you want to mask environment variables for images and containers in sensor logs and in the Container Security UI, add the "--mask-env-variable" parameter to args:

```
args: ["--k8s-mode", "--container-runtime", "cri-o", "--mask-env-variable"]
```

If you want to disable image scans for General Sensor, add the "--disableImageScan" parameter to args:

```
args: ["--k8s-mode", "--container-runtime", "cri-o", "--disableImageScan"]
```

If you want to disable log4j vulnerability scanning on your container images, add the "--disable-log4j-scanning" parameter to args:

```
args: ["--k8s-mode", "--container-runtime", "cri-o", "--disable-log4j-scanning"]
```

If you want to optimize image scans for General Sensor, add the "--optimize-image-scans" parameter to args:

```
args: ["--k8s-mode", "--container-runtime", "cri-o", "--optimize-image-scans"]
```

If you want to enable secret detection for container images, add the "--perform-secret-detection" parameter to args. Note that secret detection is supported only on CICD and registry sensors.

```
args: ["--k8s-mode", "--container-runtime", "containerd", "--perform-secret-detection"]
```

If you want to enable malware detection for container images, add the "--perform-malware-detection" parameter to args. Note that malware detection is supported only on registry sensor.

```
args: ["--k8s-mode", "--registry-sensor", "--perform-malware-detection"]
```

If you want to limit the memory usage of sensor container, add the "--limit-resource-usage" parameter to args.

```
args: ["--k8s-mode", "--limit-resource-usage"]
```

Under **resources** specify the following:

```
resources:
  limits:
    cpu: "0.2" # Default CPU usage limit on each node for sensor.
```

For example, for limiting the cpu usage to 5%, set resources:limits:cpu: "0.05". This limits the cpu usage to 5% of one core on the host.

If there are multiple processors on a node, setting the resources:limits:cpu value applies the CPU limit to one core only. For example, if you have 4 CPUs on the system and you want to set CPU limit as 20% of overall CPU capacity, then the CPU limit should be set to 0.8 i.e., 80% of one core only which becomes 20% of total CPU capacity.

To disable any CPU usage limit, set resources:limits:cpu value to 0.

Optionally, if you want to specify the memory resources for Container Sensor, you can specify it under **resources**. Recommended values for the Container Sensor's memory requests and memory limits are:

```
resources:
  limits:
    cpu: "0.2" # Default CPU usage limit on each node for sensor
    memory: "500Mi"
  requests:
    memory: "300Mi"
```

If you want to specify the memory resources for SCA Scan, or Secret Scan, or Malware Scan you can specify it under **resources**. It is recommended to use "--limit-resource-usage" argument while installing the sensor.

Recommended values for the Container Sensor's memory requests and memory limits for above mentioned scan types are:

```
resources:
  limits:
    cpu: "0.2" # Default CPU usage limit on each node for sensor
    memory: "3.5Gi"
  requests:
    memory: "1.5Gi"
```

Disclaimer: The above recommendation is based on the average image size of 1.5GB, if the image sizes are more, please increase the memory limit accordingly.

When either of the memory resource values (limits or requests) is specified for Container Sensor, we automatically apply both memory requests and memory limits to image scanning containers. Default values are 200Mi and 700Mi, respectively.

Additionally, you could overwrite one or both values by specifying the following variables under **env**. In this example, the values were changed to 300Mi and 800Mi.

- name: QUALYS\_SCANNING\_CONTAINER\_MEMORYREQUESTMB  
value: "300Mi"
- name: QUALYS\_SCANNING\_CONTAINER\_MEMORYLIMITMB  
value: "800Mi"

Under **env** specify the following:

Activation ID (Required)

- name: ACTIVATIONID  
value: XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX

Customer ID (Required)

- name: CUSTOMERID  
value: XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX

Specify POD\_URL when using docker hub image. Otherwise, remove it.

- name: POD\_URL  
value: <Specify POD URL>

Specify the scanning container launch timeout in minutes. If this env variable is not present, then 10 minutes is the default.

- name: QUALYS\_SCANNING\_CONTAINER\_LAUNCH\_TIMEOUT  
value: "10"

With each scan, we check the node status to see if the node is schedulable or not, and launch the scan only if the node is schedulable. If the node status indicates that the node is unschedulable, then we retry the scan after a default interval of 15 minutes. You can increase or decrease the time the sensor waits before retrying the scan by specifying a different scan retry interval in minutes.

- name: UNSCHEDULABLE\_NODE\_SCAN\_RETRY\_INTERVAL  
value: "30"

Uncomment and indent properly the proxy information, or keep it as is if not required:

- For the communication between the sensor and the backend (Container Management Service):

```
#- name: qualys_https_proxy  
#   value: <proxy FQDN or Ip address>:<port#>
```

- For a registry sensor version 1.21.0 or later used for a public registry:

```
#- name: https_proxy  
#   value: <proxy FQDN or Ip address>:<port#>
```

Uncomment proxy-cert-path under **volumes**, or keep it as is if not required:

```
# uncomment (and indent properly) below section if proxy (with CA cert)  
is required to connect to the Qualys Cloud Platform  
  #- name: proxy-cert-path  
  #   hostPath:  
  #     path: <proxy certificate path>  
  #     type: File
```

Activation ID and Customer ID are required. Use the Activation ID and Customer ID from your subscription. To get the Activation ID and Customer ID, login to the Container Security UI, go to Configurations > Sensors, click Download, and then click any sensor type. The installation command on the Installation Instructions screen contains your Activation ID and Customer ID. Activation ID is like a password, do not share it.

If you are using an https proxy, ensure that all Kubernetes nodes have a valid certificate file for the sensor to communicate with the Container Management Server.

If you are not using a proxy and you have kept the above-mentioned parts commented, you can keep the following part commented from **volumeMounts** as well:

```
#- mountPath: /etc/qualys/qpq/cert/custom-ca.crt  
#   name: proxy-cert-path
```

## How to deploy the Container Sensor DaemonSet

Once you have modified the **cssensor-crio-ds.yml** file, run the following command on Kubernetes master to create a DaemonSet:

```
kubectl create -f cssensor-crio-ds.yml
```

## How to remove the Container Sensor DaemonSet

To uninstall Qualys Container Sensor, run the following command on Kubernetes master:

```
kubectl delete -f cssensor-crio-ds.yml
```

**Note:** The persistent storage will need to be removed manually on each worker node.

## Launch sensor without persistent storage

You can run the sensor without using persistent storage on host. In this case data is not stored on host but stored at the `/usr/local/qualys/qpq/data` folder relative to the Sensor.

To launch sensor without persistent storage, modify the **cssensor-crio-ds.yml** file and provide "--sensor-without-persistent-storage" as an additional value in **args**.

```
args: ["--k8s-mode", "--container-runtime", "cri-o", "--sensor-  
without-persistent-storage"]
```

It is recommended to use the "--enable-console-logs" option along with "--sensor-without-persistent-storage" to preserve the logs.

Under **volumeMounts** remove/comment the persistent-volume section.

```
volumeMounts:  
- mountPath: /usr/local/qualys/qpa/data  
  name: persistent-volume
```

Under **volumes** remove/comment the persistent-volume section.

```
volumes:  
- name: persistent-volume  
  hostPath:  
    path: /usr/local/qualys/sensor/data  
    type: DirectoryOrCreate
```

## Deploy in Kubernetes - OpenShift

This section assumes you have the sensor image: [Obtain the Container Sensor Image](#)

Integrate the Container Sensor into the DaemonSet like other application containers to ensure that there is always a Sensor deployed on the Docker Host. Perform the following steps for creating a DaemonSet for the Qualys sensor to be deployed in OpenShift.

**Note:** Ensure that the Container Sensor has read and write access to the persistent storage and the docker daemon socket.

Download the **QualysContainerSensor.tar.xz** file from Qualys Cloud Portal on OpenShift master.

Untar the sensor package:

```
sudo tar -xvf QualysContainerSensor.tar.xz
```

Use the following commands to push the Qualys sensor image to a repository common to all nodes in the OpenShift cluster:

```
sudo docker load -i qualys-sensor.tar  
sudo docker tag <IMAGE NAME/ID> <URL to push image to the repository>  
sudo docker push <URL to push image to the repository>
```

For example:

```
sudo docker load -i qualys-sensor.tar
```

```
sudo docker tag c3fa63a818df mycloudregistry.com/container-  
sensor:qualys-sensor-xxx  
sudo docker push mycloudregistry.com/container-sensor:qualys-sensor-xxx
```

**Note:** Do not use the examples as is. Replace the registry/image path with your own.

## Modify the `cssensor-openshift-ds.yml` file

Modify the `cssensor-openshift-ds.yml` file (extracted from `QualysContainerSensor.tar.xz`) to provide values for the following parameters. In order for the yml file to work properly, ensure that you do not remove/comment the respective sections mentioned below. Note that you can download the yml file directly from [https://github.com/Qualys/cs\\_sensor](https://github.com/Qualys/cs_sensor)

```
serviceAccountName:  
    qualysuser
```

Ensure that the `serviceAccountName` is provided in the pod declaration.

```
containers:  
  - name: qualys-container-sensor  
    image: <CS Sensor image name in the docker hub/private registry>  
    securityContext:  
      privileged: true  
      args: ["--k8s-mode"]
```

If you want to deploy the sensor for CI/CD environment provide the **args** value as:

```
args: ["--k8s-mode", "--cicd-deployed-sensor", "--log-level", "5", "--  
log-filesize", "5M", "--log-filepurgecount", "4"]
```

If you want to deploy a Registry Sensor provide the **args** value as:

```
args: ["--k8s-mode", "--registry-sensor", "--log-level", "5", "--log-  
filesize", "5M", "--log-filepurgecount", "4"]
```

**Note:** The values for `--log-level`, `--log-filesize` and `--log-filepurgecount` in the **args** above are only samples. Specify appropriate values for your needs.

If you want print logs on the console, provide `--enable-console-logs` as an additional value in **args**.

To restrict the cpu usage to a certain value, change the following: (Optional)

Under **resources** specify the following:

```
resources:  
  limits:  
    cpu: "0.2" # Default CPU usage limit(20% of one core on the host).
```

For example, for limiting the cpu usage to 5%, set `resources:limits:cpu: "0.05"`. This limits the cpu usage to 5% of one core on the host.

If there are multiple processors on a node, setting the `resources:limits:cpu` value applies the CPU limit to one core only.

For example, if you have 4 CPUs on the system and you want to set CPU limit as 20% of overall CPU capacity, then the CPU limit should be set to 0.8 i.e., 80% of one core only which becomes 20% of total CPU capacity.

To disable any CPU usage limit, set `resources:limits:cpu` value to 0.

Under **env** specify the following:

Activation ID (Required)

```
- name: ACTIVATIONID
  value: XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX
```

Customer ID (Required)

```
- name: CUSTOMERID
  value: XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX
```

Specify proxy information, or remove if not required:

```
- name: qualys_https_proxy
  value: proxy.localnet.com:3128
```

Uncomment and indent properly the proxy information, or keep it as is if not required:

- For the communication between the sensor and the backend (Container Management Service):

```
#- name: qualys_https_proxy
#   value: <proxy FQDN or Ip address>:<port#>
```

- For a registry sensor version 1.21.0 or later used for a public registry:

```
#- name: https_proxy
#   value: <proxy FQDN or Ip address>:<port#>
```

With each scan, we check the node status to see if the node is schedulable or not, and launch the scan only if the node is schedulable. If the node status indicates that the node is unschedulable, then we retry the scan after a default interval of 15 minutes. You can increase or decrease the time the sensor waits before retrying the scan by specifying a different scan retry interval in minutes.

```
- name: UNSCHEDULABLE_NODE_SCAN_RETRY_INTERVAL
  value: "30"
```

Under **volumes** specify the proxy cert path, or remove if not required:

```
- name: proxy-cert-path
  hostPath:
    path: /root/cert/proxy-certificate.crt
```

Activation ID and Customer ID are required. Use the Activation ID and Customer ID from your subscription.

If you are using a proxy, ensure that all OpenShift nodes have a valid certificate file for the sensor to communicate with the Container Management Server.

If you are not using a proxy and you have removed the above mentioned parts, you can remove the following part from **volumeMounts** as well:

```
- mountPath: /etc/qualys/qpa/cert/custom-ca.crt
  name: proxy-cert-path
```

Once you have modified the **cssensor-openshift-ds.yml** file, run the following command on OpenShift master to create a DaemonSet:

```
oc create -f cssensor-openshift-ds.yml
```

If you need to uninstall Qualys Container Sensor, run the following command on OpenShift master:

```
oc delete ds qualys-container-sensor -n kube-system
```

## Launch sensor without persistent storage

You can run the sensor without using persistent storage on host. In this case data is not stored on host but stored at the `/usr/local/qualys/qpa/data` folder relative to the Sensor.

To launch sensor without persistent storage, modify the **cssensor-openshift-ds.yml** file and provide `--sensor-without-persistent-storage` as an additional value in **args**.

```
args: ["--k8s-mode", "--sensor-without-persistent-storage"]
```

It is recommended to use the `--enable-console-logs` option along with `--sensor-without-persistent-storage` to preserve the logs.

Under **volumeMounts** remove/comment the persistent-volume section.

```
volumeMounts:
- mountPath: /usr/local/qualys/qpa/data
  name: persistent-volume
```

Under **volumes** remove/comment the persistent-volume section.

```
volumes:
- name: persistent-volume
  hostPath:
    path: /usr/local/qualys/sensor/data
```



## Deploy in Kubernetes - OpenShift4.4+ with CRI-O Runtime

This section assumes you have the sensor image: [Obtain the Container Sensor Image](#)

**Note:** CRI-O runtime is supported by the General (host) sensor and Registry sensor. It is not supported by Build CI/CD sensor.

### Modify the `cssensor-openshift-crio-ds.yml` file

Below is the Kubernetes DaemonSet deployment template that can be used to deploy the Qualys Container Sensor. For customers' convenience this template is available in the `QualysContainerSensor.tar.xz`. Note that you can download the yml file directly from [https://github.com/Qualys/cs\\_sensor](https://github.com/Qualys/cs_sensor)

IMPORTANT: The field alignment in the yml file is very important. Please make sure to honor the formatting provided in the template.

```
kind: List
apiVersion: v1
items:
  # Create custom namespace qualys
  - kind: Namespace
    apiVersion: v1
    metadata:
      name: qualys
  # Service Account
  - kind: ServiceAccount
    apiVersion: v1
    metadata:
      name: qualys-service-account
      namespace: qualys
  # Role for all permission to qualys namespace
  - kind: Role
    # if k8s version is 1.17 and earlier then change apiVersion to
    "rbac.authorization.k8s.io/v1beta1"
    apiVersion: rbac.authorization.k8s.io/v1
    metadata:
      name: qualys-reader-role
      namespace: qualys
    rules:
      - apiGroups: [""]
        resources: ["pods"]
        verbs: ["create", "delete", "deletecollection"]
      - apiGroups: ["batch"]
        resources: ["jobs"]
        verbs: ["get","create", "delete", "deletecollection"]
      - apiGroups: [""]
        resources: ["pods/attach"]
        verbs: ["create"]
```

```
# ClusterRole for read permission to whole cluster
- kind: ClusterRole
  # if k8s version is 1.17 and earlier then change apiVersion to
"rbac.authorization.k8s.io/v1beta1"
  apiVersion: rbac.authorization.k8s.io/v1
  metadata:
    name: qualys-cluster-reader-role
  rules:
  - apiGroups: [""]
    resources: ["nodes", "pods/status",
"replicationcontrollers/status", "nodes/status"]
    verbs: ["get"]
  - apiGroups: [""]
    resources: ["pods"]
    verbs: ["get", "list", "watch"]
  - apiGroups: [""]
    resources: ["pods/exec"]
    verbs: ["create"]
  - apiGroups: ["apps"]
    resources: ["replicasets/status", "daemonsets/status",
"deployments/status", "statefulsets/status"]
    verbs: ["get"]
  - apiGroups: ["batch"]
    resources: ["jobs/status", "cronjobs/status"]
    verbs: ["get"]
  # RoleBinding to assign permissions in qualys-reader-role to qualys-
service-account
- kind: RoleBinding
  # if k8s version is 1.17 and earlier then change apiVersion to
"rbac.authorization.k8s.io/v1beta1"
  apiVersion: rbac.authorization.k8s.io/v1
  metadata:
    name: qualys-reader-rb
    namespace: qualys
  subjects:
  - kind: ServiceAccount
    name: qualys-service-account
    namespace: qualys
  roleRef:
    kind: Role
    name: qualys-reader-role
    apiGroup: rbac.authorization.k8s.io
  # ClusterRoleBinding to assign permissions in qualys-cluster-reader-
role to qualys-service-account
- kind: ClusterRoleBinding
  # if k8s version is 1.17 and earlier then change apiVersion to
```

```
"rbac.authorization.k8s.io/v1beta1"
  apiVersion: rbac.authorization.k8s.io/v1
  metadata:
    name: qualys-cluster-reader-rb
  subjects:
  - kind: ServiceAccount
    name: qualys-service-account
    namespace: qualys
  roleRef:
    kind: ClusterRole
    name: qualys-cluster-reader-role
    apiGroup: rbac.authorization.k8s.io
- kind: SecurityContextConstraints
  apiVersion: security.openshift.io/v1
  metadata:
    name: scc-qualys-sensor
  allowHostDirVolumePlugin: true
  allowHostNetwork: true
  allowHostIPC: false
  allowHostPID: false
  allowHostPorts: false
  allowPrivilegedContainer: false
  readOnlyRootFilesystem: false
  runAsUser:
    type: RunAsAny
  seLinuxContext:
    type: RunAsAny
  users:
  - system:serviceaccount:qualys:qualys-service-account
# Qualys Container Sensor pod with
- apiVersion: apps/v1
  kind: DaemonSet
  metadata:
    name: qualys-container-sensor
    namespace: qualys
    labels:
      k8s-app: qualys-cs-sensor
  spec:
    selector:
      matchLabels:
        name: qualys-container-sensor
    updateStrategy:
      type: RollingUpdate
  template:
    metadata:
```

```
labels:
  name: qualys-container-sensor
spec:
  #tolerations:
  # this toleration is to have the daemonset runnable on master
nodes
  # remove it if want your masters to run sensor pod
  #- key: node-role.kubernetes.io/master
  # effect: NoSchedule
  serviceAccountName: qualys-service-account
  containers:
  - name: qualys-container-sensor
    image: qualys/qcs-sensor:latest
    imagePullPolicy : IfNotPresent
    resources:
      limits:
        cpu: "0.2" # Default CPU usage limit on each node for
sensor.
        args: ["--k8s-mode", "--container-runtime", "cri-o"]
        env:
        - name: CUSTOMERID
          value: __customerId
        - name: ACTIVATIONID
          value: __activationId
        - name: POD_URL
          value:
        - name: QUALYS_SCANNING_CONTAINER_LAUNCH_TIMEOUT
          value: "10"
# uncomment(and indent properly) below section if proxy is required to
connect Qualys Cloud
  #- name: qualys_https_proxy
  # value: <proxy FQDN or Ip address>:<port#>
  - name: QUALYS_POD_NAME
    valueFrom:
      fieldRef:
        fieldPath: metadata.name
  - name: QUALYS_POD_NAMESPACE
    valueFrom:
      fieldRef:
        fieldPath: metadata.namespace
  volumeMounts:
  - mountPath: /var/run/crio/crio.sock
    name: socket-volume
    readOnly: true
  - mountPath: /usr/local/qualys/qpaa/data
    name: persistent-volume
```

```
- mountPath: /usr/local/qualys/qpq/data/conf/agent-data
  name: agent-volume
# uncomment (and indent properly) below section if proxy (with CA cert)
required to connect Qualys Cloud
  #- mountPath: /etc/qualys/qpq/cert/custom-ca.crt
  # name: proxy-cert-path
  securityContext:
    allowPrivilegeEscalation: false
volumes:
- name: socket-volume
  hostPath:
    path: /var/run/crio/crio.sock
    type: Socket
- name: persistent-volume
  hostPath:
    path: /usr/local/qualys/sensor/data
    type: DirectoryOrCreate
- name: agent-volume
  hostPath:
    path: /etc/qualys
    type: DirectoryOrCreate
# uncomment (and indent properly) below section if proxy (with CA cert)
required to connect Qualys Cloud
  #- name: proxy-cert-path
  # hostPath:
  #   path: <proxy certificate path>
  #   type: File
hostNetwork: true
```

Qualys Container Sensor DaemonSet should be deployed in 'qualys' namespace as a part of ServiceAccount with adequate permission to communicate with Kubernetes API Server. The Role, ClusterRole, RoleBinding, and ClusterRoleBindings are used to assign the necessary permissions to the ServiceAccount.

You'll need these permissions:

get, list, watch - to monitor the resources to be scanned for vulnerabilities across the cluster

create, delete, deletecollection - to spawn containers for image vulnerability assessment in 'qualys' namespace, scan pods across the cluster and then clean up after itself

**Note:** Ensure that the Container Sensor has read and write access to the persistent storage and the cri-o daemon socket.

## Modify parameters in the yaml file

Copy the **cssensor-openshift-crio-ds.yaml** file to Openshift4.4 cluster's master node then modify it by providing values for the following parameters. In order for the yaml file to work properly, ensure you only update the parameters/sections specified below. Note that you can download the yml file directly from [https://github.com/Qualys/cs\\_sensor](https://github.com/Qualys/cs_sensor)

General Sensor is assumed here. As mentioned earlier, CRI-O is supported by the General (host) sensor and Registry sensor. It is not supported by Build CI/CD sensor.

Uncomment the **tolerations** section under **spec** if you want Sensor daemonset to be deployed on master nodes

```
spec:
  #tolerations:
  # this toleration is to have the daemonset runnable on master nodes
  # remove it if want your masters to run sensor pod
  #- key: node-role.kubernetes.io/master
  # effect: NoSchedule
```

If you want to prioritize Qualys Sensor PODs:

Locate and Uncomment the below lines of code in the .yaml file. And change the PriorityClass value mentioned under kind: PriorityClass as per your requirement.

```
#- kind: PriorityClass
#  apiVersion: scheduling.k8s.io/v1
#  metadata:
#    name: qualys-priority-class
#  value: 0
#  preemptionPolicy: PreemptLowerPriority
#  description: Priority class for daemonset
```

Also, locate the PriorityClass name present below and Uncomment it.

```
#priorityClassName: qualys-priority-class
```

```
containers:
- name: qualys-container-sensor
  image: <CS Sensor image name in the private/docker hub registry>
  args: ["--k8s-mode", "--container-runtime", "cri-o"]
```

If you want to deploy a Registry Sensor provide the args value as:

```
args: ["--k8s-mode", "--container-runtime", "cri-o", "--registry-sensor"]
```

If you want to change the log level, provide "--log-level", "<a number between 0 and 5>" as an additional value in args, e.g if you want logs in trace provide

```
args: ["--k8s-mode", "--container-runtime", "cri-o", "--log-level", "5"]
```

If you want to launch the sensor with scan thread value other than default 4, provide "--scan-thread-pool-size", "<number of threads>" as an additional value in args.

```
args: ["--k8s-mode", "--container-runtime", "cri-o", "--scan-thread-pool-size", "6"]
```

If you want to define the number of archived qpa.log files to be generated and size per log file, provide "--log-filesize", "<digit><K/M/>" where "K" means kilobyte and "M" means megabyte, and "--log-filepurgecount", "<digit>" as an additional value in args. Default is "--log-filesize": "10M" and "--log-filepurgecount": "5" applied via config.

"--log-filesize": can be used to define the maximum size per log file. For example, "10K" (kilobytes), "10M" (megabytes) or "10" (bytes).

"--log-filepurgecount": can be used to define the number of archived log files to be generated, please note that there will always be current qpa.log file in log/ directory.

```
args: ["--k8s-mode", "--container-runtime", "cri-o", "--log-filesize", "5M", "--log-filepurgecount", "4"]
```

If you want to mask environment variables for images and containers in sensor logs and in the Container Security UI, add the "--mask-env-variable" parameter to args:

```
args: ["--k8s-mode", "--container-runtime", "cri-o", "--mask-env-variable"]
```

If you want to disable image scans for General Sensor, add the "--disableImageScan" parameter to args:

```
args: ["--k8s-mode", "--container-runtime", "cri-o", "--disableImageScan"]
```

If you want to disable log4j vulnerability scanning on your container images, add the "--disable-log4j-scanning" parameter to args:

```
args: ["--k8s-mode", "--container-runtime", "cri-o", "--disable-log4j-scanning"]
```

If you want to optimize image scans for General Sensor, add the "--optimize-image-scans" parameter to args:

```
args: ["--k8s-mode", "--container-runtime", "cri-o", "--optimize-image-scans"]
```

If you want to enable secret detection for container images, add the "--perform-secret-detection" parameter to args. Note that secret detection is supported only on CICD and registry sensors.

```
args: ["--k8s-mode", "--container-runtime", "containerd", "--
```

```
perform-secret-detection"]
```

If you want to enable malware detection for container images, add the "--perform-malware-detection" parameter to args. Note that malware detection is supported only on registry sensor.

```
args: ["--k8s-mode", "--registry-sensor", "--perform-malware-  
detection"]
```

If you want to limit the memory usage of sensor container, add the "--limit-resource-usage" parameter to args.

```
args: ["--k8s-mode", "--limit-resource-usage"]
```

Under **resources** specify the following:

```
resources:  
limits:  
cpu: "0.2" # Default CPU usage limit on each node for sensor.
```

For example, for limiting the CPU usage to 5%, set resources:limits:cpu: "0.05". This limits the CPU usage to 5% of one core on the host.

If there are multiple processors on a node, setting the resources:limits:cpu value applies the CPU limit to one core only. For example, if you have 4 CPUs on the system and you want to set CPU limit as 20% of overall CPU capacity, then the CPU limit should be set to 0.8 i.e., 80% of one core only which becomes 20% of total CPU capacity.

To disable any CPU usage limit, set resources:limits:cpu value to 0.

Optionally, if you want to specify the memory resources for Container Sensor, you can specify it under **resources**. Recommended values for the Container Sensor's memory requests and memory limits are:

```
resources:  
limits:  
cpu: "0.2" # Default CPU usage limit on each node for sensor  
memory: "500Mi"  
requests:  
memory: "300Mi"
```

If you want to specify the memory resources for SCA Scan, or Secret Scan, or Malware Scan you can specify it under **resources**. It is recommended to use "--limit-resource-usage" argument while installing the sensor.

Recommended values for the Container Sensor's memory requests and memory limits for above mentioned scan types are:

```
resources:  
limits:  
cpu: "0.2" # Default CPU usage limit on each node for sensor
```



```
memory: "3.5Gi"  
requests:  
memory: "1.5Gi"
```

Disclaimer: The above recommendation is based on the average image size of 1.5GB, if the image sizes are more, please increase the memory limit accordingly.

When either of the memory resource values (limits or requests) is specified for Container Sensor, we automatically apply both memory requests and memory limits to image scanning containers. Default values are 200Mi and 700Mi, respectively.

Additionally, you could overwrite one or both values by specifying the following variables under **env**. In this example, the values were changed to 300Mi and 800Mi.

```
- name: QUALYS_SCANNING_CONTAINER_MEMORYREQUESTMB  
value: "300Mi"  
- name: QUALYS_SCANNING_CONTAINER_MEMORYLIMITMB  
value: "800Mi"
```

Under **env** specify the following:

```
Activation ID (Required)  
- name: ACTIVATIONID  
value: XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX  
  
Customer ID (Required)  
- name: CUSTOMERID  
value: XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX
```

Specify `POD_URL` when using docker hub image. Otherwise, remove it.

```
- name: POD_URL  
value: <Specify POD URL>
```

Specify the scanning container launch timeout in minutes. If this env variable is not present, then 10 minutes is the default.

```
- name: QUALYS_SCANNING_CONTAINER_LAUNCH_TIMEOUT  
value: "10"
```

With each scan, we check the node status to see if the node is schedulable or not, and launch the scan only if the node is schedulable. If the node status indicates that the node is unschedulable, then we retry the scan after a default interval of 15 minutes. You can increase or decrease the time the sensor waits before retrying the scan by specifying a different scan retry interval in minutes.

```
- name: UNSCHEDULABLE_NODE_SCAN_RETRY_INTERVAL  
value: "30"
```

Specify proxy information, or remove if not required:

```
- name: qualys_https_proxy
  value: <PROXY_URL>
```

Uncomment and indent properly the proxy information, or keep it as is if not required:

- For the communication between the sensor and the backend (Container Management Service):

```
#- name: qualys_https_proxy
#   value: <proxy FQDN or Ip address>:<port#>
```

- For a registry sensor version 1.21.0 or later used for a public registry:

```
#- name: https_proxy
#   value: <proxy FQDN or Ip address>:<port#>
```

Uncomment proxy-cert-path under volumes, or keep it as is if not required:

```
#- name: proxy-cert-path
#   hostPath:
#     path: /root/cert/proxy-certificate.crt
#     type: File
```

Activation ID and Customer ID are required. Use the Activation ID and Customer ID from your subscription. To get the Activation ID and Customer ID, login to the Container Security UI, go to Configurations > Sensors, click Download, and then click any sensor type. The installation command on the Installation Instructions screen contains your Activation ID and Customer ID. Activation ID is like a password, do not share it.

If you are using an https proxy, ensure that all Kubernetes nodes have a valid certificate file for the sensor to communicate with the Container Management Server.

If you are not using a proxy and you have kept the above-mentioned parts commented, you can keep the following part commented from **volumeMounts** as well:

```
#- mountPath: /etc/qualys/qpa/cert/custom-ca.crt
#   name: proxy-cert-path
```

## How to deploy the Container Sensor DaemonSet

Once you have modified the **cssensor-openshift-crio-ds.yml** file, run the following command on OpenShift master to create a DaemonSet:

```
oc create -f cssensor-openshift-crio-ds.yml
```

## How to remove the Container Sensor DaemonSet

To uninstall Qualys Container Sensor, run the following command on Openshift master:

```
oc delete -f cssensor-openshift-crio-ds.yml
```

**Note:** The persistent storage will need to be removed manually on each worker node.

## Launch sensor without persistent storage

You can run the sensor without using persistent storage on host. In this case data is not stored on host but stored at the `/usr/local/qualys/qpqa/data` folder relative to the Sensor.

To launch sensor without persistent storage, modify the `cssensor-openshift-crio-ds.yml` file and provide `--sensor-without-persistent-storage` as an additional value in `args`.

```
args: ["--k8s-mode", "--container-runtime", "cri-o", "--sensor-  
without-persistent-storage"]
```

It is recommended to use the `--enable-console-logs` option along with `--sensor-without-persistent-storage` to preserve the logs.

Under **volumeMounts** remove/comment the persistent-volume section.

```
volumeMounts:  
- mountPath: /usr/local/qualys/qpqa/data  
  name: persistent-volume
```

Under **volumes** remove/comment the persistent-volume section.

```
volumes:  
- name: persistent-volume  
  hostPath:  
    path: /usr/local/qualys/sensor/data  
    type: DirectoryOrCreate
```

## Deploy in Kubernetes with TKGI - Docker Runtime

This section assumes you have the sensor image: [Obtain the Container Sensor Image](#)

Perform the following steps for creating a DaemonSet for the Qualys sensor to be deployed in Kubernetes.

**Note:** Ensure that the Container Sensor has read and write access to the persistent storage and the docker daemon socket.

### Modify the `cssensor-ds.yml` file

Below is the Kubernetes DaemonSet deployment template that can be used to deploy the Qualys Container Sensor. For customers' convenience, this template is available in the `QualysContainerSensor.tar.xz` as `cssensor-ds.yml` file. Note that you can download the yaml file directly from [https://github.com/Qualys/cs\\_sensor](https://github.com/Qualys/cs_sensor)

**IMPORTANT:** The field alignment in the yaml file is very important. Please make sure to honor the formatting provided in the template.

```
kind: List
```

```
apiVersion: v1
items:
  - kind: Namespace
    apiVersion: v1
    metadata:
      name: qualys
  # Service Account
  - kind: ServiceAccount
    apiVersion: v1
    metadata:
      name: qualys-service-account
      namespace: qualys
  # Role for read/write/delete permission to qualys namespace
  - kind: Role
    # if k8s version is 1.17 and earlier then change apiVersion to
"rbac.authorization.k8s.io/v1beta1"
    apiVersion: rbac.authorization.k8s.io/v1
    metadata:
      name: qualys-reader-role
      namespace: qualys
    rules:
      - apiGroups: [ "", "batch" ]
        resources: [ "pods", "jobs" ]
        verbs: [ "get", "list", "watch", "create", "delete",
"deletecollection" ]
      - apiGroups: [ "" ]
        resources: [ "pods/status" ]
        verbs: [ "get" ]
      - apiGroups: [ "" ]
        resources: [ "pods/attach", "pods/exec" ]
        verbs: [ "create" ]
  - kind: ClusterRole
    # if k8s version is 1.17 and earlier then change apiVersion to
"rbac.authorization.k8s.io/v1beta1"
    apiVersion: rbac.authorization.k8s.io/v1
    metadata:
      name: qualys-cluster-reader-role
    rules:
      - apiGroups: [ "" ]
        resources: [ "nodes", "pods/status",
"replicationcontrollers/status", "nodes/status" ]
        verbs: [ "get" ]
      - apiGroups: [ "apps" ]
        resources: [ "replicasets/status", "daemonsets/status",
"deployments/status", "statefulsets/status" ]
        verbs: [ "get" ]
```

```
- apiGroups: ["batch"]
  resources: ["jobs/status", "cronjobs/status"]
  verbs: ["get"]
# RoleBinding to assign permissions in qualys-reader-role to qualys-
service-account
- kind: RoleBinding
  # if k8s version is 1.17 and earlier then change apiVersion to
"rbac.authorization.k8s.io/v1beta1"
  apiVersion: rbac.authorization.k8s.io/v1
  metadata:
    name: qualys-reader-role-rb
    namespace: qualys
  subjects:
- kind: ServiceAccount
  name: qualys-service-account
  namespace: qualys
  roleRef:
    kind: Role
    name: qualys-reader-role
    apiGroup: rbac.authorization.k8s.io
- kind: ClusterRoleBinding
  # if k8s version is 1.17 and earlier then change apiVersion to
"rbac.authorization.k8s.io/v1beta1"
  apiVersion: rbac.authorization.k8s.io/v1
  metadata:
    name: qualys-cluster-reader-rb
  subjects:
- kind: ServiceAccount
  name: qualys-service-account
  namespace: qualys
  roleRef:
    kind: ClusterRole
    name: qualys-cluster-reader-role
    apiGroup: rbac.authorization.k8s.io
# Qualys Container Sensor pod with
- apiVersion: apps/v1
  kind: DaemonSet
  metadata:
    name: qualys-container-sensor
    namespace: qualys
    labels:
      k8s-app: qualys-cs-sensor
  spec:
    selector:
      matchLabels:
        name: qualys-container-sensor
```

```
updateStrategy:
  type: RollingUpdate
template:
  metadata:
    labels:
      name: qualys-container-sensor
  spec:
    #tolerations:
    # this toleration is to have the daemonset runnable on master
nodes
    # remove it if want your masters to run sensor pod
    #- key: node-role.kubernetes.io/master
    # effect: NoSchedule
    serviceAccountName: qualys-service-account
    containers:
    - name: qualys-container-sensor
      image: qualys/qcs-sensor:latest
      imagePullPolicy : IfNotPresent
      resources:
        limits:
          cpu: "0.2" # Default CPU usage limit on each node for
sensor.
      args: ["--k8s-mode"]
      env:
      - name: CUSTOMERID
        value: __customerId
      - name: ACTIVATIONID
        value: __activationId
      - name: POD_URL
        value:
      - name: QUALYS_SCANNING_CONTAINER_LAUNCH_TIMEOUT
        value: "10"
# uncomment(and indent properly) below section if using Docker HTTP
socket with TLS
  #- name: DOCKER_TLS_VERIFY
  # value: "1"
# uncomment(and indent properly) below section if proxy is required to
connect Qualys Cloud
  #- name: qualys_https_proxy
  # value: <proxy FQDN or Ip address>:<port#>
  - name: QUALYS_POD_NAME
    valueFrom:
      fieldRef:
        fieldPath: metadata.name
  - name: QUALYS_POD_NAMESPACE
    valueFrom:
```

```
        fieldRef:
          fieldPath: metadata.namespace
    volumeMounts:
      - mountPath: /var/run/docker.sock
        name: socket-volume
        readOnly: true
      - mountPath: /usr/local/qualys/qpaa/data
        name: persistent-volume
      - mountPath: /usr/local/qualys/qpaa/data/conf/agent-data
        name: agent-volume
# uncomment(and indent properly) below section if proxy(with CA cert)
required to connect Qualys Cloud
    #- mountPath: /etc/qualys/qpaa/cert/custom-ca.crt
    # name: proxy-cert-path
# uncomment(and indent properly) below section if using Docker HTTP
socket with TLS
    #- mountPath: /root/.docker
    # name: tls-cert-path
    securityContext:
      allowPrivilegeEscalation: false
  volumes:
    - name: socket-volume
      hostPath:
        path: /var/run/docker.sock
        type: Socket
    - name: persistent-volume
      hostPath:
        path: /usr/local/qualys/sensor/data
        type: DirectoryOrCreate
    - name: agent-volume
      hostPath:
        path: /etc/qualys
        type: DirectoryOrCreate
# uncomment(and indent properly) below section if proxy(with CA cert)
required to connect Qualys Cloud
    #- name: proxy-cert-path
    # hostPath:
    #   path: <proxy certificate path>
    #   type: File
# uncomment(and indent properly) below section if using Docker HTTP
socket with TLS
    #- name: tls-cert-path
    # hostPath:
    #   path: <Path of directory of client certificates>
    #   type: Directory
  hostNetwork: true
```

Qualys Container Sensor DaemonSet should be deployed in 'qualys' namespace as a part of ServiceAccount with adequate permission to communicate with Kubernetes API Server. The Role, ClusterRole, RoleBinding and ClusterRoleBinding are used to assign the necessary permissions to the ServiceAccount. If you already have Qualys Container Sensor running in a different namespace other than 'qualys', you'll need to first uninstall Qualys Container Sensor from the other namespace and then deploy it fresh in 'qualys' namespace.

You'll need these permissions:

get, list, watch - to monitor the resources in 'qualys' namespace

create, delete, deletecollection - to spawn containers for image vulnerability assessment in 'qualys' namespace then clean up after itself

## Modify parameters in the yaml file

Copy the **cssensor-ds.yml** file to Kubernetes cluster's master node then modify it by providing values for the following parameters. In order for the yaml file to work properly, ensure you only update the parameters/sections specified below. Note that you can download the yaml file directly from [https://github.com/Qualys/cs\\_sensor](https://github.com/Qualys/cs_sensor)

Uncomment the **tolerations** section under **spec** if you want Sensor daemonset to be deployed on master nodes.

```
spec:
  #tolerations:
  # this toleration is to have the daemonset runnable on master nodes
  # remove it if want your masters to run sensor pod
  #- key: node-role.kubernetes.io/master
  # effect: NoSchedule

containers:
  - name: qualys-container-sensor
    image: <CS Sensor image name in the private/docker hub registry>
    args: ["--k8s-mode"]
```

**Note:** Make sure all nodes that will be running sensor pod have access to private or docker hub registry where sensor image is stored.

If you want to deploy the sensor for CI/CD environment provide the args value as:

```
args: ["--k8s-mode", "--cicd-deployed-sensor"]
```

If you want to deploy a Registry Sensor provide the args value as:

```
args: ["--k8s-mode", "--registry-sensor"]
```

If you want print logs on the console, provide "--enable-console-logs" as an additional value in args.



If you want to change the log level, provide "--log-level", "<a number between 0 and 5>" as an additional value in args, e.g if you want logs in trace provide:

```
args: ["--k8s-mode", "--log-level", "5"]
```

If you want to launch the sensor with scan thread value other than default 4, provide "--scan-thread-pool-size", "<number of threads>" as an additional value in args.

```
args: ["--k8s-mode", "--scan-thread-pool-size", "6"]
```

If you want to define the number of archived qpa.log files to be generated provide "--log-filepurgecount", "<digit>" as an additional value in args. The default, "--log-filepurgecount", "5" is applied via config. Please note that there will always be current qpa.log file in log/ directory.

If you want to define the number of archived qpa.log files to be generated and size per log file, provide "--log-filesize", "<digit><K/M/>" where "K" means kilobyte and "M" means megabyte, and "--log-filepurgecount", "<digit>" as an additional value in args. Default is "--log-filesize": "10M" and "--log-filepurgecount": "5" applied via config.

```
args: ["--k8s-mode", "--log-filesize", "5M", "--log-filepurgecount", "4"]
```

If you want image scanning pods to be instantiated using kubernetes native `kubectl run` command provide "--use-kubectl" as an additional value in args. In this case sensor uses native kubernetes facilities to launch image scans. When this argument is omitted image containers are launched using `docker run`.

```
args: ["--k8s-mode", "--use-kubectl"]
```

If TLS authentication is enabled, specify docker client certificate, client private key and CA certificate names in the args

```
args: ["--k8s-mode", "--tls-cacert", "<file name of the CA certificate that was used to sign docker server certificate>", "--tls-cert", "<docker client certificate file name>", "--tls-key", "<docker client private key file name>"]
```

**Note:** If any of the three files have a default name such as ca.pem, cert.pem, key.pem respectively the corresponding argument can be omitted.

If you want to mask environment variables for images and containers in sensor logs and in the Container Security UI, add the "--mask-env-variable" parameter to args:

```
args: ["--k8s-mode", "--mask-env-variable"]
```

If you want to disable image scans for General Sensor, add the "--disableImageScan" parameter to args:

```
args: ["--k8s-mode", "--disableImageScan"]
```

If you want to specify a scanning policy, add the "--scanning-policy" parameter to args. The available values for the scanning policy are: "DynamicScanningOnly", "StaticScanningOnly", and "DynamicWithStaticScanningAsFallback".

```
args: ["--k8s-mode", "--scanning-policy", "StaticScanningOnly"]
```

If you want to disable log4j vulnerability scanning on your container images, add the "--disable-log4j-scanning" parameter to args:

```
args: ["--k8s-mode", "--disable-log4j-scanning"]
```

If you want to disable log4j static detection for dynamic/static image scans, add the "--disable-log4j-static-detection" parameter to args:

```
args: ["--k8s-mode", "--disable-log4j-static-detection"]
```

If you want to optimize Image scans for General Sensor, add the "--optimize-image-scans" parameter to args:

```
args: ["--k8s-mode", "--optimize-image-scans"]
```

If you have the [SCA scanning](#) feature, then you can enable SCA scanning for container images by adding the "--perform-sca-scan" parameter to args:

```
args: ["--k8s-mode", "--perform-sca-scan"]
```

By default, SCA scans run in online mode. You can choose to disable Internet access for the SCA scan and run the scan in offline mode. Note - We recommend you run the SCA scan in online mode. Quality of software package enumeration for Java substantially degrades when the SCA scan is run in offline mode. The remote maven repository may need to be consulted for an accurate package detection. This can affect accuracy of the vulnerability posture of the image.

```
args: ["--k8s-mode", "--perform-sca-scan" "--disallow-internet-access-for-sca"]
```

The default SCA scan command timeout is 5 minutes (300 seconds). You can overwrite the default timeout with a new value specified in seconds. For example, you may need to increase the SCA scan timeout when scanning large container images to ensure the SCA scan has time to finish.

```
args: ["--k8s-mode", "--perform-sca-scan" "--sca-scan-timeout-in-seconds=600"]
```

If you want to enable secret detection for container images, add the "--perform-secret-detection" parameter to args. Note that secret detection is supported only on CICD and registry sensors.

```
args: ["--k8s-mode", "--cicd-deployed-sensor", "--perform-secret-detection"]
```

```
args: ["--k8s-mode", "--registry-sensor", "--perform-secret-  
detection"]
```

If you want to enable malware detection for container images, add the "--perform-malware-detection" parameter to args. Note that malware detection is supported only on registry sensor.

```
args: ["--k8s-mode", "--registry-sensor", "--perform-malware-  
detection"]
```

Under **resources** specify the following:

```
resources:  
limits:  
cpu: "0.2" # Default CPU usage limit on each node for sensor.
```

For example, for limiting the CPU usage to 5%, set resources:limits:cpu: "0.05". This limits the CPU usage to 5% of one core on the host.

If there are multiple processors on a node, setting the resources:limits:cpu value applies the CPU limit to one core only. For example, if you have 4 CPUs on the system and you want to set CPU limit as 20% of overall CPU capacity, then the CPU limit should be set to 0.8 i.e., 80% of one core only which becomes 20% of total CPU capacity.

To disable any CPU usage limit, set resources:limits:cpu value to 0.

Optionally, if you want to specify the memory resources for Container Sensor, you can specify it under **resources**. Recommended values for the Container Sensor's memory requests and memory limits are:

```
resources:  
limits:  
  cpu: "0.2" # Default CPU usage limit on each node for sensor  
  memory: "500Mi"  
requests:  
  memory: "300Mi"
```

When either of the memory resource values (limits or requests) is specified for Container Sensor and "--use-kubectrl" is supplied in args, we automatically apply both memory requests and memory limits to image scanning containers. Default values are 200Mi and 700Mi, respectively.

Additionally, you could overwrite one or both values by specifying the following variables under **env**. In this example, the values were changed to 300Mi and 800Mi.

```
- name: QUALYS_SCANNING_CONTAINER_MEMORYREQUESTMB  
  value: "300Mi"  
- name: QUALYS_SCANNING_CONTAINER_MEMORYLIMITMB  
  value: "800Mi"
```

Under **env** specify the following:

Activation ID (Required)

- name: ACTIVATIONID  
value: XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX

Customer ID (Required)

- name: CUSTOMERID  
value: XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX

Specify POD\_URL when using docker hub image. Otherwise, remove it.

- name: POD\_URL  
value: <Specify POD URL>

Specify the scanning container launch timeout in minutes. If this env variable is not present, then 10 minutes is the default.

- name: QUALYS\_SCANNING\_CONTAINER\_LAUNCH\_TIMEOUT  
value: "10"

With each scan, we check the node status to see if the node is schedulable or not, and launch the scan only if the node is schedulable. If the node status indicates that the node is unschedulable, then we retry the scan after a default interval of 15 minutes. You can increase or decrease the time the sensor waits before retrying the scan by specifying a different scan retry interval in minutes.

- name: UNSCHEDULABLE\_NODE\_SCAN\_RETRY\_INTERVAL  
value: "30"

To enable TLS authentication uncomment the following 2 lines.

- name: DOCKER\_TLS\_VERIFY  
value: "1"

**Note:** To disable TLS use DOCKER\_TLS\_VERIFY=""(empty string) or remove it or keep it commented in yml file.

**Note:** By enabling sensor communication with docker daemon over TLS customer can restrict the sensor's access to docker socket by using docker authorization plugin.

**Note:** When TLS authentication is enabled and DOCKER\_HOST is not specified the sensor will automatically detect the FQDN of the worker node it is running on and set DOCKER\_HOST environment variable inside the sensor to <worker node's FQDN>:<2376> where 2376 is the default TLS TCP port for docker daemon

**Note:** Set DOCKER\_HOST yourself to 127.0.0.1:<port#> or localhost:<port#> by adding:

- name: DOCKER\_HOST  
value: "<loopback IPv4 address or hostname>:<port#>"

**Note:** Please make sure that FQDN, or hostname, or IPv4 address set in the DOCKER\_HOST matches the CN or Subject Alternative Name in the docker server certificate on each worker node

In TKGI setup, docker.sock is not available at the /var/run location. You must locate docker.sock on your worker nodes and change the socket-volume mapping in the yaml file. For example, if docker.sock is found at /var/vcap/data/sys/run/docker/docker.sock, then you would change the socket-volume mapping under **volumes** like this:

```
volumes:  
- name: socket-volume  
  hostPath:  
    path: /var/vcap/data/sys/run/docker/docker.sock  
    type: Socket
```

Uncomment tls-cert-path under **volumes** if TLS authentication needs to be enabled and provide directory path for client certificates, or keep it as is if not required:

```
#- name: tls-cert-path  
#   hostPath:  
#     path: <Path of directory of client certificates>  
#     type: Directory
```

Uncomment below part under **volumeMounts** as well if you are using TLS. Otherwise, keep it commented out.

```
#- mountPath: /root/.docker  
#name: tls-cert-path
```

Uncomment and indent properly the proxy information, or keep it as is if not required:

- For the communication between the sensor and the backend (Container Management Service):

```
#- name: qualys_https_proxy  
#   value: <proxy FQDN or Ip address>:<port#>
```

- For a registry sensor version 1.21.0 or later used for a public registry:

```
#- name: https_proxy  
#   value: <proxy FQDN or Ip address>:<port#>
```

Uncomment proxy-cert-path under **volumes**, or keep it as is if not required:

```
#- name: proxy-cert-path  
#   hostPath:  
#     path: /root/cert/proxy-certificate.crt  
#     type: File
```

Activation ID and Customer ID are required. Use the Activation ID and Customer ID from your subscription. To get the Activation ID and Customer ID, login to the Container Security UI, go to Configurations > Sensors, click Download, and then click any sensor type. The installation command on the Installation Instructions screen contains your Activation ID and Customer ID. Activation ID is like a password, do not share it.

If you are using an https proxy, ensure that all Kubernetes nodes have a valid certificate file for the sensor to communicate with the Container Management Server.

If you are not using a proxy and you have kept the above-mentioned parts commented, you can keep the following part commented from **volumeMounts** as well:

```
#- mountPath: /etc/qualys/qpa/cert/custom-ca.crt
#   name: proxy-cert-path
```

Once you have modified the **cssensor-ds.yml** file, run the following command on Kubernetes master to create a DaemonSet:

```
kubectl create -f cssensor-ds.yml
```

If you need to uninstall Qualys Container Sensor, run the following command on Kubernetes master:

```
kubectl delete -f cssensor-ds.yml
```

**Note:** The persistent storage will need to be removed manually on each worker node.

## Deploy in Kubernetes with TKGI - Containerd Runtime

This section assumes you have the sensor image: [Obtain the Container Sensor Image](#)

Perform the following steps for creating a DaemonSet for the Qualys sensor to be deployed in Kubernetes.

**Note:** Ensure that the Container Sensor has read and write access to the persistent storage and the docker daemon socket.

### Modify the **cssensor-containerd-ds.yml** file

Below is the Kubernetes DaemonSet deployment template that can be used to deploy the Qualys Container Sensor. For customers' convenience this template is available in the QualysContainerSensor.tar.xz. Note that you can download the yml file directly from [https://github.com/Qualys/cs\\_sensor](https://github.com/Qualys/cs_sensor)

IMPORTANT: The field alignment in the yml file is very important. Please make sure to honor the formatting provided in the template.

```
kind: List
apiVersion: v1
items:
  # Create custom namespace qualys
  - kind: Namespace
```

```
  apiVersion: v1
  metadata:
    name: qualys
# Service Account
- kind: ServiceAccount
  apiVersion: v1
  metadata:
    name: qualys-service-account
    namespace: qualys
# Role for all permission to qualys namespace
- kind: Role
  # if k8s version is 1.17 and earlier then change apiVersion to
"rbac.authorization.k8s.io/v1beta1"
  apiVersion: rbac.authorization.k8s.io/v1
  metadata:
    name: qualys-reader-role
    namespace: qualys
  rules:
  - apiGroups: [ "", "batch" ]
    resources: [ "pods", "jobs" ]
    verbs: [ "get", "list", "watch", "create", "delete",
"deletecollection" ]
  - apiGroups: [ "" ]
    resources: [ "pods/attach", "pods/exec" ]
    verbs: [ "create" ]
# ClusterRole for read permission to whole cluster
- kind: ClusterRole
  # if k8s version is 1.17 and earlier then change apiVersion to
"rbac.authorization.k8s.io/v1beta1"
  apiVersion: rbac.authorization.k8s.io/v1
  metadata:
    name: qualys-cluster-reader-role
  rules:
  - apiGroups: [ "" ]
    resources: [ "pods/exec" ]
    verbs: [ "create" ]
  - apiGroups: [ "" ]
    resources: [ "nodes", "pods", "pods/status",
"replicationcontrollers/status", "nodes/status" ]
    verbs: [ "get" ]
  - apiGroups: [ "apps" ]
    resources: [ "replicasets/status", "daemonsets/status",
"deployments/status", "statefulsets/status" ]
    verbs: [ "get" ]
  - apiGroups: [ "batch" ]
    resources: [ "jobs/status", "cronjobs/status" ]
```

```
    verbs: ["get"]
  # RoleBinding to assign permissions in qualys-reader-role to qualys-
service-account
  - kind: RoleBinding
    # if k8s version is 1.17 and earlier then change apiVersion to
"rbac.authorization.k8s.io/v1beta1"
    apiVersion: rbac.authorization.k8s.io/v1
    metadata:
      name: qualys-reader-rb
      namespace: qualys
    subjects:
      - kind: ServiceAccount
        name: qualys-service-account
        namespace: qualys
    roleRef:
      kind: Role
      name: qualys-reader-role
      apiGroup: rbac.authorization.k8s.io
  # ClusterRoleBinding to assign permissions in qualys-cluster-reader-
role to qualys-service-account
  - kind: ClusterRoleBinding
    # if k8s version is 1.17 and earlier then change apiVersion to
"rbac.authorization.k8s.io/v1beta1"
    apiVersion: rbac.authorization.k8s.io/v1
    metadata:
      name: qualys-cluster-reader-rb
    subjects:
      - kind: ServiceAccount
        name: qualys-service-account
        namespace: qualys
    roleRef:
      kind: ClusterRole
      name: qualys-cluster-reader-role
      apiGroup: rbac.authorization.k8s.io
# Qualys Container Sensor pod with
- apiVersion: apps/v1
  kind: DaemonSet
  metadata:
    name: qualys-container-sensor
    namespace: qualys
    labels:
      k8s-app: qualys-cs-sensor
  spec:
    selector:
      matchLabels:
        name: qualys-container-sensor
```



```
updateStrategy:
  type: RollingUpdate
template:
  metadata:
    labels:
      name: qualys-container-sensor
  spec:
    #tolerations:
    # this toleration is to have the daemonset runnable on master
nodes
    # remove it if want your masters to run sensor pod
    #- key: node-role.kubernetes.io/master
    # effect: NoSchedule
    serviceAccountName: qualys-service-account
    containers:
    - name: qualys-container-sensor
      image: qualys/qcs-sensor:latest
      imagePullPolicy : IfNotPresent
      resources:
        limits:
          cpu: "0.2" # Default CPU usage limit on each node for
sensor.
          args: ["--k8s-mode", "--container-runtime", "containerd"]
          env:
            - name: CUSTOMERID
              value: __customerId
            - name: ACTIVATIONID
              value: __activationId
            - name: POD_URL
              value:
            - name: QUALYS_SCANNING_CONTAINER_LAUNCH_TIMEOUT
              value: "10"
# uncomment(and indent properly) below section if proxy is required to
connect Qualys Cloud
    #- name: qualys_https_proxy
    # value: <proxy FQDN or Ip address>:<port#>
    - name: QUALYS_POD_NAME
      valueFrom:
        fieldRef:
          fieldPath: metadata.name
    - name: QUALYS_POD_NAMESPACE
      valueFrom:
        fieldRef:
          fieldPath: metadata.namespace
    volumeMounts:
    - mountPath: /var/run/containerd/containerd.sock
```

```
        name: socket-volume
        readOnly: true
      - mountPath: /usr/local/qualys/qpa/data
        name: persistent-volume
      - mountPath: /usr/local/qualys/qpa/data/conf/agent-data
        name: agent-volume
# uncomment(and indent properly) below section if proxy(with CA cert)
required to connect Qualys Cloud
    #- mountPath: /etc/qualys/qpa/cert/custom-ca.crt
    # name: proxy-cert-path
    securityContext:
      allowPrivilegeEscalation: false
volumes:
  - name: socket-volume
    hostPath:
      path: /var/run/containerd/containerd.sock
      type: Socket
  - name: persistent-volume
    hostPath:
      path: /usr/local/qualys/sensor/data
      type: DirectoryOrCreate
  - name: agent-volume
    hostPath:
      path: /etc/qualys
      type: DirectoryOrCreate
# uncomment(and indent properly) below section if proxy(with CA cert)
required to connect Qualys Cloud
    #- name: proxy-cert-path
    # hostPath:
    #   path: <proxy certificate path>
    #   type: File
hostNetwork: true
```

Qualys Container Sensor DaemonSet should be deployed in 'qualys' namespace as part of ServiceAccount with adequate permission to communicate with Kubernetes API Server. The Role, ClusterRole, RoleBinding and ClusterRoleBinding are used to assign the necessary permissions to ServiceAccount. If you already have Qualys Container Sensor running in a different namespace other than 'qualys', you'll need to uninstall Qualys Container Sensor from the other namespace and deploy it fresh in the 'qualys' namespace.

You'll need these permissions:

get, list, watch - to monitor the resources to be scanned for vulnerabilities across the cluster

create, delete, deletecollection - to spawn containers for image vulnerability assessment in 'qualys' namespace, scan pods across the cluster and then clean up after itself

**Note:** Ensure that the Container Sensor has read and write access to the persistent storage and the containerd daemon socket.

## Modify parameters in the yaml file

Copy the **cssensor-containerd-ds.yaml** file to Kubernetes cluster's master node then modify it by providing values for the following parameters. In order for the yaml file to work properly, ensure you only update the parameters/sections specified below. Note that you can download the yml file directly from [https://github.com/Qualys/cs\\_sensor](https://github.com/Qualys/cs_sensor)

General Sensor is assumed here. Uncomment the **tolerations** section under **spec** if you want Sensor daemonset to be deployed on master nodes.

```
spec:
  #tolerations:
  # this toleration is to have the daemonset runnable on master nodes
  # remove it if want your masters to run sensor pod
  #- key: node-role.kubernetes.io/master
  # effect: NoSchedule
```

If you want to prioritize Qualys Sensor PODs:

Locate and Uncomment the below lines of code in the .yaml file. And change the PriorityClass value mentioned under kind: PriorityClass as per your requirement.

```
#- kind: PriorityClass
#  apiVersion: scheduling.k8s.io/v1
#  metadata:
#    name: qualys-priority-class
#  value: 0
#  preemptionPolicy: PreemptLowerPriority
#  description: Priority class for daemonset
```

Also, locate the PriorityClass name present below and Uncomment it.

```
#priorityClassName: qualys-priority-class
```

```
containers:
- name: qualys-container-sensor
  image: <CS Sensor image name in the private/docker hub registry>
  args: ["--k8s-mode", "--container-runtime", "containerd"]
```

If you want to deploy a Registry Sensor provide the args value as:

```
args: ["--k8s-mode", "--container-runtime", "containerd", "--
registry-sensor"]
```

If you want to deploy a CICD Sensor, provide the args value as:

```
args: ["--k8s-mode", "--container-runtime", "containerd", "--cid-deployed-sensor"]
```

If you want to change the log level, provide "--log-level", "<a number between 0 and 5>" as an additional value in args, e.g if you want logs in trace provide:

```
args: ["--k8s-mode", "--container-runtime", "containerd", "--log-level", "5"]
```

If you want to launch the sensor with scan thread value other than default 4, provide "--scan-thread-pool-size", "<number of threads>" as an additional value in args.

```
args: ["--k8s-mode", "--container-runtime", "containerd", "--scan-thread-pool-size", "6"]
```

If you want to define the number of archived qpa.log files to be generated and size per log file, provide "--log-filesize", "<digit><K/M/>" where "K" means kilobyte and "M" means megabyte, and "--log-filepurgecount", "<digit>" as an additional value in args. Default is "--log-filesize": "10M" and "--log-filepurgecount": "5" applied via config.

"--log-filesize": can be used to define the maximum size per log file. For example, "10K" (kilobytes), "10M" (megabytes) or "10" (bytes).

"--log-filepurgecount": can be used to define the number of archived log files to be generated, please note that there will always be current qpa.log file in log/ directory.

```
args: ["--k8s-mode", "--container-runtime", "containerd", "--log-filesize", "5M", "--log-filepurgecount", "4"]
```

If you want to mask environment variables for images and containers in sensor logs and in the Container Security UI, add the "--mask-env-variable" parameter to args:

```
args: ["--k8s-mode", "--container-runtime", "containerd", "--mask-env-variable"]
```

If you want to disable image scans for General Sensor, add the "--disableImageScan" parameter to args:

```
args: ["--k8s-mode", "--container-runtime", "containerd", "--disableImageScan"]
```

If you want to specify a scanning policy, add the "--scanning-policy" parameter to args. The available values for the scanning policy are: "DynamicScanningOnly", "StaticScanningOnly", and "DynamicWithStaticScanningAsFallback".

```
args: ["--k8s-mode", "--scanning-policy", "StaticScanningOnly"]
```

If you want to disable log4j vulnerability scanning on your container images, add the "--disable-log4j-scanning" parameter to args:

```
args: ["--k8s-mode", "--container-runtime", "containerd", "--disable-log4j-scanning"]
```

If you want to disable log4j static detection for dynamic/static image scans, add the "--disable-log4j-static-detection" parameter to args:

```
args: ["--k8s-mode", "--container-runtime", "containerd", "--  
disable-log4j-static-detection"]
```

If you want to optimize image scans for General Sensor, add the "--optimize-image-scans" parameter to args:

```
args: ["--k8s-mode", "--container-runtime", "containerd", "--  
optimize-image-scans"]
```

If you want to enable secret detection for container images, add the "--perform-secret-detection" parameter to args. Note that secret detection is supported only on CICD and registry sensors

```
args: ["--k8s-mode", "--container-runtime", "containerd", "--  
perform-secret-detection"]
```

If you want to enable malware detection for container images, add the "--perform-malware-detection" parameter to args. Note that malware detection is supported only on registry sensor.

```
args: ["--k8s-mode", "--container-runtime", "containerd", "--  
registry-sensor", "--perform-malware-detection"]
```

If you want to scan the secured private registry (HTTPS) having a self-signed certificate, you can either provide the certificate through the yaml file or use the "--insecure-registry" argument to ignore the certificate check.

In case you provide both - the certificate through the yaml file and you use "--insecure-registry" argument - you will be able to login to the secured private registry.

```
args: ["--k8s-mode", "--container-runtime", "containerd", "--  
registry-sensor", "--insecure-registry"]
```

To provide self-signed certificate through the Yaml file, follow the steps mentioned below.

1. Open the .yaml file, and search "volumeMounts:".
2. Uncomment the following lines from the code.

```
# name: registry-cert-volume  
# subPath: ca.crt  
# readOnly: true
```

3. Similarly, uncomment the following lines and save the file.

```
# - name: registry-cert-volume  
# secret:  
#   secretName: cert-config
```

Under **resources** specify the following:

```
resources:
  limits:
    cpu: 0.2 # Default CPU usage limit (20% of one core on the host).
```

For example, for limiting the cpu usage to 5%, set `resources:limits:cpu: "0.05"`. This limits the cpu usage to 5% of one core on the host.

If there are multiple processors on a node, setting the `resources:limits:cpu` value applies the CPU limit to one core only. For example, if you have 4 CPUs on the system and you want to set CPU limit as 20% of overall CPU capacity, then the CPU limit should be set to 0.8 i.e., 80% of one core only which becomes 20% of total CPU capacity.

To disable any CPU usage limit, set `resources:limits:cpu` value to 0.

Optionally, if you want to specify the memory resources for Container Sensor, you can specify it under **resources**. Recommended values for the Container Sensor's memory requests and memory limits are:

```
resources:
  limits:
    cpu: "0.2" # Default CPU usage limit on each node for sensor
    memory: "500Mi"
  requests:
    memory: "300Mi"
```

If you want to specify the memory resources for SCA Scan, or Secret Scan, or Malware Scan you can specify it under **resources**. It is recommended to use `--limit-resource-usage` argument while installing the sensor.

Recommended values for the Container Sensor's memory requests and memory limits for above mentioned scan types are:

```
resources:
  limits:
    cpu: "0.2" # Default CPU usage limit on each node for sensor
    memory: "3.5Gi"
  requests:
    memory: "1.5Gi"
```

Disclaimer: The above recommendation is based on the average image size of 1.5GB, if the image sizes are more, please increase the memory limit accordingly.

When either of the memory resource values (limits or requests) is specified for Container Sensor, we automatically apply both memory requests and memory limits to image scanning containers. Default values are 200Mi and 700Mi, respectively.

Additionally, you could overwrite one or both values by specifying the following variables under **env**. In this example, the values were changed to 300Mi and 800Mi.

- name: QUALYS\_SCANNING\_CONTAINER\_MEMORYREQUESTMB  
value: "300Mi"
- name: QUALYS\_SCANNING\_CONTAINER\_MEMORYLIMITMB  
value: "800Mi"

Under **env** specify the following:

Activation ID (Required)

- name: ACTIVATIONID  
value: XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX

Customer ID (Required)

- name: CUSTOMERID  
value: XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX

Specify POD\_URL when using docker hub image. Otherwise, remove it.

- name: POD\_URL  
value: <Specify POD URL>

Specify the scanning container launch timeout in minutes. If this env variable is not present, then 10 minutes is the default.

- name: QUALYS\_SCANNING\_CONTAINER\_LAUNCH\_TIMEOUT  
value: "10"

With each scan, we check the node status to see if the node is schedulable or not, and launch the scan only if the node is schedulable. If the node status indicates that the node is unschedulable, then we retry the scan after a default interval of 15 minutes. You can increase or decrease the time the sensor waits before retrying the scan by specifying a different scan retry interval in minutes.

- name: UNSCHEDULABLE\_NODE\_SCAN\_RETRY\_INTERVAL  
value: "30"

Uncomment and indent properly the proxy information, or keep it as is if not required:

- For the communication between the sensor and the backend (Container Management Service):

- #- name: qualys\_https\_proxy  
# value: <proxy FQDN or Ip address>:<port#>

- For a registry sensor version 1.21.0 or later used for a public registry:

- #- name: https\_proxy  
# value: <proxy FQDN or Ip address>:<port#>

Uncomment proxy-cert-path under **volumes**, or keep it as is if not required:

```
#- name: proxy-cert-path
#   hostPath:
#     path: /root/cert/proxy-certificate.crt
#     type: File
```

Activation ID and Customer ID are required. Use the Activation ID and Customer ID from your subscription. To get the Activation ID and Customer ID, login to the Container Security UI, go to Configurations > Sensors, click Download, and then click any sensor type. The installation command on the Installation Instructions screen contains your Activation ID and Customer ID. Activation ID is like a password, do not share it.

If you are using an https proxy, ensure that all Kubernetes nodes have a valid certificate file for the sensor to communicate with the Container Management Server.

If you are not using a proxy and you have kept the above-mentioned parts commented, you can keep the following part commented from **volumeMounts** as well:

```
#- mountPath: /etc/qualys/qpq/cert/custom-ca.crt
#   name: proxy-cert-path
```



## How to deploy the Container Sensor DaemonSet

Once you have modified the `cssensor-containerd-ds.yml` file, run the following command on Kubernetes master to create a DaemonSet:

```
kubectl create -f cssensor-containerd-ds.yml
```

## How to remove the Container Sensor DaemonSet

To uninstall Qualys Container Sensor, run the following command on Kubernetes master:

```
kubectl delete -f cssensor-containerd-ds.yml
```

**Note:** The persistent storage will need to be removed manually on each worker node.

## Launch sensor without persistent storage

You can run the sensor without using persistent storage on host. In this case data is not stored on host but stored at the `/usr/local/qualys/qpa/data` folder relative to the Sensor.

To launch sensor without persistent storage, modify the `cssensor-containerd-ds.yml` file and provide `--sensor-without-persistent-storage` as an additional value in **args**.

```
args: ["--k8s-mode", "--container-runtime", "containerd", "--  
sensor-without-persistent-storage"]
```

It is recommended to use the `--enable-console-logs` option along with `--sensor-without-persistent-storage` to preserve the logs.

Under **volumeMounts** remove/comment the persistent-volume section.

```
volumeMounts:  
- mountPath: /usr/local/qualys/qpa/data  
  name: persistent-volume
```

Under **volumes** remove/comment the persistent-volume section.

```
volumes:  
- name: persistent-volume  
  hostPath:  
    path: /usr/local/qualys/sensor/data  
    type: DirectoryOrCreate
```

## How to Tag Target Images for the CICD Sensor

To tag CICD images with `qualys_scan_target:<tag>` to mark them for scanning, use the **nerdctl** tool. Use `<image-repo:tag>` for tagging an image with `qualys_scan_target:<any-tag>`.

```
nerdctl -n k8s.io tag <image-repo:tag> qualys_scan_target:<any-  
tag>
```

For example,

```
./nerdctl -n k8s.io tag docker.io/library/known:latest  
qualys_scan_target:known
```

**Notes:**

- The nerdctl binary must be available on the host.
- You must create target images in the k8s.io namespace only

## Deploy in Kubernetes with RKE1 - Docker Runtime

This section explains deploying Kubernetes with "Rancher Kubernetes Engine 1 (RKE1)" using Docker runtime. It also assumes you have the sensor image: [Obtain the Container Sensor Image](#)

### Modify the `cssensor-ds.yml` file

Modify the `cssensor-ds.yml` file (extracted from `QualysContainerSensor.tar.xz`) to provide values for the following parameters. In order for the yml file to work properly, ensure that you do not remove/comment the respective sections mentioned below. Note that you can download the yml file directly from [https://github.com/Qualys/cs\\_sensor](https://github.com/Qualys/cs_sensor)

Ensure all Kubernetes nodes have the latest Qualys sensor image from the URL provided.

```
containers:
  - name: qualys-container-sensor
    image: <CS Sensor image name in the docker hub/private registry>
    args: ["--k8s-mode"]
```

If you want to deploy the sensor for CI/CD environment provide the args value as:

```
args: ["--k8s-mode", "--cicd-deployed-sensor"]
```

If you want to deploy a Registry Sensor provide the args value as:

```
args: ["--k8s-mode", "--registry-sensor"]
```

If you want to disable image scans for General Sensor, add the `--disableImageScan` parameter to args:

```
args: ["--k8s-mode", "--disableImageScan"]
```

If you want to disable log4j vulnerability scanning on your container images, add the `--disable-log4j-scanning` parameter to args:

```
args: ["--k8s-mode", "--disable-log4j-scanning"]
```

If you want to disable log4j static detection for dynamic/static image scans, add the `--disable-log4j-static-detection` parameter to args:

```
args: ["--k8s-mode", "--disable-log4j-static-detection"]
```

If you want to optimize Image scans for General Sensor, add the `--optimize-image-scans` parameter to args:

```
args: ["--k8s-mode", "--optimize-image-scans"]
```

If you have the [SCA scanning](#) feature, then you can enable SCA scanning for container images by adding the "--perform-sca-scan" parameter to args:

```
args: ["--k8s-mode", "--perform-sca-scan"]
```

By default, SCA scans run in online mode. You can choose to disable Internet access for the SCA scan and run the scan in offline mode. Note - We recommend you run the SCA scan in online mode. Quality of software package enumeration for Java substantially degrades when the SCA scan is run in offline mode. The remote maven repository may need to be consulted for an accurate package detection. This can affect accuracy of the vulnerability posture of the image.

```
args: ["--k8s-mode", "--perform-sca-scan" "--disallow-internet-access-for-sca"]
```

The default SCA scan command timeout is 5 minutes (300 seconds). You can overwrite the default timeout with a new value specified in seconds. For example, you may need to increase the SCA scan timeout when scanning large container images to ensure the SCA scan has time to finish.

**Note:** The "--sca-scan-timeout-in-seconds" parameter also applies to secret and malware detection.

```
args: ["--k8s-mode", "--perform-sca-scan" "--sca-scan-timeout-in-seconds=600"]
```

If you want to enable secret detection for container images, add the "--perform-secret-detection" parameter to args:

```
args: ["--k8s-mode", "--perform-secret-detection"]
```

If you want to enable malware detection for container images, add the "--perform-malware-detection" parameter to args:

```
args: ["--k8s-mode", "--registry-sensor", "--perform-secret-detection"]
```

If you want print logs on the console, provide "--enable-console-logs" as an additional value in args.

To restrict the cpu usage to a certain value, change the following: (Optional)

Under **resources** specify the following:

```
resources:  
  limits:  
    cpu: "0.2" # Default CPU usage limit(20% of one core on the host).
```

For example, for limiting the cpu usage to 5%, set resources:limits:cpu: "0.05". This limits the cpu usage to 5% of one core on the host. If there are multiple processors on a node, setting the resources:limits:cpu value applies the CPU limit to one core only.

For example, if you have 4 CPUs on the system and you want to set CPU limit as 20% of overall CPU capacity, then the CPU limit should be set to 0.8 i.e., 80% of one core only which becomes 20% of total CPU capacity.

To disable any CPU usage limit, set `resources:limits:cpu` value to 0.

Optionally, if you want to specify the memory resources for Container Sensor, you can specify it under **resources**. Recommended values for the Container Sensor's memory requests and memory limits are:

```
resources:
  limits:
    cpu: "0.2" # Default CPU usage limit on each node for sensor
    memory: "500Mi"
  requests:
    memory: "300Mi"
```

When either of the memory resource values (limits or requests) is specified for Container Sensor and "--use-kubect!" is supplied in args, we automatically apply both memory requests and memory limits to image scanning containers. Default values are 200Mi and 700Mi, respectively.

Additionally, you could overwrite one or both values by specifying the following variables under **env**. In this example, the values were changed to 300Mi and 800Mi.

```
- name: QUALYS_SCANNING_CONTAINER_MEMORYREQUESTMB
  value: "300Mi"
- name: QUALYS_SCANNING_CONTAINER_MEMORYLIMITMB
  value: "800Mi"
```

Under **env** specify the following:

Activation ID (Required)

```
- name: ACTIVATIONID
  value: XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX
```

Customer ID (Required)

```
- name: CUSTOMERID
  value: XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX
```

Specify proxy information, or remove if not required:

```
- name: qualys_https_proxy
  value: proxy.localnet.com:3128
```

With each scan, we check the node status to see if the node is schedulable or not, and launch the scan only if the node is schedulable. If the node status indicates that the node is unschedulable, then we retry the scan after a default interval of 15 minutes. You can increase or decrease the time the sensor waits before retrying the scan by specifying a different scan retry interval in minutes.

```
- name: UNSCHEDULABLE_NODE_SCAN_RETRY_INTERVAL
  value: "30"
```

Under **volumes** specify the proxy cert path, or remove if not required:

```
- name: proxy-cert-path
  hostPath:
    path: /root/cert/proxy-certificate.crt
    type: File
```

Activation ID and Customer ID are required. Use the Activation ID and Customer ID from your subscription.

If you are using a proxy, ensure that all Kubernetes nodes have a valid certificate file for the sensor to communicate with the Container Management Server.

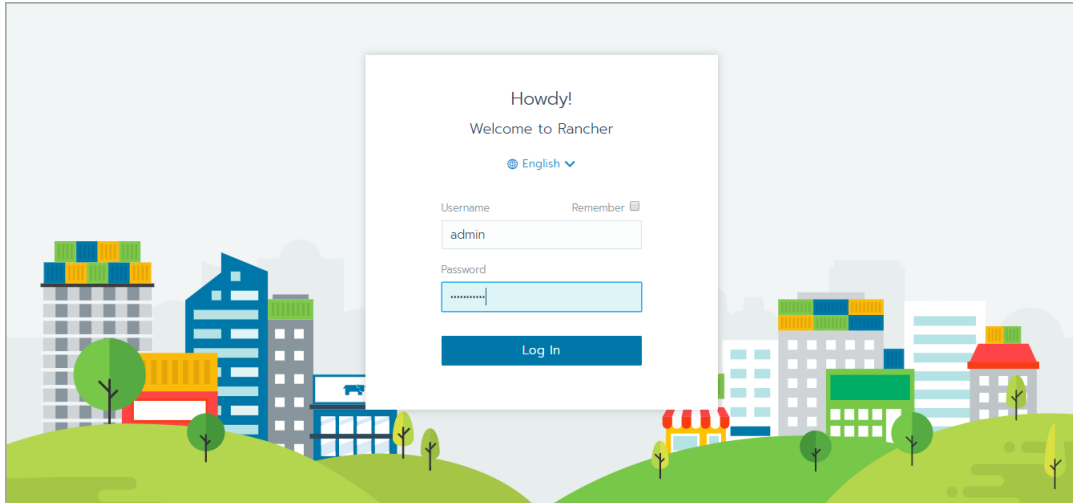
If you are not using a proxy and you have removed the above-mentioned parts, you can remove the following part from **volumeMounts** as well:

```
- mountPath: /etc/qualys/qpacert/custom-ca.crt
  name: proxy-cert-path
```

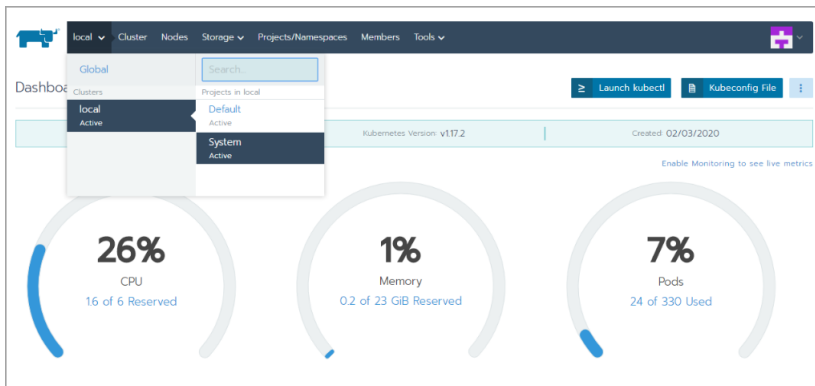
Once you have modified the **cssensor-ds.yml** file, save it.

## Create Qualys sensor DaemonSet in Rancher UI

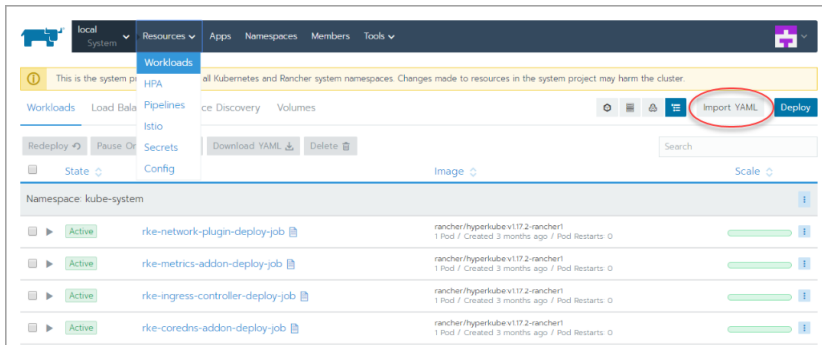
Log in to the Rancher UI to create a Qualys sensor DaemonSet. Use the credentials that were set during the creation setup.



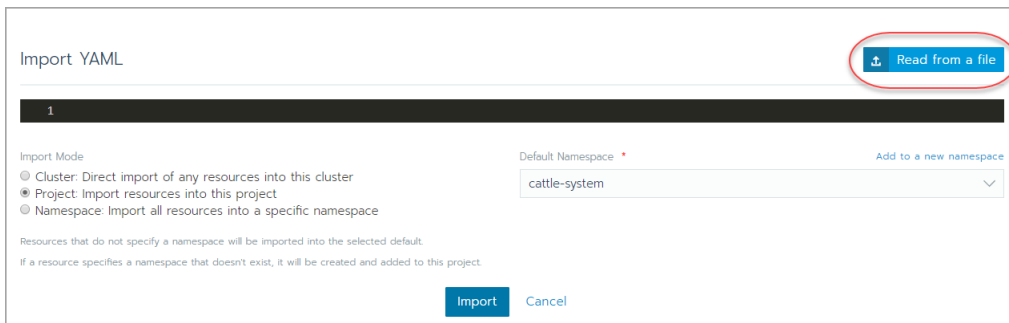
From the top menu select the Cluster and Project under which the DaemonSet for the Qualys sensor is to be deployed in Rancher.



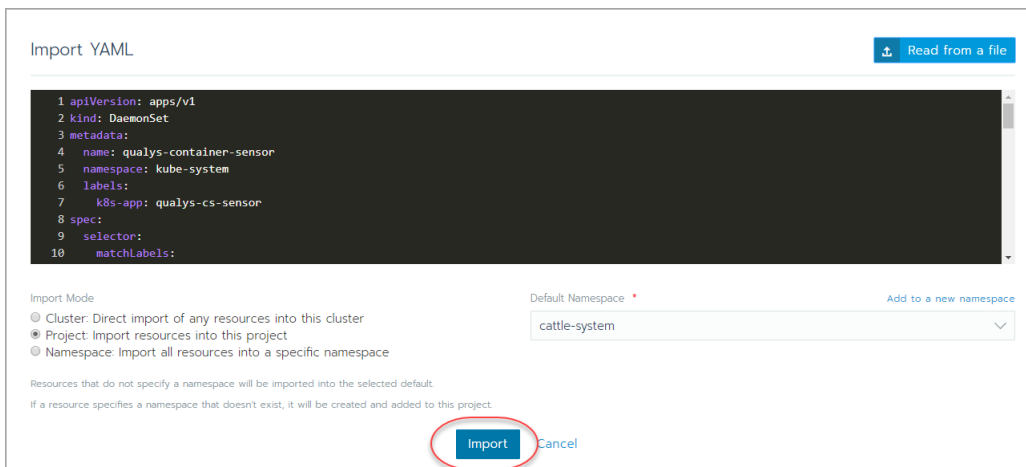
You will be navigated to the Resources tab. Click the "Import YAML" button.



Click the "Read from a file" button, then browse for and select the **cssensor-ds.yml** file that you've modified.

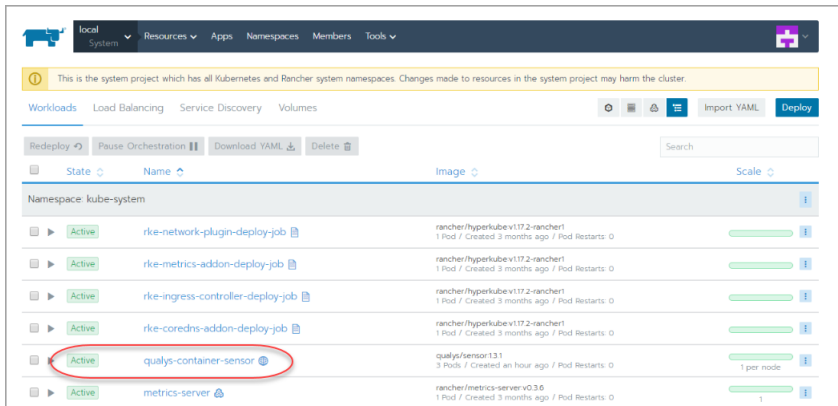


Click the Import button.

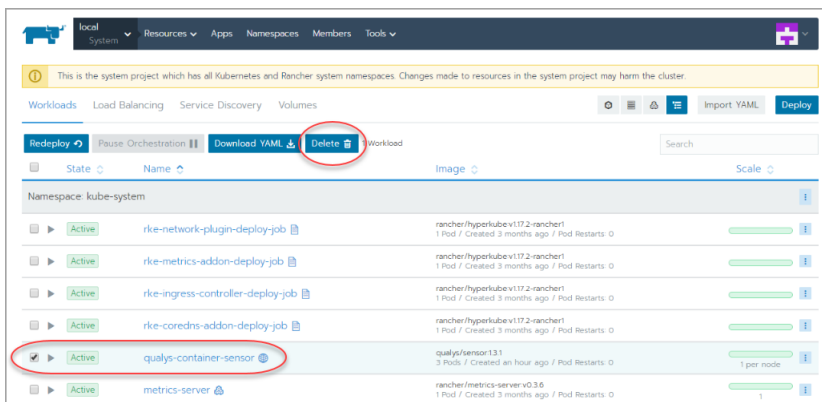




On the Workloads page under kube-system namespace ensure that the qualys-container-sensor DaemonSet is deployed and active.



If you need to uninstall the Qualys Container Sensor, then under kube-system namespace, select the check box next to qualys-container-sensor and click Delete.



## Launch sensor without persistent storage

You can run the sensor without using persistent storage on host. In this case data is not stored on host but stored at the /usr/local/qualys/qpq/data folder relative to the Sensor.

To launch sensor without persistent storage, modify the **cssensor-ds.yml** file and provide "--sensor-without-persistent-storage" as an additional value in args.

```
args: ["--k8s-mode", "--sensor-without-persistent-storage"]
```

It is recommended to use the "--enable-console-logs" option along with "--sensor-without-persistent-storage" to preserve the logs.

Under **volumeMounts** remove/comment the persistent-volume section.

```
volumeMounts:  
- mountPath: /usr/local/qualys/qpq/data  
  name: persistent-volume
```

Under **volumes** remove/comment the persistent-volume section.

```
volumes:  
- name: persistent-volume  
  hostPath:  
    path: /usr/local/qualys/sensor/data  
  type: DirectoryOrCreate
```

## Deploy in Kubernetes with RKE2 - Containerd Runtime

This section assumes you have the sensor image: [Obtain the Container Sensor Image](#)

### Modify the `cssensor-containerd.yml` file

Modify the `cssensor-containerd.yml` file (extracted from `QualysContainerSensor.tar.xz`) to provide values for the following parameters. In order for the yml file to work properly, ensure that you do not remove/comment the respective sections mentioned below. Note that you can download the yml file directly from [https://github.com/Qualys/cs\\_sensor](https://github.com/Qualys/cs_sensor)

Ensure all Kubernetes nodes have the latest Qualys sensor image from the URL provided.

```
containers:  
- name: qualys-container-sensor  
  image: <CS Sensor image name in the docker hub/private registry>  
  args: ["--k8s-mode", "--container-runtime", "containerd"]
```

If you want to deploy the sensor for CI/CD environment provide the args value as:

```
args: ["--k8s-mode", "--container-runtime", "containerd", "--cidc-  
deployed-sensor"]
```

If you want to deploy a Registry Sensor provide the args value as:

```
args: ["--k8s-mode", "--container-runtime", "containerd", "--  
registry-sensor"]
```

If you want to disable image scans for General Sensor, add the `--disableImageScan` parameter to args:

```
args: ["--k8s-mode", "--container-runtime", "containerd", "--  
disableImageScan"]
```

If you want to disable log4j vulnerability scanning on your container images, add the `--disable-log4j-scanning` parameter to args:

```
args: ["--k8s-mode", "--container-runtime", "containerd", "--  
disable-log4j-scanning"]
```

If you want to disable log4j static detection for dynamic/static image scans, add the "--disable-log4j-static-detection" parameter to args:

```
args: ["--k8s-mode", "--container-runtime", "containerd", "--  
disable-log4j-static-detection"]
```

If you want to optimize Image scans for General Sensor, add the "--optimize-image-scans" parameter to args:

```
args: ["--k8s-mode", "--container-runtime", "containerd", "--  
optimize-image-scans"]
```

If you have the [SCA scanning](#) feature, then you can enable SCA scanning for container images by adding the "--perform-sca-scan" parameter to args:

```
args: ["--k8s-mode", "--container-runtime", "containerd", "--  
perform-sca-scan"]
```

By default, SCA scans run in online mode. You can choose to disable Internet access for the SCA scan and run the scan in offline mode. Note - We recommend you run the SCA scan in online mode. Quality of software package enumeration for Java substantially degrades when the SCA scan is run in offline mode. The remote maven repository may need to be consulted for an accurate package detection. This can affect accuracy of the vulnerability posture of the image.

```
args: ["--k8s-mode", "--container-runtime", "containerd", "--  
perform-sca-scan" "--disallow-internet-access-for-sca"]
```

The default SCA scan command timeout is 5 minutes (300 seconds). You can overwrite the default timeout with a new value specified in seconds. For example, you may need to increase the SCA scan timeout when scanning large container images to ensure the SCA scan has time to finish.

**Note:** The "--sca-scan-timeout-in-seconds" parameter also applies to secret and malware detection.

```
args: ["--k8s-mode", "--container-runtime", "containerd", "--  
perform-sca-scan" "--sca-scan-timeout-in-seconds=600"]
```

If you want to enable secret detection for container images, add the "--perform-secret-detection" parameter to args:

```
args: ["--k8s-mode", "--container-runtime", "containerd", "--  
perform-secret-detection"]
```

If you want to enable malware detection for container images, add the "--perform-malware-detection" parameter to args:

```
args: ["--k8s-mode", "--container-runtime", "containerd", "--  
registry-sensor", "--perform-secret-detection"]
```

If you want print logs on the console, provide "--enable-console-logs" as an additional value in args.

To restrict the cpu usage to a certain value, change the following: (Optional)

Under **resources** specify the following:

```
resources:
  limits:
    cpu: "0.2" # Default CPU usage limit (20% of one core on the host).
```

For example, for limiting the cpu usage to 5%, set resources:limits:cpu: "0.05". This limits the cpu usage to 5% of one core on the host. If there are multiple processors on a node, setting the resources:limits:cpu value applies the CPU limit to one core only.

For example, if you have 4 CPUs on the system and you want to set CPU limit as 20% of overall CPU capacity, then the CPU limit should be set to 0.8 i.e., 80% of one core only which becomes 20% of total CPU capacity.

To disable any CPU usage limit, set resources:limits:cpu value to 0.

Optionally, if you want to specify the memory resources for Container Sensor, you can specify it under **resources**. Recommended values for the Container Sensor's memory requests and memory limits are:

```
resources:
  limits:
    cpu: "0.2" # Default CPU usage limit on each node for sensor
    memory: "500Mi"
  requests:
    memory: "300Mi"
```

When either of the memory resource values (limits or requests) is specified for Container Sensor and "--use-kubectrl" is supplied in args, we automatically apply both memory requests and memory limits to image scanning containers. Default values are 200Mi and 700Mi, respectively.

Additionally, you could overwrite one or both values by specifying the following variables under **env**. In this example, the values were changed to 300Mi and 800Mi.

```
- name: QUALYS_SCANNING_CONTAINER_MEMORYREQUESTMB
  value: "300Mi"
- name: QUALYS_SCANNING_CONTAINER_MEMORYLIMITMB
  value: "800Mi"
```

Under **env** specify the following:

Activation ID (Required)

```
- name: ACTIVATIONID
  value: XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX
```

Customer ID (Required)

- name: CUSTOMERID  
value: XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX

Specify proxy information, or remove if not required:

- name: qualys\_https\_proxy  
value: proxy.localnet.com:3128

With each scan, we check the node status to see if the node is schedulable or not, and launch the scan only if the node is schedulable. If the node status indicates that the node is unschedulable, then we retry the scan after a default interval of 15 minutes. You can increase or decrease the time the sensor waits before retrying the scan by specifying a different scan retry interval in minutes.

- name: UNSCHEDULABLE\_NODE\_SCAN\_RETRY\_INTERVAL  
value: "30"

Under **volumes** specify the proxy cert path, or remove if not required:

- name: proxy-cert-path  
hostPath:  
path: /root/cert/proxy-certificate.crt  
type: File

Activation ID and Customer ID are required. Use the Activation ID and Customer ID from your subscription.

If you are using a proxy, ensure that all Kubernetes nodes have a valid certificate file for the sensor to communicate with the Container Management Server.

If you are not using a proxy and you have removed the above-mentioned parts, you can remove the following part from **volumeMounts** as well:

- mountPath: /etc/qualys/qpacert/custom-ca.crt  
name: proxy-cert-path

Under **volumes** specify the volume path of Containerd socket.

- volumes:
  - name: socket-volume  
hostPath:  
path: /run/k3s/containerd/containerd.sock  
type: Socket

Once you have modified the **cssensor-containerd.yml** file, save it.

## Deploy in Google Kubernetes Engine (GKE) with multi-node clusters

Google Kubernetes Engine (GKE) treats persistent volumes as cluster resources and they are shared by all nodes of a cluster. Our current method for using a Persistent Volume Claim (PVC) for retaining state and config of a sensor does not work in a GKE cluster of more than 1 node, as each sensor will attempt to write to the same PVC storage location causing the sensors to fail.

See steps below for deploying the sensor to multi-node clusters of GKE. Be sure to choose the correct yaml file according to the runtime. For Docker Runtime, use **cssensor-ds.yml** (see steps under [GKE node version 1.18 and older](#)). For Containerd Runtime, use **cssensor-containerd-ds.yml** (see steps under [GKE node version 1.19 and later](#)).

### GKE node version 1.18 and older

Follow the steps below for deploying the sensor to multi-node clusters of GKE. The steps in this section apply to GKE node version 1.18 and older.

Modify the **cssensor-ds.yml** with the following **args**:

```
args: ["--k8s-mode", "--sensor-without-persistent-storage", "--enable-console-logs"]
```

Under **volumeMounts** remove/comment persistent-volume and agent-volume sections.

```
volumeMounts:
- mountPath: /usr/local/qualys/qpq/data
  name: persistent-volume
- mountPath: /usr/local/qualys/qpq/data/conf/agent-data
  name: agent-volume
```

Under **volumes** remove/comment the persistent-volume and agent-volume sections.

```
volumes:
- name: persistent-volume
  hostPath:
    path: /usr/local/qualys/sensor/data
    type: DirectoryOrCreate
- name: agent-volume
  hostPath:
    path: /etc/qualys
    type: DirectoryOrCreate
```

### GKE node version 1.19 and later

Follow the steps below for deploying the sensor to multi-node clusters of GKE. The steps in this section apply to GKE node version 1.19 and later.

## Steps for deploying without persistent storage

Modify the `cssensor-containerd-ds.yml` with the following `args`:

```
args: ["--k8s-mode", "--container-runtime", "containerd", "--  
sensor-without-persistent-storage", "--enable-console-logs"]
```

Under `volumeMounts` remove/comment persistent-volume and agent-volume sections.

```
volumeMounts:  
- mountPath: /usr/local/qualys/qpq/data  
  name: persistent-volume  
- mountPath: /usr/local/qualys/qpq/data/conf/agent-data  
  name: agent-volume
```

Under `volumes` remove/comment the persistent-volume and agent-volume sections.

```
volumes:  
- name: persistent-volume  
  hostPath:  
    path: /usr/local/qualys/sensor/data  
    type: DirectoryOrCreate  
- name: agent-volume  
  hostPath:  
    path: /etc/qualys  
    type: DirectoryOrCreate
```

## Steps for deploying with persistent storage

Follow the deployment instructions as outlined in the section [Deploy in Kubernetes - Containerd Runtime](#) but make the following change to the persistent storage path under `volumes`.

Under `volumes`, change the persistent storage path from `"/usr/local/qualys/sensor/data"` to `"/tmp/qualys/sensor/data"` since the default path is not writable on GKE nodes.

```
volumes:  
- name: persistent-volume  
  hostPath:  
    path: /tmp/qualys/sensor/data  
    type: DirectoryOrCreate
```

## Deploy in Kubernetes using Helm Charts

Helm is the package manager for Kubernetes and Helm Charts can be used to deploy the Container Security Sensor in Kubernetes.

### Before you begin:

- Ensure that you have the following application versions:

- **Kubernetes:**
  - Kubernetes 1.17+
  - Helm v3.x.x
- **RedHat OpenShift:**
  - OpenShift v4.0+ (The default container engine is CRI-O.)
- Know your runtime.  
Get the details of your container runtime by running the following command:  

```
kubectl get nodes -o wide
```

## Helm Chart Support for Qualys Container Security Sensor Versions

Sensor Version	Helm Chart Version
1.19.2-0	1.0.0

## Install the Helm Chart

- 1 Download the Qualys Helm chart from the following location:  
'<https://artifacthub.io/packages/helm/qualys-helm-chart/qcs-sensor>'
- 2 Install the Helm chart using the following command:

```
helm install [NAME] [CHART] [flags]
```

For example,

```
helm install qcs-sensor-demo qcs-sensor --namespace qualys --create-namespace
```

- 3 Configure the parameters for the Qualys helm chart. Specify each parameter using the '--set key=value[,key=value]' argument to 'helm install'.

For example,

```
helm install qcs-sensor-demo qcs-sensor --namespace qualys --set containerd.enabled=true
```

**Note:** A command line parameter value takes precedence over the parameter value from values.yaml.

## Configuration Parameters for the Helm Chart

The following are the configurable parameters for the Helm chart:



Parameter	Mandatory?	Description	Default Value
containerd.enabled	Enable only one runtime environment.	Set to true, if the container runtime is containerd.	true
containerd.socketPath	Optional	The path of the mounted volume for the containerd socket.	/var/run/containerd/containerd.sock
crio.enabled	Enable only one runtime environment.	Set to true, if the container runtime is CRI-O.	false
crio.socketPath	Optional	The path of the mounted volume for the CRI-O socket.	/var/run/crio/crio.sock
docker.enabled	Enable only one runtime environment.	Set to true, if the container runtime is docker.	false
docker.socketPath	Optional	The path of the mounted volume for the docker socket.	/var/run/docker.sock
docker.tlsVerify.enabled	Optional	Enables the TLS authentication. The value should be 0 or 1.	false
docker.tlsVerify.tlsCertPath	Optional (Mandatory if DOCKER_TLS_VERIFY=1 is defined.)	Provide the path of the client certificate directory.	-
docker.tlsVerify.dockerHost	Optional (Mandatory if DOCKER_TLS_VERIFY=1 is defined.)	Specify the address on which the docker daemon is configured to listen.	-
docker.tlsVerify.dockerHostValue	Optional	Specify the loopback IPv4 address or hostname, and port <IPv4 address or hostname>:<port#>.	-
openshift	Optional	Set to true, if deploying in OpenShift.	false
qualys.createNamespace	Optional	Set to true, if you want to create a new custom namespace.	false
qualys.namespace	Optional	Provide the namespace to be used. Use the same namespace in values.yaml and on command line when using the helm install command.	qualys
qualys.customerID	Mandatory	Provide the Qualys customer id.	-

Parameter	Mandatory?	Description	Default Value
qualys.activationID	Mandatory	Provide the Qualys activation id.	-
qualys.pod_url	Mandatory	Provide the URL of a Qualys POD.	-
qualys.containerLaunchTimeout	Optional	Specify the launch timeout for the scanning container in minutes.	10
qualys.image	Optional	Specify the name of the Container Security sensor image in the private/dockerhub registry.	qualys/qcs-sensor:1.16.0-0
qualys.imagePullPolicy	Optional	Specify how to pull (download) the specified image.	IfNotPresent
qualys.cpu	Optional	Specify the CPU usage limit in percentage for the sensor. Range: 0-100.	0.2 (20% per core on the host)
qualys.args.withoutPersistentStorage	Optional	Runs the sensor without using the persistent storage on the host.	false
qualys.args.enableConsoleLogs	Optional	Prints logs on the console.	false
qualys.args.cicdDeployedSensor	Optional	Runs the sensor in a CI/CD environment.	false
qualys.args.registrySensor	Optional	Runs the sensor to list and scan the registry assets.	false
qualys.args.concurrentScans	Optional	Specify the number of docker/registry asset scans to run in parallel. Range: 1-20.	4
qualys.args.disableLog4jScanning	Optional	Disables the log4j vulnerability scanning for container images.	false
qualys.args.disableLog4jStaticDetection	Optional	Disables the log4j static detection for dynamic/static image scans.	false
qualys.args.logFilePurgeCount	Optional	The maximum number of sensor log files to archive.	5
qualys.args.logFileSize	Optional	The maximum size for a sensor log file in bytes. You can specify "<digit><K/M/>", where K is kilobytes and M is megabytes.	10M

Parameter	Mandatory?	Description	Default Value
qualys.args.logLevel	Optional	Sets the logging level for sensor. It determines the type of sensor data you want to log. Specify a value from 0 to 5. 0= Error, 1= Warning, 3= Information, 4= Verbose, 5= Trace	3 (Information)
qualys.args.maskEnvVariable	Optional	Masks the environment variables for images and containers.	false
qualys.args.optimizeImageScans	Optional	Optimizes the image scans for the General sensor. It is available for the General sensor type only.	false
qualys.args.disableImageScan	Optional	Disables the image scans for the General sensor.	false
qualys.readOnly	Optional	Runs the sensor in read-only mode.	-
qualys.tolerations.enabled	Optional	Allows the DaemonSet runnable on master nodes.	false
qualys.tolerations.tolerationKey	Optional	Specify the toleration key.	-
qualys.tolerations.tolerationOperator	Optional	Specify the toleration operator.	-
qualys.tolerations.tolerationValue	Optional	Specify the toleration value.	-
qualys.tolerations.tolerationEffect	Optional	Specify the toleration effect.	-
qualys.proxy.enabled	Optional	Set to true, if a proxy is required to connect to the Qualys cloud.	false
qualys.proxy.proxyValue	Optional	Specify the IPv4/IPv6 address or FQDN of the proxy server.	-
qualys.proxy.proxyCertPath	Optional	Specify the path of the proxy certificate file. proxycertpath is applicable only if the proxy has a valid certificate file.	-
qualys.sensorContainerResources.enabled	Optional	Specifies the memory resources for the Sensor container.	false
qualys.sensorContainerResources.memoryLimit	Optional	Specify the memory usage limit for the sensor container.	-

Parameter	Mandatory?	Description	Default Value
qualys.sensorContResources.memoryRequest	Optional	Specify the memory usage request for the sensor container.	-
qualys.scanningContResources.enabled	Optional	Specifies the memory resources for the scanning container.	false
qualys.scanningContResources.memoryLimit	Optional	Specify the memory usage limit for the scanning container.	-
qualys.scanningContResources.memoryRequest	Optional	Specify the memory usage request for the scanning container.	-
qualys.persistentVolumeHostPath	Optional	Specify the directory where the sensor will store the files.	/usr/local/qualys/sensor/data
qualys.persistentVolumeClaim.enabled	Optional	Requests for a storage of a specific size from the gross persistent volume.	false
qualys.persistentVolumeClaim.storageClassName	Mandatory if `qualys.persistentVolumeClaim.enabled` is `true`.	Specify the storage class name used by Kubernetes PersistentVolume.	-
qualys.persistentVolumeClaim.storage	Mandatory if `qualys.persistentVolumeClaim.enabled` is `true`.	Specify the storage memory required. For example, 1Gi for the general/cicd sensor and 10Gi for the registry sensor.	-
qualys.priorityClass.enabled	Optional	Set to true, if you want to use the priority and preemption on PODs.	false
qualys.priorityClass.priorityClassName	Optional	Specify the priority class name used in DaemonSet.	-
qualys.priorityClass.priorityClassValue	Optional	Specify the value that determines the priority. For example, "1000000". Enter an integer less than or equal to 1 billion (1000000000). The higher the value, the higher the priority. Values are relative to the values of other priority classes in the cluster. Reserve very high numbers for system critical pods that you don't want to be preempted (removed).	-

Parameter	Mandatory?	Description	Default Value
qualys.priorityClass.preemptionPolicy	Optional	Specify the preemption policy for a POD.	-
qualys.priorityClass.globalDefault	Optional	Indicates that the value of this PriorityClass should be used for PODs without a priorityClassName.	-

## Namespace Usage

- To create a new custom namespace, use the following command:

```
helm install qcs-sensor-demo qcs-sensor --set  
qualys.createNamespace=true --set  
qualys.namespace=namespace_name -n namespace_name --create-  
namespace
```

- To use an existing namespace, use the following command:

```
helm install qcs-sensor-demo qcs-sensor --set  
qualys.namespace=existing_namespace_name -n  
existing_namespace_name
```

- To use the default (Qualys) namespace, use the following command

```
helm install qcs-sensor-demo qcs-sensor -n qualys
```

## Upgrade the Helm Chart

Use the following command to upgrade the chart:

```
helm upgrade [RELEASE] [CHART] [flags]
```

Where, [RELEASE] is the release name and [CHART] is the chart path.

For example,

```
helm upgrade qcs-sensor-demo qcs-sensor --namespace qualys
```

## Uninstall the Helm Chart

Use the following command to uninstall the chart:

```
helm uninstall RELEASE_NAME [...] [flags]
```

For example,

```
helm uninstall qcs-sensor-demo --namespace qualys
```

## Collection of Kubernetes Cluster Attributes

We added the collection of Kubernetes cluster attributes starting in Sensor version 1.8 and Container Security version 1.10. You can search Kubernetes cluster attributes collected by the sensor when searching containers or sensors in the Container Security UI. Kubernetes cluster attributes include node details, pod details, controller details and more. Use Container Security APIs to see the Kubernetes cluster attributes collected for your containers and sensors.

**Important** - Kubernetes attributes will only be processed for containers discovered after the Container Security version 1.10 release. Kubernetes attributes are collected as part of container inspect processing when containers are discovered for the first time. To fetch Kubernetes cluster attributes for an existing deployment in Kubernetes, you will have to "rollout restart" the existing deployment, which will create new containers and this will start the container inspect processing. Kubernetes attributes will get collected for the newly created containers on Kubernetes clusters.

Use the following command for the "rollout restart":

```
kubectl rollout restart deployment <deployment-name> -n <namespace>
```

### What are the Kubernetes cluster attributes?

- Cluster type (Kubernetes)
- Cluster version
- Project name (collected for projects in Google Cloud Platform)
- Node name and flag indicating whether the node is the master node
- Pod name
- Pod UUID
- Pod namespace
- Pod labels (key and value pairs)
- Controller name
- Controller UUID
- Controller type (e.g. DaemonSet, Deployment, ReplicaSet, etc)

## Update the sensor deployed in Kubernetes

You can update the Container Sensor DaemonSet to the latest version in Kubernetes. This information is applicable for Amazon Elastic Container Service for Kubernetes (Amazon EKS), Google Kubernetes Engine (GKE), and Azure Kubernetes Service (AKS).

Ensure that the Container Sensor has read and write access to the persistent storage and the docker daemon socket.

Perform the following steps on Kubernetes master for updating the Container Sensor.

**Note:** Ensure the Container Sensor DaemonSet is running in the Kubernetes environment.

**Note:** If you already have Qualys Container Sensor running in a namespace other than 'qualys', then you must first uninstall the sensor from the other namespace. Use the new yml extracted from the latest QualysContainerSensor.tar.xz or you can download the yml file directly from [https://github.com/Qualys/cs\\_sensor](https://github.com/Qualys/cs_sensor). Deploy fresh Qualys Container Sensor in 'qualys' namespace. You should use the same path for persistent storage as earlier deployment under hostpath for persistent-volume:

```
- name: persistent-volume
  hostPath:
    path: /usr/local/qualys/sensor/data
    type: DirectoryOrCreate
```

Download the **QualysContainerSensor.tar.xz** file from Qualys Cloud Portal on Kubernetes master.

Untar the sensor package:

```
sudo tar -xvf QualysContainerSensor.tar.xz
```

Copy the Sensor version from the **version-info** file (extracted from QualysContainerSensor.tar.xz).

## Modify the cssensor-ds.yml file

Modify the **cssensor-ds.yml** file (extracted from QualysContainerSensor.tar.xz) to provide values for the following parameters. In order for the yml file to work properly, ensure that you do not remove/comment the respective sections mentioned below. Note that you can download the yml file directly from [https://github.com/Qualys/cs\\_sensor](https://github.com/Qualys/cs_sensor)

Ensure all Kubernetes nodes have the latest Qualys sensor image from the URL provided.

```
containers:
  - name: qualys-container-sensor
    image: <CS Sensor image name in the docker hub/private registry>
    args: ["--k8s-mode"]
```

The image value must be in the format:

```
registryurl/qualys/sensor:<version-info>
```

If you want to deploy the sensor for CI/CD environment provide the **args** value as:

```
args: ["--k8s-mode", "--cicd-deployed-sensor", "--log-level", "5", "--log-file-size", "5M", "--log-file-purge-count", "4"]
```

If you want to deploy a Registry Sensor provide the **args** value as:

```
args: ["--k8s-mode", "--registry-sensor", "--log-level", "5", "--log-file-size", "5M", "--log-file-purge-count", "4"]
```

**Note:** The values for "--log-level", "--log-filesize" and "--log-filepurgecount" in the **args** values above are only samples. Specify appropriate values for your needs.

If you want print logs on the console, provide "--enable-console-logs" as an additional value in **args**.

To restrict the cpu usage to a certain value, change the following: (Optional)

Under **resources** specify the following:

```
resources:
  limits:
    cpu: "0.2" # Default CPU usage limit(20% of one core on the host).
```

For example, for limiting the cpu usage to 5%, set resources:limits:cpu: "0.05". This limits the cpu usage to 5% of one core on the host.

If there are multiple processors on a node, setting the resources:limits:cpu value applies the CPU limit to one core only.

For example, if you have 4 CPUs on the system and you want to set CPU limit as 20% of overall CPU capacity, then the CPU limit should be set to 0.8 i.e., 80% of one core only which becomes 20% of total CPU capacity.

To disable any CPU usage limit, set resources:limits:cpu value to 0.

Under **env** specify the following:

Activation ID (Required: Use the same Activation ID provided in the existing Container Sensor DaemonSet that you are upgrading)

```
- name: ACTIVATIONID
  value: XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX
```

Customer ID (Required: Use the same Customer ID provided in the existing Container Sensor DaemonSet that you are upgrading)

```
- name: CUSTOMERID
  value: XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX
```

Specify proxy information, or remove if not required:

```
- name: qualys_https_proxy
  value: proxy.localnet.com:3128
```

Under **volumes** specify the proxy cert path, or remove if not required:

```
- name: proxy-cert-path
  hostPath:
    path: /root/cert/proxy-certificate.crt
    type: File
```



Activation ID and Customer ID are required. Use the Activation ID and Customer ID from your subscription.

If you are using a proxy, ensure that all Kubernetes nodes have a valid certificate file for the sensor to communicate with the Container Management Server.

If you are not using a proxy and you have removed the above mentioned parts, you can remove the following part from **volumeMounts** as well:

```
- mountPath: /etc/qualys/qp/cert/custom-ca.crt
  name: proxy-cert-path
```

Once you have modified **cssensor-ds.yml**, save the file, and then perform docker login to the registry on Kubernetes master before running the update script (k8s-rolling-update.sh).

For example:

```
docker login mycloudregistry.com
```

The registry should be accessible from all Kubernetes nodes and the Kubernetes master from where the update is being performed.

To update the Container Sensor DaemonSet to the latest version, run the following command on Kubernetes master:

```
./k8s-rolling-update.sh Registry_Url=mycloudregistry.com
```

**Note:** k8s-rolling-update.sh will do docker load, docker tag and docker push to the registry.

## Setting up the Priority to your POD

Prioritizing PODs is more helpful in case of resource contention. You can assign the highest priority to your POD using `PriorityClass` parameter.

To assign priority to your pod, follow the steps mentioned below.

1. Open the Sensor Deployment file (.yml).
2. Locate the below lines of code in the file.

```
#- kind: PriorityClass
#  apiVersion: scheduling.k8s.io/v1
#  metadata:
#    name: qualys-priority-class
#  value: 0
#  preemptionPolicy: PreemptLowerPriority
#  description: Priority class for daemonset
```

and

```
#priorityClassName: qualys-priority-class
```

3. Uncomment the above mentioned lines of code by removing the “#” present preceding the lines.
4. You can change the PriorityClass value as per your requirement.

## Deploying sensor in Docker Swarm

Integrate the Container Sensor into the DaemonSet like other application containers and set the replication factor to 1 to ensure there is always a sensor deployed on the Docker Host.

Perform the following steps for creating a DaemonSet for the Qualys sensor to be deployed in Docker Swarm.

Download the **QualysContainerSensor.tar.xz** file from Qualys Cloud Portal on a Linux computer.

Untar the sensor package:

```
sudo tar -xvf QualysContainerSensor.tar.xz
```

Use the following commands to push the qualys sensor image to a repository common to all nodes in the Docker Swarm cluster:

```
sudo docker load -i qualys-sensor.tar
sudo docker tag <IMAGE NAME/ID> <URL to push image to the repository>
sudo docker push <URL to push image to the repository>
```

For example:

```
sudo docker load -i qualys-sensor.tar
sudo docker tag c3fa63a818df myregistry.com/qualys_sensor:xxx
sudo docker push myregistry.com/qualys_sensor:xxx
```

**Note:** Do not use the examples as is. Replace the registry/image path with your own.

### Modify the **cssensor-swarm-ds.yml** file

Modify the **cssensor-swarm-ds.yml** file (extracted from QualysContainerSensor.tar.xz) to provide values for the following parameters. In order for the yml file to work properly, ensure that you do not remove/comment the respective sections mentioned below. Note that you can download the yml file directly from [https://github.com/Qualys/cs\\_sensor](https://github.com/Qualys/cs_sensor)

Ensure that all master and worker nodes have the latest Qualys sensor image from the URL provided.

```
qualys-container-sensor:
  image: <CS Sensor image name in the docker hub/private registry>
  deploy:
    mode: global # Deploy 1 container on each node == DaemonSet
    command: ["--swrm-mode"]
```

If you want to deploy the sensor for CI/CD environment provide the **command** value as:

```
command: ["--swrm-mode", "--cicd-deployed-sensor", "--log-level", "5", "--log-filesize", "5M", "--log-filepurgecount", "4"]
```

If you want to deploy a Registry Sensor provide the **command** value as:

```
command: ["--swrm-mode", "--registry-sensor", "--log-level", "5", "--log-filesize", "5M", "--log-filepurgecount", "4"]
```

**Note:** The values for "--log-level", "--log-filesize" and "--log-filepurgecount" in the **command** values above are only samples. Specify appropriate values for your needs.

If you want print logs on the console, provide "--enable-console-logs" as an additional value in **command**.

If you want to mask environment variables for images and containers in sensor logs and in the Container Security UI, add "--mask-env-variable" as an additional value in **command**.

To restrict the cpu usage to a certain value, change the following: (Optional)

Under **deploy** specify the following:

```
mode: global # Deploy 1 container on each node == DaemonSet
resources:
  limits:
    cpus: '0.20' # Default CPU usage limit(20% of one core on the host.
```

For example, for limiting the cpu usage to 5%, set deploy:resources:limits:cpus: "0.05". This limits the cpu usage to 5% of one core on the host.

If there are multiple processors on a node, setting the deploy:resources:limits:cpus value applies the CPU limit to one core only.

For example, if you have 4 CPUs on the system and you want to set CPU limit as 20% of overall CPU capacity, then the CPU limit should be set to 0.8 i.e., 80% of one core only which becomes 20% of total CPU capacity.

To disable any CPU usage limit, set deploy:resources:limits:cpus value to 0.

Under **environment** specify the following:

```
environment:
  ACTIVATIONID: XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX
  CUSTOMERID: XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX
  qualys_https_proxy: proxy.qualys.com:3128
```

Activation ID and Customer ID are required. Use the Activation ID and Customer ID from your subscription. You can remove the proxy information if not required.

Under **volumes** ensure that you provide the following information:

```
volumes:
- type: bind
  source: /var/run/
  target: /var/run/
- type: volume
```

```

    source: persistent-volume
    target: /usr/local/qualys/qpq/data/
  - type: bind
    source: /etc/qualys # Must exist !
    target: /usr/local/qualys/qpq/data/conf/agent-data

```

Keep source as "persistent-volume". This ensures that the source directory in volume mapping is set to docker swarm root directory (i.e. /data/docker/volumes).

**/etc/qualys** directory must exist on all masters and worker nodes for successful volume mapping.

```

volumes:
  persistent-volume:

```

Under **configs** ensure that you provide the following information:

```

configs:
  proxy-cert-path:
    file: /root/cert/proxy-certificate.crt

```

If you are using a proxy, ensure that all masters and worker nodes have a valid certificate file for the sensor to communicate with the Container Management Server.

If you are not using a proxy and you have removed `qualys_https_proxy` from environment, you can remove the following parts as well:

```

configs:
  - source: proxy-cert-path
    target: /etc/qualys/qpq/cert/custom-ca.crt

```

```

configs:
  proxy-cert-path:
    file: /root/cert/proxy-certificate.crt

```

Once you have modified the **cssensor-swarm-ds.yml** file, run the following command on docker swarm master/leader to create a stack:

```
docker stack deploy -c cssensor-swarm-ds.yml qualys-container-sensor
```

If you need to uninstall Qualys Container Sensor, run the following command on docker swarm master/leader:

```
docker stack rm qualys-container-sensor
```

## Launch sensor without persistent storage

You can run the sensor without using persistent storage on host. In this case data is not stored on host but stored at the `/usr/local/qualys/qpq/data` folder relative to the Sensor.

To launch sensor without persistent storage, modify the `cssensor-swarm-ds.yml` file and provide `--sensor-without-persistent-storage` as an additional value in **command**.

```
command: ["--swrm-mode", "--sensor-without-persistent-storage"]
```

It is recommended to use the `--enable-console-logs` option along with `--sensor-without-persistent-storage` to preserve the logs.

Under **volumes** (outside **services**) remove/comment the persistent-volume section.

```
volumes:
  persistent-volume:
```

Under **volumes** (inside **services**) remove/comment the persistent-volume section.

```
services:
  volumes:
    - type: volume
      source: persistent-volume
      target: /usr/local/qualys/qpq/data/
```

# Deploying sensor in AWS ECS Cluster

Perform the following steps to deploy Qualys Container Sensor as a daemon service in Amazon ECS cluster.

**Prerequisites:** AWS ECS Cluster should be up and running.

Download the **QualysContainerSensor.tar.xz** file from Qualys Cloud Portal on a Linux computer.

Untar the sensor package:

```
sudo tar -xvf QualysContainerSensor.tar.xz
```

Use the following commands to push the qualys sensor image to a repository common to all nodes in the cluster:

```
sudo docker load -i qualys-sensor.tar
sudo docker tag <IMAGE NAME/ID> <URL to push image to the repository>
sudo docker push <URL to push image to the repository>
```

For example:

```
sudo docker load -i qualys-sensor.tar
sudo docker tag c3fa63a818df 20576712438.dr.ecr.us-east-1.amazonaws.com/container-sensor:qualys-sensor-xxx
sudo docker push 20576712438.dr.ecr.us-east-1.amazonaws.com/container-sensor:qualys-sensor-xxx
```

**Note:** Do not use the examples as is. Replace the registry/image path with your own.

## Modify the **cssensor-aws-ecs.json** file

Modify the **cssensor-aws-ecs.json** file (extracted from QualysContainerSensor.tar.xz) to provide values for the following parameters. In order for the json file to work properly, ensure that you do not remove/comment the respective sections mentioned below. Note that you can download the json file directly from [https://github.com/Qualys/cs\\_sensor](https://github.com/Qualys/cs_sensor)

```
"containerDefinitions": [
  {
    "name": "qualys-container-sensor",
    "image": "20576712438.dr.ecr.us-east-1.amazonaws.com/container-sensor:qualys-sensor-xxx",
    "cpu": 10,
    "memory": 512,
    "essential": true,
    "command": [
      "--ecs-mode"
    ],
  },
]
```

Specify appropriate values for **cpu** (no. of vcpu) and **memory** (size in MB).

If you want to deploy the sensor for CI/CD environment provide the **command** value as:

```
"command": [
  "--ecs-mode",
  "--cicd-deployed-sensor",
]
```

If you want to deploy a Registry Sensor provide the **command** value as:

```
"command": [
  "--ecs-mode",
  "--registry-sensor",
]
```

If you want to change the log level, provide "--log-level", "<a number between 0 and 5>" as an additional value in **args**, e.g if you want logs in trace provide:

```
"command": [ "--ecs-mode", "--log-level", "5", ]
```

If you want to define the number of archived qpa.log files to be generated and size per log file, provide "-log-filesize", "<digit></K/M>" where "K" means kilobyte and "M" means megabyte, and "log-filepurgecount", "<digit>" as an additional value in **args**. Default is "log-filesize": "10M" and "log-filepurgecount": "5" applied via config.

--log-filesize": can be used to define the maximum size per log file. For example, "10K" (kilobytes), "10M" (megabytes) or "10" (bytes).

--log-filepurgecount": can be used to define the number of archived log files to be generated. Please note that there will always be current qpa.log file in log/directory.

```
"command": [ "--ecs-mode", "--log-level", "5", "--log-filesize",
  "5M", "--log-filepurgecount", "4"]
```

If you want to print logs on the console, provide "--enable-console-logs" as an additional value in **command**.

If you want to disable image scans for General Sensor (supported for all Runtimes), add the "--disableImageScan" parameter:

```
"command": [ "--ecs-mode", "--disableImageScan"]
```

--disable-log4j-scanning" can be used to disable log4j vulnerability scanning for container images.

```
"command": [ "--ecs-mode", "--disable-log4j-scanning"]
```

--disable-log4j-static-detection" can be used to disable log4j static detection for dynamic/static image scans.

```
"command": [ "--ecs-mode", "--disable-log4j-static-detection"]
```



If you want to optimize image scans for General Sensor, add the "--optimize-image-scans" parameter:

```
"command": [ "--ecs-mode", "--optimize-image-scans"]
```

If you want to enable secret detection for container images, add the "--perform-secret-detection" parameter. Note that secret detection is supported only on CICD and registry sensors.

```
"command": [ "--ecs-mode", "--perform-secret-detection"]
```

If you want to enable malware detection for container images, add the "--perform-malware-detection" parameter. Note that malware detection is supported only on registry sensor.

```
"command": [ "--ecs-mode", "--perform-malware-detection"]
```

Under **environment** specify the following:

```
"environment": [
  {
    "name": "ACTIVATIONID",
    "value": "XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX"
  },
  {
    "name": "CUSTOMERID",
    "value": "XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX"
  },
  {
    "name": "POD_URL",
    "value": "<Specify POD URL>"
  },
  {
    "name": "qualys_https_proxy",
    "value": "<proxy FQDN or IP address>:<port#>"
  }
]
```

Activation ID and Customer ID are required. Use the Activation ID and Customer ID from your subscription. Specify proxy information, or remove the section if not required. If you remove the proxy section, ensure that json indentation is correct.

If you are not using a proxy and you have removed **qualys\_https\_proxy** from environment, you can remove the following parts from **mountPoints** and **volumes**:

```
configs:
```

```

- source: proxy-cert-path
  target: /etc/qualys/qpa/cert/custom-ca.crt

configs:
  proxy-cert-path:
    file: /root/cert/proxy-certificate.crt

```

If proxy section is removed from environment, then remove **proxy-cert-path** sections under **mountPoints** and **volumes** as well:

```

"mountPoints": [
  {
    "sourceVolume": "proxy-cert-path",
    "containerPath": "/etc/qualys/qpa/cert/custom-ca.crt"
  },
]

"volumes": [
  {
    "name": "proxy-cert-path",
    "host": {
      "sourcePath": "/root/cert/proxy-certificate.crt"
    }
  }
]

```

Under **volumes**, provide information for persistent\_volume. If you specify a custom location for persistent\_volume, it would get created if not already available on the Docker Host. Once you are done with the changes, save the **cssensor-aws-ecs.json** file.

## Import the json file into Amazon ECS UI to complete the sensor deployment

On the Amazon ECS UI, under Task Definitions, click **Create New Task Definition**.

Select the launch type compatibility as EC2. Provide the Task Definition name, and then provide Task Role, Network Mode, and Task Execution Role, if applicable.

Scroll to the bottom of the page and select **Configure via JSON**. Remove any existing content and then copy-paste the entire contents of the **cssensor-aws-ecs.json** file.

Click **Create** to create the Task Definition. Once created, it should get listed under Task Definitions.

Now go to Clusters, and click the cluster name on which you want to deploy the sensor.

Under Services tab, click **Create**. Select the launch type as EC2. Select the Task Definition you created above and its revision, and then select a cluster. Provide the Service name, Service type as "DAEMON", and then configure Network, Load Balancing, and Auto Scaling

if applicable. Review the information, and then click **Create** to create the Service. Once created, it should get listed under Services. Verify that the service status is Active. In the tasks tab, verify that tasks are running on all ECS containers.

## Stopping Qualys sensor on Amazon ECS Cluster

If you want to stop the Qualys container sensor from running on all containers, simply delete the service from the Services tab. This will kill the qualys-container-sensor service, but will not remove the sensor from the AWS ECS instances.

## Launch sensor without persistent storage

You can run the sensor without using persistent storage on host. In this case data is not stored on host but stored at the `/usr/local/qualys/qpa/data` folder relative to the Sensor.

To launch sensor without persistent storage, modify the `cssensor-aws-ecs.json` file and provide `--sensor-without-persistent-storage` as an additional value in **command**.

```
"command": [
    "--ecs-mode",
    "--sensor-without-persistent-storage"
],
```

It is recommended to use the `--enable-console-logs` option along with `--sensor-without-persistent-storage` to preserve the logs.

Under **mountPoints** remove the persistent-volume section.

```
"mountPoints": [
  {
    "sourceVolume": "persistent_volume",
    "containerPath": "/usr/local/qualys/qpa/data"
  },
```

Under **volumes** remove the persistent-volume section.

```
"volumes": [
  {
    "name": "persistent_volume",
    "host": {
      "sourcePath": "/usr/local/qualys/sensor/data"
    }
  },
```

# Scan Container Images in AWS Fargate (ECS)

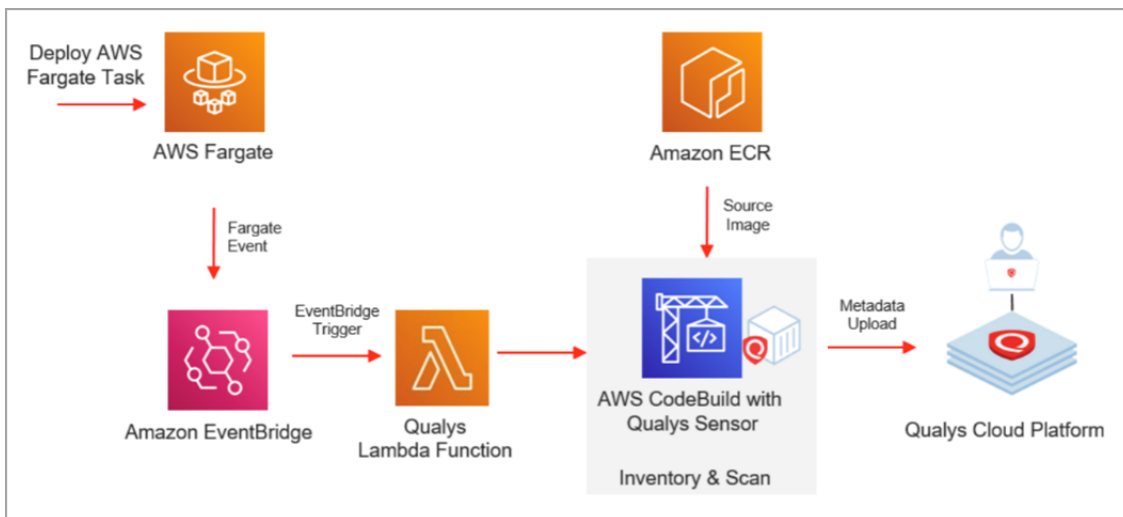
Qualys Container Security can be used to secure AWS Fargate. AWS Fargate is a serverless compute engine for containers that works with Amazon Elastic Container Service (ECS). This feature allows you to know the containers running on AWS Fargate, perform vulnerability and compliance scanning on container images launched by Amazon Fargate tasks (ECS), and view the findings to take remediation actions.

Since AWS Fargate is serverless, the solution launches a sensor whenever a new Fargate task is deployed. We will use AWS CloudFormation and a Qualys Lambda function to trigger scanning automatically. You'll configure a CloudFormation template with your subscription details and a Qualys Lambda function with the Qualys S3 bucket name & S3 bucket key to trigger image scanning of images pulled from Amazon Elastic Container Registry (ECR).

## How it works

We support scanning Docker images pulled from Amazon Elastic Container Registry (Amazon ECR) with x86\_64 architecture.

When an AWS ECS Fargate task is launched, the AWS EventBridge rule created during Qualys deployment consumes the event. The EventBridge rule is set in such a way that it triggers the Qualys scanning Lambda function. The Qualys Lambda function then processes the event received from EventBridge to decide on image scanning. The Qualys Lambda function launches the AWS CodeBuild to run the Qualys sensor, which pulls the image from Amazon ECR and then performs the vulnerability and compliance scan on the image. After a successful image scan, image metadata gets uploaded to the Qualys Cloud Platform for evaluation, and users can view details from the Container Security UI and API.



For Qualys Private Cloud Platform (PCP), we have provided guidelines for setting up the connectivity between AWS and your Private Cloud Platform. Refer to [Qualys Container Security - Securing AWS Fargate on Qualys Private Cloud Platform \(PCP\)](#).

## Qualys AWS ECS Fargate Image Scan Stack Deployment Steps

Follow the steps below to set up AWS ECS Fargate image scanning. You'll create a CloudFormation stack using a Qualys CloudFormation template and a Qualys Lambda function.

### Prerequisites

Before you begin, make sure you have the following items ready to successfully launch the CloudFormation AWS ECS Fargate image scanning stack.

- The AWS region where you want to deploy the stack.
- Qualys CloudFormation template URL: <https://qualys-cs-image-scanning-cloud-formation-template.s3.amazonaws.com/qcs-ecs-fargate-image-scanning-cf.template>
- Environment details for your Qualys subscription: POD URL, Activation ID, Customer ID. To get the Activation ID and Customer ID auto-generated for your subscription, go to **Configurations** > **Sensors** in the UI, click **Download Sensor**. Then click any sensor type. The installation command on the **Installation Instructions** page contains your Activation ID and Customer ID.
- Qualys Lambda function (Zip file). You'll need the S3 bucket name and bucket key for configuring the ECS scanning Lambda function. See the following section to get the S3 bucket name and bucket key for each AWS region: [Qualys CS Lambda Function S3 Bucket Names and Keys](#)
- Qualys sensor image (version 1.18 or later). Refer to [How to Get the Qualys Sensor Image](#)

### How to Get the Qualys Sensor Image

You have these options for the Qualys Container Security Sensor image:

- Use from Docker Hub directly
- Use from Docker Hub but push the image to your ECR repository (public)
- Load from tar and push it to your ECR repository (public)

The sections that follow describe these options in more detail.

#### Use from Docker Hub directly

You can use the sensor image directly from Docker Hub. The Container Security Sensor on Docker Hub is available as:

```
qualys/qcs-sensor: <tag>
qualys/qcs-sensor:latest
```

Look up the most recent tag in Docker Hub.

**Use from Docker Hub but push the image to your ECR repository (public)**

Use the following commands to push the qualys sensor image to the ECR public repository:

```
sudo docker pull qualys/qcs-sensor:latest
sudo docker tag qualys/qcs-sensor:latest <URL to push image to ECR public repository>
sudo docker push <URL to push image to ECR public repository>
```

For example:

```
sudo docker pull qualys/qcs-sensor:latest
sudo docker tag c3fa63a818df
public.ecr.aws/y4h7m2t8/qualys/sensor:latest
sudo docker push public.ecr.aws/y4h7m2t8/qualys/sensor:latest
```

**Load from tar and push the image to your ECR repository (public)**

Download the **QualysContainerSensor.tar.xz** file from Qualys Cloud Portal on a Linux computer. In the Container Security UI, download the Binary (tar.xz) file by going to **Configurations > Sensors > Download Sensor** and click any sensory type. Then pick **Linux** and the **Binary (tar.xz)** tab. Click **Download Now** to get the tar file.

Untar the sensor package:

```
sudo tar -xvf QualysContainerSensor.tar.xz
```

Use the following commands to push the qualys sensor image to the ECR public repository:

```
sudo docker load -i qualys-sensor.tar
sudo docker tag <IMAGE NAME/ID> <URL to push image to ECR public repository>
sudo docker push <URL to push image to ECR public repository>
```

For example:

```
sudo docker load -i qualys-sensor.tar
sudo docker tag c3fa63a818df
public.ecr.aws/y4h7m2t8/qualys/sensor:latest
sudo docker push public.ecr.aws/y4h7m2t8/qualys/sensor:latest
```

## How to deploy the stack using AWS Console

We use AWS CloudFormation for scanning container images in AWS Fargate ECS.

Follow these deployment instructions:

- 1) Log into your AWS Console.
- 2) Go to **CloudFormation**, click **Create Stack** and select **With new Resources**.
- 3) Under **Specify template**, in the **Amazon S3 URL** field, enter the Qualys CloudFormation Template S3 URL (see URL in the [Prerequisites](#) section). Then, click **Next** to continue to the template configuration.

**Specify template**  
A template is a JSON or YAML file that describes your stack's resources and properties.

Template source  
Selecting a template generates an Amazon S3 URL where it will be stored.

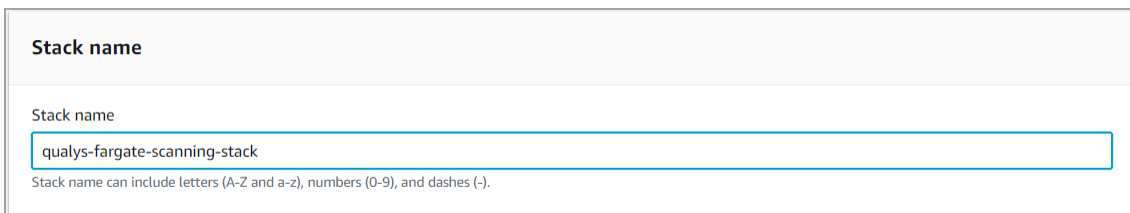
Amazon S3 URL  Upload a template file

Amazon S3 URL  
https://  
Amazon S3 template URL

S3 URL: Will be generated when URL is provided View in Designer

Cancel Next

- 4) Under **Stack name**, enter a name for the Qualys AWS Fargate scanning stack, such as qualys-fargate-scanning-stack.



**Stack name**

Stack name  
qualys-fargate-scanning-stack  
Stack name can include letters (A-Z and a-z), numbers (0-9), and dashes (-).

- 5) Under **Qualys Global Configuration**, provide environment details for your subscription, including POD URL, Activation ID, and Customer ID.

To get the Activation ID and Customer ID auto-generated for your subscription, go to **Configurations > Sensors** in the UI, click **Download Sensor**, and then click any sensor type. The installation command on the **Installation Instructions** page contains your Activation ID and Customer ID.

**QualysPodUrl:** Enter the Container Security Server URL for the Qualys Platform where your account is located (e.g. <https://cmsqagpublic.qg2.apps.qualys.com/ContainerSensor> for US2 Platform). If you are not sure of the URL, refer to the following link to identify your Qualys platform and get the Container Security Server URL:

<https://www.qualys.com/platform-identification/#container-security-servers>

**QualysCustomerId:** Enter your Qualys subscription's customer ID.

**QualysActivationId:** Enter your Qualys subscription's activation ID.

**Qualys Global Configuration**

QualysPodUrl  
Enter your Qualys POD URL

QualysCustomerId  
Enter your Qualys Customer id

QualysActivationId  
Enter your Qualys Activation id

6) Under **Qualys CS Lambda function Configuration**, provide Lambda S3 bucket name, key and log level. See the following link to get the bucket name and key values: [Qualys CS Lambda Function S3 Bucket Names and Keys](#)

**QualysLambdaFunctionS3BucketName:** Enter the S3 bucket name for the Qualys Lambda function.

**QualysLambdaFunctionS3BucketKey:** Enter the S3 bucket key as the name of the Qualys Lambda function (e.g., qcslambda-1.0.0-34-PUBLIC.zip).

**QualysLambdaLogLevel:** Select a log level for the Qualys Lambda function. The default value is "info". Keep the default value or select another log level if more verbose logging is needed.

**Qualys CS Lambda function Configuration**

QualysLambdaFunctionS3BucketName  
Enter the S3 bucket name for Qualys lambda function deployment package

QualysLambdaFunctionS3BucketKey  
Enter the S3 bucket key for Qualys lambda function deployment package

QualysLambdaLogLevel  
Enter the log level for Qualys lambda function



7) Under **Qualys CS Sensor Configuration**, provide the following details:

**QualysSensorImage**: Enter the name of the CS 1.18.0 sensor image.

**QualysSensorLogLevel**: Select a log level (0-5) for the Qualys sensor. The default value is 3 (Information). Keep the default value or select another log level if more verbose logging is needed.

**QualysSensorCliParameters**: You can keep this field empty.

**Qualys CS Sensor Configuration**

**QualysSensorImage**  
Enter the Qualys CS sensor image name with tag

**QualysSensorLogLevel**  
Enter the log level for Qualys Container Security Scanner

**QualysSensorCliParameters**  
Enter the list of CLI parameters that needs to configured for Qualys CS sensor

8) Click **Next** to continue through the workflow. On the final page, you will need to select the **I acknowledge that AWS CloudFormation might create IAM resources** check box.

Capabilities

**i** The following resource(s) require capabilities: [AWS::IAM::Role]

This template contains Identity and Access Management (IAM) resources that might provide entities access to make changes to your AWS account. Check that you want to create each of these resources and that they have the minimum required permissions. [Learn more](#)

I acknowledge that AWS CloudFormation might create IAM resources.

9) Click **Create stack**. That's it!

## Resources Created

When the stack creation is successful, several resources are created and they'll appear in the **Resources** section, as shown below. In this example, the resources were created for a stack named fargate-demo.

Logical ID	Physical ID	Type	Status	Status reason	Module
QualysECSFargateImageScanningBuildProject	QualysECSFargateImageScanning	AWS::CodeBuild::Project	UPDATE_COMPLETE	-	-
QualysECSFargateImageScanningInvokeLambdaPermission	fargate-demo- QualysECSFargateImageScanningInvokeLambdaPermission-10M4H7PQZVL8G	AWS::Lambda::Permission	CREATE_COMPLETE	-	-
QualysECSFargateImageScanningLambda	QualysECSFargateImageScanningLambda	AWS::Lambda::Function	UPDATE_COMPLETE	-	-
QualysECSFargateImageScanningLambdaRole	fargate-demo- QualysECSFargateImageScanningLambdaRole-10TJJD585AXX	AWS::IAM::Role	CREATE_COMPLETE	-	-
QualysECSFargateImageScanningRule	QualysECSFargateImageScanningRule	AWS::Events::Rule	CREATE_COMPLETE	-	-
QualysECSFargateImageScanningServiceRole	fargate-demo- QualysECSFargateImageScanningServiceRole-SP6940MZM9LU	AWS::IAM::Role	CREATE_COMPLETE	-	-

Here's another look at the resources created.

Logical ID	Type
QualysECSFargateImageScanningBuildProject	AWS::CodeBuild::Project
QualysECSFargateImageScanningInvokeLambdaPermission	AWS::Lambda::Permission
QualysECSFargateImageScanningLambda	AWS::Lambda::Function
QualysECSFargateImageScanningLambdaRole	AWS::IAM::Role
QualysECSFargateImageScanningRule	AWS::Events::Rule
QualysECSFargateImageScanningServiceRole	AWS::IAM::Role

## Qualys CS Lambda Function S3 Bucket Names and Keys

### CS Lambda Function S3 Bucket Name

Refer to the table below to get the Qualys CS Lambda function S3 bucket name for your region.

AWS Region	Bucket Name
us-east-1	qualys-cs-image-scanning-lambda-function-us-east-1
us-east-2	qualys-cs-image-scanning-lambda-function-us-east-2
us-west-1	qualys-cs-image-scanning-lambda-function-us-west-1
us-west-2	qualys-cs-image-scanning-lambda-function-us-west-2
me-south-1	qualys-cs-image-scanning-lambda-function-me-south-1
eu-central-1	qualys-cs-image-scanning-lambda-function-eu-central-1
eu-west-1	qualys-cs-image-scanning-lambda-function-eu-west-1
eu-west-2	qualys-cs-image-scanning-lambda-function-eu-west-2
eu-north-1	qualys-cs-image-scanning-lambda-function-eu-north-1

### CS Lambda Function S3 Bucket Key

The Qualys CS Lambda function S3 bucket key is the same across all AWS regions.

The bucket key you'll enter is: **qcslambda-1.0.0-34-PUBLIC.zip**

## Compliance with CIS Benchmark for Docker

Qualys Container Security adheres to the CIS Benchmark for Docker for our Sensor image. This section provides guidance on how to use the Sensor image in a way that complies with the CIS Benchmark for Docker. We've provided instructions below for a number of controls so you can operate the Sensor in a compliant manner.

<b>CIS Docker Benchmark</b>	<b>5.9 Ensure that the host's network namespace is not shared (Automated)</b>
Qualys Control	CID 10811 "Status of the network mode set for the Docker containers on the host system"
Resolution	<p>To meet compliance with this control, Qualys Container Sensor should run without the command line argument <code>--net=host</code>.</p> <p>However, it should be noted that when sensor is launched without <code>--net=host</code>, sensor will not be able to detect its host IP address. By default, the sensor container will be assigned a default IP from the pool assigned to the network. All the containers on the host will be mapped to the IP assigned to the sensor container and each container's host IP association will be missing. Hence, not to lose the asset/host association we recommend the sensor to be launched with the argument.</p>
Installsensor.sh Command	Remove <code>--net=host</code> from the <code>installsensor.sh</code> script to run sensor without this command.
Docker Run Command	<p>Do not specify <code>--net=host</code> as part of the Docker run command when deploying a sensor.</p> <pre>sudo docker run -d --restart on-failure -v /var/run/docker.sock:/var/run/docker.sock:ro -v /usr/local/qualys/sensor/data:/usr/local/qualys/qpa/data -e ACTIVATIONID=&lt;Activation id&gt; -e CUSTOMERID=&lt;Customer id&gt; -e POD_URL=&lt;POD URL&gt; --name qualys-container-sensor qualys/qcs-sensor:latest</pre>
Kubernetes DaemonSet	For deployments in Kubernetes with Docker Runtime, remove <code>hostNetwork: true</code> from <code>cssensor-ds.yml</code> .

<b>CIS Docker Benchmark</b>	<b>5.10 Ensure that the memory usage for containers is limited (Automated)</b>
Qualys Control	CID 10812 "Status of the memory usage limitation for the Docker containers on the host system"

Resolution	To meet compliance with this control, Qualys Container Sensor should run with the command line argument MemoryUsageLimit for the installsensor.sh script or -m as part of Docker run command when deploying a sensor. The value should be formatted as <digit><unit> where unit can be any of the following: b (bytes), k (kilobytes), m (megabytes), g (gigabytes). The recommended value is 500m for 500 megabytes.
Installsensor.sh Command	Specify MemoryUsageLimit as a command line argument for installsensor.sh script.  sudo ./installsensor.sh ActivationId=<Activation id> CustomerId=<Customer id> Storage=/tmp/qualys/sensor/data MemoryUsageLimit=500m -s
Docker Run Command	Specify -m as part of the Docker run command when deploying a sensor.  sudo docker run -d --restart on-failure -m 500m -v /var/run/docker.sock:/var/run/docker.sock:ro -v /usr/local/qualys/sensor/data:/usr/local/qualys/qpa/data -e ACTIVATIONID=<Activation id> -e CUSTOMERID=<CustomerId> -e POD_URL=<POD URL> --net=host --name qualys-container-sensor qualys/qcs-sensor:latest
Kubernetes DaemonSet	For deployments in Kubernetes with Docker Runtime, add the memory usage limit to the resources section in the cssensor-ds.yml file, as shown below.  resources: limits: memory: "500M"

<b>CIS Docker Benchmark</b>	<b>5.11 Ensure that CPU priority is set appropriately on containers (Automated)</b>
Qualys Control	CID 10813 "Status of the CPU share weighting set for the Docker containers on the host system"
Resolution	To meet compliance with this control, Qualys Container Sensor should run with the command line argument CpuShares for the installsensor.sh script or --cpu-shares as part of Docker run command when deploying a sensor. This defines the CPU shares for the sensor container. A valid value is a non-zero, positive integer other than 1024.

Installsensor.sh Command	Specify CpuShares as a command line argument for installsensor.sh script.  sudo ./installsensor.sh ActivationId=<Activation id> CustomerId=<Customer id> Storage=/tmp/qualys/sensor/data CpuShares=1023 -s
Docker Run Command	Specify --cpu-shares as part of the Docker run command when deploying a sensor.  sudo docker run -d --restart on-failure --cpu-shares 1023 -v /var/run/docker.sock:/var/run/docker.sock:ro -v /usr/local/qualys/sensor/data:/usr/local/qualys/qpa/data -e ACTIVATIONID=<Activation id> -e CUSTOMERID=<Customer id> -e POD_URL=<POD URL> --name qualys-container-sensor qualys/qcs-sensor:latest

<b>CIS Docker Benchmark</b>	<b>5.12 Ensure that the container's root filesystem is mounted as read only (Automated)</b>
Qualys Control	CID 10825 "Status of the read-only filesystem for the Docker containers on the host system"
Resolution	To meet compliance with this control, Qualys Container Sensor should run with the command line argument --read-only for the installsensor.sh script or as part of Docker run command when deploying a sensor.
Installsensor.sh Command	Specify --read-only as a command line argument for installsensor.sh script.  sudo ./installsensor.sh ActivationId=<Activation id> CustomerId=<Customer id> Storage=/tmp/qualys/sensor/data --read-only -s
Docker Run Command	Specify --read-only as part of the Docker run command when deploying a sensor.  sudo docker run -d --restart on-failure --read-only -v /var/run/docker.sock:/var/run/docker.sock:ro -v /usr/local/qualys/sensor/data:/usr/local/qualys/qpa/data -e ACTIVATIONID=<Activation id> -e CUSTOMERID=<Customer id> -e POD_URL=<POD URL> --name qualys-container-sensor qualys/qcs-sensor:latest
Kubernetes DaemonSet	For deployments in Kubernetes with Docker Runtime, add readOnlyRootFilesystem: true in the securityContext section of the cssensor-ds.yml file.  securityContext: readOnlyRootFilesystem: true

<b>CIS Docker Benchmark</b>	<b>5.25 Ensure that the container is restricted from acquiring additional privileges (Automated)</b>
Qualys Control	CID 10855 “Status of the 'no-new-privileges' security option set for the Docker containers on the host system”
Resolution	To meet compliance with this control, Qualys Container Sensor should run with the command line argument --security-opt=no-new-privileges as part of the Docker run command when deploying a sensor.
Installsensor.sh Command	There is no new option for the installsensor.sh script. Support for installer script for this argument will be added in a future release. Alternatively, you can use docker run command as specified below to meet this control.
Docker Run Command	Specify --security-opt=no-new-privileges as part of the Docker run command when deploying a sensor.  <pre>sudo docker run -d --restart on-failure --security-opt=no-new-privileges -v /var/run/docker.sock:/var/run/docker.sock:ro -v /usr/local/qualys/sensor/data:/usr/local/qualys/qpa/data -e ACTIVATIONID=&lt;Activation id&gt; -e CUSTOMERID=&lt;Customer id&gt; -e POD_URL=&lt;POD URL&gt; --name qualys-container-sensor qualys/qcs-sensor:latest</pre>

<b>CIS Docker Benchmark</b>	<b>5.28 Ensure that the PIDs cgroup limit is used (Automated)</b>
Qualys Control	CID 10829 “Status of the Process ID (PID) cgroup limit for Docker containers”
Resolution	To meet compliance with this control, Qualys Container Sensor should run with the command line argument PidLimit for installsensor.sh script or --pids-limit as part of Docker run command when deploying a sensor. This defines the Pid limit for the sensor container. The value provided must be a positive integer.
Installsensor.sh Command	Specify PidLimit as a command line argument for installsensor.sh script.  <pre>sudo ./installsensor.sh ActivationId=&lt;Activation id&gt; CustomerId=&lt;Customer id&gt; Storage=/tmp/qualys/sensor/data PidLimit=100 -s</pre>
Docker Run Command	Specify --pids-limit as part of the Docker run command when deploying a sensor.  <pre>sudo docker run -d --restart on-failure --pids-limit 100 -v /var/run/docker.sock:/var/run/docker.sock:ro -v /usr/local/qualys/sensor/data:/usr/local/qualys/qpa/data -e ACTIVATIONID=&lt;Activation id&gt; -e CUSTOMERID=&lt;Customer id&gt; -e POD_URL=&lt;POD URL&gt; --name qualys-container-sensor qualys/qcs-sensor:latest</pre>

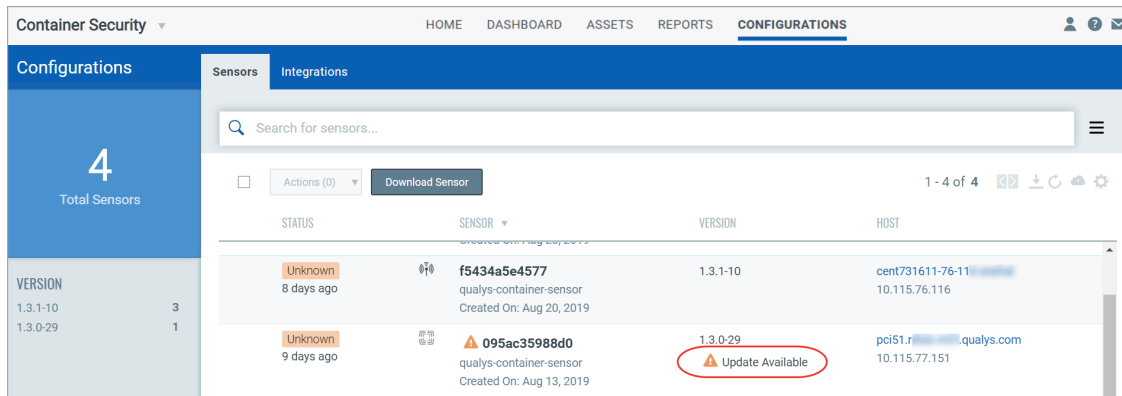
CIS Docker Benchmark	Multiple Controls
Installsensor.sh Command	<p>This sample includes all the command line arguments for installsensor.sh for meeting compliance, including MemoryUsageLimit, CpuShares, PidLimit, and --read-only. Note that --net=host is excluded.</p> <pre>sudo ./installsensor.sh ActivationId=&lt;Activation id&gt; CustomerId=&lt;Customer id&gt; Storage=/tmp/qualys/sensor/data MemoryUsageLimit=500m CpuShares=1023 PidLimit=100 -s --read-only</pre>
Docker Run Command	<p>This sample includes all the command line arguments for the Docker run command for meeting compliance, including --security-opt=no-new-privileges, --cpu-shares, --pids-limit, -m, and --read-only. Note that --net=host is excluded.</p> <pre>sudo docker run -d --restart on-failure --read-only --security- opt=no-new-privileges --cpu-shares 1023 --pids-limit 100 -m 500m -v /var/run/docker.sock:/var/run/docker.sock:ro -v /usr/local/qualys/sensor/data:/usr/local/qualys/qpa/data -e ACTIVATIONID=&lt;Activation id&gt; -e CUSTOMERID=&lt;Customer id&gt; -e POD_URL=&lt;POD URL&gt; --name qualys-container-sensor qualys/qcs-sensor:latest</pre>



# Administration

## Sensor updates

When a newer sensor version is available than the one deployed, you'll see "Update Available" next to the sensor name in the UI. You should update the sensor to the newer version to take advantage of new features, bug fixes and to remediate vulnerabilities.



### For sensors downloaded from the Qualys UI

Sensors deployed on docker with the `installsensor.sh` script or `docker run` command will be updated automatically (unless the `--disable-auto-update` option was used for the install script). Sensors are not updated automatically for Kubernetes deployments. Refer to [Update the sensor deployed in Kubernetes](#) for instructions.

### For sensors installed from Docker Hub

The Qualys Container Sensor image hosted on Docker Hub does not support auto update. See [Upgrading the sensor](#) in the section [Installing the sensor from Docker Hub](#) for instructions on how to upgrade to the latest version.

## How to uninstall the sensor

The `QualysContainerSensor.tar.xz` file (which you download for sensor installation from Qualys Cloud Platform) has the script **`uninstallsensor.sh`** for uninstalling the sensor.

### To uninstall a sensor:

If the docker host is configured to communicate over `docker.sock`, use the following command:

```
./uninstallsensor.sh -s
```

If the docker host is configured to communicate over TCP socket then provide the address on which docker daemon is configured to listen:

```
./uninstallsensor.sh DockerHost=<<IPv4 address or FQDN>>:<Port#> -s
```

For example,

```
./uninstallsensor.sh DockerHost=10.11.12.13:1234 -s
```

Follow the on-screen prompts to uninstall the sensor. Qualys recommends not to clear the persistent storage.

# Troubleshooting

## Check sensor logs

The sensor log file is located at the following location by default:

```
/usr/local/qualys/sensor/data/logs/qpa.log
```

If you are running a sensor without persistent storage by using the argument "--sensor-without-persistent-storage" it means the sensor will not write any data to the host. In this case, the Container Security UI may not show any sensor details and the logs will be available inside the sensor container at the following location:

```
/usr/local/qualys/qpa/data/
```

Note: If the sensor is installed without any persistent storage, the Container Summary page may not display any sensor details, and instead, it may show the error "There is no sensor activity recorded".

## Sensor health status

For standalone deployments of Container Security Sensor v1.10 and later, we'll show the container health status in 'docker ps'. For the first 24 hours from deployment, the health status will be "health: starting" as we monitor the health of the sensor. After 24 hours, the status will change from "health: starting" to "healthy". Please note that as long as the sensor STATUS in 'docker ps' is "Up" the sensor is running successfully.

Here's an example of sensor health status for first 24 hours of new sensor deployment:

```
root@ip-10-11-12-13:/home/ubuntu# docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
5e234d567c89 qualys/sensor:latest "/usr/bin/tini -- /u..." 2 days ago Up
10 minutes (health: starting) qualys-container-sensor
```

## Diagnostic script

Qualys provides a script to collect diagnostic information about the sensor. You must run the script on the host on which you want to collect the diagnostic information from.

The diagnostic script is present in the QualysContainerSensor.tar.xz that you downloaded for installing the sensor.

The script is called Sensor\_Diagnostic\_Script.py. You must have Python installed on the host in order to run the script. The script collects the following information from the host and puts it in a tar file called SensorDiagnostic.tar. You can send that file to Qualys Support for further assistance.

The SensorDiagnostic.tar includes 'ScanInfo.json', 'qpa.log' of qualys-container-sensor from given persistent storage, docker logs of qualys-container-sensor, and all information described below in the 'SensorDiagnostic.log' file. If 'ScanInfo.json' and Sensor logs are not available on the Docker host then this script creates empty 'ScanInfo.json' and qpa.log files, and appends "File not found" to them.

- Operating System Information (Type of OS i.e. Linux or Mac and other details)
- Proxy Configuration (Type of proxy set e.g. system, docker, cloud-agent proxy)
- CPU Architecture (Details about model, CPUs, cores, etc)
- RAM Usage (Memory allocation and utilization on host)
- Docker Version (Docker version installed on host)
- Socket Configuration (Docker socket configuration on host e.g. TCP/unix domain)
- Number of docker images (Count of all docker images and their details)
- Number of docker containers (Count of all docker containers and their details)
- CPU and Memory usage of running containers (First result of all resource usage statistics)

## Sensor crashes during upgrade

Use `installsensor.sh` to reinstall Qualys container sensor keeping the "Storage" value as it was for earlier Sensor. This will ensure that the new sensor will not be marked as another Sensor and will simply upgrade the existing one.

For help on install command, see [Deploying Container Sensor](#).

**Note:** At any given point in time, DO NOT delete the persistent storage. Else, the sensor deployed thereafter will be marked as a new sensor.

## What if sensor restarts?

The Sensor is designed to handle restart scenarios and will continue functioning normally after restart. No customer intervention is needed until the sensor crashes. Sensor will restart according to the sensor restart policy.

### Sensor restart policy

Exceptions will be handled gracefully and the sensor will restart as per its restart policy for recoverable and irrecoverable errors, as described below.

#### Recoverable errors

The sensor will return a recoverable error code 24 in cases like the sensor has crashed or the sensor caught an exception. In these cases, the sensor will recover on its own, and will keep on restarting. There is no max limit set on the number of restarts, but the time between two restarts will increase with the number of restarts needed, up to 16 minutes.

For example, the time between two restarts could be 1 minute to start, then 2 minutes, then 4 minutes, then 8 minutes, then 16 minutes. Once 16 minutes is reached, the time between restarts will remain at 16 minutes. No core dump file will be created.

### **Irrecoverable errors**

If the sensor returns an irrecoverable error code, it means the sensor will not recover on its own and the sensor will exit. For standalone deployments, the sensor will exit upon receiving the irrecoverable error code. For DaemonSet deployments, when the sensor exits with an irrecoverable error code, the Kubernetes Pod restart policy will restart the exited container. Irrecoverable error codes must be resolved by making changes to the deployment files and deployment arguments.

## Duplicate Kubernetes containers

While searching for containers you may see duplicates of containers orchestrated by Kubernetes. This is because Kubernetes spins up a monitoring container for every service container it brings up. Qualys container sensor sees them as two different containers and reports and scans both of the containers.

To see results without duplicate containers add the following string to queries used for searching Kubernetes containers.

```
not label.key:POD
```

For example, use this query to find running containers in Kubernetes:

```
state:"RUNNING" and not label.key:POD
```

## Get container runtime details

There are several commands you can run to get details about the container runtime in use and configuration setup that you may want to share with Qualys Support to troubleshoot an issue.

### Get nodes

Use the following command to get information about the cluster with details for each node like name, status, roles, age, version, internal and external IP addresses, OS image, kernel version, and container runtime.

```
kubectl get nodes -o wide
```

### Get container runtime info

Use the following command to get information about the container runtime like the status and configuration.

```
crictl info
```

### List containers

Use the following command to list all containers with details, including container ID, image ID, when the container was created (number of minutes, days, weeks or months ago), current state (e.g. Running, Exited), container name, attempt number and POD ID.

```
crictl ps -a
```

### List images

Use the following command to list images with details, including image name, tag, image ID and image size.

```
crictl images
```