

Carbon Black Container User Guide

15 February 2024

You can find the most up-to-date technical documentation on the VMware by Broadcom website at:

<https://docs.vmware.com/>

VMware by Broadcom
3401 Hillview Ave.
Palo Alto, CA 94304
www.vmware.com

Copyright © 2024 Broadcom. All Rights Reserved. The term “Broadcom” refers to Broadcom Inc. and/or its subsidiaries. For more information, go to <https://www.broadcom.com>. All trademarks, trade names, service marks, and logos referenced herein belong to their respective companies. [Copyright and trademark information](#).

Contents

VMware Carbon Black Container User Guide	8
1 Carbon Black Container Overview	9
Why Deploy Security for Containers	11
How Carbon Black Secures Containers	11
Provides Visibility into Kubernetes Security Posture	12
Secures the Complete Lifecycle of Kubernetes Applications	13
Automates Runtime Cluster Scanning	13
Enables Compliance and Policy Automation	15
Containers Architecture	15
Container Agent Components Architecture	17
cbcontainers-operator	19
cbcontainers-monitor	20
cbcontainers-node-agent	21
cbcontainers-hardening-enforcer	23
cbcontainers-hardening-state-reporter	24
cbcontainers-runtime-resolver	25
cbcontainers-image-scanning-reporter	26
Containers Concepts and Terminology	27
2 Setting up the Container Security Environment	33
Roles and Users for Containers	33
Using and Creating Roles for Containers	34
Add a Container Role	36
Create a User Account for Containers	39
Adding Clusters and Installing Kubernetes Sensors	39
Add a Cluster and Install the Kubernetes Sensor	40
Private Container Registry	44
Check the Kubernetes Sensor Status and Health	44
Installing a Containerized Sensor	46
Set up a Containerized Sensor	47
Install a Containerized Sensor	49
Install a Containerized Sensor on a Docker Client	52
Install a Containerized Sensor on an ECS Cluster	53
Validate the Container Image Signature	56
Setting up CLI Client for Image Scanning	58
Download a CLI Client	59
Add and Configure a CLI Client	60

- Carbon Black Container Operator Technical Reference 63
 - Manually Deploy the Container Operator 63
 - Uninstall the Container Operator 64
 - Manually Deploy the Container Agent 64
 - Openshift 65
 - Reading Metrics by using Prometheus 68
 - Custom Resources Definitions 69
 - Changing Components Resources 73
 - Configuring Container Services to use HTTP Proxy 75
 - Changing the Image Source 77
 - Operator Role-based Access Control 78
 - Container Operator Developer Instructions 79
 - Helm Charts 81

3 Configuring Container Security 86

- Kubernetes Scopes 86
 - Kubernetes Scopes Hierarchy 87
 - Built-in Kubernetes Scopes 89
 - Add a Kubernetes Applications Scope to Kubernetes Resources 90
 - Add a Kubernetes Deploy Location Scope to Kubernetes Resources 91
 - Add a Kubernetes Container Images Scope to Kubernetes Resources 92
 - View a Kubernetes Scope 94
 - Edit or Delete a Kubernetes Scope 95
 - Kubernetes Scope Baselines for Runtime Policies 96
 - View a Kubernetes Scope Baseline for a Runtime Policy 96
 - Add a Behavior to a Kubernetes Scope Baseline 97
 - Add a False Positive as Normal Behavior to the Scope Baseline 97
 - Reset a Kubernetes Scope Baseline 98
- Egress Groups 99
 - Create an Egress Group 99
 - Edit or Delete an Egress Group 100
- Kubernetes Policies 101
 - Kubernetes Runtime Policies 101
 - Create a Kubernetes Runtime Policy 101
 - Edit a Kubernetes Runtime Policy 103
 - Enable a Kubernetes Runtime Policy Draft 103
 - View Kubernetes Runtime Policy Details 103
 - Kubernetes Hardening Policies 105
 - Built-in Kubernetes Hardening Policies 105
 - Create a Kubernetes Hardening Policy 105
 - Enforcement Presets 107

- Edit a Kubernetes Hardening Policy 110
- Enable a Kubernetes Hardening Policy Draft 111
- Save a Hardening Policy as a Template 111
- Duplicate a Hardening Policy 112
- Kubernetes Policy Rules 112
- Kubernetes Policy Templates 126
- Subscribe to Alert Notifications 128
- Setting up API Access 129
 - Create and Manage an API Key 130
 - Delete API Key with Attached Notification Rule 132
 - Setting Access Levels 133
 - Create Access Levels 133
 - Apply Access Level to an API Key 133

4 Scanning Images 135

- Manually Rescan a Container Image 136

5 Monitoring and Analyzing Containers 138

- Severity Scoring 138
 - Kubernetes Risk Severity Scoring 138
 - Risk Evaluation for Container Images 139
 - Color Indicators for Image Vulnerabilities Scoring 140
- Monitoring Container Images 141
 - View Container Images - Overview 141
 - View Deployed Container Image Details 143
 - View Container Image Repositories 146
 - View Image Scan Report - Scan Log Details 147
 - View Container Image Scan Report 148
 - View a Container Image Scan Report - Overview 149
 - View a Container Image Scan Report - Layers 150
 - View a Container Image Scan Report - Packages 153
 - View a Container Image Scan Report - Suspicious Files 154
 - View a Container Image Scan Report - Vulnerabilities 156
 - View a Container Image Scan Report - Vulnerability Details 158
 - View a Container Image Scan Report - K8s Workloads 160
 - View a Container Image Scan Report - Scan Log 161
 - Investigate Container Image Vulnerabilities 161
 - Allow an Exception for a Vulnerability 163
- Managing and Viewing File Reputations in Container Images 164
 - Detect Malware in a Container Image 164
 - Override a File Reputation in a Container Image 166

- Manage File Reputations for Container Images 167
- Adding File Reputations in Container Images 168
 - Add a File to the Banned List 169
 - Add a Reputation to the Approved List 170
 - Expiration of Approved Certificates 171
- Detecting and Preventing Secrets 172
 - Detect Secrets in Containers on the Scan Log Page 175
 - Prevent Secrets in Containers 176
- Monitoring Kubernetes Workloads 177
 - View Kubernetes Workloads 178
 - View a Kubernetes Workload - Overview 181
 - View a Kubernetes Workload - Runtime Policy 181
 - View a Kubernetes Workload - Hardening Policy 182
 - View a Kubernetes Workload - Network Connections 183
 - View a Kubernetes Workload - Risks 184
 - View a Kubernetes Workload - Behavior Models 185
 - Kubernetes Virtual Workloads 186
- Analyzing Network Activity 186
 - Investigate Cluster Activity in the Network Map 187
 - Visualizing Namespace Data on the Network Map 189
 - Visualizing Workloads Data on the Network Map 192

6 Investigating and Remediating Container Security Issues 195

- Exploring Kubernetes Events (Hardening) 195
 - Explore Kubernetes Events - Overview 195
 - Explore Kubernetes Events - Details 197
- Investigating Container Events on the Investigate Page 199
 - Investigate Container Events 203
 - Investigate Kubernetes Clusters 204
 - Investigate Kubernetes Namespaces 206
 - Investigate Kubernetes Workloads 207
- Investigate Containers Events on the Process Analysis Page 208
- Triaging Kubernetes Alerts 211
 - Search for Kubernetes Alerts 211
 - View Kubernetes Alert Details 212
- Identify Available Fixes and Patches 213

7 Managing Clusters and Kubernetes Sensors 217

- View Clusters 217
- Edit a Cluster 218
- Delete a Cluster and its Sensor 219

- Upgrading or Downgrading the Kubernetes Sensor 220
 - Upgrade or Downgrade the Kubernetes Sensor through the Command Line 220
 - Upgrade or Downgrade the Kubernetes Sensor through the Console 221
 - Upgrade or Downgrade the Kubernetes Sensor Remotely through the Console 223
- Delete a CLI Client 226

8 Carbon Black Container Operator Technical Reference 227

- Manually Deploy the Container Operator 228
 - Uninstall the Container Operator 228
- Manually Deploy the Container Agent 229
- Openshift 230
- Reading Metrics by using Prometheus 232
- Custom Resources Definitions 233
- Changing Components Resources 237
- Configuring Container Services to use HTTP Proxy 239
- Changing the Image Source 241
- Operator Role-based Access Control 242
- Container Operator Developer Instructions 243
- Helm Charts 245

VMware Carbon Black Container User Guide

VMware Carbon Black Container™ is a comprehensive security solution for both on-premise and cloud-native workloads by offering visibility, hardening, vulnerability management, and runtime protection capabilities.

Carbon Black Container helps reducing risk by identifying vulnerabilities and misconfigurations to harden workloads.

This solution provides security teams with the visibility and ability to enforce compliance while integrating into existing DevOps processes. With VMware Carbon Black, organizations can reduce risk, maintain compliance, and simplify security for Kubernetes environments at scale.

Intended Audience

Carbon Black Container is for both Security and DevOps teams. As security shifts left, developers must take an increased ownership in security and implementing security measures through the code and the build stage of the modern application lifecycle.

Security teams must enforce compliance requirements and keep applications secure during the deployment and runtime stages.

This guide is written for Security Analysts, DevSecOps, and DevOps teams. It assumes you have knowledge of containers and Kubernetes clusters.

Carbon Black Container Overview

1

The Carbon Black Container solution can provide the visibility and control that DevOps and security teams need to make sure that their Kubernetes clusters and the applications deployed on them are secure. This topic provides a condensed view of Carbon Black Container benefits.

VMware Carbon Black Container Essentials and VMware Carbon Black Container Advanced

Carbon Black offers two Carbon Black Container packages that are described in the following table.

VMware Carbon Black Container Essentials	VMware Carbon Black Container Advanced
Security posture dashboard	Container Essentials +
Compliance policy automation	Threat detection
Prioritized risk assessment	Anomaly detection
Governance control and enforcement	Egress security
Image scanning and vulnerability management	SIEM integration
Shift-left security with CI/CD hardening	
Topology map	
Auto Enforce	

At a Glance

Carbon Black Container delivers policy-based reporting and enforcement of your organization's security posture across all workloads deployed in Kubernetes clusters.

With Carbon Black Container, you can:

- Secure the complete lifecycle of Kubernetes applications.
- Detect and fix vulnerabilities and misconfigurations before deployment.
- Meet compliance standards.

- Achieve simple, secure multi-cloud and hybrid cloud Kubernetes at scale.

Use Cases

- Kubernetes Security Posture Management (KSPM)
- Container image scanning
- Container image hardening
- Increased visibility into Kubernetes environments
- Ensured security compliance, governance, and enforcement
- Build behavior models for applications to identify and alert on anomalies
- Secure containers and Kubernetes applications
- Increase visibility into Kubernetes environments

Key Benefits for DevOps Teams

- Fast and easy deployment
- Seamless integration into the CI/CD pipeline and existing processes
- Address vulnerabilities and misconfigurations at build
- Enable speed of delivery without compromising security
- Gain visibility into application connectivity and configuration with in-cluster network visibility map
- Risk-prioritized vulnerability assessment of container images at runtime
- Understand misconfiguration of secret management in Kubernetes

Key Benefits for Security Teams

- Gain complete visibility into the Kubernetes security posture
- Enable prioritized vulnerability reporting
- Define and customize security policies
- Enable developers to address vulnerabilities and misconfigurations at build
- Enable speed of delivery without compromising security
- Connect image vulnerabilities to specific running workloads
- Secure egress connections to private and public destinations
- Identify malicious egress connections by using IP reputation
- Use machine learning and AI to build network behavior model for workloads

- Identify malicious network activity
- Consolidate events and alerts to a single dashboard
- Gain visibility into Kubernetes clusters, networking flow, and application architecture

Read the following topics next:

- [Why Deploy Security for Containers](#)
- [How Carbon Black Secures Containers](#)
- [Containers Architecture](#)
- [Containers Concepts and Terminology](#)

Why Deploy Security for Containers

You can integrate security into your DevOps processes to easily deploy quality apps faster with Carbon Black Container. When you secure apps early in development, you reduce vulnerabilities in production.

Container Security is a critical part of a comprehensive security assessment. It is the practice of protecting containerized applications from potential risk using a combination of security tools and policies. Container Security manages risks throughout the environment, including all aspects of the software supply chain or CI/CD pipeline, infrastructure, container runtime, and lifecycle management applications that run on containers.

A unified security strategy from development to production is critical for detecting vulnerabilities and misconfigurations early in development to minimize the attack surface that containers pose. By starting with the build phase, DevOps and Security teams can create workloads that are secure by design. These teams require visibility into workloads at the runtime layer to secure Kubernetes clusters and their applications.

Security must be integrated at each layer throughout the development lifecycle to effectively protect against attacks. To address threats in increasingly complex environments, security requires a multilayered approach that spans the full application lifecycle.

Organizations adopting Kubernetes must provide visibility for security teams and set guardrails for development teams through configuration and compliance policies to avoid vulnerabilities and misconfigurations. These policies ensure steady governance and minimal disruption to DevOps workflows and protect the complete deployment lifecycle without impacting business agility and speed to market.

How Carbon Black Secures Containers

Carbon Black Container delivers visibility into all workloads and provides the ability to enforce compliance, security, and governance from a single dashboard.

Carbon Black Container enables enterprise-grade container security at the speed of DevOps from development to production. This solution provides DevOps and Security teams with detailed visibility, context, and the ability to enforce compliance while integrating into the existing application build and deploying processes. With Carbon Black Container, organizations of all sizes can reduce risk, maintain compliance, and simplify security for Kubernetes environments at scale.

Provides Visibility into Kubernetes Security Posture

With Carbon Black Container, Security and DevOps teams gain full visibility into Kubernetes environments to proactively harden workloads and better identify and reduce the risks posed by vulnerabilities and misconfigurations. Organizations can use the image repository to take inventory of the risks associated with an image, and directly align that vulnerability with a running workload.

Security Posture Dashboard

A single pane of glass provides complete visibility into your security posture across Kubernetes clusters or applications, including:

- Visibility into Kubernetes clusters and workload inventory.
- A combined view of all vulnerabilities, misconfiguration, and rules violations.
- A consolidated risk score aggregated for all workload attributes to prioritize remediation.

Network Map

The network visibility map lets you view workload connections in a single map of the application architecture. The network visibility map provides detailed information and context to better understand the application architecture and network traffic behavior.

To get a clean view of an application, filters allow the connectivity of the map to remove unnecessary noise such as system namespaces. You can use similar filters to better understand what connection is encrypted or not encrypted to gain full visibility into your application traffic posture. The goal of the networking visibility map is to give teams a better understanding of the connectivity and configuration of applications that are installed in the Kubernetes cluster.



Secures the Complete Lifecycle of Kubernetes Applications

Carbon Black Container integrates into the developer lifecycle to analyze and control application risks before they are deployed into production.

This purpose-built solution automates DevSecOps, delivering continuous cloud native security and compliance for the full lifecycle of workloads running in Kubernetes.

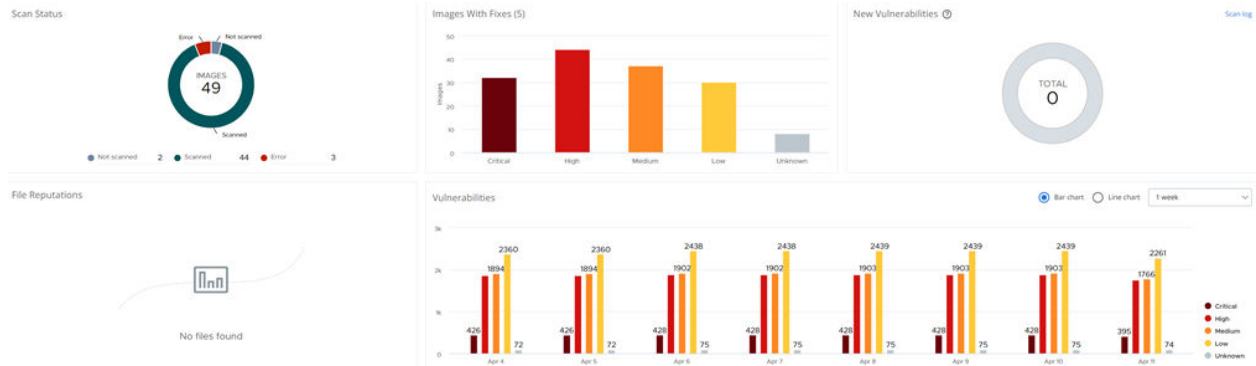
- Integrates with the CI/CD pipeline.
- Scans container images for vulnerabilities at build and runtime.
- Creates and enforces content-based security policies quickly and easily.
- Customizes and automates security policies and controls to harden the desired state and avoid configuration drift.
- Enables reporting and enforcement of the security posture across all workloads deployed in Kubernetes clusters.

Automates Runtime Cluster Scanning

CI/CD integration and a shift-left approach is an effective strategy; however, continuously monitoring the security posture in production is also required.

Cluster scanning provides the same level of visibility as scanning applications developed in CI/CD to third-party and infrastructure-level components. It is critical to ensure container images used in any running workload are up-to-date and can detect vulnerabilities.

Runtime cluster scanning ensures all running images are scanned for misconfigurations and vulnerabilities to better evaluate overall risk. For example, to confirm that the applied configuration and manifest still aligns with the policy, thereby identifying vulnerable misconfigurations and making sure that the cluster itself does not have any clear text secrets or malicious containers running. This capability enables DevOps and Security teams to understand the level of security in the run state, and to make any necessary changes to the pipeline to better secure workloads.



Container images present some security challenges. Images are usually built by layering other images, which could contain vulnerabilities, and those vulnerabilities can find their way into production systems. Defects and malware can also affect container images. When the provenance of a container is unknown, these risks increase.

Container image registries with the following functionality can reduce these risks:

- Scan images for vulnerabilities found in the Common Vulnerabilities and Exploits (CVE) database.
- Sign images as known and trusted by using a notary.
- Set up secure, encrypted channels for connecting to the registry.
- Authenticate users and control access by using existing enterprise accounts managed in a standard directory service, such as Active Directory.
- Tightly control access to the registry using the principles of least privilege and separation of duties.
- Enact policies that let users consume only those images that meet your organization's thresholds for vulnerabilities.

Vulnerability Scanning

Most applications use components that are sourced from third-party image registries. Having realized this, attackers often insert malicious code into these registries. Containers often use base images of operating systems like Ubuntu and CentOS from a public image repository such as DockerHub. The packages of an operating system and the applications on it can contain vulnerabilities.

Vulnerability scanning helps detect known vulnerabilities to reduce the risk of security breaches. Identifying an image vulnerability or malware on an image, and keeping those from going into production, reduces the attack surface of a containerized application.

Enables Compliance and Policy Automation

In this context, compliance refers to industry standards such as CIS Benchmarks and your own organizational requirements. Typically, the SecOps team defines the security policies for the organization, and the DevOps team creates the policies and ensures compliance.

Carbon Black Container solution’s compliance and policy automation capabilities:

- Shift into the development cycle to detect and prevent vulnerabilities at build.
- Create automated policies to enforce secure configuration.
- Ensure compliance with organizational requirements and industry standards such as CIS benchmarking.
- Leverage pre-built templates and customized policies.

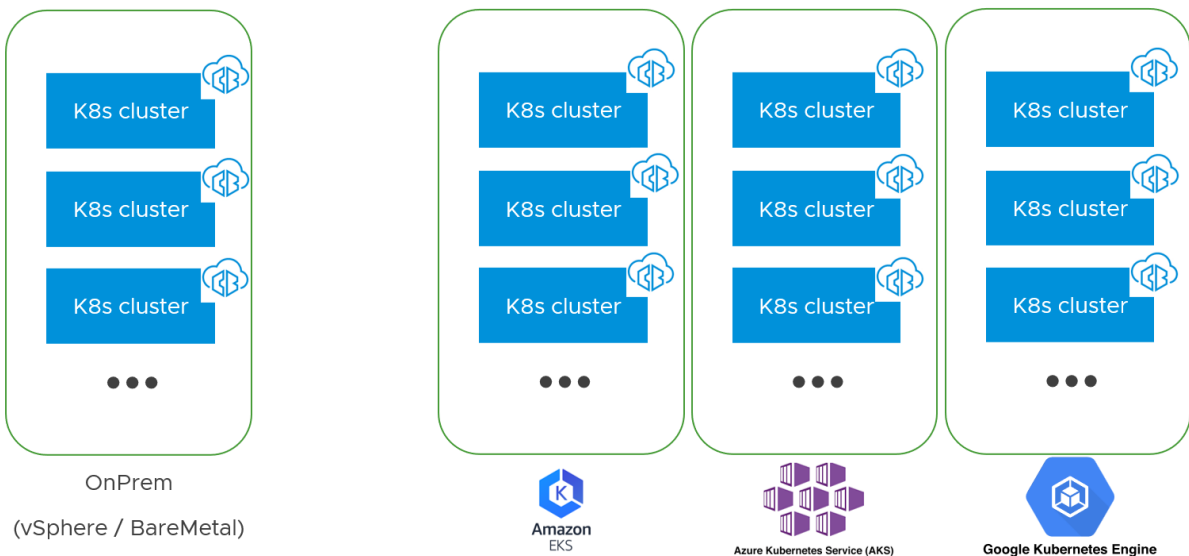
Containers Architecture

This topic discusses Carbon Black Container and Kubernetes architecture.

Carbon Black Cloud is a cloud-native SAAS solution. It can protect multiple Kubernetes clusters on-prem and in the public cloud (Amazon EKS, Azure Kubernetes Service, Google Kubernetes Engine).

Kubernetes multi cloud architecture

Onprem and in public cloud



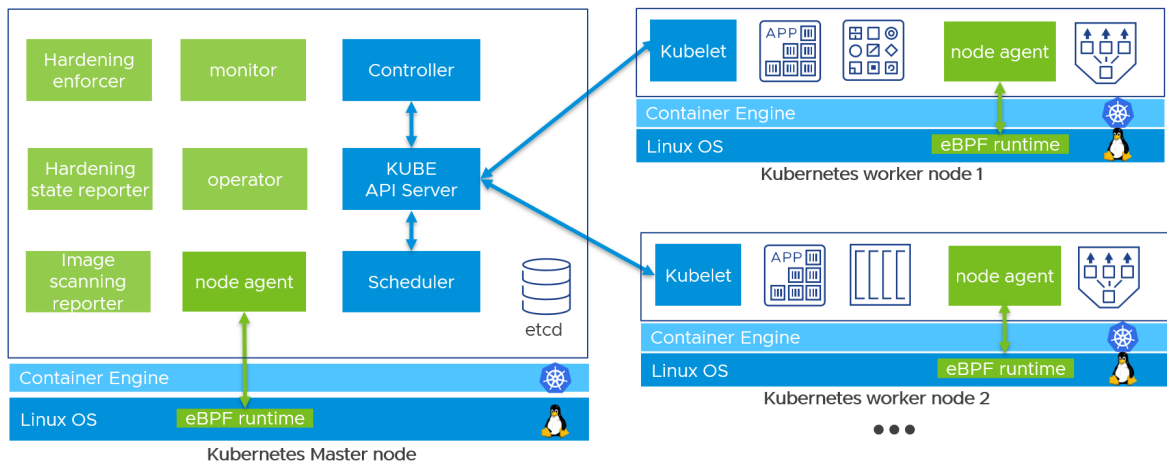
Kubernetes Cluster Components

On a Kubernetes cluster, Carbon Black Container consists of key components that interact with each other.

All Carbon Black Container pods run in a dedicated namespace called *cbcontainers-dataplane*. The pods must all connect to Carbon Black Cloud through a direct connection or a proxy.

VMware Carbon Black Container architecture

Kubernetes cluster



In the preceding diagram, all Carbon Black Cloud components are displayed in green and all Kubernetes components are displayed in blue.

Carbon Black Container uses [eBPF](#) technology (external link) to add the runtime security layer in Linux. eBPF extends the kernel capabilities safely and efficiently without requiring changing kernel source code or load kernel modules. Carbon Black uses eBPF in Carbon Black Container for all Linux kernels version 4.4+. With eBPF, Carbon Black Container can monitor all ingress, egress, and internal network connections. eBPF detects ports scanning, anomalous behaviors, and connections to malicious IPs and URLs.

The *node agent* pod is a Kubernetes DaemonSet. It makes sure that all nodes run a copy of this pod; therefore, you can add more nodes to your Kubernetes cluster, and Carbon Black automatically protects them. One node agent exists on each worker node. Daemonset are commonly used for monitoring, networking, and security solutions. This technology is available in all Kubernetes.

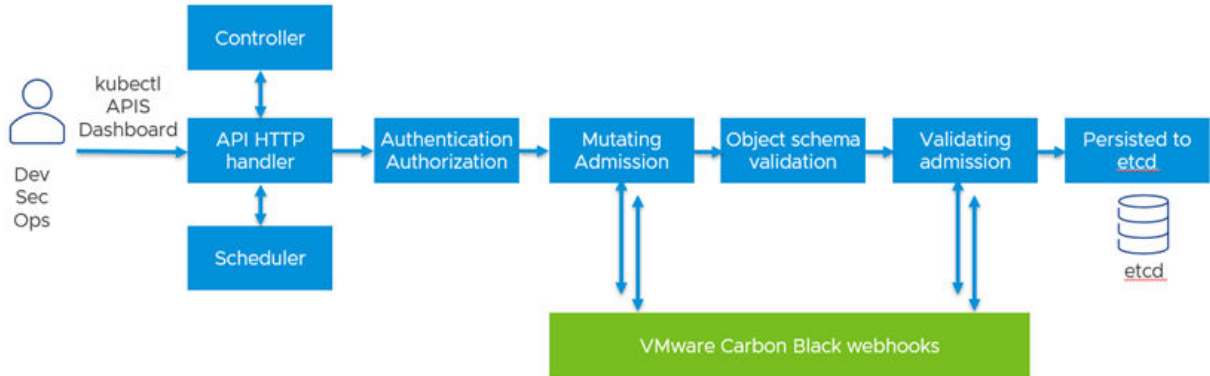
Kubernetes Admission Controller

VMware Carbon Black Container provides two kinds of policies:

- Runtime policy
- Hardening policy

Hardening policies include webhooks, which can extend the [admission control](#) (external link) of Kubernetes clusters. Carbon Black Container can automatically enforce (or *mutate*), block, or alert if an admin deploys a resource that is not compliant with Carbon Black Container hardening policies.

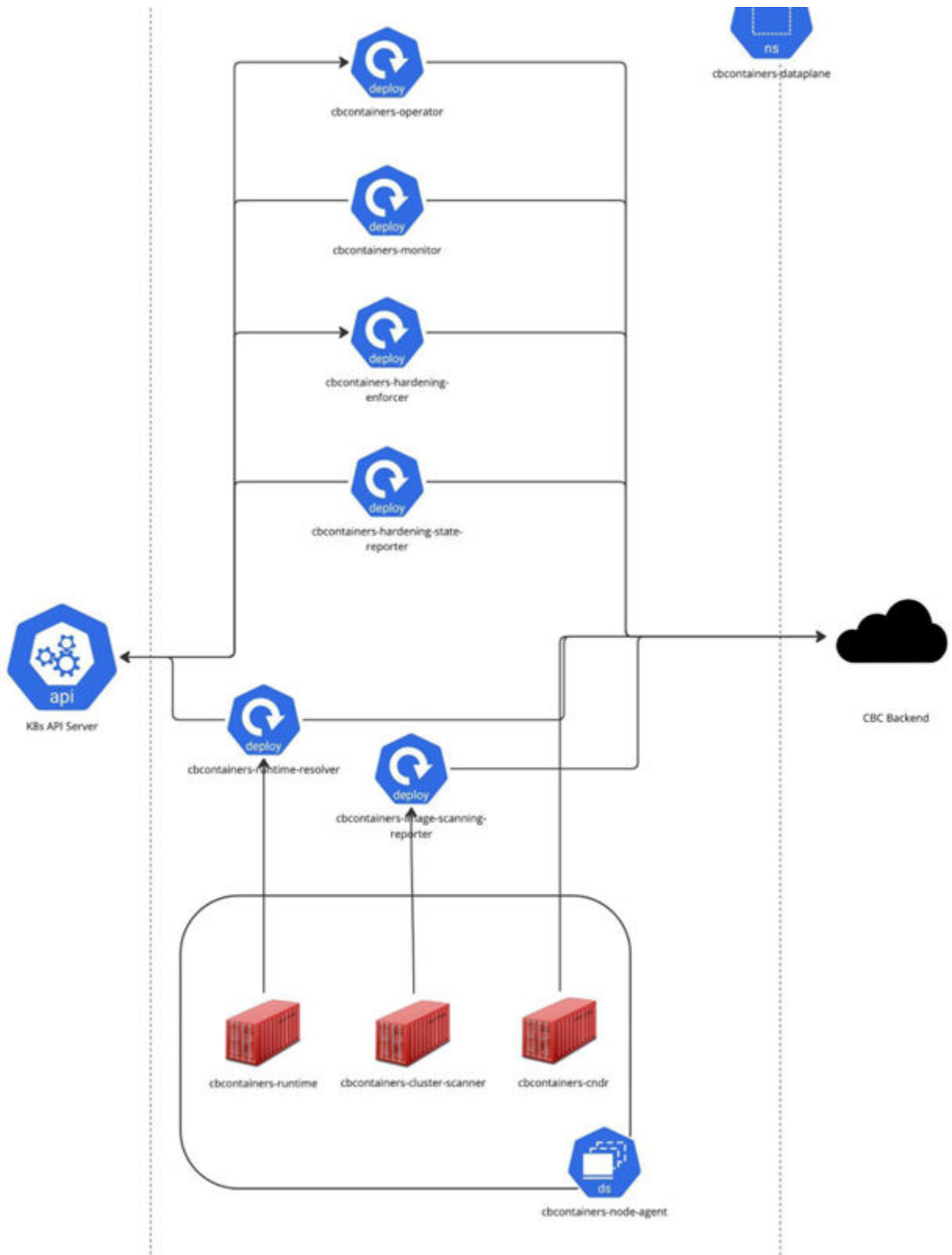
Kubernetes Admission controller



Container Agent Components Architecture

All Carbon Black agent components exist in a single namespace. By default, this namespace is called `cbcontainers-dataplane`.

Common components are shared between features.



The following topics describe each component.

cbcontainers-operator

The `cbcontainers-operator` is a set of controllers that deploy and manage the Carbon Black Container components. It is deployed as a Kubernetes Deployment and typically has only one pod.

manager

The manager is the main container within the `cbcontainers-operator` component. It acts as a Kubernetes Custom Resource Controller and it monitors instances of any object type from the `cbcontainersagents.operator.containers.carbonblack.io` API group. It provisions other Carbon Black Agent components through a CRD object. It requires a connection to the API server and it does not have any open ports.

Image	<code>cbartifactory/octarine-operator</code>
Opened ports	None
Connects to Kubernetes services	<code>kubernetes.default.svc</code> (Kubernetes API server)
Connects to backend	<code>defense-prod05.conferdeploy.net:443</code>
NO_PROXY requirements	The Kubernetes API server IP addresses (resolved from <code>kubernetes.default.svc</code> within the cluster)
Requested resources	CPU- 100m, Memory - 64Mi
Resource limits	CPU- 500m, Memory - 256Mi
Replica count (min & def)	Min- 1, Default - 1
Horizontal Scaling	Not required
Tolerances	<code>node.kubernetes.io/memory-pressure:NoSchedule op=Exists</code> <code>node.kubernetes.io/not-ready:NoExecute op=Exists for 300s</code> <code>node.kubernetes.io/unreachable:NoExecute op=Exists for 300s</code>
Is privileged	No

kube-rbac-proxy

The `kube-rbac-proxy` container acts as a sidecar to the operator's manager container. Its purpose is to protect the manager from malicious attacks. It protects the operator's metrics endpoint by requiring callers to have the metrics-reader ClusterRole assigned.

Image	<code>cbartifactory/kube-rbac-proxy</code>
Opened ports	8443/TCP
Connects to Kubernetes services	<code>kubernetes.default.svc</code> (Kubernetes API server)
Connects to backend	No

NO_PROXY requirements	The Kubernetes API server IP addresses (resolved from <code>kubernetes.default.svc</code> within the cluster)
Requested resources	CPU- 20m, Memory - 64Mi
Resource limits	CPU- 500m, Memory - 128Mi
Replica count (min & def)	Min- 1, Default - 1
Horizontal Scaling	Not required
Tolerances	<code>node.kubernetes.io/memory-pressure:NoSchedule op=Exists</code> <code>node.kubernetes.io/not-ready:NoExecute op=Exists for 300s</code> <code>node.kubernetes.io/unreachable:NoExecute op=Exists for 300s</code>
Is privileged	No

cbcontainers-monitor

The `cbcontainers-monitor` is responsible for reporting the health and metadata of the agent to the Carbon Black Cloud backend. This responsibility includes the state of agent components (running, waiting, failed) and determines whether the agent is in a healthy state.

The `cbcontainers-monitor` also acts as a health check for the cluster in Carbon Black Cloud. Agents that stop reporting this data for 24 hours are considered unhealthy.

Image	<code>cbartifactory/monitor</code>
Opened ports	None
Connects to Kubernetes services	<code>kubernetes.default.svc</code> (Kubernetes API server)
Connects to backend	<code>events.containers.carbonblack.io:443</code> (gRPC) <code>defense-prod05.confdeploy.net:443</code>
NO_PROXY requirements	The Kubernetes API server IP addresses (resolved from <code>kubernetes.default.svc</code> within the cluster)
Requested resources	CPU- 30m, Memory - 64Mi
Resource limits	CPU- 200m, Memory - 256Mi
Replica count (min & def)	Min- 1, Default - 1
Horizontal Scaling	Manual
Tolerances	<code>node.kubernetes.io/not-ready:NoExecute op=Exists for 300s</code> <code>node.kubernetes.io/unreachable:NoExecute op=Exists for 300s</code>
Is privileged	No

cbcontainers-node-agent

Need a description here.

cbcontainers-runtime

The `cbcontainers-runtime` container is part of every pod within the `cbcontainers-node-agent` DaemonSet. It is a privileged container that uses eBPF to attach to the Linux kernel on each Kubernetes node and generate a stream of events of observed network connections. These events are batched together and sent by gRPC to the `cbcontainers-runtime-resolver` deployment. The `cbcontainers-runtime` container does not connect directly to the Carbon Black Cloud backend.

Image	<code>cbartifactory/runtime-kubernetes-sensor</code>
Opened ports	None
Connects to Kubernetes services	<code>cbcontainers-runtime-resolver.cbcontainers-dataplane.svc.cluster.local:8080</code>
Connects to backend	No
NO_PROXY requirements	<code>cbcontainers-runtime-resolver.cbcontainers-dataplane.svc.cluster.local</code> and the Kubernetes API server IP addresses (resolved from <code>kubernetes.default.svc</code> within the cluster)
Requested resources	CPU- 30m, Memory - 64Mi
Resource limits	CPU- 2, Memory - 4Gi
Replica count (min & def)	Min- 1, Default = Kubernetes node count
Horizontal Scaling	Because it is a part of DaemonSet, new Kubernetes nodes automatically get a replica. There is no need for manual scaling.
Tolerances	<pre>node.kubernetes.io/disk-pressure:NoSchedule op=Exists node.kubernetes.io/memory-pressure:NoSchedule op=Exists node.kubernetes.io/network-unavailable:NoSchedule op=Exists node.kubernetes.io/not-ready:NoExecute op=Exists node.kubernetes.io/pid-pressure:NoSchedule op=Exists node.kubernetes.io/unreachable:NoExecute op=Exists node.kubernetes.io/unschedulable:NoSchedule op=Exists</pre>
Is privileged	Yes

cbcontainers-cluster-scanner

The `cbcontainers-cluster-scanner` container is part of every pod within the `cbcontainers-node-agent` DaemonSet. Different container runtime endpoints (Containerd, dockershim, CRI-O) are mounted inside the pod to communicate with the container runtime of the node. The cluster scanner calls the container runtime using gRPC to list containers and images, read their contents, and perform scans on the images that detect vulnerabilities, malware, and secrets.

For clusters utilizing CRI-O, additional paths from the host are mounted and utilized. These are paths where CRI-O stores image data and are required to fully scan images because some operations are not natively supported by the CRI-O API.

Most communication from `cbcontainers-cluster-scanner` goes through the `cbcontainers-image-scanning-reporter` before reaching the Carbon Black Cloud backend — except for generating certificates for mTLS connections, which is done by directly calling the Carbon Black Cloud backend.

Image	cbartifactory/cluster-scanner
Opened ports	None
Connects to Kubernetes services	cbcontainers-image-scanning-reporter.cbcontainers-dataplane.svc.cluster.local:443 kubernetes.default.svc (Kubernetes API server)
Connects to backend	defense-prod05.conferdeploy.net:443
NO_PROXY requirements	cbcontainers-runtime-resolver.cbcontainers-dataplane.svc.cluster.local and the Kubernetes API server IP addresses (resolved from kubernetes.default.svc within the cluster)
Requested resources	CPU- 30m, Memory - 64Mi
Resource limits	CPU- 2, Memory - 4Gi
Replica count (min & def)	Min- 1, Default = Kubernetes node count
Horizontal Scaling	Because it is a part of DaemonSet, new Kubernetes nodes automatically get a replica. There is no need for manual scaling.
Tolerances	node.kubernetes.io/disk-pressure:NoSchedule op=Exists node.kubernetes.io/memory-pressure:NoSchedule op=Exists node.kubernetes.io/not-ready:NoExecute op=Exists node.kubernetes.io/pid-pressure:NoSchedule op=Exists node.kubernetes.io/unreachable:NoExecute op=Exists node.kubernetes.io/unschedulable:NoSchedule op=Exists
Is privileged	Yes

cbcontainers-cndr

TheCNDR container contains the Carbon Black Cloud Linux Sensor. It uses eBPF probes for monitoring container process actions, file access events and network events.

Events are processed, attributed to workloads, passed through a rules engine to generate alerts if needed, and sent to the Carbon Black Cloud backend for presentation and analysis.

Image	cbartifactory/cndr
Opened ports	None
Connects to Kubernetes services	kubernetes.default.svc (Kubernetes API server)
Connects to backend	runtime.events.containers.carbonblack.io:443 (gRPC) defense-prod05.confdeploy.net:443
NO_PROXY requirements	Kubernetes API server IP addresses (resolved from kubernetes.default.svc within the cluster)
Requested resources	CPU- 30m, Memory - 64Mi
Resource limits	CPU- 500m, Memory - 1Gi
Replica count (min & def)	Min- 1, Default = Kubernetes node count
Horizontal Scaling	Because it is a part of DaemonSet, new Kubernetes nodes automatically get a replica. There is no need for manual scaling.
Tolerances	node.kubernetes.io/disk-pressure:NoSchedule op=Exists node.kubernetes.io/memory-pressure:NoSchedule op=Exists node.kubernetes.io/network-unavailable:NoSchedule op=Exists node.kubernetes.io/not-ready:NoExecute op=Exists node.kubernetes.io/pid-pressure:NoSchedule op=Exists node.kubernetes.io/unreachable:NoExecute op=Exists node.kubernetes.io/unschedulable:NoSchedule op=Exists
Is privileged	Yes

cbcontainers-hardening-enforcer

The `cbcontainers-hardening-enforcer` component is responsible for enforcing container security hardening policies.

The `cbcontainers-hardening-enforcer` component:

- Evaluates policy block rules through a validating webhook and blocks creating and updating Kubernetes objects accordingly.
- Evaluates policy enforce rules through a mutating webhook and modifies created and updated Kubernetes objects accordingly.

Image	cbartifactory/guardrails-enforcer
Opened ports	443/TCP (Kubernetes Service), 8080/TCP (Kubernetes Pods) - mutating and validating webhooks entry point Note You might need to open port 8080 from the master nodes to kubelet nodes in the FW.
Connects to Kubernetes services	kubernetes.default.svc (Kubernetes API server)
Connects to backend	events.containers.carbonblack.io:443 (gRPC) defense-prod05.confederdeploy.net:443
NO_PROXY requirements	The Kubernetes API server IP addresses (resolved from kubernetes.default.svc within the cluster)
Requested resources	CPU- 30m, Memory - 64Mi
Resource limits	CPU- 200m, Memory - 256Mi
Replica count (min & def)	Min- 1, Default - 1
Horizontal Scaling	Scaling is done by the operator. You can manually set the number of replicas in the CRD. <spec.components.basic.enforcer.replicasCount>
Tolerances	node.kubernetes.io/not-ready:NoExecute op=Exists for 300s node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
Is privileged	No

cbcontainers-hardening-state-reporter

The `cbcontainers-hardening-state-reporter` component is responsible for reporting Kubernetes objects to the backend.

The `cbcontainers-hardening-state-reporter` component monitors Kubernetes objects (such as pods, deployments, services, and so forth) by using a Kubernetes API watch method. It reports changes of objects to the backend.

Image	cbartifactory/guardrails-state-reporter
Opened ports	None
Connects to Kubernetes services	The Kubernetes API server IP addresses (resolved from kubernetes.default.svc within the cluster)
Connects to backend	events.containers.carbonblack.io:443 (gRPC) defense-prod05.confederdeploy.net:443
NO_PROXY requirements	The Kubernetes API server IP addresses (resolved from kubernetes.default.svc within the cluster)
Requested resources	CPU- 30m, Memory - 64Mi

Resource limits	CPU- 200m, Memory - 256Mi
Replica count (min & def)	Min- 1, Default - 1
Horizontal Scaling	None
Tolerances	node.kubernetes.io/not-ready:NoExecute op=Exists for 300s node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
Is privileged	No

cbcontainers-runtime-resolver

Runtime protection enables the use of policy rules to help secure deployed workloads. The `cbcontainers-runtime-resolver` component is responsible for the enrichment of network events together with their Kubernetes context and sending the events to the Carbon Black Cloud backend.

The `cbcontainers-runtime-resolver` component receives network events from the `cbcontainers-runtime` container within the `cbcontainers-node-agent` DaemonSet pods using inbound gRPC connections. The events have their Kubernetes context attached and are then batched together and sent via gRPC to the Carbon Black Cloud backend.

The Kubernetes information is taken from the API server by using standard Kubernetes in-cluster authentication and communication with the API server. `List` and `watch` operations are used with the API server; however, the information is cached locally in the `cbcontainers-runtime-resolver` to avoid unnecessary network traffic and improve response times.

Image	cbartifactory/runtime-kubernetes-resolver
Opened ports	8080/TCP
Connects to Kubernetes services	kubernetes.default.svc (Kubernetes API server)
Connects to backend	runtime.events.containers.carbonblack.io:443 (gRPC) defense-prod05.conferdeploy.net:443
NO_PROXY requirements	The Kubernetes API server IP addresses (resolved from <code>kubernetes.default.svc</code> within the cluster)
Requested resources	CPU- 200m, Memory - 64Mi
Resource limits	CPU- 900m, Memory - 1Gi
Replica count (min & def)	Min- 1, Default - 1

Horizontal Scaling	<p>By default, <code>cbcontainers-runtime-resolver</code> is scaled automatically by the operator. It uses the following formula:</p> <pre><node_count>/ <spec.components.runtimeProtection.resolver.nodesToReplicasRatio></pre> <p>Where <code><node_count></code> is the current number of nodes in the cluster and <code><spec.components.runtimeProtection.resolver.nodesToReplicasRatio></code> is taken from the CRD (by default this value is 5, but it can be lowered to accommodate network traffic intensive clusters).</p>
Tolerances	<pre>node.kubernetes.io/memory-pressure:NoSchedule op=Exists node.kubernetes.io/not-ready:NoExecute op=Exists for 300s node.kubernetes.io/unreachable:NoExecute op=Exists for 300s</pre>
Is privileged	No

Note See also [cbcontainers-runtime](#).

cbcontainers-image-scanning-reporter

Cluster image scanning enables an initial scan and automatic rescanning of cluster images. The `cbcontainers-image-scanning-reporter` component is responsible for aggregating and sending all scanned image results to the Carbon Black Cloud backend.

The `cbcontainers-image-scanning-reporter` component acts as a proxy for some calls to the Carbon Black Cloud backend. Because it maintains a local cache, this action avoids a large number of calls in the case that the cluster has many nodes (respectively, cluster-scanners).

Image	<code>bartifactory/image-scanning-reporter</code>
Opened ports	443/TCP
Connects to Kubernetes services	None
Connects to backend	<code>defense-prod05.confederdeploy.net:443</code>
NO_PROXY requirements	N/A
Requested resources	CPU- 200m, Memory - 64Mi
Resource limits	CPU- 900m, Memory - 1Gi
Replica count (min & def)	Min- 1, Default - 1
Horizontal Scaling	None

Tolerances	node.kubernetes.io/not-ready:NoExecute op=Exists for 300s node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
Is privileged	No

Note See also [cbcontainers-cluster-scanner](#).

Containers Concepts and Terminology

This topic introduces concepts and defines common terms that are used in Carbon Black Container.

General Terminology

Term	Definition
Admission Controller	A piece of code that intercepts requests to the Kubernetes API server. Admission controllers limit requests to create, delete, or modify objects.
cbctl	Command-line tool that lets you control your Carbon Black Cloud Container and Kubernetes workload security. Carbon Black Cloud CLI Client scans container images and reports their health to the Carbon Black Cloud console.
Common Vulnerabilities and Exposures (CVE)	A reference method for publicly known information-security vulnerabilities and exposures.
Container	Lightweight, portable executable image. Containers let you virtualize multiple application runtime environments on the same operating system (kernel) instance.
Container Orchestration	Exposes the API and interfaces. Helps manage the container lifecycle.
Control Plane	Manages worker nodes and pods.
Controller	Watches the state of your cluster and makes or requests changes where needed. Each controller tries to move the current cluster state closer to the desired state.
Cluster	A set of nodes. Each cluster contains at least one node.
DaemonSet	Node agent pod in the pod that ensures that all nodes run a copy of this pod. Using this node enables you to add more nodes to the pod and have them automatically protected by Carbon Black. Daemonset are commonly used for monitoring, networking, and security solutions. This technology is available in all Kubernetes.
DevOps	Integration of traditional development and IT operations teams.

Term	Definition
Docker	Technology that provides operating system level virtualization (containers). The Docker environment includes a container runtime as well as container build and image management. builds an OCI-standard container image: therefore, Docker images run on any OCI-compliant container runtime.
eBPF	Technology that extends the capabilities of the kernel safely and efficiently without changing kernel source code or loading kernel modules.
Egress	The traffic going from the cluster to another network (public or private).
Ingress	Exposes HTTP and HTTPS routes from outside the cluster to services within the cluster.
Kubelet	Agent that runs on nodes.
Kubernetes	An open-source container orchestrator. Automates deployment, load balancing, resource allocation, and security enforcement for containers. Keeps containerized applications running in their desired state to ensure that they are scalable and resilient.
Manifest digest	A hash of a container image that is encrypted with SHA-256 and is deterministic based on the image build.
Microservice	An application that is divided into a suite of independent, loosely integrated services.
Namespace	A mechanism for isolating groups of resources in a single cluster.
Node	Worker machine that runs containerized applications.
Node Agent	Makes sure that all nodes run a copy of the pod. The node agent allows you to add more nodes to your Kubernetes cluster and have them be automatically protected by Carbon Black Container.
Pod	A set of running containers.
Registry	Docker Hub and other third party repository hosting services are called registries. A registry stores a collection of repositories.
Repository	Stores one or more versions of a specific image.
Scope	A way to group Kubernetes resources for targeted security protection and analysis. For example, you can group resources by cluster and namespace and then create policies for that scope.
Templated Policies	Carbon Black Cloud Container deploys with 3 templated policies: <i>Basic</i> , <i>Restrictive</i> , and <i>CIS Benchmark</i> .

Term	Definition
Vulnerability Scanning	Vulnerability scanning helps detect known vulnerabilities to reduce the risk of security breaches. Reduces the attack surface of a containerized application.
Workload	An application running in a container.

Tip For a full glossary of Kubernetes terms, see <https://kubernetes.io/docs/reference/glossary/?fundamental=true>.

Runtime Policies Concepts and Terminology

Runtime policies include rules for egress network control, threat protection, and anomaly detection in your Kubernetes environment. They provide the benchmark to control Kubernetes workloads behavioral changes. Control of the Kubernetes runtime environment happens at two levels:

- **Scope:** you can monitor all Kubernetes resources in a defined scope.
- **Workload,** you can track the behavior of a specific workload.

Actions

All rules have an associated action: *Monitor* or *Alert*. Either action causes an alert in the Carbon Black Cloud console.

- **Monitor:** Monitor actions create an event record for informational purposes.
- **Alert:** Alert actions create an event record signifying a change in behavior. Alert is the default action for each rule unless it is changed.

Built-in Rules

Runtime policies include built-in rules from the following categories:

- **Egress Traffic (Scope)** — A list of allowed domains or IP addresses
- **Malicious Egress Traffic (Scope)** — A list of malicious IP addresses and domains that have bad reputations
- **Workload Anomaly Detection** — A change in workload behavior
- **Workload Threat Detection** — A port scan

Learning Period

The learning period is the time during which all the Kubernetes resources in a scope are monitored for egress network connections. All egress destinations are recorded in the scope baseline. After the learning period is complete, the system actively tracks workloads behavior. Subsequent violations of the Kubernetes runtime policies trigger alerts.

If the learning period of a policy is modified, the policy stops alerting and the learning period is reset. If you add a new rule, the learning period starts running only for the new rule.

You can see and analyze the alerts in the [Triaging Kubernetes Alerts](#) page in the Carbon Black Cloud console.

Protection Level to Use for Selecting Rules

The runtime policy rules are split among the following protection levels:

Basic

Covers the issues that have the highest priority.

Moderate

Extends the rules included in the **Basic** protection level.

Strict

Extends the rules included in the **Moderate** protection level. Provides the broadest coverage of issues.

Runtime Policy Scope

Kubernetes scope is a grouping of Kubernetes resources, such as clusters or workloads. With the Kubernetes runtime policies, scopes explicitly define deploy phase or target complete applications.

Scope Baseline

The scope baseline determines the normal allowed behavior for all Kubernetes resources inside a scope. You can establish a scope baseline by monitoring the egress traffic of all workloads in the scope for a certain time, called a *learning period*. Deviation from the baseline triggers an alert. The baseline is at scope level and you can amend or reset the final behavior list in the scope.

Hardening Policies Terminology and Concepts

Actions

All rules have an action associated with them: **Alert**, **Block**, or **Enforce**. The rules configuration sets an expected value. If the value is not met, a rule violation is triggered.

An **Alert** action violation displays as a notification.

A **Block** action blocks the Kubernetes resources. This violation displays as an alert and block notification.

An **Enforce** action enforces the value for a rule. Enforce overwrites the value of one or more fields to the value that is defined in the rule's preset. In other words, Enforce changes the setting instead of blocking it. For example, you might set CPU and Memory for all workloads.

Note When you enforce values, the running workload is different from the deployed workload. This difference can impact workload behavior and cause confusion if troubleshooting is required.


Built-in Rules

Built-in rules are available for direct use in Kubernetes hardening policies and are based on the Kubernetes security configuration.

Built-in Policies and Scopes

Policies and scopes that are available are with the Carbon Black Cloud console to facilitate the initial setup of Kubernetes policies. You can update and delete these policies and scopes. For more information, see [Built-in Kubernetes Hardening Policies](#) and [Built-in Kubernetes Scopes](#).

Built-in Rules for Container Images

Rules that display the container-shaped icon  apply to scopes in the build phase by using the CLI Client. The rules also apply to Kubernetes workloads based on container images in the deploy phase. These rules enforce container image properties and behavior. Rules that do not display this icon are not applicable for the build phase. See [Built-in Kubernetes Policy Rules](#).

Custom Rules

Custom rules use JSONPaths to specify Kubernetes resources and properties.

Custom Templates

A combination of built-in and custom rules.

Exceptions

Exclusion of workloads from the coverage of a Kubernetes policy due to known and accepted behavior.

- For most rules, the exceptions are based on a workload name.
- For Role-Based Access Control (RBAC) rules, the exceptions are based on resource name and username.
- For rules that allow the **Enforce** action, the exceptions are based on workload name or workload label.

Hardening Policies

Policies that check rules on your Kubernetes environment configuration.

Kubernetes Scope

Grouping of Kubernetes resources with a definitive purpose; for example, to apply a policy.

Predefined Templates

Predefined rule sets of built-in rules.

Violations

Notifications on changes that happen in your Kubernetes environment after enabling Kubernetes hardening policies. Violations trigger actions at the block or alert rule level. Potential violations can be identified before enabling a policy, thus allowing planning security strategies such as adding exceptions, enforcing actions, or disabling and enabling rules.

Setting up the Container Security Environment

2

This section describes how to prepare your environment for securing Kubernetes with Carbon Black Container.

Follow these basic steps to set up your container environment: for Carbon Black Container security:

- 1 Make sure your Kubernetes environment meets the supported Operating Environment Requirements for the Kubernetes Sensor. See [Kubernetes Sensor OER](#).
- 2 Add users and assign user roles so that the appropriate people can install, configure, and manage Carbon Black Container security features.
- 3 Optionally review and manually install the Operator and Agent. (This deployment happens automatically during Step 4.)
- 4 Add Kubernetes clusters to the Carbon Black Cloud console and install a Kubernetes sensor into each Kubernetes cluster that you want to protect.
- 5 Optionally install Containerized Sensors for non-Kubernetes environments.
- 6 Download, add, and configure a CLI client to scan local images.

You will then be ready to create scopes and policies to manage your containers.

Read the following topics next:

- [Roles and Users for Containers](#)
- [Adding Clusters and Installing Kubernetes Sensors](#)
- [Check the Kubernetes Sensor Status and Health](#)
- [Installing a Containerized Sensor](#)
- [Setting up CLI Client for Image Scanning](#)
- [Carbon Black Container Operator Technical Reference](#)

Roles and Users for Containers

You can add users and assign appropriate roles for their work in Containers.

By setting up and managing users and their roles, you give the users access to the Carbon Black Cloud console and Containers security functionality.

Note This section specifically describes setting up users and user roles for Containers. For information about managing all Carbon Black Cloud users and their roles, see [Managing Users](#) and [User Roles](#).

Using and Creating Roles for Containers

Every Carbon Black Cloud console user is assigned to a role that defines permissions. The role is assigned when you create the new user account; this assignment can be modified at any time.

Carbon Black Cloud includes four Kubernetes-related pre-defined roles that you can assign to users (or you can create custom roles: see [Add a Container Role](#)).

- Kubernetes SecOps View Only
- Kubernetes SecOps
- Kubernetes DevOps
- Kubernetes Security Developer

Kubernetes Security DevOps are responsible for the Kubernetes workload posture.

Responsibilities include setting up clusters, scopes, and security policies for Kubernetes workloads. Security DevOps can monitor the health of the Kubernetes environment, investigate workloads and violations, and take appropriate actions.

Role Definitions and Recommendations

The following table describes Carbon Black Cloud permissions and recommendations for user roles for Containers.

Table 2-1. User Roles/Permissions Matrix - by Role

Role	Description	Permissions	Workflow
Kubernetes SecOps View Only	Monitors environment. Cannot take any actions.	<ul style="list-style-type: none"> ■ View Notifications ■ View Kubernetes Security ■ View Images ■ View Workloads 	N/A
Kubernetes SecOps	Assess and control the workload's attack surface from build to runtime. Focus on detecting, responding to, and preventing container runtime threads —can quickly detect runtime threads. This role is appropriate for SOC Analysts.	<ul style="list-style-type: none"> ■ Dismiss Alerts ■ View and Manage Alerts, Notes, and Tags ■ View and Manage Notifications ■ View and Manage API Keys ■ Manage Users ■ View and Manage Kubernetes Security ■ View Images ■ Manage Image Exceptions 	<ol style="list-style-type: none"> 1 Monitor and analyze Containers. See Monitoring and Analyzing Containers. 2 Take action and remediate security issues. See Investigating and Remediating Container Security Issues. 3 Triage alerts. See Triaging Kubernetes Alerts.

Table 2-1. User Roles/Permissions Matrix - by Role (continued)

Role	Description	Permissions	Workflow
Kubernetes DevOps	<p>Assess and control the workload's attack surface from build to runtime. Troubleshooting and remediation of security issues.</p> <p>Responsible for determining the Kubernetes workload posture. Responsibilities include setting up Kubernetes policies, scopes, and clusters in the Carbon Black Cloud console. Security DevOps can monitor the health of the Kubernetes environment, investigate workloads and violations, and take appropriate actions.</p>	<ul style="list-style-type: none"> ■ Dismiss Alerts ■ View and Manage Notifications ■ View and Manage API Keys ■ Manage Users ■ View and Manage Kubernetes Security ■ View Images ■ Manage Image Exceptions 	<ol style="list-style-type: none"> 1 Set up user roles and manage users. See Roles and Users for Containers. 2 Add clusters to the console and install Kubernetes Sensors. See Adding Clusters and Installing Kubernetes Sensors. 3 Configure Containers. See Configuring Container Security. 4 Monitor and analyze Containers. See Monitoring and Analyzing Containers. 5 Triage alerts. See Triaging Kubernetes Alerts. 6 Take action and remediate security issues. See Investigating and Remediating Container Security Issues.
Kubernetes Security Developer	<p>Inspects a single container for security posture and compliance.</p>	<ul style="list-style-type: none"> ■ View and Manage Kubernetes Security ■ View Images ■ Manage Image Exceptions 	<ol style="list-style-type: none"> 1 Monitor and analyze Kubernetes workloads. See Monitoring Kubernetes Workloads. 2 Triage alerts. See Triaging Kubernetes Alerts.

Add a Container Role

To add a new role for Containers work, perform the following procedure.

Procedure

- 1 On the left navigation pane, click **Settings > Roles**.
- 2 In the upper right of the page, click **Add Role**.
- 3 Enter a unique name and description for the new role. Special characters are not allowed.

- 4 Optionally, select a role from the **Copy permissions from** dropdown to use an existing role as a template. This allows you to add and remove permissions from an existing set of role permissions.

5 Expand the **Permissions** categories and select or deselect permissions for the role.

* Role name

IX-SecOps

* Description

Security Operations Users for IX Environment

Copy permissions from

View All (with K8s) ▼

* Permissions


+ Alerts	<input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input checked="" type="radio"/> <input checked="" type="radio"/>
+ AlertsTest	<input type="radio"/>
+ API Keys	<input type="radio"/> <input type="radio"/> <input type="radio"/>
+ Appliances	<input type="radio"/> <input checked="" type="radio"/> <input type="radio"/>
+ assignsubroletocustomerrole	<input type="radio"/>
+ Auto-close	<input type="radio"/> <input type="radio"/>
+ cat	<input type="radio"/>
+ Compliance Assessment	<input type="radio"/> <input type="radio"/>
+ Container Security Management	<input type="radio"/>
+ Custom Detections	<input type="radio"/> <input type="radio"/> <input checked="" type="radio"/> <input checked="" type="radio"/>
+ Deobfuscation	<input type="radio"/>
+ Device Control	<input type="radio"/> <input type="radio"/> <input checked="" type="radio"/>
+ dgutinTestSubrole	<input type="radio"/>
+ Endpoint Management	<input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input checked="" type="radio"/> <input type="radio"/> <input checked="" type="radio"/>
+ Files and Reputations	<input type="radio"/> <input type="radio"/> <input checked="" type="radio"/>
+ Host Based Firewall	<input type="radio"/>
+ Investigate	<input checked="" type="radio"/> <input checked="" type="radio"/>
+ Labs	<input type="radio"/> <input type="radio"/>
+ Live Query	<input type="radio"/> <input checked="" type="radio"/>
+ Live Response	<input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/>
+ My test subrole category	<input type="radio"/>
+ name	<input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/> <input type="radio"/>

Save

Cancel

See [Using and Creating Roles for Containers](#) for more information about Container role permissions.

- 6 Click **Save**.

Tip Click the **Duplicate**  icon next to the role in the table to make a copy of that role. Use copied roles to easily make minor adjustments for new roles.

What to do next

- Use the icons to the right of your new role in the table to duplicate, edit, export, or delete the role.
- [Create a User Account for Containers](#)

Create a User Account for Containers

To create a new user account for Containers work, perform the following procedure.

Prerequisites

We recommend that you study the available user roles before you create a user for Containers. Users are granted specific permissions based on their assigned role. Pre-defined user roles are available for selection. If existing roles do not suffice for your environment, you can create custom roles. See [Using and Creating Roles for Containers](#).

Procedure

- 1 On the left navigation pane, click **Settings > Users**.
- 2 In the upper right of the page, click **Add User**.
- 3 Enter the details for the new user including name, email address, and role.
- 4 Click **Save**.

Results

- An email is sent to the input email address. The email prompts the user to log in and create a password.
- Added usernames display after the users have confirmed their login credentials.

Adding Clusters and Installing Kubernetes Sensors

To enable Carbon Black Container, you must install one Carbon Black Kubernetes Sensor for each Kubernetes cluster. To do so, you must add a cluster to the console.

A Kubernetes extension called *Operator* and a custom resource definition are used to deploy the Kubernetes Sensor. Operators consist of set of controllers that deploy and manage user-defined components and report on their health. You define the components with a custom resource definition.

The Carbon Black operator deploys the Kubernetes Sensor inside the cluster and manages its lifecycle. The data in the custom resource file defines which features are enabled for the sensor. The essential steps of the sensor deployment procedure are:

- Setup and install the Carbon Black Operator
- Deploy the Carbon Black Agent on top of the Operator.
- Allow access to the Carbon Black Cloud console
- Configure the Kubernetes Sensor and scanner

Note

- The **Add Cluster** wizard walks you through these steps in [Add a Cluster and Install the Kubernetes Sensor](#).
 - A technical overview and separate deployment instructions for the Operator and Agent are included in [Carbon Black Container Operator Technical Reference](#). You generally do not need to separately install these components, but the background information and deployment content is added here for your convenience.
-

Add a Cluster and Install the Kubernetes Sensor

To add a cluster to the Carbon Black Cloud console and install the Kubernetes Sensor into that cluster, perform the following procedure.

Prerequisites

Before you begin, open both the Carbon Black Cloud console and a terminal window.

Procedure

- 1 On the left navigation pane of the console, do one of the following depending on your system configuration and role:
 - If you have the Kubernetes Security DevOps or SecOps role and your system has the Containers Security feature only, click **Inventory > Clusters**.
 - If you have any other role and your system has Container security and other Carbon Black Cloud features, click **Inventory > Kubernetes > Clusters**.
- 2 In the upper right of the page, click **Add Cluster**.

3 Add the **Cluster Detail** information.

Add Cluster ✕

1 — 2 — 3 — 4
 CLUSTER DETAIL AUTHENTICATION SENSOR FINISH SETUP

CLUSTER DETAIL [Cluster setup guide](#)

* Cluster name Cluster group ?

> Cluster labels

Next Cancel

- a Enter a unique cluster name using lowercase letters, numbers, and hyphens. The name cannot contain a colon (:) symbol.
- b Type or select an existing cluster group to help specify resources in scopes and policies. The cluster group is also used for observing the network activity map of your clusters.

When no group is provided, the cluster is added to the *default* group.

- c Optionally add cluster labels. A label consists of a key and a value. You can add multiple labels.

4 Click **Next**.

5 Provide a dedicated API key to establish the communication between your Kubernetes cluster and the console.

- Click **Generate a new API key** and enter an API key name that is unique to your Carbon Black Cloud organization.

Add Cluster
✕

AUTHENTICATION [Cluster setup guide](#)

New API Key
Existing API Key

⚠ Record this API Secret Key in a secure location for later reference. This information will not be stored in Carbon Black.

API ID	API Secret Key
VYUIRF76AL	HZPFPMZEEK3S5MGMIM1M3CWF

Next
Back
Cancel

- Click **Use existing API key** and select an existing API key.

Important Do not reuse keys between clusters. Use a separate Carbon Black Cloud API key for each cluster.

- 6 Select the version of the Kubernetes Sensor to install on your cluster. The latest sensor version is set by default.
- 7 Under **Advanced Settings**, optionally set up a proxy server or a private container registry.
 - **Proxy server** can include a proxy URL or remain empty. The field is empty by default.
 - **Private container registry** can include a private registry URL or remain empty. The field is empty by default. For important information about using a private container registry, see [Private Container Registry](#).

Add Cluster
✕

SENSOR

[Cluster setup guide](#)

*** Sensor version**

Version main (latest) ▾

Features include:

- K8s security posture management for workload risk
- Automated scans of running images for vulnerability, malware and secrets
- Runtime workload and container protection with threat detection and prevention

Advanced Settings

Proxy server (HTTP/HTTPS)

Private container registry [?](#)

Next
Back
Cancel

Note

- 8 On the **Finish Setup** page, select **Kubectl** or **Helm Charts**.
- 9 Copy and run each command in sequence into your terminal:

Add Cluster
✕

FINISH SETUP

[Cluster setup guide](#)

Deployment tool Kubectl Helm Charts

Run these commands in your terminal and click **Done**

1 — Detect K8s version and install appropriate operator Bash ▾

```
curl -s https://setup.dev.containers.carbonblack.io/develop/operator-apply.sh | bash
```

2 — Create secret for this cluster to connect with Carbon Black

```
kubectl create secret generic cbcontainers-access-token --namespace cbcontainers-dataplane --from-literal=accessToken=HZPFPMZEEK355HGMI1H3CWF/VYU1RF76AL
kubectl create secret generic cbcontainers-company-code --namespace cbcontainers-dataplane --from-literal=companyCode=47H1CAW7HR45WR4S32G03R14U1G5Y
```

3 — Apply cluster configuration and install sensor [View YAML details](#)

```
kubectl apply -f https://setup.dev.containers.carbonblack.io/cr-e841ab46-733d-4ac3-b1ed-890aeffa2e49
```

Done
Back
Cancel

- 10 In the console, click **Done**.
- 11 Refresh the console browser page to view the new cluster.

The cluster status will be **Pending install**.

It takes up to 5 minutes for the cluster to stabilize during the initial setup. During this time, the status might display an error. Wait three to five minutes after submitting the install request to verify the correct status.

Results

After completing the setup procedure successfully, the status changes to **Running**.

What to do next

- 1 [Check the Kubernetes Sensor Status and Health](#)
- 2 [Download a CLI Client](#)
- 3 [Add and Configure a CLI Client](#)

Private Container Registry

You can use a private container registry to reduce traffic costs or to provide application teams with a source of verified container images.

To use Carbon Black Container security through a private container registry, Carbon Black Container provides you with a script to simplify the task of mapping, downloading, and storing the container images.

Before you deploy the sensor:

- Upload and tag the container images to your site
- Configure the registry

For configuration information and instructions, see <https://github.com/octarinesec/octarine-operator/tree/main/charts> (external link).

Check the Kubernetes Sensor Status and Health

To view the status of a Kubernetes Sensor in a cluster, perform the following procedure.

Procedure

- 1 On the left navigation pane of the console, do one of the following depending on your system configuration and role:
 - If you have the Kubernetes Security DevOps or SecOps role and your system has the Containers Security feature only, click **Inventory > Clusters**.
 - If you have any other role and your system has Container security and other Carbon Black Cloud features, click **Inventory > Kubernetes > Clusters**.

- 2 On the Kubernetes Clusters page, click the **Clusters** tab and then click the **General** tab.
- 3 In the left pane, you can filter the list of displayed clusters by:
 - Status
 - Sensor Version
 - Operator Version
 - Cluster Label Key
 - Cluster Label Value
- 4 In the Clusters panel, you can search for a cluster, and you can select a displayed cluster name to view sensor health data.
- 5 Select the cluster and view **Status** in the right panel.

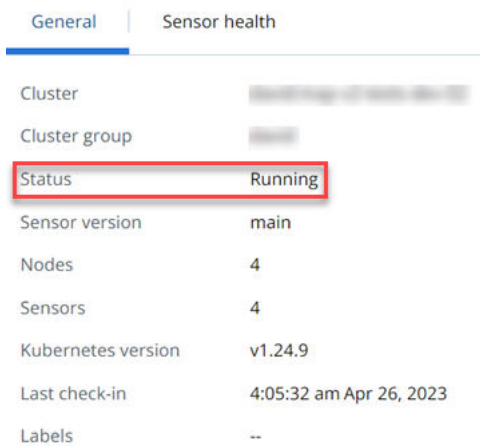


Table 2-2. Kubernetes Sensor Status

Status	Description
Critical	No activity has been detected from any cluster components for more than 24 hours
Error	A critical component is down or the status cannot be detected
Pending install	Cluster setup is in progress
Running	All components are up and running without errors
Warning	A non-critical component is down, or the status cannot be detected

- 6 Click the **Sensor health** tab.

To expand any entry, click the arrow  icon on the left. For example:

General | **Sensor health**

Deployment

- ✓ cbcontainers-runtime-resolver
 - ✓ cbcontainers-runtime-resolver-b9f7674c9-2dnwb
- ✓ cbcontainers-hardening-enforcer
 - ✓ cbcontainers-hardening-enforcer-6f9585dfd4-mq8l5
- ✓ cbcontainers-hardening-state-reporter
 - ✓ cbcontainers-hardening-state-reporter-77fb77cf7c-4ksfb
- ✓ cbcontainers-monitor
 - ✓ cbcontainers-monitor-64b49fc687-lvcc7

Operator

- ✓ cbcontainers-operator
 - ✓ cbcontainers-operator-555f9fb769-jdbrj

Webhook

- ✓ cbcontainers-hardening-enforcer (validating)
- ✓ cbcontainers-hardening-enforcer (mutating)

DaemonSet

- ✓ cbcontainers-node-agent
 - ✓ cbcontainers-node-agent-c84v4
 - ✓ cbcontainers-node-agent-dzmp9
 - ✓ cbcontainers-node-agent-hcm8q
 - ✓ cbcontainers-node-agent-mdgf9

Installing a Containerized Sensor

The Containerized Sensor is an agent that includes both Carbon Black EDR and Image Scanning capabilities. It is used for non-Kubernetes container environments.

The sensor runs as a container, and provides container context to the regular Carbon Black EDR capabilities. This context is known as Cloud Native Detection and Response (CNDR). The sensor scans the containers on the node for vulnerabilities, malware, and secrets.

Required Dependencies

Before you install the Containerized Sensor, make sure that the following requirements are satisfied:

- The sensor is installed as a privileged container on the host network. The installing user must have permissions that allow the sensor to be installed as a privileged container on the host network, as well as the permissions to mount root folders and unix sockets to the container.
- Carbon Black Container
- Carbon Black EDR
- 2GB of memory.
- An API key that has these settings:

Setting	Description
Access Level type	Set to Custom and select KUBERNETES_SECURITY_DATAPLANE.
Access Token	Record the provided API ID and API Secret Key in the format of API Secret Key/API ID, and use it as the access token.

See [Create and Manage an API key](#) for more information.

Set up a Containerized Sensor

Before you can install the containerized sensor, you must set it up.

The Containerized Sensor includes both Carbon Black EDR and Image Scanning capabilities. It is used for non-Kubernetes container environments.

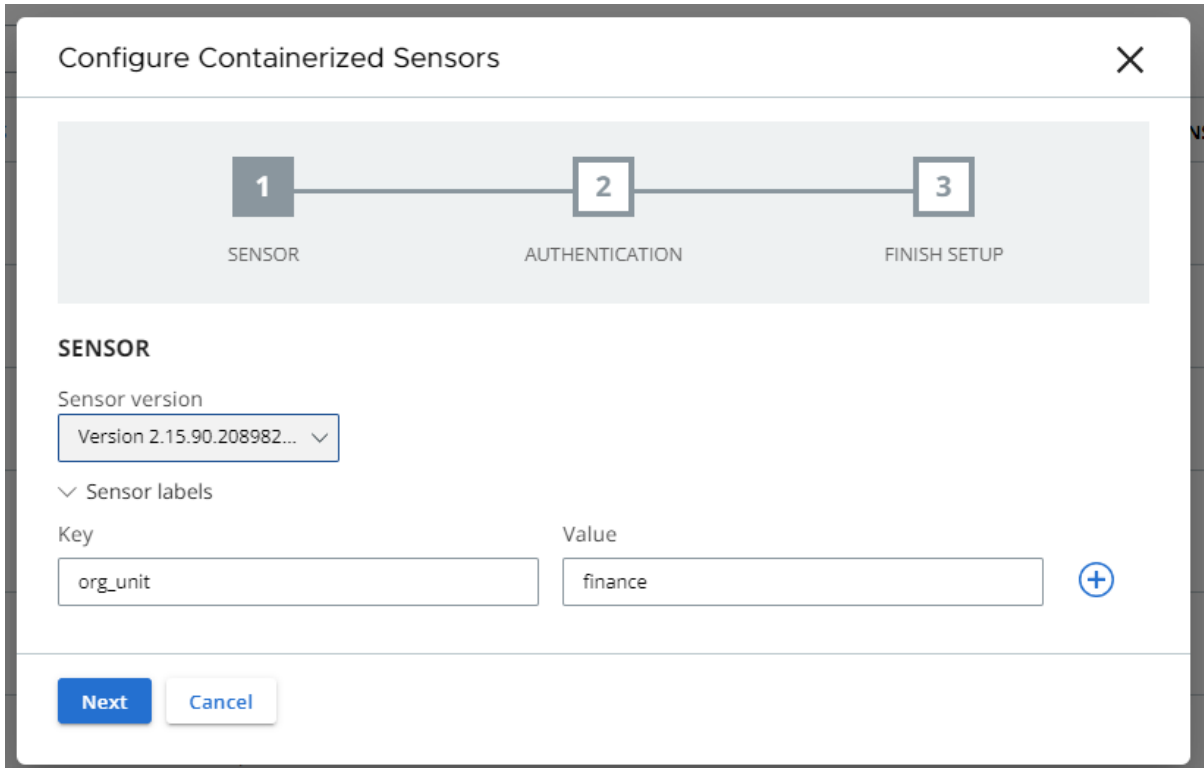
Prerequisites

See [Installing a Containerized Sensor](#).

Procedure

- 1 On the left navigation pane, click **Inventory > Endpoints**.
- 2 In the upper right corner, click **Sensor Options** and select **Configure containerized sensors**.
- 3 Select the sensor version. The sensor version is in the format: `a.b.c.d-e.f`, where `a.b.c.d` pertains to the original Linux sensor version and `e.f` is the version of the image scanning agent. For example: 2.14.0.12349-1.3.

- Click **Show** next to **Sensor Labels** and enter a key and value; for example, `org_unit` and `finance`.



- Click **Next**.

- On the Authentication page, click **Use existing API key** and select the pre-existing API key from the drop down menu.

Configure Containerized Sensors

1 SENSOR — 2 AUTHENTICATION — 3 FINISH SETUP

AUTHENTICATION [Cluster setup guide](#)

API KEY NAME	API ID	ACCESS LEVEL TYPE
hardening api key	XXXXXXXXXX	Kubernetes DevOps
knox-k8s	XXXXXXXXXX	KUBERNETES_SECURITY_DATAPLANE
knox-cli	XXXXXXXXXX	Container Image CLI tool
ben-gke-cndr	XXXXXXXXXX	KUBERNETES_SECURITY_DATAPLANE

Showing 1-20 of 54 Items per page Jump to page < 1 2 3 >

- Click **Next**.
- On the Finish Setup page, download the configuration settings as either a JSON or YAML file.
- Click **Done** to finish setup and close the setup wizard.

What to do next

[Install a Containerized Sensor](#)

Install a Containerized Sensor

After you set up a Containerized Sensor, you can install it.

The Containerized Sensor includes both Carbon Black EDR and Image Scanning capabilities. It is used for non-Kubernetes container environments.

Prerequisites

See:

- [Installing a Containerized Sensor](#)
- [Set up a Containerized Sensor](#)

Procedure

- 1 Run the container image `cbartifactory/cb-containers-sensor:{sensor-version}` together with your selected sensor version.
- 2 Attach these volume mounts to the container:
 - a Container runtime unix socket. Currently only supports docker - `/var/run/docker.sock:/var/run/docker.sock:ro`
 - b Host root path - `:/var/opt/root`
 - c Host hostname - `/etc/hostname:/etc/hostname`
 - d Host boot folder - `/boot:/boot`
 - e Host operating system identification data - `/etc/os-release:/etc/os-release`
 - f Carbon Black Metadata Mount - `/var/opt/carbonblack:/var/opt/carbonblack`
- 3 During sensor setup, the setup wizard provided these environment variables:

Environment Variable	Description
CBC_ACCOUNT	Your Carbon Black Organization Key.
CBC_ACCESS_TOKEN	API key with appropriate permissions.
CB_COMPANY_CODES	Your Carbon Black Company Codes.
CBC_API_HOST	Your Carbon Black environment API host.
HOST_ROOT_PATH	The mounted location of the root path.
CONTAINER_REPORTER_HOSTNAME_FILEPATH	The mounted location of the hostname path.
CONTAINER_REPORTER_LABELS	Key Value labels used to identify the host. For example: <code>key1=value1, key2=value2.</code>

4 (Optional) You can configure the sensor image with additional advanced environment variables:

Environment Variable	Description
CONTAINER_REPORTER_HOST	Value you can set as the container's hostname. You can set the hostname instead of <code>CONTAINER_REPORTER_HOSTNAME_FILEPATH</code> . If both values are set, this variable takes priority. If this value is set, you can delete the hostname volume mount.
ENDPOINT	Value of the host's container-runtime endpoint Unix socket. This value is set to docker's <code>/var/run/docker.sock</code> by default. Note Currently only the docker container runtime is supported.
CONTAINER_RUNTIME	The name of the host container runtime. This value is set to <code>docker-daemon</code> by default. Note Currently only docker container runtime is supported.
SCANNER_CLI_FLAGS_ENABLE_SECRET_DETECTION	Boolean flag to enable/disable container scanning secret detection. This value is set to <code>true</code> (enabled) by default.
SCANNER_CLI_FLAGS_IGNORE_BUILD_IN_REGEX	Boolean flag to determine whether to ignore filenames' built-in regexes and scan every file for secrets. This value is set to <code>false</code> by default.
SCANNER_CLI_FLAGS_SCAN_BASE_LAYERS	Boolean flag used to decide whether to scan the image base layers for secrets. This value is set to <code>false</code> by default.
SCANNER_CLI_FLAGS_SKIP_DIRS_OR_FILES	List of files and directories (in Regexes) to ignore when detecting secrets. This value is set to <code>empty</code> by default.
SCANNER_CLI_FLAGS_CONCURRENT_FILE_LIMIT	Number of files to scan at one time for secrets. This value is set to <code>200</code> by default. You can increase or decrease this number to determine the speed of the scan. If the number is higher, the service requires more resources (memory and CPU).
DISABLE_SCANNER	Boolean flag to disable the container scanner capability. This value is set to <code>false</code> by default.
DISABLE_SENSOR	Boolean flag to disable CNDR capability. This value is set to <code>false</code> by default.

5 Install the sensor:

- Using a docker compose file: [Install a Containerized Sensor on a Docker Client.](#)
- On an AWS ECS cluster: [Install a Containerized Sensor on an ECS Cluster.](#)

Install a Containerized Sensor on a Docker Client

You can run the Carbon Black Containerized Sensor on a host that has the Docker client to detect and enforce EDR and Container Scanning capabilities. Additionally, the Containerized Sensor can detect vulnerabilities, malware, and secrets in the runtime in a Docker container.

Prerequisites

You must have the following products and information:

- Linux Host with docker installed
- Carbon Black Cloud Container
- Carbon Black EDR
- API key with appropriate permissions
- See:
 - [Installing a Containerized Sensor](#)
 - [Set up a Containerized Sensor](#)
 - [Install a Containerized Sensor](#)

Procedure

- 1 Add the environment variables you received from the setup wizard you ran in [Set up a Containerized Sensor](#) to the `docker-compose.yaml` file.

```
version: "3.3"
services:
  sensor:
    pid:host
    network_mode: host
    image: docker.io/cbartifactory/cb-containers-sensor:{sensor-version}
    privileged: true
    environment:
      # fill environment variables here
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock:ro
      - /boot:/boot
      - /var/opt/carbonblack:/var/opt/carbonblack
      - /etc/os-release:/etc/os-release
      - /:/var/opt/root
      - /etc/hostname:/etc/hostname
```

- 2 Deploy the agent container by running the following command:

```
docker-compose up -d
```

Install a Containerized Sensor on an ECS Cluster

You can run the Carbon Black Containerized Sensor on an ECS cluster to detect and enforce EDR and Container Scanning capabilities. Additionally, the Containerized Sensor can detect vulnerabilities, malware, and secrets in the runtime in an ECS Cluster.

Prerequisites

You must have the following products and information:

- ECS Cluster
- Carbon Black Carbon Black Cloud Container
- Carbon Black EDR
- API key with appropriate permissions
- See:
 - [Installing a Containerized Sensor](#)
 - [Set up a Containerized Sensor](#)
 - [Install a Containerized Sensor](#)

Procedure

- 1 Register the agent task definition and update it with the relevant environment variables from the setup wizard you ran in [Set up a Containerized Sensor](#):

```
{
  "family": "cbcontainers-daemon",
  "pidMode": "host",
  "networkMode": "bridge",
  "executionRoleArn": "<arn role with ec2 deployment permissions>",
  "containerDefinitions":
  [
    {
      "name": "host-container-scanner",
      "image": "docker.io/cbartifactory/cb-containers-sensor:{sensor-version} >",
      "cpu": 512,
      "memory": 1024,
      "privileged": true,
      "environment":
      [
        // fill environment variables list here
      ],
      "mountPoints":
      [
        {
          "sourceVolume": "dockersock",
          "containerPath": "/var/run/docker.sock"
        },
        {
          "sourceVolume": "hostname",
```

```

        "containerPath": "/etc/hostname"
    },
    {
        "sourceVolume": "boot",
        "containerPath": "/boot"
    },
    {
        "sourceVolume": "cb-data-dir",
        "containerPath": "/var/opt/carbonblack"
    },
    {
        "sourceVolume": "os-release",
        "containerPath": "/etc/os-release"
    },
    {
        "sourceVolume": "root",
        "containerPath": "/var/opt/root"
    }
],
"healthCheck": {
    "command": [
        "CMD-SHELL",
        "cat /tmp/ready || exit 1"
    ],
    "interval": 60,
    "timeout": 15,
    "retries": 3,
    "startPeriod": 60
}
],
"volumes":
[
    {
        "name": "dockersock",
        "host":
        {
            "sourcePath": "/var/run/docker.sock"
        }
    },
    {
        "name": "hostname",
        "host":
        {
            "sourcePath": "/etc/hostname"
        }
    },
    {
        "name": "boot",
        "host":
        {
            "sourcePath": "/boot"
        }
    },
    {

```

```

    "name": "cb-data-dir",
    "host":
    {
      "sourcePath": "/var/opt/carbonblack"
    }
  },
  {
    "name": "os-release",
    "host":
    {
      "sourcePath": "/etc/os-release"
    }
  },
  {
    "name": "root",
    "host":
    {
      "sourcePath": "/"
    }
  }
],
"requiresCompatibilities":
[
  "EC2"
]
}

```

- 2 Register the agent task definition by using the AWS ECS user interface or the AWS CLI:

```

aws ecs register-task-definition --cli-input-json file://cbcontainers-daemon.json --region
<region-to-apply-at>

```

- 3 (Optional) To write agent logs to AWS CloudWatch, add the `logConfiguration` section inside the container definition element in the task definition:

```

{
  "logConfiguration":
  {
    "logDriver": "awslogs",
    "options":
    {
      "awslogs-group": "cbcontainers-agent",
      "awslogs-region": "<region>",
      "awslogs-stream-prefix": "cbcontainers-agent"
    }
  }
}

```

Add the `cbcontainers-agent` `awslogs-group` and add the `logs:CreateLogStream` and `logs:PutLogEvents` Actions to the ECS Role Policy.

- 4 To run the agent, create a service to run the task: `cbcontainers-daemon-svc`:

```
aws ecs create-service \
  --region <region-to-apply-at> \
  --cluster <your-cluster-name> \
  --service-name cbcontainers-daemon-svc \
  --launch-type EC2 \
  --task-definition cbcontainers-daemon \
  --scheduling-strategy DAEMON
```

- 5 To run the agent as an ECS task, add a role with the following permissions in the `executionRoleArn` section of the task definition:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ec2:DescribeTags",
        "ecs:CreateCluster",
        "ecs:DeregisterContainerInstance",
        "ecs:DiscoverPollEndpoint",
        "ecs:Poll",
        "ecs:RegisterContainerInstance",
        "ecs:StartTelemetrySession",
        "ecs:UpdateContainerInstancesState",
        "ecs:Submit*",
        "ecr:GetAuthorizationToken",
        "ecr:BatchCheckLayerAvailability",
        "ecr:GetDownloadUrlForLayer",
        "ecr:BatchGetImage"
      ],
      "Resource": "*"
    }
  ]
}
```

- 6 (Optional) To write agent logs to AWS CloudWatch, add the Actions `logs:CreateLogStream` and `logs:PutLogEvents` to the Actions list.

Note To write the containers logs, the policy must have cloudwatch access and permissions to pull images and run ECS tasks.

Validate the Container Image Signature

To verify the security and integrity of the container image, you can validate the container signature.

During verification, use this public key:

```
-----BEGIN PUBLIC KEY-----
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAE1ivoAvFrHG19lM01ecsBN1juD0p5
6kGA7G5M0WnOS2zc5qNPQSN1fzwOc/EgEIskERJY/NMmCjq0rcZzzKgfxQ==
-----END PUBLIC KEY-----
```

Prerequisites

Before you can verify the container image signing, you must download the [cosign tool](#).

Procedure

- 1 Download the containerized sensor image: `cbartifactory/cb-containers-sensor` using an image management tool, such as `docker`.
- 2 Run the signature verification command using the public key above:

```
cosign verify --key container-signing-key.pub cbartifactory/cb-containers-
sensor:<sensor-version>
```

Results

An example of a successful verification:

```
Verification for docker.io/cbartifactory/cb-containers-sensor:<sensor-version> --
The following checks were performed on each of these signatures:
- The cosign claims were validated
- Existence of the claims in the transparency log was verified offline
- The signatures were verified against the specified public key
[
  {
    "critical": {
      "identity": {
        "docker-reference": "docker.io/cb/cbartifactory/cb-containers-sensor"
      },
      "image": {
        "docker-manifest-digest":
"sha256:ala0dfe211c0fdcbae68fccb7629e79f3d9775891584daddc8aff5050237911"
      },
      "type": "cosign container image signature"
    },
    "optional": {
      "Bundle": {
        "SignedEntryTimestamp":
"MEUCIBiIc38wiBow7FT09yIlanYEki248tu4kYcJYr3dSwRUkAiEA9R9pK6SnTaTnhPKmK592n0keUGj8mdxTIA1Fc75j7
i4=",
        "Payload": {
          "body":
"eyJhcGlWZXJzaW9uIjoia0dfe211c0fdcbae68fccb7629e79f3d9775891584daddc8aff5050237911"

```

```

KT01XcDFSRT13T1FvMmEwZEJOMGMxVFRCWGJrOVRNbnBqTlhGT1VGR1RUakZtZW5kUF15OUZaMFZKYzJ0RlVrcFpMMDVOY
1VOcWNUQnlZMxA2ZWt0blpuaFJQVDBLTfMwdExTMUZUa1FnVUZWQ1RFbERJRXRGV1MwdExTMHRDZz09In19fX0=",
    "integratedTime": 1699443190,
    "logIndex": 48394752,
    "logID": "c0d23d6ad406973f9559f3ba2d1ca01f84147d8ffc5b8445c224f98b9591801d"
  }
}
}
}
]

```

Setting up CLI Client for Image Scanning

To include image scanning in your continuous integration script, configure and use the Carbon Black Cloud CLI Client (`cbctl`). This client is available for Linux and macOS.

You can install the CLI client on a Dev/Sec/Ops machine, or you can include it in a CI/CD pipeline — for example, Jenkins or Gitlab. The CLI client requires an Internet connection to Carbon Black Cloud and access to your container registries.

Carbon Black CLI Client performs an image scan for known vulnerabilities and enforces security or compliance rules. The CLI Client performs the following tasks:

- **Vulnerabilities scanning of container images.**

Container images are matched against a known vulnerabilities database. The image details include operating system and non-operating system packages, libraries, licenses, binaries, and metadata. The vulnerabilities scan result is included in the image metadata.

- **Enforcing standards for container images.**

To evaluate policy violations, the image scan results are matched against a specific policy that is configured for the CLI scope. The CLI run fails the build pipeline step if policy violations are detected. The violation of policy rules is added to the image metadata together with image rule exceptions.

- **Enforcing standards for Kubernetes workloads.**

Kubernetes workloads are matched against a Kubernetes hardening policy to evaluate the workload compliance for security risks. By leveraging the information from both image vulnerabilities and workload configuration, a complete picture of the workload risk exposure is available.

The CLI client presents the following interface and command options:

```

$ cbctl
A client CLI for image scanning, and instrumenting Carbon Black services.

Usage:
  cbctl [command]

Available Commands:
  auth          Set auth for cbctl
  completion    generate the autocompletion script for the specified shell
  config        Manage Carbon Black configuration
  help          Help about any command
  image         Commands related to image analysis
  k8s-object    Commands related to k8s-object analysis
  user          Manage cbctl user profiles
  version       Show the cli tool version and build info

Flags:
  -c, --config string          config file (default "/home/slist/.cbctl/.cbctl.yaml")
  --debug string[="/home/slist/.cbctl/debug.log"]  enable debug log (default "/home/slist/.cbctl/debug.log")
  -h, --help                  help for cbctl
  --plain-mode                display ui on plain mode
  -u, --user-profile string    user profile

Use "cbctl [command] --help" for more information about a command.
$

```

Secrets File Detection

If secret detection is enabled, Carbon Black Cloud detects all text files in an image. Files can be ignored; these are specified by using CLI flags. System files are ignored by default to reduce scan time.

Note To enable or disable secrets detection, see [Add a Cluster and Install the Kubernetes Sensor](#).

Table 2-3. CLI Flags

Flag	Description	Default Setting
<code>enableSecretDetection</code>	Indicates whether the scan should scan for secrets	False
<code>skipDirsOrFiles</code>	Files or directories to not scan for secrets	N/A
<code>scanBaseLayers</code>	Indicates whether the scan should scan the base layers for secrets	False
<code>ignoreBuildInRegex</code>	Indicates whether the scan should ignore the build-in regexes of files	False

Download a CLI Client

Add configured CLI instances to enable local scanning of images, workload vulnerability assessment, and CI integration. The CLI instance scans container images and reports their health to the Carbon Black Cloud console.

Procedure

- 1 On the left navigation pane of the console, do one of the following depending on your system configuration and role:
 - If you have the Kubernetes Security DevOps or SecOps role and your system has the Containers Security feature only, click **Inventory > Clusters**.
 - If you have any other role and your system has Container security and other Carbon Black Cloud features, click **Inventory > Kubernetes > Clusters**.
- 2 Click the **CLI Config** tab.
- 3 In the upper right of the page, click **Download CLI**.
- 4 Select and download the CLI client for your operating system (macOS or Linux).

Download CLI

OS	VERSION	DETAILS	ACTION
CLI client (Mac)	v1.9.2	MD5SUM	ca8eb6bdb0f825a2ed4ce329e869f256
		SHA1SUM	295338b3b54bf53bb99beb800aa78d250b80e816
		SHA256SUM	c0d51bbbe7247d91c1f5647f714b6c3683c93887aafef311a3077bf270b883e0
CLI client (Linux)	v1.9.2	MD5SUM	ff954bdec199d856b1bed37c5335059f
		SHA1SUM	f6d151e8d745248c47d05621d5b0c11cac1c21b7
		SHA256SUM	9f3dfe307f02c139c8ef3a22a25807245fe61a11f2094462e2bf57d3dc2a2b20

Close

- 5 Click **Close**.

What to do next

Add and Configure a CLI Client

Add and Configure a CLI Client

To set up a CLI instance for image scanning, perform the following procedure.

Add configured CLI instances to enable local scanning of images, workload vulnerability assessment, and CI integration. The CLI instance scans container images and reports their health to the Carbon Black Cloud console.

Prerequisites

[Download a CLI Client](#)

Before you begin, open both the Carbon Black Cloud console and a terminal window.

Procedure

- 1 On the left navigation pane of the console, do one of the following depending on your system configuration and role:
 - If you have the Kubernetes Security DevOps or SecOps role and your system has the Containers Security feature only, click **Inventory > Clusters**.
 - If you have any other role and your system has Container security and other Carbon Black Cloud features, click **Inventory > Kubernetes > Clusters**.
- 2 Click the **CLI Config** tab.
- 3 In the upper right of the page, click **Add CLI**.
 - a Enter a unique name for this CLI instance (different from the API key name).

Use lowercase characters, numbers, and hyphens only. The name helps identify and manage the CLI in the console.
 - b Enter the build step name (for example, development, production, compliance) to be used as the default field for CLI runs.

Build steps are used as reference IDs in build-phase scopes to establish a connection with related configured CLIs. The build step parameter is used to match a scope in Carbon Black Cloud, and consecutively to apply the policy for that scope. The default scope is stored in the configuration file.

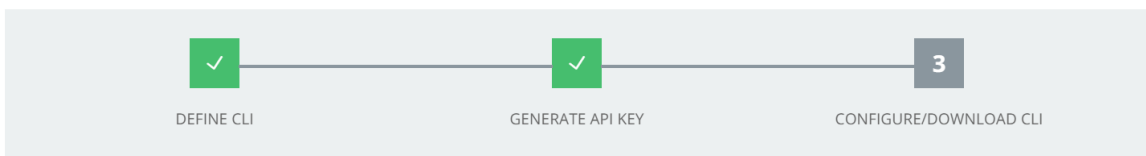
Note

 - The default build step is not unique. Multiple CLI instances can use the same default scope. The **Default build step** cannot be modified after the initial setup, unless you directly edit the configuration file.
 - If a scan is invoked without a build step parameter, the default build step from the configuration file is used.
 - Create a Build Phase scope using this value on the **Kubernetes > Scopes** page in **Build steps**. See [Kubernetes Scopes](#).
 - You must use the CLI `validate` command.
 - c Add an optional description (recommended).
- 4 Click **Next**.
- 5 Enter a unique API key name and click **Generate**.
- 6 Click **Next**.

7 Copy and run the following command in your terminal window.

```
mkdir -p ~/.cbctl
cat > ~/.cbctl/.cbctl.yaml <<EOF
active_user_profile: cbctl_default
cbctl_default:
  cb_api_id: UHSZCDKMI1
  cb_api_key: 4AYGEJ1T9ILZTQ6VZG9TTEH8
  org_key: EWRTY2PK
  saas_url: https://defense-dev01.cbdtest.io/containers
  default_build_step: ix-test
EOF
```

Add CLI



CONFIGURE CLI CLIENT

[CLI setup guide](#)

Copy this command into your terminal and run it

```
mkdir -p ~/.cbctl
cat > ~/.cbctl/.cbctl.yaml <<EOF
active_user_profile: cbctl_default
cbctl_default:
  cb_api_id: UHSZCDKMI1
  cb_api_key: 4AYGEJ1T9ILZTQ6VZG9TTEH8
  org_key: EWRTY2PK
  saas_url: https://defense-dev01.cbdtest.io/containers
  default_build_step: ix-test
EOF
```

DOWNLOAD CLI CLIENT

Already have the CLI client? Skip this step and click **Done**

- + CLI client (Mac) v1.9.2 [Download](#)
- + CLI client (Linux) v1.9.2 [Download](#)

Done

Back

Cancel

8 If you did not already download the CLI client, you can select and download the CLI instance binary file now, and run it in your build environment.

9 Click **Done**.

Results

You can operate the configured CLI Client in a terminal to observe the results from vulnerabilities scans on your container images.

What to do next

To run the Image Scanning CLI API, see [Container Security API and Integrations](#).

To monitor the Vulnerabilities scan for container images that are deployed on Kubernetes, go to the **Inventory > Kubernetes > Container Images** page.

To see the image scanning results for container images that are in particular repositories but not yet deployed, go to the **Inventory > Kubernetes > Container Images** page and click the **Image Repos** tab.

Carbon Black Container Operator Technical Reference

The Carbon Black Container Operator runs within a Kubernetes cluster. The Container Operator is a set of controllers that deploy and manage the Carbon Black Container components.

The Operator handles the following actions:

- Deploys and manages the Carbon Black Container product, including the configuration and the image scanning for Kubernetes security.
- Automatically fetches and deploys the Carbon Black Container private image registry secret.
- Automatically registers the Carbon Black Container cluster.
- Manages the Carbon Black Container validating webhook and dynamically manages the admission control webhook to avoid possible downtime.
- Monitors and reports agent availability to the Carbon Black Cloud console.

The Carbon Black Container Operator uses the operator-framework to create a GO operator that is responsible for managing and monitoring the Carbon Black Container components deployment.

To review the Operator compatibility matrix, see [Kubernetes Sensor Operator Distributions and Kubernetes Version](#).

Note We recommend that you deploy the Operator by using the **Add Cluster** wizard (see [Add a Cluster and Install the Kubernetes Sensor](#)). However, this technical reference section of the *User Guide* also includes manual Operator and Agent installation instructions.

Manually Deploy the Container Operator

To manually deploy the Carbon Black Container Operator, perform the following procedure.

These instructions use an Operator image. To deploy the Operator without using an image, see [Container Operator Developer Instructions](#).

Prerequisites

Your cluster must be running Kubernetes 1.18+.

Procedure

- ◆ You can initiate the Operator deployment in two ways:

- **Script:**

```
export OPERATOR_VERSION=v6.0.2
export OPERATOR_SCRIPT_URL=https://setup.containers.carbonblack.io/$OPERATOR_VERSION/
operator-apply.sh
curl -s $OPERATOR_SCRIPT_URL | bash
```

{OPERATOR_VERSION} is of the format "v{VERSION}".

- **Source code:**

- a Clone the GIT project and deploy the operator from the source code.

By default, the Operator uses `CustomResourceDefinitions v1`, which requires Kubernetes 1.16+. You can also deploy an Operator by using `CustomResourceDefinitions v1beta1` (deprecated in Kubernetes 1.16, removed in Kubernetes 1.22).

- b Create the Operator image:

```
make docker-build docker-push IMG={IMAGE_NAME}
```

- c Deploy the Operator resources:

```
make deploy IMG={IMAGE_NAME}
```

What to do next

[Manually Deploy the Container Agent](#)

Uninstall the Container Operator

To uninstall the Carbon Black Container Operator, perform the following procedure.

Procedure

- ◆ To uninstall the Carbon Black Container Operator, run the following command:

```
export OPERATOR_VERSION=v6.0.2
export OPERATOR_SCRIPT_URL=https://setup.containers.carbonblack.io/$OPERATOR_VERSION/
operator-apply.sh
curl -s $OPERATOR_SCRIPT_URL | bash -s -- -u
```

This command deletes the Carbon Black Container custom resource definitions (CRDs) and instances.

Manually Deploy the Container Agent

To manually deploy the Carbon Black Container Agent, perform the following procedure.

Prerequisites

Manually Deploy the Container Operator

Procedure

- 1 Apply the Carbon Black Container API token secret:

```
kubectl create secret generic cbcontainers-access-token \
--namespace cbcontainers-dataplane --from-literal=accessToken=\
{API_Secret_Key}/{API_ID}
kubectl create secret generic cbcontainers-company-code --namespace cbcontainers-dataplane
--from-literal=companyCode=RXXXXXXXXXXG\!XXXX
```

- 2 Apply the Carbon Black Container Agent custom resource:

Deploy `cbcontainersagents.operator.containers.carbonblack.io` to prompt the Operator to deploy the dataplane components:

```
apiVersion: operator.containers.carbonblack.io/v1
kind: CBContainersAgent
metadata:
  name: cbcontainers-agent
spec:
  account: {ORG_KEY}
  clusterName: {CLUSTER_GROUP}:{CLUSTER_NAME}
  version: {AGENT_VERSION}
  gateways:
    apiGateway:
      host: {API_HOST}
    coreEventsGateway:
      host: {CORE_EVENTS_HOST}
    hardeningEventsGateway:
      host: {HARDENING_EVENTS_HOST}
    runtimeEventsGateway:
      host: {RUNTIME_EVENTS_HOST}
```

Note See also [Custom Resources Definitions](#).

Openshift

The Carbon Black Container Operator and Agent require elevated permissions to operate properly. However, this requirement violates the default `SecurityContextConstraints` on most Openshift clusters, thereby causing the components to fail to start.

You can resolve this issue by applying the following custom security constraint configurations on the cluster. This action requires cluster administrator privileges.

```
kind: SecurityContextConstraints
apiVersion: security.openshift.io/v1
metadata:
  name: scc-anyuid
runAsUser:
```

```

  type: MustRunAsNonRoot
allowHostPID: false
allowHostPorts: false
allowHostNetwork: false
allowHostDirVolumePlugin: false
allowHostIPC: false
allowPrivilegedContainer: false
readOnlyRootFilesystem: true
seLinuxContext:
  type: RunAsAny
fsGroup:
  type: RunAsAny
supplementalGroups:
  type: RunAsAny
users:
- system:serviceaccount:cbcontainers-dataplane:cbcontainers-operator
- system:serviceaccount:cbcontainers-dataplane:cbcontainers-enforcer
- system:serviceaccount:cbcontainers-dataplane:cbcontainers-state-reporter
- system:serviceaccount:cbcontainers-dataplane:cbcontainers-monitor
- system:serviceaccount:cbcontainers-dataplane:cbcontainers-runtime-resolver
---
kind: SecurityContextConstraints
apiVersion: security.openshift.io/v1
metadata:
  name: scc-image-scanning # This probably needs to be fixed in the actual deployment
runAsUser:
  type: RunAsAny
allowHostPID: false
allowHostPorts: false
allowHostNetwork: false
allowHostDirVolumePlugin: false
allowHostIPC: false
allowPrivilegedContainer: false
readOnlyRootFilesystem: false
seLinuxContext:
  type: RunAsAny
fsGroup:
  type: RunAsAny
supplementalGroups:
  type: RunAsAny
allowedCapabilities:
- 'NET_BIND_SERVICE'
users:
- system:serviceaccount:cbcontainers-dataplane:cbcontainers-image-scanning
---
kind: SecurityContextConstraints
apiVersion: security.openshift.io/v1
metadata:
  name: scc-node-agent
runAsUser:
  type: RunAsAny
allowHostPID: true
allowHostPorts: false
allowHostNetwork: true
allowHostDirVolumePlugin: true

```

```

allowHostIPC: false
allowPrivilegedContainer: true
readOnlyRootFilesystem: false
seLinuxContext:
  type: RunAsAny
fsGroup:
  type: RunAsAny
supplementalGroups:
  type: RunAsAny
volumes:
- configMap
- downwardAPI
- emptyDir
- hostPath
- persistentVolumeClaim
- projected
- secret
users:
- system:serviceaccount:cbcontainers-dataplane:cbcontainers-agent-node

```

Uninstalling the Operator on Openshift

Add this `SecurityContextConstraints` before running the operator uninstall command:

```

kind: SecurityContextConstraints
apiVersion: security.openshift.io/v1
metadata:
  name: scc-edr-cleaner
runAsUser:
  type: RunAsAny
allowHostPID: true
allowHostPorts: false
allowHostNetwork: true
allowHostDirVolumePlugin: true
allowHostIPC: false
allowPrivilegedContainer: true
readOnlyRootFilesystem: false
seLinuxContext:
  type: RunAsAny
fsGroup:
  type: RunAsAny
supplementalGroups:
  type: RunAsAny
volumes:
- configMap
- downwardAPI
- emptyDir
- hostPath
- persistentVolumeClaim
- projected
- secret
users:
- system:serviceaccount:cbcontainers-edr-sensor-cleaners:cbcontainers-edr-sensor-cleaner

```

Reading Metrics by using Prometheus

Operator metrics are protected by `kube-auth-proxy`. You must grant permissions to a Prometheus server before it can scrape the protected metrics.

You can create a `ClusterRole` and bind it with `ClusterRoleBinding` to the service account that your Prometheus server uses.

If you have not configured this cluster role and cluster role binding, you can use the following configuration:

Cluster Role

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: cbcontainers-metrics-reader
rules:
  - nonResourceURLs:
    - /metrics
    verbs:
    - get
```

Cluster Role Binding

```
kubectl create clusterrolebinding metrics --clusterrole=cbcontainers-metrics-reader --
serviceaccount=<prometheus-namespace>:<prometheus-service-account-name>
```

Use the following `ServiceMonitor` to scrape metrics from the Carbon Black Container Operator. Your Prometheus custom resource service monitor selectors must match this configuration.

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  labels:
    control-plane: operator
  name: cbcontainers-operator-metrics-monitor
  namespace: cbcontainers-dataplane
spec:
  endpoints:
  - bearerTokenFile: /var/run/secrets/kubernetes.io/serviceaccount/token
    path: /metrics
    port: https
    scheme: https
    tlsConfig:
      insecureSkipVerify: true
  selector:
    matchLabels:
      control-plane: operator
```

Custom Resources Definitions

The Carbon Black Container Operator implements controllers for Carbon Black Container custom resources definitions (CRDs).

Carbon Black Container Agent Custom Resource

Deploy `cbcontainersagents.operator.containers.carbonblack.io` to prompt the Operator to deploy the dataplane components.

Table 2-4. Required Parameters

Parameter	Description
<code>spec.account</code>	Carbon Black Container org key
<code>spec.clusterName</code>	Carbon Black Container cluster name (<code><cluster_group:cluster_name></code>)
<code>spec.version</code>	Carbon Black Container Agent version
<code>spec.gateways.apiGateway.host</code>	Carbon Black Container API host
<code>spec.gateways.coreEventsGateway.host</code>	Carbon Black Container core events host (for example, health checks)
<code>spec.gateways.hardeningEventsGateway.host</code>	Carbon Black Container hardening events host (for example, deleted, validated, and blocked resources)
<code>spec.gateways.runtimeEventsGateway.host</code>	Carbon Black Container runtime events host (for example, traffic events)

Table 2-5. Optional Parameters

Parameter	Description	Default Value
<code>spec.apiGateway.port</code>	Carbon Black Container API port	443
<code>spec.accessTokenSecretName</code>	Carbon Black Container API access token secret name	<code>cbcontainers-access-token</code>
<code>spec.gateways.coreEventsGateway.port</code>	Carbon Black Container core events port	443
<code>spec.gateways.hardeningEventsGateway.port</code>	Carbon Black Container hardening events port	443
<code>spec.gateways.runtimeEventsGateway.port</code>	Carbon Black Container runtime events port	443

Table 2-6. Basic Components Optional Parameters

Parameter	Description	Default Value
<code>spec.components.basic.enforcer.replicasCount</code>	Carbon Black Container Hardening Enforcer number of replicas	1
<code>spec.components.basic.monitor.image.repository</code>	Carbon Black Container Monitor image repository	<code>cbartifactory/monitor</code>

Table 2-6. Basic Components Optional Parameters (continued)

Parameter	Description	Default Value
<code>spec.components.basic.enforcer.image.repository</code>	Carbon Black Container Hardening Enforcer image repository	<code>cbartifactory/guardrails-enforcer</code>
<code>spec.components.basic.stateReporter.image.repository</code>	Carbon Black Container Hardening State Reporter image repository	<code>cbartifactory/guardrails-state-reporter</code>
<code>spec.components.basic.monitor.resources</code>	Carbon Black Container Monitor resources	<code>{requests: {memory: "64Mi", cpu: "30m"}, limits: {memory: "256Mi", cpu: "200m"}}</code>
<code>spec.components.basic.enforcer.resources</code>	Carbon Black Container Hardening Enforcer resources	<code>{requests: {memory: "64Mi", cpu: "30m"}, limits: {memory: "256Mi", cpu: "200m"}}</code>
<code>spec.components.basic.stateReporter.resources</code>	Carbon Black Container Hardening State Reporter resources	<code>{requests: {memory: "64Mi", cpu: "30m"}, limits: {memory: "256Mi", cpu: "200m"}}</code>

Table 2-7. Runtime Components Optional Parameters

Parameter	Description	Default Value
<code>spec.components.runtimeProtection.enabled</code>	Carbon Black Container flag to control Runtime components deployment	<code>True</code>
<code>spec.components.runtimeProtection.resolver.image.repository</code>	Carbon Black Container Runtime Resolver image repository	<code>cbartifactory/runtime-kubernetes-resolver</code>
<code>spec.components.runtimeProtection.sensor.image.repository</code>	Carbon Black Container Runtime Sensor image repository	<code>cbartifactory/runtime-kubernetes-sensor</code>
<code>spec.components.runtimeProtection.internalGrpcPort</code>	Carbon Black Container Runtime gRPC port that the resolver exposes for the sensor	<code>443</code>
<code>spec.components.runtimeProtection.resolver.logLevel</code>	Carbon Black Container Runtime Resolver log level	<code>"panic", "fatal", "error", "warn", "info", "debug", "trace" (default info)</code>
<code>spec.components.runtimeProtection.resolver.resources</code>	Carbon Black Container Runtime Resolver resources	<code>{requests: {memory: "64Mi", cpu: "200m"}, limits: {memory: "1024Mi", cpu: "900m"}}</code>
<code>spec.components.runtimeProtection.sensor.logLevel</code>	Carbon Black Container Runtime Sensor log level	<code>"panic", "fatal", "error", "warn", "info", "debug", "trace" (default info)</code>
<code>spec.components.runtimeProtection.sensor.resources</code>	Carbon Black Container Runtime Sensor resources	<code>{requests: {memory: "64Mi", cpu: "30m"}, limits: {memory: "1024Mi", cpu: "500m"}}</code>

Table 2-8. Cluster Scanning Components Optional Parameters

Parameter	Description	Default Value
<code>spec.components.clusterScanning.enabled</code>	Carbon Black Container flag to control Cluster Scanning components deployment	True
<code>spec.components.clusterScanning.imageScanningReporter.image.repository</code>	Carbon Black Container Image Scanning Reporter image repository	<code>cbartifactory/image-scanning-reporter</code>
<code>spec.components.clusterScanning.clusterScanner.image.repository</code>	Carbon Black Container Scanner Agent image repository	<code>cbartifactory/cluster-scanner</code>
<code>spec.components.clusterScanning.imageScanningReporter.resources</code>	Carbon Black Container Image Scanning Reporter resources	<code>{requests: {memory: "64Mi", cpu: "200m"}, limits: {memory: "1024Mi", cpu: "900m"}}</code>
<code>spec.components.clusterScanning.clusterScanner.resources</code>	Carbon Black Container Cluster Scanner resources	<code>{requests: {memory: "64Mi", cpu: "30m"}, limits: {memory: "1024Mi", cpu: "500m"}}</code>
<code>spec.components.clusterScanning.clusterScanner.k8sContainerEngine.engineType</code>	Carbon Black Container Cluster Scanner Kubernetes container engine type. One of these options: <code>containerd / docker-daemon / cri-o</code>	N/A
<code>spec.components.clusterScanning.clusterScanner.k8sContainerEngine.endpoint</code>	Carbon Black Container Cluster Scanner Kubernetes container engine endpoint path	N/A
<code>spec.components.clusterScanning.clusterScanner.k8sContainerEngine.CRIO.storagePath</code>	Carbon Black Container Cluster Scanner override default image storage path (CRI-O only)	N/A
<code>spec.components.clusterScanning.clusterScanner.k8sContainerEngine.CRIO.storageConfigPath</code>	Carbon Black Container Cluster Scanner override default image storage config path (CRI-O only)	N/A
<code>spec.components.clusterScanning.clusterScanner.k8sContainerEngine.CRIO.configPath</code>	Carbon Black Container Cluster Scanner override default CRI-O config path (CRI-O only)	N/A
<code>spec.components.clusterScanning.clusterScanner.cliFlags.enableSecretDetection</code>	Carbon Black Container Cluster Scanner flag of whether the scan should scan for secrets	False
<code>spec.components.clusterScanning.clusterScanner.cliFlags.skipDirsOrFiles</code>	Carbon Black Container Cluster Scanner flag of files or directories to not scan for secrets	N/A

Table 2-8. Cluster Scanning Components Optional Parameters (continued)

Parameter	Description	Default Value
<code>spec.components.clusterScanning.clusterScanner.cliFlags.scanBaseLayers</code>	Carbon Black Container Cluster Scanner flag of whether the scan should include the base layers scan for secrets	False
<code>spec.components.clusterScanning.clusterScanner.cliFlags.ignoreBuildInRegex</code>	Carbon Black Container Cluster Scanner flag of whether the scan should ignore the built-in regexes of files to skip secret detection	False

Table 2-9. Components Common Optional Parameters

Parameter	Description	Default Value
<code>labels</code>	Carbon Black Container component deployment and pod labels	Empty map
<code>deploymentAnnotations</code>	Carbon Black Container component deployment annotations	Empty map
<code>podTemplateAnnotations</code>	Carbon Black Container component pod annotations	{}
<code>env</code>	Carbon Black Container component pod environment variables	Empty map
<code>image.tag</code>	Carbon Black Container component image tag	Agent version
<code>image.pullPolicy</code>	Carbon Black Container component pull policy	IfNotPresent
<code>probes.port</code>	Carbon Black Container component probes port	8181
<code>probes.scheme</code>	Carbon Black Container component probes scheme	HTTP
<code>probes.initialDelaySeconds</code>	Carbon Black Container component probes initial delay seconds	3
<code>probes.timeoutSeconds</code>	Carbon Black Container component probes timeout seconds	1
<code>probes.periodSeconds</code>	Carbon Black Container component probes period seconds	30
<code>probes.successThreshold</code>	Carbon Black Container component probes success threshold	1
<code>probes.failureThreshold</code>	Carbon Black Container component probes failure threshold	3
<code>prometheus.enabled</code>	Carbon Black Container component enable Prometheus scraping	False
<code>prometheus.port</code>	Carbon Black Container component Prometheus server port	7071

Table 2-9. Components Common Optional Parameters (continued)

Parameter	Description	Default Value
<code>nodeSelector</code>	Carbon Black Container component node selector	<code>{}</code>
<code>affinity</code>	Carbon Black Container component affinity	<code>{}</code>

Table 2-10. Centralized Proxy Parameters

Parameter	Description	Default Value
<code>spec.components.settings.proxy.enabled</code>	Enables applying the centralized proxy settings to all components	False
<code>spec.components.settings.proxy.httpProxy</code>	HTTP proxy server address to use	Empty string
<code>spec.components.settings.proxy.httpsProxy</code>	HTTPS proxy server address to use	Empty string
<code>spec.components.settings.proxy.noProxy</code>	A comma-separated list of hosts to which to connect without using a proxy	Empty string
<code>spec.components.settings.proxy.noProxySuffix</code>	A comma-separated list of hosts to which to append the <code>noProxy</code> list of values	The API server IP addresses followed by <code>cbcontainers-dataplane.svc.cluster.local</code>

Table 2-11. Other Components Optional Parameters

Parameter	Description	Default Value
<code>spec.components.settings.daemonSetsTolerations</code>	Carbon Black DaemonSet component tolerances	Empty array

Changing Components Resources

Needs description/intro.

```
spec:
  components:
    basic:
      monitor:
        resources:
          limits:
            cpu: 200m
            memory: 256Mi
          requests:
            cpu: 30m
            memory: 64Mi
      enforcer:
        resources:
          ##### DESIRED RESOURCES SPEC - for hardening enforcer container
      stateReporter:
```

```

resources:
  ##### DESIRED RESOURCES SPEC - for hardening state reporter container
runtimeProtection:
resolver:
  resources:
    ##### DESIRED RESOURCES SPEC - for runtime resolver container
sensor:
  resources:
    ##### DESIRED RESOURCES SPEC - for node-agent runtime container
clusterScanning:
imageScanningReporter:
  resources:
    ##### DESIRED RESOURCES SPEC - for image scanning reporter pod
clusterScanner:
  resources:
    ##### DESIRED RESOURCES SPEC - for node-agent cluster-scanner container

```

Cluster Scanner Component Memory

By default, the `clusterScanning.clusterScanner` component attempts to scan images of sizes up to 1GB. Its recommended resources are:

```

resources:
  requests:
    cpu: 100m
    memory: 1Gi
  limits:
    cpu: 2000m
    memory: 6Gi

```

To scan images larger than 1GB, allocate higher memory resources in the component's `requests.memory` and `limits.memory`, and add an environment variable `MAX_COMPRESSED_IMAGE_SIZE_MB` to override the maximum images size in MB that the scanner tries to scan.

For example, to set the cluster scanner to scan images up to 1.5 GB. the configuration is:

```

spec:
  components:
    clusterScanning:
      clusterScanner:
        env:
          MAX_COMPRESSED_IMAGE_SIZE_MB: "1536" // 1536 MB == 1.5 GB
        resources:
          requests:
            cpu: 100m
            memory: 2Gi
          limits:
            cpu: 2000m
            memory: 5Gi

```

If your nodes have low memory and you want the cluster scanner to consume less memory, you must reduce the component's `requests.memory` and `limits.memory`, and override the `MAX_COMPRESSED_IMAGE_SIZE_MB` parameter to be less than 1GB (1024MB).

For example, assign lower memory resources and set the cluster-scanner to scan images up to 250MB:

```
spec:
  components:
    clusterScanning:
      clusterScanner:
        env:
          MAX_COMPRESSED_IMAGE_SIZE_MB: "250" // 250 MB
        resources:
          requests:
            cpu: 100m
            memory: 250Mi
          limits:
            cpu: 2000m
            memory: 1Gi
```

Configuring Container Services to use HTTP Proxy

You can configure the Carbon Black Container to use an HTTP proxy by enabling the centralized proxy settings or by manually setting `HTTP_PROXY`, `HTTPS_PROXY`, and `NO_PROXY` environment variables.

The centralized proxy settings apply an HTTP proxy configuration for all components. The manual setting of environment variables allows you to set the configuration parameters on a per component basis. If both HTTP proxy environment variables and centralized proxy settings are provided, the environment variables take precedence. The Operator does not use the centralized proxy settings, so you must use the environment variables for it instead.

Configure Centralized Proxy Settings

To configure the proxy environment variables in the Operator, use the following command to patch the Operator deployment:

```
kubectl set env -n cbcontainers-dataplane deployment cbcontainers-operator HTTP_PROXY="<proxy-url>" HTTPS_PROXY="<proxy-url>" NO_PROXY="<kubernetes-api-server-ip>/<range>"
```

Update the `CBContainersAgent` CR to use the centralized proxy settings (`kubectl edit cbcontainersagents.operator.containers.carbonblack.io cbcontainers-agent`):

```
spec:
  components:
    settings:
      proxy:
```

```

enabled: true
httpProxy: "<proxy-url>"
httpsProxy: "<proxy-url>"
noProxy: "<exclusion1>,<exclusion2>"

```

You can disable the centralized proxy settings without deleting them by setting the `enabled` key to `false`.

By default, the centralized proxy settings determine the API server IP address(es) and the necessary proxy exclusions for the `cbcontainers-dataplane` namespace. These determined values are automatically appended to the `noProxy` values or the specified `NO_PROXY` environment variable for a particular component. To change those pre-determined values, you can specify the `noProxySuffix` key at the same level as the `noProxy` key. It has the same format as the `noProxy` key and its values are treated as if they were pre-determined. You can also force nothing to be appended to `noProxy` or `NO_PROXY` by setting `noProxySuffix` to an empty string.

Configure HTTP Proxy Per-Component Environment Variables

To configure environment variables for the `basic`, `Runtime`, and `Image Scanning` components, update the `CBContainersAgent` CR using the proxy environment variables (`kubectl edit cbcontainersagents.operator.containers.carbonblack.io cbcontainers-agent`):

```

spec:
  components:
    basic:
      enforcer:
        env:
          HTTP_PROXY: "<proxy-url>"
          HTTPS_PROXY: "<proxy-url>"
          NO_PROXY: "<kubernetes-api-server-ip>/<range>"
      stateReporter:
        env:
          HTTP_PROXY: "<proxy-url>"
          HTTPS_PROXY: "<proxy-url>"
          NO_PROXY: "<kubernetes-api-server-ip>/<range>"
      runtimeProtection:
        resolver:
          env:
            HTTP_PROXY: "<proxy-url>"
            HTTPS_PROXY: "<proxy-url>"
            NO_PROXY: "<kubernetes-api-server-ip>/<range>"
        sensor:
          env:
            HTTP_PROXY: "<proxy-url>"
            HTTPS_PROXY: "<proxy-url>"
            NO_PROXY: "<kubernetes-api-server-ip>/<range>,<cbcontainers-runtime-resolver.cbcontainers-dataplane.svc.cluster.local>"
      clusterScanning:
        clusterScanner:
          env:
            HTTP_PROXY: "<proxy-url>"
            HTTPS_PROXY: "<proxy-url>"

```

```

    NO_PROXY: "<kubernetes-api-server-ip>/<range>,cbcontainers-image-scanning-
reporter.cbcontainers-dataplane.svc.cluster.local"
  imageScanningReporter:
    env:
      HTTP_PROXY: "<proxy-url>"
      HTTPS_PROXY: "<proxy-url>"
      NO_PROXY: "<kubernetes-api-server-ip>/<range>"

```

Important You must configure the `NO_PROXY` environment variable to use the value of the Kubernetes API server IP address. To find the API-server IP address, run the following command:

```
kubectl -n default get service kubernetes -o=jsonpath='{..clusterIP}'
```

Additional Proxy Considerations

When using a non-transparent HTTPS proxy, you must configure the agent to use the proxy certificate authority:

```

spec:
  gateways:
    gatewayTLS:
      rootCAsBundle: <Base64 encoded proxy CA>

```

Alternatively, you can allow the agent to communicate without verifying the certificate. We do not recommend this option because it exposes the agent to an MITM attack.

```

spec:
  gateways:
    gatewayTLS:
      insecureSkipVerify: true

```

Changing the Image Source

By default, all images for the Operator and Agent deployments are pulled from Docker Hub. If you prefer to mirror the images in your internal repositories, you can specify the image by modifying the `CBContainersAgent` resource that you apply to your cluster.

Modify the following properties to specify the image for each service:

- `monitor` - `spec.components.basic.monitor.image`
- `enforcer` - `spec.components.basic.enforcer.image`
- `state-reporter` - `spec.components.basic.stateReporter.image`
- `runtime-resolver` - `spec.components.runtimeProtection.resolver.image`
- `runtime-sensor` - `spec.components.runtimeProtection.sensor.image`
- `image-scanning-reporter` - `spec.components.clusterScanning.imageScanningReporter.image`
- `cluster-scanner` - `spec.components.clusterScanning.clusterScanner.image`

The image object consists of four properties:

- `repository` - the repository of the image; for example, `docker.io/my-org/monitor`
- `tag` - the version tag of the image; for example, `1.0.0`, `latest`, and so forth.
- `pullPolicy` - the pull policy for that image; for example, `IfNotPresent`, `Always`, or `Never`. See [Image pull policy](#) (external link).
- `pullSecrets` - the image pull secrets that are going to be used to pull the container images. The secrets must already exist in the cluster. See [Pull an Image from a Private Registry](#) (external link).

Sample configuration:

```
spec:
  monitor:
    image:
      repository: docker.io/my-org/monitor
      tag: 1.0.0
      pullPolicy: Always
      pullSecrets:
        - my-pull-secret
```

In this case, the operator attempts to run the monitor service from the `docker.io/my-org/monitor:1.0.0` container image and the kubelet is instructed to always pull the image by using the `my-pull-secret` secret.

Using a Shared Secret for all Images

To use just one pull secret to pull all the custom images, specify it under `spec.settings.imagePullSecrets`.

The secret is added to the `imagePullSecrets` list of all Agent workloads.

Operator Role-based Access Control

This section describes how to configure and use Carbon Black Container Operator Role-based Access Control (RBAC).

RBAC Definition and Design

Following the principle of least-privilege, any permission given to the Operator should have good reason and be scoped as tightly as possible.

In practice, this means:

- If the resource is namespaced and part of the agent, use a `Role` to give permissions in the agent's namespace only.
- If the resource is namespaced and not part of the agent:
 - To read it, use a `ClusterRole` unless you are sure what the namespace will be.

- To modify it, examine whether this is absolutely necessary.
- If the resource is non-namespaced, use a `ClusterRole` and restrict `delete`, `get`, `update`, and `patch` through `resourceNames`. `Create`, `list`, and `watch` either do not support this restriction or require extra care.

Changing the Operator Access Levels

Operator access level permissions are generated by `controller-gen` and controlled by using `+kubebuilder` directives. See [controller definitions](#) (external link). Any change to those directives requires running `make manifests` to update the respective `role.yaml` file. You must also propagate changes to the helm charts.

Changing the Agent Component Access Levels

Agent component access levels, service accounts, and role bindings are manually maintained in `dataplane_roles.yaml` and the helm equivalent. You must apply changes in both locations.

The roles should follow the least-privilege principle. Agent components often need more permissions than the Operator to work as expected.

Container Operator Developer Instructions

This topic describes instructions using the SDK version 1.29.0 for the Operator.

Deploy the Operator without using an Image

To install dependencies to verify the `kubeconfig` context:

```
make deploy OPERATOR_REPLICAS=0
```

To run the Operator from the terminal to verify the `kubeconfig` context:

```
make run
```

From your editor, run and debug `main.go` to verify the `KUBECONFIG` environment variable.

Install the Dataplane on your own Control Plane

Under the Carbon Black Container Cluster CR:

```
spec:
  apiGatewaySpec:
    adapter: {MY-ADAPTER-NAME}
```

where `{MY-ADAPTER-NAME}` is your control plane adapter name. The default value is `containers`.

Uninstall the Container Operator

From a terminal, run the following command:

```
make undeploy
```

Note This command does not clean up the Carbon Black directory on the dataplane nodes.

Changing Security Context Settings

Hardening enforcer/state_reporter security context settings:

You can change the values under `cbcontainers/state/hardening/objects` for `enforcer_deployment.go` Or `state_reporter_deployment.go`.

Using defaults:

Defaults in the `OpenAPISchema` is a feature in `apiextensions/v1` version of `CustomResourceDefinitions`. These default values are supported by `kubebuilder` by using tags; for example, `kubebuilder:default=something`. For backwards compatibility, all defaults should also be implemented and set in the controllers to make sure that they work on clusters v1.15 and below.

Note `kubebuilder` does not support an empty object as a default value. See [related issue](#) (external link). The root issue is in regard to maps, but the same code causes issues with objects.

Therefore, the following specification will not apply the default for `test` unless the user specifies `bar`.

```
spec:
  properties:
    bar:
      properties:
        test:
          default: 10
          type: integer
```

Applying this YAML will save an empty object for `bar: spec: {}`.

Instead, applying `spec: { bar: {} }` works as expected and saves the following object:

```
spec: { bar: { test: 10 }}
```

For example:

```
spec:
  properties:
    bar:
      default: {}
```



```
properties:
  test:
    default: 10
    type: integer
```

kubebuilder cannot currently produce that output. Therefore, replacing all instance of `<>` with `{ }` so that using `kubebuilder:default=<>` produces the correct output.

Defaulting is not supported by v1beta1 versions of CRD.

Local Debugging

To debug locally, run `make run-delve`. This command builds and starts a delve debugger in headless mode. Then use an editor to start a remote session and connect to the delve instance.

For `goland`, the built-in go remote configuration works.

Custom Namespace

If the Operator is not deployed in the default namespace (`cbcontainers-dataplane`), you must set the `OPERATOR_NAMESPACE` environment variable when using `make run` or `make run-delve`.

Helm Charts

This topic describes the official Helm charts for installing the Carbon Black Container Agent (Operator, CRD, and Agent components).

cbcontainers-operator

The [cbcontainer-operator](#) chart (external link) is the official Helm chart for installing the Carbon Black Container Operator and CRD. Helm 3 is supported.

You can install the chart without any customizations or modifications, and you can create the Helm release in any namespace. You can customize the namespace in which the Operator is installed.

To install the Helm chart from the source:

```
cd charts/cbcontainers-operator
helm install cbcontainers-operator ./cbcontainers-operator-chart
```

Table 2-12. Customization

Parameter	Description	Default Value
<code>spec.operator.image.repository</code>	Repository of the Operator image	<code>cbartifactory/octarine-operator</code>
<code>spec.operator.image.version</code>	Version of the Operator image	The latest version of the Operator image
<code>spec.operator.resources</code>	Carbon Black Container Operator resources	<code>{requests: {memory: "64Mi", cpu: "30m"}, limits: {memory: "256Mi", cpu: "200m"}}</code>

Table 2-12. Customization (continued)

Parameter	Description	Default Value
<code>spec.rbacProxy.resources</code>	Kube RBAC proxy resources	{requests: {memory: "64Mi", cpu: "30m"}, limits: {memory: "256Mi", cpu: "200m"}}
<code>spec.operator.environment</code>	Environment variables to be set to the Operator pod	[]

Namespace

By default, the Carbon Black Container Operator is installed in the `cbcontainers-dataplane` namespace.

To change the namespace, set the `operatorNamespace` field in your `values.yaml` file.

The chart automatically creates the namespace. If you do not want to do that (because you have already created the namespace), set the `createOperatorNamespace` field in your `values.yaml` file to `false`.

If the namespace is pre-created, then it must also be labeled properly or the Operator and Agent might not reconcile successfully. The following commands show an example of creating a custom namespace and labeling and installing the operator inside.

```
NAMESPACE=<your_value>
kubectl create namespace $NAMESPACE
kubectl label namespace $NAMESPACE control-plane=operator octarine=ignore
helm install cbcontainers-operator ./cbcontainers-operator-chart --set
createOperatorNamespace=false,operatorNamespace=$NAMESPACE
```

CRD Installation

By default, installing the chart will also create the `CBContainersAgent` CRD.

To manage the CRD in a different way and not install it together with the chart, set the `installCRD` field in your `values.yaml` file to `false`.

HTTP Proxy

To use an HTTP proxy for the communication with the Carbon Black Cloud backend, you must set 3 environment variables. These variables are exposed through the `Values.operator.proxy` parameters in the `values.yaml` file:

- `Values.operator.proxy.http`
- `Values.operator.proxy.https`
- `Values.operator.proxy.noProxy`

See also [Configuring Container Services to use HTTP Proxy](#) .

Templates

The `cbcontainers-operator` chart consists of four [templates](#) (external link).

The [operator.yaml](#) file (external link) contains all resources except for the Operator deployment. It is generated by kustomize. For more info see [config/default_chart](#) (external link).

The `deployment.yaml` file contains the Operator `Deployment` resource. It is derived from [this Kustomize configuration](#). Because it must be configurable through Helm, it is heavily templated. Therefore, it cannot be generated automatically, so it must be maintained by hand. If any changes are made to the [Kustomize configuration](#), they must also be reflected in the `deployment.yaml` file.

The `dataplane_rbac.yaml` and `dataplane_service_accounts` files contain necessary RBAC objects for the Agent to work as expected.

cbcontainers-agent

The [cbcontainer-agent](#) chart (external link) is the official Helm chart for installing the Carbon Black Container Agent components. Helm 3 is supported.

Note Before installing the Agent components, you must install the Operator and the CRD.

Installation

Before you can install the chart, you must configure it. You must provide the following eight required fields:

Parameter	Description
<code>spec.orgKey</code>	Org key of the organization using Carbon Black Cloud
<code>spec.clusterName</code>	Name of the cluster that will be added to Carbon Black Cloud
<code>spec.clusterGroup</code>	The group that the cluster belongs to in Carbon Black Cloud
<code>spec.version</code>	Version of the Agent images
<code>spec.gateways.apiGatewayHost</code>	URL of the Carbon Black Cloud API gateway
<code>spec.gateways.coreEventsGatewayHost</code>	URL of the Carbon Black Cloud core events gateway
<code>spec.gateways.hardeningEventsGatewayHost</code>	URL of the Carbon Black Cloud hardening events gateway
<code>spec.gateways.runtimeEventsGatewayHost</code>	URL of the Carbon Black Cloud runtime events gateway

After setting these required fields in a `values.yaml` file, you can install the chart from source:

```
cd charts/cbcontainers-agent
helm install cbcontainers-agent ./cbcontainers-agent-chart -n cbcontainers-dataplane
```

Customization

The way in which the Carbon Black Container components are installed is highly customizable.

You can set different properties for the components or enable and disable components by using the `spec.components` section of your `values.yaml` file.

For a list of all possible values, see [Custom Resources Definitions](#).

Namespace

The Carbon Black Cloud Containers Agent will run in the same namespace as the deployed Operator. This is by design because only one running agent per cluster is supported. To customize that namespace, see [operator-chart](#) (external link).

The actual namespace where Helm tracks the release (see `--namespace flag`, external link) is not important to the Agent chart, but the recommended approach is to also use the same namespace as the Operator chart.

The `agentNamespace` value is only required if the Agent chart is responsible for deploying the Agent's secret as well. If the secret is pre-created before deploying the agent, then `agentNamespace` has no effect.

Secret Creation

Carbon Black API Key

For the Agent components to function correctly and communicate with the Carbon Black Cloud backend, an access token is required. This token is located in a *secret*. By default, the secret is named `cbcontainers-access-token`, but that name is configurable through the `accessTokenSecretName` property. If that secret does not exist, the Operator will not start any of the Agent components.

To create the secret as part of the chart installation, provide the `accessToken` value to the chart.

Inject this value as part of your pipeline in a secure way: store the secret as plain text in your `values.yaml` file.

To create the secret in an alternative and more secure way, do not set the `accessToken` value: the chart will not create the secret objects.

Important Do not store the token in your source code.

Carbon Black Company Codes

For the agent CNDR component to function correctly and communicate with the Carbon Black Cloud backend, a company code is required. This code is located in a secret. By default, the secret is named `cbcontainers-company-code`, but that name is configurable through the `components.cndr.companyCodeSecretName` property.

If that secret does not exist, the CNDR component will fail.

If you want to create the secret as part of the chart installation, provide the `companyCode` value to the chart.

Inject this value as part of your pipeline in a secure way: store the secret as plain text in your `values.yaml` file.

To create the secret in an alternative and more secure way, do not set the `companyCode` value: the chart will not create the secret objects.

Important Do not store the code in your source code.

Configuring Container Security

3

This section describes the tasks involved in configuring Container security.

Read the following topics next:

- [Kubernetes Scopes](#)
- [Egress Groups](#)
- [Kubernetes Policies](#)
- [Subscribe to Alert Notifications](#)
- [Setting up API Access](#)

Kubernetes Scopes

Kubernetes scopes are groups of Kubernetes resources that share a purpose. For example, clusters are Kubernetes resources that qualify for a scope definition. You can use a scope as a filter or to apply an identical security policy across Kubernetes resources.

Grouping Kubernetes resources in scopes provides a foundation for targeted planning of security policies. You can add and edit scopes, and you can delete scopes that are not attached to a Kubernetes policy.

Default Scope

The default scope is a predefined scope that encompasses all clusters and namespaces. The default scope is called *Any*. The *Any* scope is always available and cannot be deleted. It is the highest scope in the hierarchy of scopes. The scope resolution process searches for the most precise scope definition into which a Kubernetes resource falls to apply the policy. If no more precise scope is found, the policy that is attached to the default scope is considered.

Scopes for the Build Phase

Build Phase refers to defining the container images or Kubernetes objects for scanning or validating with CLI Client commands. You can integrate the commands in a CI/CD pipeline. You can define a scope for all resources in the build phase, for Kubernetes namespaces, or for a particular build step. The build step is a parameter that the CLI Client uses for performing image scanning. See [Setting up CLI Client for Image Scanning](#) and [Chapter 4 Scanning Images](#).

Scopes for the Deploy Phase

Deploy Phase refers to a grouping of Kubernetes workloads that are going to be deployed or are already deployed.

Scopes can overlap by hierarchy from the most general to the most specific according to the following order: *all clusters*, *cluster group*, *cluster*, *namespace*, and *workload*. For workloads that are part of overlapping scopes, the policy attached to the narrowest scope is applied. In that way, a workload resolves to a single policy.

See [Kubernetes Scopes Hierarchy](#).

Example: Examples

Example Scope	Purpose
A cluster group for all production clusters	Filters or assigns a policy for all clusters in the same tier.
One or more Kubernetes clusters	Filter or assigns a policy to different clusters.
Application across clusters by choosing a Kubernetes namespace that is defined on many clusters	Filters or assigns policies to a group of resources forming an application regardless of where they are deployed.

Application Scopes

Application scopes include container images in both build phase and deploy phase. The scope reflects the practice of separating the applications in their own Kubernetes namespaces. If a scope is defined as an application scope, the policy assigned to the scope is applied to all container images in the namespace, regardless of the development phase and regardless of the clusters where this namespace is located. This scope ensures the same hardening criteria while building or deploying the application.

Kubernetes Scopes Hierarchy

The Kubernetes scopes hierarchy is important for the scope resolution process. The scope resolution process finds the most specific scope in which a workload exists, and the scope then defines the policy to apply on the Kubernetes workload.

Scope Resolution for Kubernetes Workloads in Overlapping Scopes

Scopes are overlapping by design, which means that the workloads might belong to several overlapping scopes. However, each Kubernetes workload is associated with a single *policy*. By implementing a scope resolution logic, the system finds the policy that is related to the most specific scope for each workload.

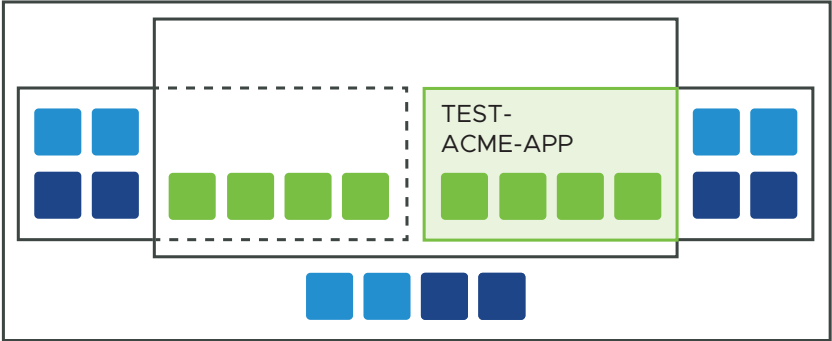
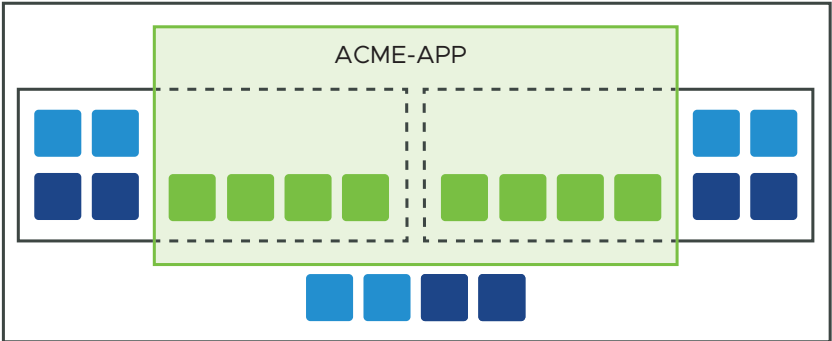
By planning the scopes, you can determine which policy to apply to specific areas in your Kubernetes environment without affecting the rest of the system.

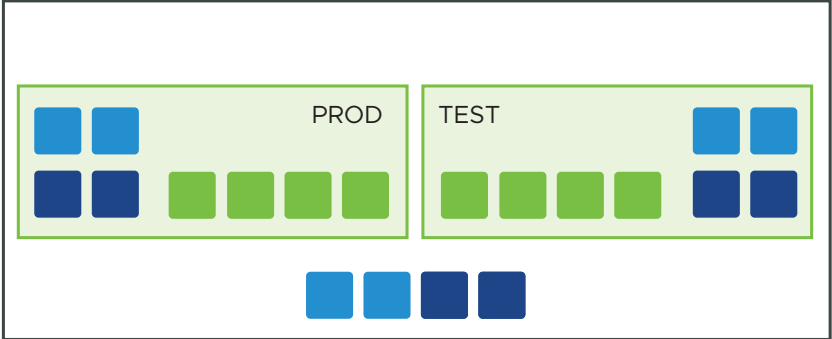
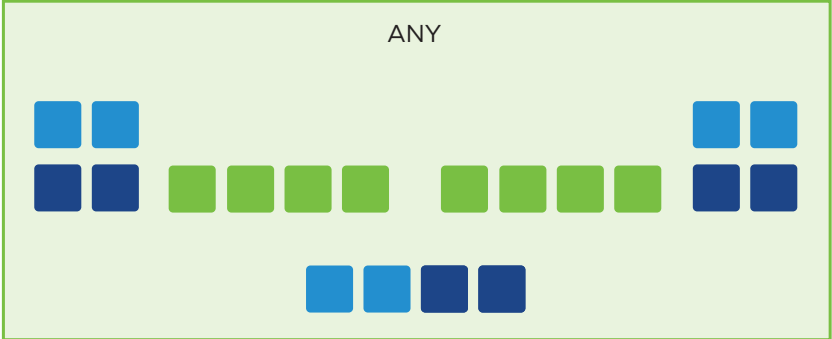
Scope Ranking

Scopes are ranked by specificity. Specific scopes take precedence over general scopes.

The following diagrams rank scopes. The diagrams display boxes in various colors for workloads in cluster groups and namespaces, and a green box for the scope that encompasses them. The most specific scope is at the top of the hierarchy.

Example: Example Illustration of Scopes

Ranking	Description
Resources in specific namespaces in specific clusters	The most specific definition of a scope for using a particular Kubernetes hardening policy.
Resources in specific namespaces in specific cluster groups	Only these particular namespaces inside cluster groups are covered.
Resources in specific clusters	<p>All namespaces in a cluster are covered. Example scope: <i>test-acme-app</i> to test the application in an isolated testing cluster:</p> 
Resources in specific namespaces in any cluster	<p>Application scopes that are defined for a namespace and are valid for all clusters that contain the namespace. Example scope across the Kubernetes environment to cover the namespace: <i>acme-app</i>:</p> 

Ranking	Description
Resources in specific cluster groups	<p>This high-level scope covers groups of clusters. Example two scopes for Production and Testing environments:</p> 
All resources - refer to the Any scope	<p>The default Any scope contains all workloads in the system and overlaps with all other scopes. Scopes for specific Kubernetes resources take precedence over the default scope.</p> 

Built-in Kubernetes Scopes

When you install and set up your Kubernetes clusters, the system includes three ready-to-use scopes: **Kubernetes System**, **CBContainers dataplane**, and **Default Namespace**.

The built-in scopes are assigned to built-in hardening policies. The scopes are available as a starting point for your configuration, and you can either edit or delete them. For more information about built-in hardening policies, see [Built-in Kubernetes Hardening Policies](#).

Pre-Packaged Scope	Scope Target	Scope Description
Kubernetes System	Target: Deploy phase Namespaces: kube-system	Matches the namespace for objects created by the Kubernetes system. This system typically contains services for DNS, proxy, controller manager, and other system components.
CBContainers dataplane	Target: Deploy phase Namespaces: cbcontainers-dataplane octarine-dataplane	Matches the namespace where the Carbon Black Kubernetes agent runs and deploys its resources. Note Two namespaces are listed here. <i>Octarine-dataplane</i> is the namespace name before version 3.0.0 of the agent. <i>Cbcontainers-dataplane</i> is the current namespace name.
Default Namespace	Target: Deploy phase Namespaces: default	Matches Kubernetes built-in default namespace that holds objects that have no specified no namespace.

Note If the built-in scopes are not modified, the **Last modified by** parameter is **Carbon Black**. After you edit a scope, the **Last modified by** parameter changes.

Add a Kubernetes Applications Scope to Kubernetes Resources

You can group Kubernetes resources in a scope. The scope target is *Applications*.

Prerequisites

Set up your Kubernetes clusters. See [Adding Clusters and Installing Kubernetes Sensors](#).

Procedure

- On the left navigation pane, do one of the following depending on your system configuration and role:
 - If you have the Kubernetes Security DevOps or SecOps role and your system has only the Container security feature, click **Inventory > Scopes**.
 - If you have any other role and your system has Container security and other Carbon Black Cloud features, click **Inventory > Kubernetes > Scopes**.
- Click **Add Scope**.
- Enter a **Name** for the scope.
- For target resources, select **Applications**. This scope will target applications in specific namespaces. A policy can be enforced during the build, deployment, and execution phases.
- Click **Next**.
- Select the namespaces from the dropdown menu.

7 Click **Save**.

The scope is ready for use in a Kubernetes Hardening Policy.

What to do next

[Create a Kubernetes Hardening Policy](#)

Add a Kubernetes Deploy Location Scope to Kubernetes Resources

You can group Kubernetes resources in a scope. The scope target is *Deploy Locations*.

Prerequisites

Set up your Kubernetes clusters. See [Adding Clusters and Installing Kubernetes Sensors](#).

Procedure

- 1 On the left navigation pane, do one of the following depending on your system configuration and role:
 - If you have the Kubernetes Security DevOps or SecOps role and your system has only the Container security feature, click **Inventory > Scopes**.
 - If you have any other role and your system has Container security and other Carbon Black Cloud features, click **Inventory > Kubernetes > Scopes**.
- 2 Click **Add Scope**.
- 3 Enter a **Name** for the scope.
- 4 For target resources, select **Deploy Locations**. This scope will target workloads in specific clusters or cluster groups. A policy can be enforced during the deployment and execution phases.
- 5 Click **Next**.

6 Select your scope targets.

Add Scope
✕

SCOPE DEFINITION

After a policy is assigned, workloads in the specified cluster groups or clusters will be subject to content screening, configuration hardening, and behavior analysis.

Cluster groups
 Select

Clusters

A scope can target individual namespaces in the specified cluster groups or clusters; it will take precedence over generic scopes covering the same applications.

Apply only to specific namespaces

Save
Back
Cancel

- You can group by clusters, namespaces, or both.
- To apply the same policy to multiple clusters, use the cluster group as a basis for your scope. You can also select individual clusters instead of a cluster group. A cluster group includes all its existing or future clusters. Thus, cluster group is a broader selection than choosing a list of clusters.
- If you have namespaces with the same name in multiple clusters, the scope you define per namespace will span across clusters for that namespace.
- To determine a particular namespace inside a particular cluster, you can point to a cluster or cluster group and to a specific namespace.

7 Click **Save**.

The scope is ready for use in a Kubernetes Hardening Policy.

What to do next

[Create a Kubernetes Hardening Policy](#)

Add a Kubernetes Container Images Scope to Kubernetes Resources

You can use scopes to enforce policies on container images that are not yet deployed. The scope target is Build Phase.

Prerequisites

Set up your Kubernetes clusters. See [Adding Clusters and Installing Kubernetes Sensors](#).

Procedure

- 1 On the left navigation pane, do one of the following depending on your system configuration and role:
 - If you have the Kubernetes Security DevOps or SecOps role and your system has only the Container security feature, click **Inventory > Scopes**.
 - If you have any other role and your system has Container security and other Carbon Black Cloud features, click **Inventory > Kubernetes > Scopes**.
- 2 Click **Add Scope**.
- 3 Enter a **Name** for the scope.
- 4 For target resources, select **Container images**. This scope will target specific container images. A policy can be enforced during the build phase.
- 5 Click **Next**.
- 6 Select the target criteria from the dropdown menus.

Add Scope
✕

SCOPE DEFINITION [CLI Clients help](#)

Harden images by assigning a policy and configuring CLI instances to perform validation during the build phase

Apply only to specific build steps dev Select build steps

A scope can target images in particular namespaces; it will take precedence over generic scopes covering the same workloads

Apply only to specific namespaces acme-fe Select namespaces

Save
Back
Cancel

Option	Description
Apply only to specific build steps	Harden images by assigning a policy and configuring CLI instances to perform validation during the build phase.
Apply only to specific namespaces	A scope can target images in particular namespaces; it will take precedence over generic scopes covering the same workloads.

7 Click **Save**.

The scope is ready for use in a Kubernetes Hardening Policy.

What to do next

[Create a Kubernetes Hardening Policy](#)

View a Kubernetes Scope

To view a Kubernetes scope, perform the following procedure.

Procedure

- 1 On the left navigation pane, do one of the following depending on your system configuration and role:
 - If you have the Kubernetes Security DevOps or SecOps role and your system has only the Container security feature, click **Inventory > Scopes**.
 - If you have any other role and your system has Container security and other Carbon Black Cloud features, click **Inventory > Kubernetes > Scopes**.

The left pane shows the scopes, attached policies, and the number of workloads affected by each scope.

- 2 In the left pane, select the scope.
 - Click the **General** tab to view the scope details.

Scope Details	
Name	IXDeployScope
Target	Deployment locations
Hardening policy	IX-Hardening
Runtime policy	IX-Runtime
Clusters	default:acme-test, default:rsdemo
Namespaces	--
Workloads	0
Last modified	7:02:55 am Mar 21, 2023
Last modified by	

- If there are policies attached to the scope, you can click a policy name to view that policy summary. For example:

Policy Details
✕

Status	Enabled
Name	eks runtime policy
Scope	eks scope
Last modified	5:41:49 am Feb 14, 2023
Last modified by	

RULE ▼	ACTION
Medium risk malicious destinations ⓘ	Alert
Medium or low risk internal connections ⓘ	Alert
Medium or low risk ingress connections ⓘ	Alert
Medium or low risk egress connections ⓘ	Alert

Close

- You can also view the namespaces and workloads covered by this scope. Click the **Workloads** tab to view namespaces. To view workloads within that namespace, click the namespace.

Edit or Delete a Kubernetes Scope

You can update the configuration of a Kubernetes scope. You can only update the scope name and the included resources. You cannot update the scope target.

Procedure

- 1 On the left navigation pane, do one of the following depending on your system configuration and role:
 - If you have the Kubernetes Security DevOps or SecOps role and your system has only the Container security feature, click **Inventory > Scopes**.
 - If you have any other role and your system has Container security and other Carbon Black Cloud features, click **Inventory > Kubernetes > Scopes**.
- 2 In the left pane, select the scope. On the **General** tab, click **Edit** from the **Options** dropdown menu.

Tip To delete the scope, click **Delete** and then click **OK** to confirm the deletion.

- 3 Click **Next**.
- 4 Modify the name or included resources and click **Save**.

Kubernetes Scope Baselines for Runtime Policies


Kubernetes scope baselines apply to runtime policies. Baseline behaviors reflect the normal activity for all workloads grouped in a scope as discovered during the learning period. The learning period is the time during which all the Kubernetes resources in a scope are monitored for egress network connections. All egress destinations are recorded in the scope baseline.

The scope baseline determines the normal allowed behavior for all Kubernetes resources inside a scope. Deviation from the baseline triggers an alert. The baseline is at scope level, and you can amend or reset the final behavior list.

View a Kubernetes Scope Baseline for a Runtime Policy

To view a Kubernetes scope baseline for a runtime policy, perform the following procedure.

Procedure

- 1 On the left navigation pane, click **Enforce > K8s Policies**.
- 2 Click the **Runtime Policies** tab.
- 3 Select the policy and click the arrow  at the end of the row to open the **Policy Details** panel.
- 4 In the **Policy details** panel, click **View scope baseline**.

Baseline behaviors display in the left pane. You can remove a behavior from the baseline, or you can select a behavior to view additional information in the right pane. For example:

Behavior was learned because the workloads below connected to [\[redacted\]](#)

WORKLOAD	
Name	cbcontainers-hardening-state-reporter
Kind	Deployment
Cluster	[redacted]
Namespace	cbcontainers-dataplane
Name	cbcontainers-hardening-enforcer
Kind	Deployment
Cluster	[redacted]
Namespace	cbcontainers-dataplane

What to do next

You can add a behavior to the scope baseline or reset the scope baseline.

Add a Behavior to a Kubernetes Scope Baseline

You can change the scope baseline for a Kubernetes runtime policy after the completion of the learning period without resetting the learning period and without removing anything from the baseline.

Procedure

- 1 On the left navigation pane, click **Enforce > K8s Policies**.
- 2 Click the **Runtime Policies** tab.
- 3 Select the policy and click the caret **>** at the end of the row to open the **Policy Details** panel.
- 4 In the **Policy details** panel, click **View scope baseline**.
- 5 Click **Add Behavior**.
- 6 Select the type of destination. Enter the public or private domain, subdomain, or IP range, and click **Add**.

Results

You successfully added a destination of egress traffic to the scope baseline.

Add a False Positive as Normal Behavior to the Scope Baseline

You can adjust the scope baseline of Kubernetes runtime policies for alerts that indicate false positive workloads behavior. To do so, you can close alerts or add egress traffic destinations to the scope baseline.

You generally review alerts after you enable or update a Kubernetes runtime policy and after the learning period completes. You can reduce the number of alerts by resolving the issues or by closing the alerts.

Note Closing alerts is only recommended for excluding specific workloads that exhibit known behaviors from the alerts list.

Procedure

- 1 On the left navigation pane, select **Alerts**.
- 2 Locate and select the alerts of interest and do one of the following:
 - On the **Actions** dropdown menu, click **Add to baseline**. Click **OK** to confirm.
 - On the **Actions** dropdown menu, click **Close**.

Close Alert ×

1 alert will be closed on **frontend**

Containers Runtime 7c18a9bb-6c29-34d8-0205-1c6734cd853d
Detected an abnormal internal connection with medium or low risk

Close as Resolved - Benign/Known... ▾

Manage Related Alerts ⓘ

Threat ID b38bbeb12385cd27ad64c85f80b53be7d8809c58cc3ffe9beb9e6c130039c1e4

Close all existing alerts with this threat ID [View 175 alerts](#) ↗

Automatically close all future alerts with this threat ID?

Yes, close all future alerts

No, do not close all future alerts

Note

Close Alert Cancel

- a In the **Close as** dropdown menu, select a reason for closing the alert, for example, **Resolved - Benign/Known**.
- b Optionally select the check box to close all existing alerts that have the same threat ID.
- c Optionally automatically close all future alerts that have this threat ID.
- d Enter an optional note about the reason for closing the alert.
- e Click **Close Alert**.


Reset a Kubernetes Scope Baseline

To reset a Kubernetes scope baseline, perform the following procedure.

Resetting the baseline is valuable when an image has changed and the new behavior differs from the previously learned behavior.

Procedure

- 1 On the left navigation pane, click **Enforce > K8s Policies**.
- 2 Click the **Runtime Policies** tab.

- 3 Locate the policy that is attached to the scope and click the arrow  icon at the end of the row.
- 4 Click **Reset**.

Results

The scope baseline and the policy learning period are reset.

Egress Groups

Egress groups organize the presentation of egress traffic from your cluster on the network map. You define egress groups for your clusters based on domains and IP addresses.

There are two default egress groups: public and private. Public egress contains the traffic that goes outside of your network. Private egress comprises the egress traffic that uses a private address space of IP addresses.

Note If a destination classifies for two or more egress groups, the traffic appears under the most specific egress group.

Create an Egress Group

To define an egress group, perform the following procedure.

Procedure

- 1 On the left navigation pane, do one of the following depending on your system configuration and role:
 - If you have the Kubernetes Security DevOps or SecOps role and your system has only theContainers security feature, click **Inventory > Network**.
 - If you have any other role and your system has Container security and other Carbon Black Cloud features, click **Inventory > Kubernetes > Network**.
- 2 Click the **Egress Groups** tab and then click **Add Group**.
- 3 Enter the **Name** and **Description** for the group.
- 4 Define the **Destination subnet and domains** for the group. Configure the destination as a set of rules with logical AND operator. The possible options to configure are the following.
 - a **DNS domain name** — exact match of the domain name
 - b **DNS domain name and subdomains** — all domain names containing the subdomain suffix
 - c **IP range** — classless inter-domain routing (CIDR) using subnet masks or IPv6 notation

Example:

Add Egress Group

✕

*** Name**

Description

Destination subnets and domains

Domain ▼	vmware.com	-
Domain and subdomains ▼	*.vmware.com	-
IP range (CIDR) ▼	[blurred]	- +



Save
Cancel

5 Click **Save**.

Edit or Delete an Egress Group

To edit or delete an egress group, perform the following procedure.

Procedure

- 1 On the left navigation pane, do one of the following depending on your system configuration and role:
 - If you have the Kubernetes Security DevOps or SecOps role and your system has only the Container security feature, click **Inventory > Network**.
 - If you have any other role and your system has Container security and other Carbon Black Cloud features, click **Inventory > Kubernetes > Network**.
- 2 Click the **Egress Groups** tab.
- 3 Select the egress group to edit or delete.
 - To edit the egress group, click the **Edit**  icon. Update the group configuration and click **Save**.
 - To delete the egress group, click the trashcan icon ; then click **Delete** to confirm.

Kubernetes Policies

Kubernetes policies in Carbon Black Cloud group security rules into policies to help harden the Kubernetes environment.

Carbon Black Container Kubernetes policies are defined by the type of environment they protect — *runtime* or *hardening*. Each Kubernetes policy binds to a particular Kubernetes scope, and each scope is assigned to a single policy. A runtime policy and a hardening policy can share a common scope. This architecture helps track the root of a policy violation.

Note When Kubernetes policies are referenced without specifying type, the reference is to both types of policy.

Kubernetes Runtime Policies

Kubernetes runtime policies are groups of rules that monitor behavior and changes in the Kubernetes environment related to egress traffic, threats, and anomalies. Kubernetes runtime policies define the allowed behavior while the Kubernetes workloads are running.

See [Runtime Policies Concepts and Terminology](#).

Create a Kubernetes Runtime Policy

To create a Kubernetes runtime policy, perform the following procedure.

Prerequisites

All prerequisites are optional.

- Read [Runtime Policies Concepts and Terminology](#).
- Create a Kubernetes scope to link to the Kubernetes runtime policy. To create a Kubernetes scope, see [Kubernetes Scopes](#). If you do not create the scope in advance, you can do so when you create the Kubernetes runtime policy.

Procedure

- 1 On the left navigation pane, click **Enforce > K8s Policies**.
- 2 Click the **Runtime Policies** tab.
- 3 Click **Add Policy**.
- 4 On the **Define Policy** page, name the policy, select the scope from the list of available scopes, and click **Next**.

Note If you have not configured a scope for use with this policy, click **Add Scope**. For detailed instructions, see [Add a Kubernetes Applications Scope to Kubernetes Resources](#).


- On the **Add Rules** page, select the rules to include in the policy.

You can add rules from the **Basic**, **Moderate**, and **Strict** templates. For more information about these templates, see [Kubernetes Policy Templates](#).

Important Carbon Black recommends that you start with the rules from the **Basic** template to provide alerts for issues that have the highest severity.

For example, to add all rules from the **Basic** template:

- Select the **Basic** rule template on the left.
- Select the type of alerting action (**Monitor** or **Alert**) at the top right. **Alert** is the default action.
- Click **Add all 5 rules** at the top right.

You can add individual rules from templates instead of adding rules in bulk. To do so, click the arrow  icon at the right of the rule.

After you have added rules, they display in the right pane of the page. From here, you can remove individual rules or all rules.

Note You can create your own templates. See [Create a Kubernetes Policy Template](#).

- Click **Next**.
- Review the policy settings. Set the learning period for the scope baseline. The default value is 7 days. To see the progress of the scope baseline during the learning period, see [View a Kubernetes Scope Baseline for a Runtime Policy](#).
 - Click **Enable Policy** to create and activate the policy.

- Click **Save as Draft** to save the policy in a draft state. In this case, Carbon Black Cloud saves the policy as **Disabled**. You can edit and enable the policy. See [Edit a Kubernetes Runtime Policy](#) and [Enable a Kubernetes Runtime Policy Draft](#).

What to do next

After you configure your Kubernetes runtime policies and after the learning period ends, the behavioral baseline is established, and protection is active. All alerts that are caused by violations of the runtime policies display on the **Alerts** page. See [Triaging Kubernetes Alerts](#).

Edit a Kubernetes Runtime Policy

To edit a Kubernetes runtime policy, perform the following procedure.

Procedure

- 1 On the left navigation pane, click **Enforce > K8s Policies**.
- 2 Click the **Runtime Policies** tab.
- 3 Select the policy to edit and click **Edit policy** in the **Actions** dropdown menu.

Note For more details about fields and rules in a runtime policy, see [Kubernetes Runtime Policies](#) and [Create a Kubernetes Runtime Policy](#).

- a Change the scope to which the policy is linked and click **Next**.
- b Add or remove rules as necessary and click **Next**.
- c Adjust the learning period if necessary and click **Save**.

Enable a Kubernetes Runtime Policy Draft

You can enable a Kubernetes policy that is **Disabled**. Policies in a **Disabled** state have been saved as a draft during creation.

Procedure

- 1 On the left navigation pane, click **Enforce > K8s Policies**.
- 2 Click the **Runtime Policies** tab.
- 3 Select the policy that has a **Disabled** status and click **Enable policy** in the **Actions** dropdown menu.


Results

The policy is immediately enabled.

View Kubernetes Runtime Policy Details

To view Kubernetes runtime policy details, perform the following procedure.

Procedure

- 1 On the left navigation pane, click **Enforce > K8s Policies**.
- 2 Click the **Runtime Policies** tab.
- 3 Select the policy to view and click the arrow  icon at the right of the rule.

POLICY DETAILS ▼

Status	Enabled
Name	IX-Runtime
Scope	IX-Scope
Last modified	6:26:43 am Mar 30, 2023
Last modified by	

RULES

Baseline behaviors are being learned. Egress rules will be enabled in 8 minutes.


[View scope baseline](#)

RULE ▼	ACTION
Internal port scan ⓘ	Alert
High risk malicious destinations ⓘ	Alert
High risk egress connections ⓘ	Alert
Egress port scan ⓘ	Alert

4 You can view the following details:

- **Status, Name, Scope, Last modified date, and Last modified by** data. To view additional scope details, click the **Scope** name.

Scope Details ✕

Name	IX-Scope
Target	Deployment locations
Hardening policy	--
Runtime policy	IX-Runtime
Cluster groups	default
Namespaces	--
Workloads	0
Last modified	5:18:19 am Mar 30, 2023
Last modified by	

[Close](#)

- Rules status (if in learning mode)

- Rules and their actions
- Scope baseline. Click **View scope baseline** to view and manage the baseline. See [Kubernetes Scope Baselines for Runtime Policies](#).

Kubernetes Hardening Policies

Kubernetes hardening policies combine predefined and user-defined policy rules that describe the target configuration of Kubernetes resources. Kubernetes hardening policies assure the security of the workloads configuration.

See [Hardening Policies Terminology and Concepts](#).

Built-in Kubernetes Hardening Policies

When you install and set up your Kubernetes clusters, the system includes two ready-to-use policies: **Kube system** and **CBContainers dataplane**.

The built-in policies are associated with built-in scopes. For more information about built-in scopes, see [Built-in Kubernetes Scopes](#).

The policies are available as a starting point for your configuration, and you can either edit or delete them.

Tip You can duplicate the policies and modify the duplicates, thereby maintaining the original policies for reference.

Built-in Policy	Assigned Scope
Kube system	Kubernetes System
CBContainers dataplane	CBContainers dataplane

As long as the built-in policies are not modified, the **Last modified by** parameter is **Carbon Black**. After you edit a policy, the **Last modified by** parameter changes.

The built-in policies include a subset of the built-in rules that are available for use in all Kubernetes hardening policies.

Create a Kubernetes Hardening Policy

You can create Kubernetes hardening policies to enforce rules on your Kubernetes workloads and container images.

Prerequisites

All prerequisites are optional.

- Read [Hardening Policies Terminology and Concepts](#).
- Create a Kubernetes scope to link to the Kubernetes hardening policy. See [Add a Kubernetes Applications Scope to Kubernetes Resources](#). If you do not create the scope beforehand, you can perform this task when you create the Kubernetes hardening policy.

- To use a custom rule in a Kubernetes hardening policy, you must create the custom rule before you create the hardening policy. See [Custom Rules for Kubernetes Hardening Policies](#).
- Create custom rule templates to apply to new policies. See [Create a Kubernetes Policy Template](#).
- To apply the **Enforce** action to a rule, you must add an enforcement preset. See [Enforcement Presets](#).

Procedure

- 1 On the left navigation pane, click **Enforce > K8s Policies**.
- 2 Click the **Hardening Policies** tab.
- 3 Click **Add Policy**.
- 4 On the **Define Policy** page:
 - a Name the policy.
 - b Select the scope from the list of available scopes or click **Add Scope** to configure a new scope for use with this policy. See [Add a Kubernetes Applications Scope to Kubernetes Resources](#).
 - c To enable init containers, select the **Include init containers** text box.

Init containers are special containers that run before app containers in a Kubernetes pod. Init containers can contain utilities or setup scripts that are not present in an application image. Init containers often have more privileges, but a shorter life span. They may have less impact on the overall security of your clusters.
 - d Ephemeral containers are selected by default.

Ephemeral containers are a special type of container that are useful for debugging within pods. If you do not want ephemeral containers associated with this policy, deselect the **Include ephemeral containers** check box. For more information about ephemeral containers, see [Ephemeral Containers](#).
 - e Click **Next**.
- 5 On the **Add Rules** page, select the rules to include in the policy.
 - You can add all rules in a category or all rules from a template. All rules have the **Alert** action by default. You can reset the action to **Block** or **Enforce**.

Important

- Enforcement rules do not operate on the `kube-system` namespace. In that namespace, they act as blocking rules to prevent unexpected changes to critical system resources.
 - When required, include a defined or add an enforcement preset for the **Enforce** action. The Enforcement preset drop-down menu displays if the rule requires user input. See [Enforcement Presets](#).
-

- You can add individual rules from templates instead of adding rules in bulk. To do so, click the arrow > icon at the right of the rule.
- After you have added rules, they display in the right pane of the page. From there, you can remove individual rules or all rules.

6 Click **Next**.

7 On the **Review Violations** page, review the possible violations for which notifications are sent after you enable the policy.

Review Violations ?

RULE ▲	ACTION	VIOLATIONS ▼	EXCEPTIONS	
Access to host namespace ⓘ	Alert	0	No	<input checked="" type="checkbox"/> On
Additional capabilities ⓘ	Alert	0	No	<input checked="" type="checkbox"/> On
Allow privilege escalation ⓘ	Alert	0	No	<input checked="" type="checkbox"/> On
Allow privileged container ⓘ	Enforce	0	No	<input checked="" type="checkbox"/> On
Cluster role binding ⓘ	Block	0	No	<input checked="" type="checkbox"/> On
SecComp profile ⓘ	Alert	0	No	<input checked="" type="checkbox"/> On

Violations | Exceptions

No violations for selected rule

Note You can create exceptions: click the **Exceptions** tab and then click **Add Criteria**. See [Create an Exception for a Kubernetes Hardening Policy Rule](#).

8 Toggle rules **On** or **Off** to define the rules that are currently active in the hardening policy.

9 Click **Next**.

10 On the **Confirm Policy** page, click **Enable Policy**.

- Click **Enable Policy** to create and activate the policy.
- Click **Save as Draft** to save the policy in a draft state. In this case, Carbon Black Cloud saves the policy as **Disabled**. You can edit and enable the policy. See [Edit a Kubernetes Hardening Policy](#) and [Enable a Kubernetes Hardening Policy Draft](#).

What to do next

After you configure your Kubernetes hardening policies, you can observe rule violations on the **Workload Details** pane of the Kubernetes Workloads page.

Enforcement Presets

Carbon Black lets you enforce actions on resources by creating rule enforcement presets. The presets are pre-defined requirements that enforce specific fields and values by automatically mutating resources that deviate from your organizational standards.

As a DevSecOps, you can take control of your environment and reduce the number of violations by enforcing rules instead of changing the configuration sets for existing resources to meet company-introduced requirements.

Assign an Enforcement Preset to a Kubernetes Hardening Policy

To add an enforcement preset to a Kubernetes hardening policy, perform the following procedure.

Note This procedure uses the **Hardening Policies** tab in the **Enforce > K8s Policies** page. You can alternatively assign an enforcement preset to a rule on the **Rules** tab.

Procedure

- 1 On the left navigation pane, click **Enforce > K8s Policies**.
- 2 Click the **Hardening Policies** tab.
- 3 Click a policy name to edit it or add a new policy. See [Create a Kubernetes Custom Rule for Container Images](#) and [Edit a Kubernetes Hardening Policy](#).
- 4 Click **Next**.
- 5 On the Add Rules page, locate a rule that has an **Enforce** option and select **Enforce**.
The **Enforcement preset** dropdown menu displays if the rule requires user input.
- 6 To assign a preset to the rule, do one of the following:
 - Select an existing preset from the **Enforcement preset** dropdown menu.
 - Click **Add new preset** to create a new preset.

- 7 To create a new preset, click **Add new preset**.
 - a Enter a name for the preset and select the rule-specific fields from the **Field** dropdown menu.
 - b Select an action from the **Action** dropdown menu and enter the enforce value.

To add more fields, click the plus + icon.

Add Enforcement Preset
✕

Automate rule compliance by enforcing predefined fields and values.
If non-conforming resources are found, they'll be mutated to match the enforcement preset.

*** Name**

CPU Limits Preset

CPU limits Enforcements

Field	Action	Value	
spec.containers[*].resources.limits.cpu	Enforce value	500	+

Save
Cancel

- c Click **Save**.
The newly defined preset displays in the **Enforcement preset** dropdown menu.

- 8 To add the rule to the policy, click the caret > icon to the right of the rule.
- 9 Click **Next**.

The modified rule appears in the **Review Violations** section and the rule enforcement preset name is available in the **Action** column.

Review Violations ⓘ

RULE ▲	ACTION	VIOLATIONS ▼	EXCEPTIONS	
Access to host namespace ⓘ	Alert	0	No	<input checked="" type="checkbox"/>
Additional capabilities ⓘ	Alert	0	No	<input checked="" type="checkbox"/>
Allow privilege escalation ⓘ	Alert	0	No	<input checked="" type="checkbox"/>
Allow privileged container ⓘ	Enforce	0	No	<input checked="" type="checkbox"/>
Cluster role binding ⓘ	Block	0	No	<input checked="" type="checkbox"/>
CPU limits ⓘ	Enforce CPU Limits Preset	0	No	<input checked="" type="checkbox"/>
SecComp profile ⓘ	Alert	0	No	<input checked="" type="checkbox"/>

When a new resource deploys, the system uses the predefined fields for enforcement.

10 Click **Save**.

Add or Delete an Enforcement Preset

To add or delete an enforcement preset, perform the following procedure.

Note You cannot delete a preset that is currently in use.

Procedure

- 1 On the left navigation pane, click **Enforce > K8s Policies**.
- 2 Click the **Rules** tab.
- 3 Locate and double-click the rule that has enforcement presets.
- 4 In the **Rule Details** pane to the right, click the **Enforcement Presets** dropdown menu.
All available presets for this rule display.
- 5 Locate an enforcement preset, click the dropdown menu, and select an action.
 - To update the value fields of the preset, select **Edit** and save your changes.
 - To delete the preset, select **Delete** and confirm your action.

Note If the preset is used in a policy, the dropdown menu is deactivated.

Results

After editing the preset, existing workloads are not changed until they are re-deployed in the environment.

Edit a Kubernetes Hardening Policy

To edit a Kubernetes hardening policy, perform the following procedure.

Procedure

- 1 On the left navigation pane, click **Enforce > K8s Policies**.
- 2 Click the **Hardening Policies** tab.

- 3 Click the policy name to edit it or click **Edit policy** in the **Actions** dropdown menu.

Note For more details about fields and rules in a runtime policy, see [Kubernetes Runtime Policies](#) and [Create a Kubernetes Runtime Policy](#).

- a Change the scope to which the policy is linked and click **Next**.
- b Add or remove rules as necessary and click **Next**.

Note To modify or add an enforcement preset, see [Assign an Enforcement Preset to a Kubernetes Hardening Policy](#).

- c Confirm the policy details and click **Save**.

Note You can deactivate a rule if it triggers too many violations until the issues in your environment are resolved. To exclude the rule from the policy, toggle the state of the rule to `off`.

Enable a Kubernetes Hardening Policy Draft

You can enable a Kubernetes policy that is **Disabled**. Policies in a **Disabled** state have been saved as a draft during creation.

Procedure

- 1 On the left navigation pane, click **Enforce > K8s Policies**.
- 2 Click the **Hardening Policies** tab.
- 3 Select the policy that has a **Disabled** status and click **Enable policy** in the **Actions** dropdown menu.

Results

The policy is immediately enabled.

Save a Hardening Policy as a Template

To save the rules from a Kubernetes hardening policy for use in other policies, save a policy as a template.

Procedure

- 1 On the left navigation pane, click **Enforce > K8s Policies**.
- 2 Click the **Hardening Policies** tab.
- 3 For the selected policy, click **Save as template** in the **Actions** dropdown menu.
- 4 Enter name for the new template and click **Save**.

Results

The newly created template is saved. The **Templates** tab is displayed on the Kubernetes Policies page, with focus on the new template. See [Kubernetes Policy Templates](#).

Duplicate a Hardening Policy

To use the same rules configuration of a Kubernetes hardening policy for another scope, you can duplicate the policy.

Procedure

- 1 On the left navigation pane, click **Enforce > K8s Policies**.
- 2 Click the **Hardening Policies** tab.
- 3 For the selected policy, click **Duplicate** in the **Actions** dropdown menu.
- 4 Enter name for the new template and click **Save**.

The wizard for creating a new policy populates with all the data from the original policy.

- 5 Modify the policy name and scope and save the duplicated policy.

Kubernetes Policy Rules

Rules are the primary components of Kubernetes policies. Rules are applied on Kubernetes resources. You can use predefined rules or create custom ones.

- Built-in rules are based on the Kubernetes security configuration. They are divided into categories and used in predefined templates.
- Custom rules are user-defined rules for Kubernetes workloads or container images. If you update a custom rule, the change impacts all policies in which the rule is applied.

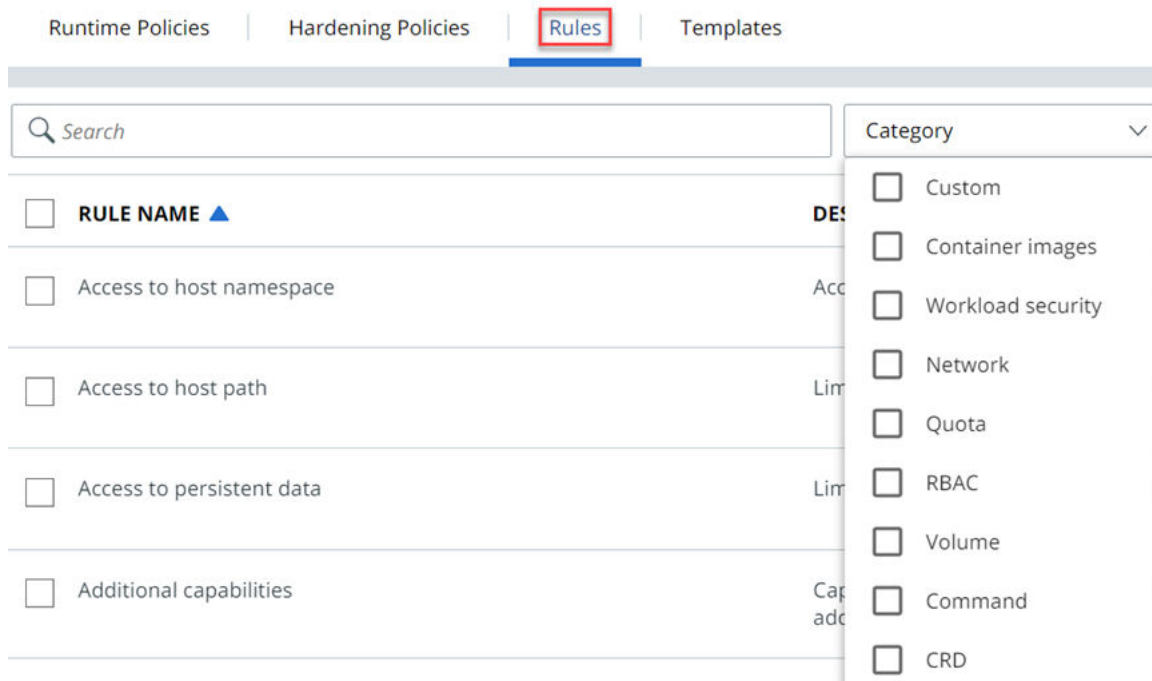
View Hardening Policy Rules

To view existing hardening policy rules, perform the following procedure.

Procedure

- 1 On the left navigation pane, click **Enforce > K8s Policies**.
- 2 Click the **Rules** tab.

You can filter the list of rules by category.

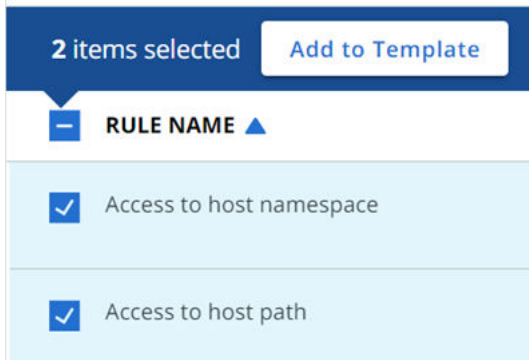


Add Hardening Rules to a Template

To add hardening policy rules to a template, perform the following procedure.

Procedure

- 1 On the left navigation pane, click **Enforce > K8s Policies**.
- 2 Click the **Rules** tab.
- 3 Select the rules to add to the template.



- 4 Click **Add to Template** and select the template from the dropdown list.
- 5 Click **Save**.

Built-in Kubernetes Policy Rules

This topic lists the built-in rules for Kubernetes hardening policies in alphabetical order.

Built-in Rules

Rule Name	Description	Category
Access to host namespace	Access to the host's network, PID, and IPC namespace.	Workload Security
Access to host path	Limits usage of host directory at the container.	Volume
Access to persistent data	Limits use of non-core volume types to those defined through PersistentVolumes.	Volume
Additional capabilities	Capabilities turn the binary "root/non-root" dichotomy into a fine-grained access control system. This rule helps to enforce the capabilities being added when running containers.	Workload Security
Allow privilege escalation	AllowPrivilegeEscalation controls whether a process can gain more privileges than its parent process.	Workload Security
Allow privileged container	Runs container in privileged mode. Processes in privileged containers are essentially equivalent to root on the host.	Workload Security
AppArmor	AppArmor (Application Armor) is a Linux security module that protects an operating system and its applications from security threats. To use it, a system administrator associates an AppArmor security profile with each program.	Workload Security
Cluster role binding	Binds a user or service account to a role in a cluster and all its namespaces.	RBAC
Company banned list	Prevents deployment of images with company banned files.	Container Images
CPU limits	Distributes CPU across workloads and ensures that a single container cannot bring the system down by exhausting resources.	Quota
Critical vulnerabilities	Prevents deployment of images with critical vulnerabilities in OS packages or libraries.	Container Images
Deny ephemeral containers	Ephemeral containers help debug workloads with limited tool sets or access by running an ad-hoc container within the pod context. While powerful for an admin, ephemeral containers can be maliciously used by adversaries to gain privileged access to workloads.	Command
Deny latest tag	Identifies container images with a "latest" tag. Latest tags make it difficult to track image versions and roll back properly.	Container Images
Deny new resources	Identify the deployment of new resources in the associated scope.	Workload Security
Deploy new CRD	Extends Kubernetes resources by customizing a particular Kubernetes installation. Once a custom resource is installed, users can create and access its objects using kubectl.	CRD
Enforce not root	Containers should be prevented from running with a root primary or supplementary GID. Specifying the user/group ID for the container or setting runAsNonRoot to true should indicate the container must run as a non-root user or group.	Workload Security
Exec to container	Kubectl exec allows a user to execute a command in a container. Attackers with permissions could run 'kubectl exec' to execute malicious code and compromise resources within a cluster.	Command
Host port	Allows workloads to be exposed by a host port.	Network

Rule Name	Description	Category
Image not scanned	Identifies workloads with images that have not been scanned within 20 minutes of deployment.	Container Images
Ingress controller	Allows workloads to be exposed by an ingress controller.	Network
Known malware	Prevents deployment of images with known malware.	Container Images
Load balancer	Allows workloads to be exposed by a load balancer.	Network
Memory limits	Distributes memory across workloads and ensures that a single container cannot bring the system down by exhausting resources.	Quota
Node port	Allows workloads to be exposed by a node port.	Network
Port forward	Kubectl port-forward allows you to bypass the cluster's perimeter security and interact directly with internal Kubernetes cluster processes from your localhost.	Command
Require hash tags	Identify container images with named tags. Hash tags are required to prevent issues with overwritten named tags	Container Images
Role binding	Binds a user or service account to a role in a namespace.	RBAC
SecComp profile	The seccomp options to be used by this container. If seccomp options are provided at both the pod and container level, the container options override the pod options.	Workload Security
Secret found	Prevents deployment of images that have secrets.	Container Images
SELinux	The SELinux context to be applied to the container. If unspecified, the container runtime will allocate a random SELinux context for each container.	Workload Security
Sysctl	Sysctls holds a list of namespaced sysctls used for the pod. Pods with unsupported sysctls (by the container runtime) might fail to launch.	Workload Security
Unmasked proc mount	ProcMount indicates the type of proc mount to use for containers. By default, it uses the container runtime defaults for read-only paths and masked paths.	Workload Security
Vulnerabilities with fixes	Prevents deployment of images with medium, high, or critical vulnerabilities—if fixes are available.	Container Images
Writable file system	Allows files to be written to the system, which makes it easier for threats to be introduced and persist in your environment.	Workload Security

Built-in Rules Specification

Note Due to the width of the Built-in Rules Specification table, it can only be viewed in HTML. See **Built-in Rules Specification** in [Built-in Policy Rules](#).

Custom Rules for Kubernetes Hardening Policies

Use the concepts and procedures in this section to create custom rules for Kubernetes hardening policies.

Each rule type is described in a separate topic. The common characteristics are as follows:

Characteristic	Description
Name	The name of the rule must be unique
Description	<p>Short description of the rule. This information displays in several places in the Carbon Black Cloud console:</p> <ul style="list-style-type: none"> ■ Enforce > K8s Policies > Rules ■ Enforce > K8s Policies > Templates ■ Enforce > K8s Policies > Hardening Policies > Add Policy > Review Violations

Basic JSONPath Rules

The JSONPath option for adding custom rules is a guided configuration of a Manageable Access-Control Policy Language (MAPL) rule that has limited capabilities. MAPL is a language for rules that controls access in a microservices environment. Use this kind of rule to define the desired state of your Kubernetes resources.

[JSONPath](#) custom rules can contain multiple conditions that are linked with logical operands. Conditions include a Kubernetes resource — **Resource Kind** — that is connected to an expected valued.

You can configure a basic JSONPath custom rule using the guided configuration in the console.

Characteristic	Description
Resource kind	Type of Kubernetes resource to which the rule refers.
JSONPath	<p>The JSONPath selector is used to get to a certain setting and specify its value in the configuration file of a Kubernetes resource.</p> <p>Note You must start the JSONPath selector string with the \$ sign.</p> <p>A custom rule can have multiple JSONPath criteria that use AND logic to match individual resources. JSONPath is a way to represent an element or a selection of elements in a JSON or YAML file. A jsonpath expression is built as a tree:</p> <pre>{.element} {.child} {.grand-child}</pre> <p>A jsonpath expression starts with a dot (.) to start matching from the root of the configuration, followed by the name of a child, then grandchild, and so on.</p> <p>Use <code>[:]</code> to match any element inside an array, such as any label name inside <code>\$.metadata.labels</code>. For example: <code>\$.metadata.labels[:].name*</code>.</p>

Characteristic	Description
Method	<p>The method to evaluate the resource value:</p> <ul style="list-style-type: none"> ■ EQ - equal ■ NE- not equal ■ RE - match a regular expression ■ NRE - does not match a regular expression ■ LT - lower than ■ LE - lower or equal than ■ GT - greater than ■ GE - greater or equal than ■ EX - exists ■ NEX - does not exist ■ IN - in list of values [val1,val2,val3,...] ■ NIN - not in list of values [val1,val2,val3,...]
Value	The threshold value to match the resource value. If the value is not matched, the rule is violated.

Example: Example JSON

```
{
  "apiVersion": "v1",
  "kind": "Namespace",
  "metadata": {
    "creationTimestamp": "2021-04-09T00:52:44Z",
    "managedFields": [
      {
        "apiVersion": "v1",
        "fieldsType": "FieldsV1",
        "fieldsV1": {
          "f:status": {
            "f:phase": {}
          }
        }
      }, ...
    ]
  }
}
```

Example: Example Custom Rule 1

Do not allow workloads that have more than 5 replicas:

```
$.spec.replicas GT 5
```

Example: Example Custom Rule 2

Requires presence of CPU quotas for all containers:

```
$.spec.template.spec.containers[:].resources.limits.cpu NEX
```

Example: Example Custom Rules 3 and 4

Requires each workload to have a label named `serviceOwner` and a value that looks like an email address (2 rules):

- `$.spec.template.metadata.label.serviceOwner NEX`

```
■ $.spec.template.metadata.label.serviceOwner NRE .+@example\.com
```

Create a JSONPath Kubernetes Custom Rule

The Carbon Black Cloud console provides some optional steps for creating and validating JSONPath criteria.

To build a correct JSONPath selector, you can enter a sample resource configuration or import the configuration of an already deployed resource in your Kubernetes environment. Based on this configuration, the Carbon Black Cloud console displays a preview of the selector's result; you can then build the selector.


Prerequisites

See [Basic JSONPath Rules](#).

Procedure

- 1 On the left navigation pane, click **Enforce > K8s Policies**.
- 2 Click the **Rules** tab.
- 3 Click **Add Rule**.
- 4 Define the rule.
 - a Enter a unique custom rule name and a description.
 - b Select **JSONPath, methods, values** as the rule criteria.
 - c Click **Next**.
- 5 Optionally enter the **Resource kind** from the dropdown menu. The default value is **Any**.
- 6 Click **Import** to open an existing resource file from your Kubernetes environment. You can also copy/paste your content into the **Sample resource JSON** text box.

The resource file or copied content displays in the **Sample resource JSON** text box to the left of the page.

- 7 In **JSONPath**, enter a string (that you can copy from the displayed JSON file), and click the  icon to the right of the text box.
- 8 Enter a **Method** from the dropdown menu and type in a **Value**.


- 9 Preview your selection in the **Results for JSONPath** area on the right of the page. If the string you entered is not returning any resources, a message displays to that effect. If you see a number, for example, [1], there is one matching resource.

The screenshot shows the 'Configure Rule' page in a web interface. At the top, there are three steps: 1. DEFINE RULE (with a green checkmark), 2. CONFIGURE RULE (highlighted with a '2' in a box), and 3. CONFIRM RULE (highlighted with a '3' in a box). Below the steps, the page is titled 'Configure Rule' and includes the instruction: 'Provide resource and JSON criteria. Including a sample JSON is optional.'

The 'Resource kind' is set to 'All kinds'. The 'Sample resource JSON' is displayed in a code editor with line numbers 22-41. A red box highlights the 'roleRef' object in the JSON, and a red arrow points from this box to the 'Results for JSONPath "\$roleRef"' area on the right. The results area shows a single matching resource: an array containing one object with 'apiGroup': 'rbac.authorization.k8s.io', 'kind': 'Role', and 'name': 'tkg-metadata-reader'.

At the bottom, the 'JSONPath' field contains '\$roleRef', the 'Method' is set to 'GET', and the 'Value' field is empty. A red arrow points from the 'roleRef' field to a search icon. Below the fields are 'Back', 'Cancel', and 'Next' buttons.

- 10 Click **Next**.
- 11 On the **Confirm Rule** page, review the summary of the rule criteria and the matching Kubernetes resources and click **Save**.

The custom rule is added to the Rules page. To review its details, click the arrow  icon at the right of the rule.

Create a Kubernetes Custom Rule for Container Images

You can create a custom rule for container images that is based on built-in rules.

Procedure

- 1 On the left navigation pane, click **Enforce > K8s Policies**.
- 2 Click the **Rules** tab.
- 3 Click **Add Rule**.

- 4 Define the rule.
 - a Enter a unique custom rule name and a description.
 - b Select **Container image criteria** as the rule criteria.
 - c Click **Next**.
- 5 Configure the rule. Options are:

Image Criteria	Vulnerability Severity or Registry Domains
<p>Critical vulnerabilities</p> <p>Note The vulnerabilities with Critical severity are part of the default Critical vulnerabilities built-in rule. If you select Critical (9.0 - 10.0), you duplicate the existing built-in rule.</p>	<ul style="list-style-type: none"> ■ Critical (9.0 - 10.0) ■ High and above (7.0 - 10.0) ■ Medium and above (4.0 - 10.0) ■ Low and above (0.1 - 10.0)
<p>Vulnerabilities with fixes</p>	<ul style="list-style-type: none"> ■ Critical (9.0 - 10.0) ■ High and above (7.0 - 10.0) ■ Medium and above (4.0 - 10.0) ■ Low and above (0.1 - 10.0)
<p>Allowed registries</p>	<p>Specify registries you want to allow as source. For example, docker.io.</p>

- 6 Click **Next**.
- 7 On the **Confirm Rule** page, review the summary of the rule criteria and the matching Kubernetes resources and click **Save**.

Confirm Rule

General

Name IX-Container-Custom-Rule

Description Custom rule for IX container images

Rule criteria

Image criteria Critical vulnerabilities

Vulnerability severity High and above

Matching resources



Rule criteria matches **34** resources across all scopes

Cluster	2
DaemonSet	6
Deployment	11
Namespace	7
Pod	17

Create an Advanced Kubernetes Custom Rule

To create an advanced Kubernetes custom rule, use a YAML file to describe MAPL rules for Kubernetes resources and applicable conditions.

MAPL rules in YAML format give more specificity in how you can configure a custom rule for a Kubernetes environment.

Prerequisites

To successfully configure an advanced custom rule, you must have the YAML file written in MAPL language that is applicable for your Kubernetes environment.

Procedure

- 1 On the left navigation pane, click **Enforce > K8s Policies**.
- 2 Click the **Rules** tab.
- 3 Click **Add Rule**.
- 4 Define the rule.
 - a Enter a unique custom rule name and a description.
 - b Select **Advanced - MAPL access control rule (YAML format)** as the rule criteria.
 - c Click **Next**.
- 5 Enter YAML code in the text area or click **Import** to import a YAML file.

Note

- The YAML file must include one-attribute conditions, using logical operands, which are tested against the Kubernetes configuration data.
- The attribute is a JSONpath.
- The method is one of the following (the value is a fixed value):

EQ - equal	EX - exists	GE - greater than or equal to
GT - greater than	IN - in list of values [val1,val2,val3,...]	LE - lower than or equal to
LT - lower than	NE- not equal	NEX - not exists
NIN - not in list of values [val1,val2,val3,...]	NRE - does not match a regular expression	RE - matches a regular expression

For example:

Configure Rule

Provide MAPL access control rule (YAML format) [Learn more](#)

* MAPL rule configuration

Import

```

1 conditions:
2   conditionsTree:
3     ANY:
4       parentJsonpathAttribute: 'jsonpath:$spec.containers[:]'
5       condition:
6         OR:
7           - condition:
8             attribute: 'jsonpath:$RELATIVE.resources.limits.cpu'
9             method: NEX
10          - condition:
11            attribute: 'jsonpath:$RELATIVE.resources.limits.memory'
12            method: NEX
13

```

See [MAPL \(Manageable Access-control Policy Language\)](#) (external link).

- 6 Click **Next**.
- 7 On the **Confirm Rule** page, review the summary of the rule criteria and the matching Kubernetes resources and click **Save**.


Edit or Delete a Kubernetes Custom Rule

To edit or delete a Kubernetes custom rule, perform the following procedure.

Note

- You can edit custom rules after creating them, even if you have included them in Kubernetes hardening policies.
- You cannot delete custom rules if they are part of Kubernetes hardening policies.

Procedure

- 1 On the left navigation pane, click **Enforce > K8s Policies**.
- 2 Click the **Rules** tab.
- 3 Locate the rule to edit and click the arrow  icon at the end of the row.
- 4 In the dropdown menu in the right panel, select an action.
 - To add a rule to template, click **Add to templates**, select one or more custom templates, and click **Save**.
 - To update the rule, click **Edit**. The **Edit Custom Rule** window shows. You cannot change the rule type. Click **Next** and follow the configuration wizard steps to modify the rule. Click **Save**.
 - To duplicate a rule, click **Duplicate**. Rename and customize the new rule.

- To delete a rule, click **Delete** and then click **OK** to confirm the deletion.

Create an Exception for a Kubernetes Hardening Policy Rule

You can review violations when you create or update a Kubernetes hardening policy and you can reduce the number of violations by creating rule exceptions. Creating exceptions omits workloads from the rule action.

Important Carbon Black recommends that you only create exceptions to exclude specific workloads that exhibit known behaviors. Remediate as many violations as possible before considering an exception.

Tip You can deactivate a rule if it triggers too many violations until the issues in your environment are resolved. To exclude the rule from the policy, toggle the state of the rule to `Off`.

Procedure

- 1 On the left navigation pane, click **Enforce > K8s Policies**.
- 2 Click the **Hardening Policies** tab.
- 3 Click the policy name to edit it.
- 4 Click **Next** two times to go to the **Review Violations** page.
- 5 Select a rule that has an **Alert** or **Enforce** action and click the **Exceptions** tab.
- 6 Click **Add Criteria**.
- 7 Define the exception criteria in the **Resource name** dropdown menu. Your options are:
 - **Resource name:** Set to **is equal to**, **starts with**, or **ends with**. Type in the name criteria.

You can specify either a particular workload or criteria that matches multiple workloads — for example, workloads that have the same prefix or the same suffix.
 - **Workload label:** Define the key value pair.
 - **Username:** **is equal to** the entered name.

Violations | Exceptions


Create rule exception criteria for resources in this scope

Exceptions will be applied in an ongoing basis to resources that meet the criteria

Resource name ▼ starts with ▼ IX ▼

The exception criteria match current and future workloads that are part of the policy scope.

8 Click **Add**.

Note You can remove an exception by clicking the trash can icon  next to the exception criteria.

Results

The total count of violations decreases. The workloads that are excluded from the rules violations show in the **Exceptions** tab.

Mutate Hardening Rules

You can enforce the values of selected resource properties to temporarily remediate an issue. When you set an **Enforce** action for a rule, the mutated value is considered and a violation alert displays. If a workload still violates the rule after remediation, it is blocked from deployment.

Note In this context, mutation means that a policy changes Kubernetes resources based on new criteria. For example, allowing privilege escalation.

The rules for which you can apply an **Enforce** action are described in the following table.

Rules Category	Rules that Allow Enforce Action	Resource Field	Enforced Value
Workload Security	Access to host namespace	<code>spec.hostNetwork</code>	False
		<code>spec.hostPID</code>	
		<code>spec.hostIPC</code>	
	Allow privilege escalation	<code>spec.containers[*].securityContext.allowPrivilegeEscalation</code>	False
	Allow privilege container	<code>spec.containers[*].securityContext.privileged</code>	False
Writable file system	<code>spec.containers[*].securityContext.readOnlyRootFilesystem</code>	True	

Rules Category	Rules that Allow Enforce Action	Resource Field	Enforced Value
	SecComp profile	metadata.annotations['container.seccomp.security.alpha.kubernetes.io/*']	User-Defined
		metadata.annotations['seccomp.security.alpha.kubernetes.io/pod*']	
		spec.securityContext.seccompProfile.type	
		spec.containers[*].securityContext.seccompProfile	
	Sysctl	spec.securityContext.sysctls	User-Defined
	Additional capabilities	spec.containers[*].securityContext.capabilities.add	User-Defined
	AppArmor	metadata.annotations['container.apparmor.security.beta.kubernetes.io/*']	User-Defined
	Unmasked proc mount	spec.containers[*].securityContext.procMount	Empty (removes the field)
	Enforce not root	spec.securityContext.runAsNonRoot	User-Defined user and group ID
		spec.containers[*].securityContext.runAsNonRoot	
		spec.containers[*].securityContext.runAsGroup	
		spec.containers[*].securityContext.runAsUser	
		securityContext.runAsGroup	
		securityContext.runAsUser	
Quota	CPU limits	spec.containers[*].resources.limits.cpu	User-Defined
		spec.containers[*].resources.requests.cpu	
	Memory limits	spec.containers[*].resources.limits.memory	User-Defined
		spec.containers[*].resources.requests.memory	

Mutate a Rule Outcome

During the policy creation process, you can set the **Enforce** action for a rule. This action sets a predefined value to the rule outcome. You must select a preset for enforcement rules that require a user-defined value.

Prerequisites

For a list of rules that allow the **Enforce** action, see [Mutate Hardening Rules](#).

Procedure

- 1 On the left navigation pane, click **Enforce > K8s Policies**.
- 2 Click the **Hardening Policies** tab.
- 3 Click the policy name to edit or click **Edit** in the **Actions** dropdown menu.
- 4 Click **Next**.
- 5 On the Add Rules page, review the **Added Rules** in the right pane, or scroll to the **Workload Security** category in the middle pane.
- 6 For each of the rules listed in the table in [Mutate Hardening Rules](#), select the **Enforce** action.
- 7 Click **Next** two times and click **Save**.

Results

You have set the property values of the rule to comply with the security standards. No violations will be triggered.

Kubernetes Policy Templates

Kubernetes policy templates are groups of predefined or custom rules that do not include exceptions.

Predefined rule sets cover the following categories:

Category	Purpose
Command	Limits Kubernetes command-line commands
Container Images	Identifies vulnerabilities in container images
CRD	Limits usage of custom resources
Custom	All custom rules that exist in the system
Network	Ensures that service types are not exposed outside of Kubernetes
Quota	Establishes CPU and memory quotas
RBAC	Limits new roles with extensive privileges

Category	Purpose
Volume	Limits access to data
Workload Security	Rules based on the Kubernetes security configuration. See Pod Security Standards (external link).

Create a Kubernetes Policy Template

You can group specific rules in custom templates to reuse them across Kubernetes hardening policies. Custom templates are a combination of built-in rules and custom rules. They are applicable when you create a policy.

Note You configure the **Alert** or **Block** action in the policy, but not in the template. Thus, you can have the same rule in different policies with different applied actions.

Procedure

- 1 On the left navigation pane, click **Enforce > K8s Policies**.
- 2 Click the **Templates** tab.
- 3 Click **Add Template**.
- 4 Enter the name for the custom template and click **Save**.
The template is created and visible in the list of **Custom Templates**.
- 5 To add rules to the newly created custom template, click **Options > Edit template**.
- 6 Select the rules to add to the custom template.
- 7 Click **Save**.

Save a Hardening Policy as a Template

To save the rules from a Kubernetes hardening policy for use in other policies, save a policy as a template.

Procedure

- 1 On the left navigation pane, click **Enforce > K8s Policies**.
- 2 Click the **Hardening Policies** tab.
- 3 For the selected policy, click **Save as template** in the **Actions** dropdown menu.
- 4 Enter name for the new template and click **Save**.

Results

The newly created template is saved. The **Templates** tab is displayed on the Kubernetes Policies page, with focus on the new template. See [Kubernetes Policy Templates](#).

Subscribe to Alert Notifications

To receive notifications when alerts occur, perform the following procedure.

Prerequisites

Email addresses must be associated with registered Carbon Black Cloud console users.

Procedure

- 1 On the left navigation pane, click **Settings > Notifications**.

Add Notification
✕

*** Name**

When do you want to be notified?

Alert crosses a threshold
▼

Alert severity 1 [-] [0] [2] [+]

*** Alert types**

All types
 Select types

USB Device Control Containers Runtime Host Based Firewall

Intrusion Detection System (email only)

*** Policy**

All policies
 Select policies

How do you want to be notified?

Email

Search for a user

Send only 1 email notification for each threat type per day

API Key

Search for an API key

Save
Cancel

- 2 Click **Add Notification** and populate the required text fields.
 - a Select the **Alert crosses a threshold** notification type from the dropdown menu. This setting notifies you if an alert crosses a specified severity threshold.
 - b Specify the alert severity threshold.

- c Select the alert types for which to receive notifications. The default value is **All types**.
- d Select all policies or specific ones. **All policies** is the default value.

If you select more than one policy, the Carbon Black Cloud console sends a separate notification for each policy.

- e Select how to get the notifications:

Select either the **Email** option or the **API Key**. For either option, select one or more users.

- f Optional. To reduce the number of emails that you receive, select the check box for **Send only 1 email notification for each threat type per day**.

3 Click **Save**.

Results

The notification displays in the **Notifications List**. When an alert surfaces that matches your notification criteria, you will receive a notification email. For example:

CARBON BLACK CLOUD ALERT

Detected an abnormal egress connection with medium or low risk

Target value	MEDIUM
Remote host	[REDACTED]
Port	443
Protocol	TCP
Workload	coredns
Workload kind	Deployment
Namespace	kube-system
Cluster	[REDACTED]
SHA-256	506ffc437f5d3c4803a45b895b02557e7280eb3c6eb7d8ff8bd9073990e989d5
Process name	KUBERNETES_RUNTIME_NODE_AGENT
Reputation	NOT_LISTED
Alert ID	b6e1e3a1-f1fa-9aab-30e3-dda6a5c8c899
Threat score	4

[View in Carbon Black Cloud](#)

This alert is based on notification settings specified in 'IX'. [Update settings](#)

Thank you for using VMware Carbon Black Cloud.

vmware Carbon Black

Setting up API Access

You can use the Carbon Black Open API platform to integrate with a variety of security products, including SIEMs, ticket tracking systems, and your own custom scripts.

To find integration partners, see <https://www.vmware.com/products/vmware-marketplace.html> and visit the Carbon Black Developer Network at <https://developer.carbonblack.com/>.

Tip You can also use the [Access Profiles and Grants API](#) to manage (create/read/update/delete) roles for a principal in your organization.

Create and Manage an API Key

You add and manage services integrations into your environment by setting their access level through creating and managing your API keys.

When creating your API Keys, you must take into account the following limitations and implications:

- All current APIs use a key of type `Custom`. Create an Access Level that supports least privilege.
- Types `SIEM` and `API` keys are deprecated and scheduled for deactivation on 31 October 2024 and 31 July 2024, respectively.
- For API migration instructions, see [API and Schema Migration](#) (external link).
- We recommend that new integrations use one of the following mechanisms to receive all available data:
 - **Data Forwarders:** to stream alerts or events to your own S3 bucket where you can control retention.
 - [Alerts v7 API](#) (external link): to search up to 180 days of historical alert data.
- It is important that you safeguard the API ID and the API Secret key.

Prerequisites

To use the **Custom** access permissions for your integrations, you must create an access level. See [Setting Access Levels](#).

Procedure

- 1 On the left navigation pane, click **Settings > API Access**.
- 2 For `Custom` API keys, create an access level. See [Setting Access Levels](#).
For more details, see the [Carbon Black Cloud API Access](#) (external link).

3 Click **Add API Key**.

Add API Key X

* Name
IX-API-Custom

Description
Custom API key for IX access

* Access Level type Custom ▼ * Custom Access Level IX-SecOps ▼

⚠ This permission set may contain unversioned APIs. Visit developer.carbonblack.com for all currently supported/versioned APIs.

Authorized IP addresses
Specify a comma separated list of single IP address, or an IP address range in CIDR notation (for example, 203.0.113.5/32).

Save Cancel

- a Enter a unique name and description.
- b Select the appropriate **Access Level Type**. The default type is `Custom`.
- c Set the **Custom Access Level**.
- d **Optional:** Add authorized IP addresses.

You can restrict the use of an API key to a specific set of IP addresses for security reasons.

4 To apply the changes, click **Save**.

Results

A pop-up window displays the new API credentials:

API Credentials
✕

API ID
HDLZ1CGWKI

API Secret Key
E1M4HSB7H3C4GU3UG5PL9M57

What to do next

Purpose	Action
To update the name, description, or the IP addresses for a specific API key:	Click the Edit icon in the Actions column.
To view the credentials for a specific API key:	Click the Actions dropdown menu and select API Credentials .
To generate new credentials:	Click the Actions dropdown menu, select API credentials , and click Generate new API Secret Key .
To see all notifications sent to the API key within a timeframe:	Click the Actions dropdown menu , click Notification History , and then select the timeframe.
To delete the API key:	Click the Actions dropdown menu and select Delete . Note You cannot use this procedure to delete API Keys that are associated with a notification rule. See Delete API Key with Attached Notification Rule .

Delete API Key with Attached Notification Rule

To delete an API key with attached notification rules, you must delete all of the associated notifications rules first and then the API key.

Procedure

- 1 On the left navigation pane, click **Settings > API Access**.
- 2 Locate the **API ID** of the API key to delete.
- 3 On the left navigation pane, click **Settings > Notifications**.

- 4 Find the API ID in the **Subscribers** column and click the **Delete** icon to delete all associated notification rules.

NAME	POLICY	CRITERIA	SUBSCRIBERS	ACTIONS
IX	All policies	Alert severity: >= 4 Alert types: All types	API key: 11ZQ3U7EW6	

- 5 On the left navigation pane, click **Settings > API Access** and click **Delete** in the **Actions** column to delete the API key.

Setting Access Levels

Access levels offer the ability to create custom levels of access for your integrations with other security products. Create custom access levels with specific, granular permissions to apply to an API key.

Create Access Levels

To access the data in your Carbon Black Cloud integrations through APIs, you must determine the appropriate access level for your API.

Procedure

- 1 On the left navigation pane, click **Settings > API Access**.
- 2 Click the **Access Levels** tab and click **Add Access Level**.
- 3 Enter a name and description for your access level.
- 4 Select the boxes of the permission functions to include in your access level.
- 5 Click **Save**.

Results

You can view the newly created access level listed in the **Access Levels** tab.

What to do next

To modify or delete an access level, use the **Actions** column. If you export an access level, you download a JSON file holding the role definition details.

Apply Access Level to an API Key

You apply a custom access level to an API key when granting access to your integrations.

Note Select a user role from the **Custom Access Level** drop-down menu for testing purposes only. User roles can contain unversioned APIs. For information on all currently supported and versioned APIs, see [Carbon Black Developer Network](#).

Prerequisites

Create a custom access level. See [Create Access Levels](#).

Procedure

- 1 On the left navigation pane, click **Settings > API Access**.
- 2 Click the **API Keys** tab and click **Add API Key**.
- 3 Enter a name for your API Key and a short description.

- 4 Select **Custom** from the **Access Level Type** dropdown menu.
- 5 Select either a user role or an access level that is available in your organization from the **Custom Access Level** dropdown menu.
- 6 To apply the changes, select **Save**.

Results

The newly created API key displays in the **API Keys** tab.

What to do next

Use the **Actions** column to edit the API key, or the dropdown menu to view the associated API key credentials and notifications history.

Scanning Images

4

You can scan container images for known vulnerabilities and you can observe the results from a system cluster scan or a manual scan in the Carbon Black Cloud console.

Note

- Image scanning is only applicable for images that are based on Linux operating system packages.
 - Image scanning requires CLI Client. See [Setting up CLI Client for Image Scanning](#).
-

Container images are scanned under the following circumstances:

- Scan is triggered by the Continuous Integration / Continuous Deployment (CI/CD) pipeline or a manual scan. See [Manually Rescan a Container Image](#).
- Kubernetes sensor version update. See [Upgrading or Downgrading the Kubernetes Sensor](#).
- Initial cluster scan of container images at cluster setup. See [Adding Clusters and Installing Kubernetes Sensors](#).
- New vulnerabilities in the Carbon Black Cloud vulnerabilities database.
- Updated file reputation.

Cluster image scanning provides the following benefits:

- Visibility for the container images in your environment.
- Information for found vulnerabilities and available fixes.
- Capability to create exceptions at image level from inside the image scan report.
- Kubernetes policies prevent container images that have substantial vulnerabilities from progressing through the CI/CD pipeline. See [Kubernetes Policies](#).
- File reputation scanning of all deployed images and malware detection. See [Detect Malware in a Container Image](#).

To have the latest information on file reputations, you must refresh the file reputation data that comes in from third-party feed providers, and you must consistently rescan your clusters for newly deployed images.

Read the following topics next:

- [Manually Rescan a Container Image](#)

Manually Rescan a Container Image

You can run the scan for a container image in the Carbon Black Cloud console or in a terminal using the CLI Client. The following procedure performs an image scan in the Carbon Black Cloud console.

If a container image is built, pushed to a public repository, and deployed to a Kubernetes cluster between two scans, it will be displayed in the list with a **Pending** status. If the image scan has a status **Error**, you can run the scan for that image in the Carbon Black Cloud console or in a terminal, using the CLI Client.


Note You can run the manual scan for images in public repositories only. If the image belongs to a private repository, the **Rescan** button is inactive.

Prerequisites

Download and configure CLI Client. See [Setting up CLI Client for Image Scanning](#). To use the CLI Client in a terminal, see [Container Security API and Integrations](#) (external link).

Procedure

- 1 On the left navigation pane, do one of the following depending on your system configuration and role:
 - If you have the Kubernetes Security DevOps or SecOps role and your system has only the Container security feature, click **Inventory > Container Images**.
 - If you have any other role and your system has Container security and other Carbon Black Cloud features, click **Inventory > Kubernetes > Container Images**.
- 2 Click the **Deployed Images** tab.
- 3 If they are contracted, expand the filter options by clicking the carets >> in the top left. For the **Scan Status** filter, select **Error**.

The table displays only images that have an **Error** status.
- 4 Either use the search field to find a particular image or choose a container image from the list. Click the arrow  icon at the right of the selected image.

5 Click **Rescan** in the **Image Details** panel.

IMAGE DETAILS [Rescan](#) [View more](#)

Name `docker.io/vmwareallspark/acme-load-gen:latest`
 Registry `docker.io`
 Repository `vmwareallspark/acme-load-gen`
 Manifest digest `sha256:ef020f9fd070600f427f35fca05541b8aefa8813e926556cbabdc540d974418`
 Repo digests
 Scan status `Scanned`
 Last scan `Last scanned at 5:52 AM on Apr 18, 2022`

KUBERNETES

Clusters `1`
 Deployments `1`
 Namespaces `1`
 Workloads `1` [🔗](#)

VULNERABILITIES

CVE	PACKAGE	FIX	EXCEPTION
> CVE-2018-250...	libwebp-dev-0.6...	Yes	Yes
> CVE-2018-250...	libwebp6-0.6.1-2	Yes	No
> CVE-2018-250...	libwebpdemux2-...	Yes	No
> CVE-2018-250...	libwebpmux3-0...	Yes	No
> CVE-2018-250...	libwebp-dev-0.6...	Yes	No

[Show all \(2583\)](#) [🔗](#)

FILE REPUTATIONS

No records found

Monitoring and Analyzing Containers

5

After you create roles and users, set up your Kubernetes clusters, and configure scopes and policies, you are ready to use the system to monitor and analyze behaviors.

Read the following topics next:

- [Severity Scoring](#)
- [Monitoring Container Images](#)
- [Managing and Viewing File Reputations in Container Images](#)
- [Detecting and Preventing Secrets](#)
- [Monitoring Kubernetes Workloads](#)
- [Analyzing Network Activity](#)

Severity Scoring

This section describes two types of security scoring methods that are used in Containers.

Kubernetes Risk Severity Scoring

Risk Severity is a metric that represents the risk of security vulnerability for your Kubernetes workload. It uses the Kubernetes Common Configuration Scoring System (KCCSS), which is a framework for rating security risks associated with misconfigurations.

Note The risk rating for Kubernetes workloads is different than the risk severity for container image vulnerabilities because they are evaluated using different scales. For more information about container image risk scores, see [Risk Evaluation for Container Images](#).

Kubernetes Common Configuration Scoring System

KCCSS scores both risks and remediations as separate rules. It calculates risk for every runtime setting of a workload and then the total risk of the workload. For each workload, a risk score ranging from 0 (no risk) to 10 (high risk) is assigned.

Measures of Risk

KCCSS shows the potential impact of risky configuration settings in three areas:

Confidentiality

Exposure of Personal Identifiable Information (PII), potential access to keys, and so on.

Integrity

Unwanted changes to the container, host, or cluster; for example, being able to change the runtime behavior, launch new processes, new pods, and so on.

Availability

Exhaustion of resources, denial of service, and so on.

KCCSS accounts for whether the risk is limited to the container or impacts the entire cluster, the ease of exploiting the risk, and whether an attack requires local access. It combines all security risks associated with a workload together with the required remediations to attribute an overall risk score to the workload.

Risk Score

The scoring system takes into account over 30 security settings for Kubernetes configurations. The exact rules and scoring formula are part of KCCSS. Based on the score, workloads are filtered by the level of severity: high, medium, or low. The higher the risk score, the higher is the severity. Every workload is assigned a risk score of between 0 (low risk) and 10 (high risk).

Score Range	Severity
0 - 3	Low
4 - 6	Medium
7 - 10	High

Risk Evaluation for Container Images

The Common Vulnerability Scoring System (CVSS) is a standard measurement system for describing characteristics and severity of software vulnerabilities. Every vulnerability is assigned a risk score of between 0.0 (no risk) and 10.0 (maximum risk).

Note The risk rating for container image vulnerabilities is different than the risk severity for workloads because they are evaluated using different scales. For more information about Kubernetes workloads risk scores, see [Kubernetes Risk Severity Scoring](#).

CVSS consists of three metric groups:

- **Base:** characteristics of a vulnerability that are constant over time and across user environments.

- **Temporal:** characteristics of a vulnerability that might change over time but does not span user environments.
- **Environmental:** characteristics of a vulnerability that is relevant and unique to a particular user environment.

For more details, refer to the [Common Vulnerability Scoring System SIG](#) (external link).

The risk score range and severity are defined as follows.

Rating	Score
None	0.0
Low	0.1 to 3.9
Medium	4.0 to 6.9
High	7.0 to 8.9
Critical	9.0 to 10.0







Note The vulnerabilities for which the threat vectors are not yet known are grouped under **Unknown** severity. This means that the system was able to identify a given artifact as vulnerable, but there might not be CVE attached to the vulnerability. Unknown severity can range between 0-10.

Color Indicators for Image Vulnerabilities Scoring

The Common Vulnerability Scoring System (CVSS) is used for estimating the severity of discovered vulnerabilities. In addition to the risk scores that are defined in CVSS, the **Unknown** category displays in the Carbon Black Cloud console.

For more information about CVSS, see [Risk Evaluation for Container Images](#).

On various Carbon Black Cloud console pages, color bars for the different vulnerabilities risk scores are displayed. The color bars correspond to the following ratings:

Color Name	Color Bar	Rating (refer to CVSS)
Green		None
Yellow		Low
Orange		Medium
Red		High
Dark Red		Critical
Gray		Unknown

The numbers inside the color bars represent number of vulnerabilities and number of fixes.

Note The risk rating for container image vulnerabilities is different than the risk severity for workloads because they are evaluated using different scales. For more information about Kubernetes workloads risk scores, see [Kubernetes Risk Severity Scoring](#).

Monitoring Container Images

A container is a lightweight, portable executable image. Container images exist in either the build or deploy stage in your continuous integration environment.

This section discusses how to monitor and analyze data on container images.

View Container Images - Overview

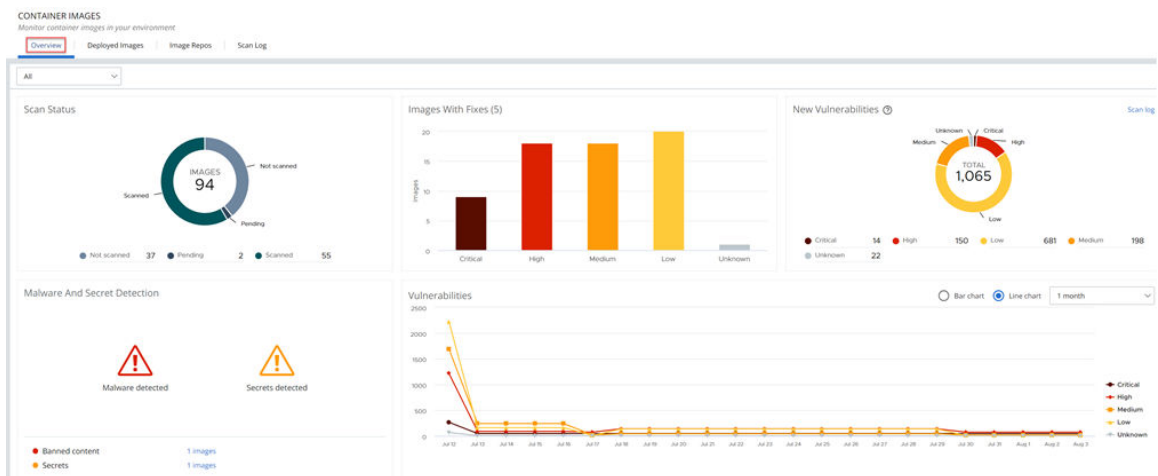
This topic provides an overview of the container image data you can retrieve on the Container Images page in the Carbon Black Cloud console.

Procedure

- ◆ On the left navigation pane, do one of the following depending on your system configuration and role:
 - If you have the Kubernetes Security DevOps or SecOps role and your system has only the Container security feature, click **Inventory > Container Images**.
 - If you have any other role and your system has Container security and other Carbon Black Cloud features, click **Inventory > Kubernetes > Container Images**.

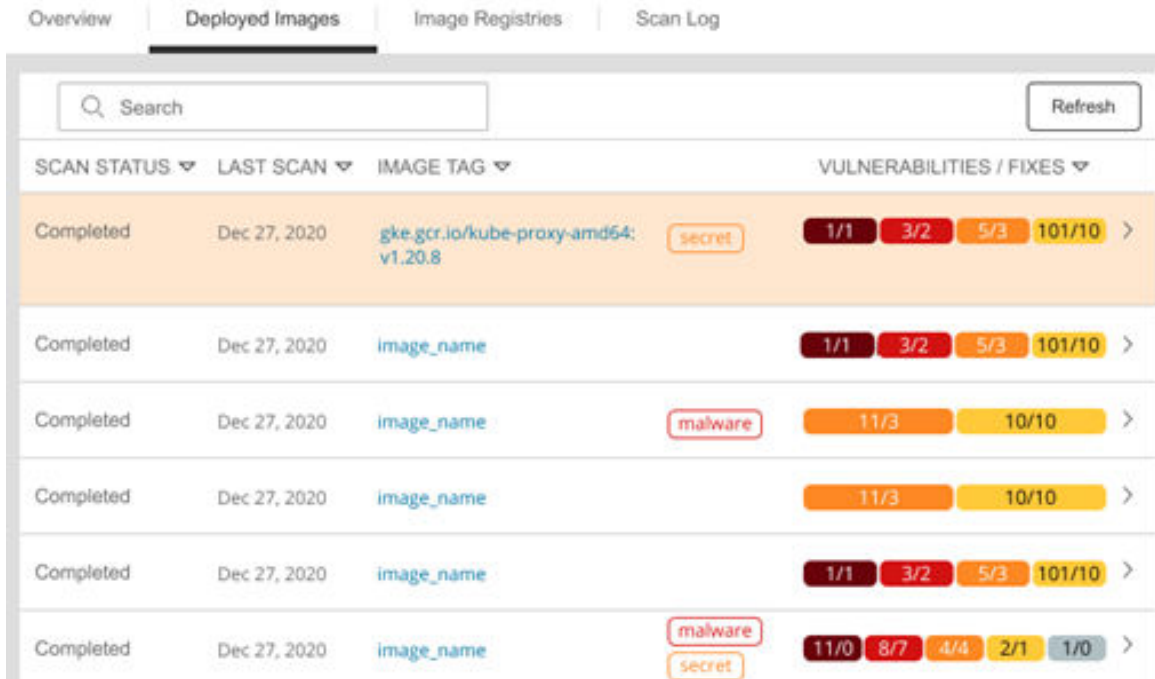
The Container Images page includes the following tabs and information.

- The **Overview** tab shows the following details:



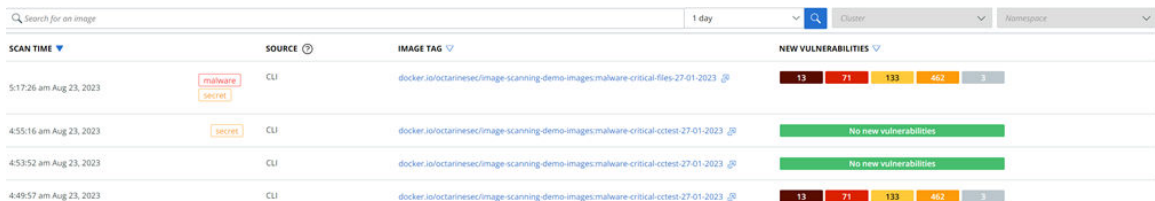
- A summary of the latest scan status
- New vulnerabilities
- Vulnerabilities with fixes

- Malware and Secret Detection
- Bar or line chart showing all vulnerabilities that were discovered within a specified timeframe
- The **Deployed Images** tab shows an inventory of container images running on your Kubernetes clusters together with vulnerability scan results and available fixes for each image.



On the **Deployed Images** tab, you can:

- View detailed container data — click the **Image Tag**.
- View information about the workload — click the link icon in the **Workloads** column.
- View details about an image — click the arrow icon to the right of the row. See [View Deployed Container Image Details](#).
- The **Image Repos** tab shows an inventory of the repositories where your container images reside. All the images in a repository are displayed, including old tags that are no longer in use, images that have not yet been deployed, and images that are deployed.
- The **Scan Log** tab shows searchable scan activity. For example:



You can click the **Image Tag** for an entry to view scan result details. See [View Image Scan Report - Scan Log Details](#).

View Deployed Container Image Details

To view deployed image scan and vulnerabilities details, perform the following procedure.

Procedure

- 1 On the left navigation pane, do one of the following depending on your system configuration and role:
 - If you have the Kubernetes Security DevOps or SecOps role and your system has only the Container security feature, click **Inventory > Container Images**.
 - If you have any other role and your system has Container security and other Carbon Black Cloud features, click **Inventory > Kubernetes > Container Images**.
- 2 Click the **Deployed Images** tab.

- 3 To expand the **Image Details** panel, click the arrow  icon at the right of the row.

IMAGE DETAILS

Rescan
[View more](#)



Name	gke.gcr.io/kube-proxy-amd64:v1.20.8
Registry	gke.gcr.io
Repository	kube
Image digest	sha256:51130e63b8158e7b3b3507e4dec916aa7e07a6cb7ce9279f6afe4803036e4128
Scan status	Completed
Last scan	06:00 am on Dec 27, 2020

SECRETS (1)


TYPE	FILE	EXCEPTION
File	.eslintignore	No

VULNERABILITIES (110)

CVE	PACKAGE	FIX	EXCEPTION
> CVE-lorem-ips...	libgnutls30-3.6.7-4...	Yes	No
> CVE-2020-3453	libhogweed4-3.4.1-1	--	No
> CVE-2020-3453	passwd-1:4.5-1.1	--	No
> CVE-2020-3453	libgnutls30-3.6.7	--	No
> CVE-2020-3453	debian3942.1	--	No

- To rescan the image, click **Rescan**. See [Manually Rescan a Container Image](#).
- To view more information about a Kubernetes workload, click the link  icon next to **Workloads** in the **Kubernetes** section.
- To access information about a file that contains secrets, click the filename in the **Secrets** section.
- To view a short description of the CVE code and the package where the vulnerability is identified, click the carat  icon to the left of the **CVE**.

CVE	PACKAGE	FIX	EXCEPTION
▼ CVE-2018-250...	libwebp-dev-0.6....	Yes	Yes
<div style="background-color: #f0f0f0; padding: 5px;"> <p>Severity 9.1</p> <p>Package libwebp-dev-0.6.1-2</p> <p>Fix 0.6.1-2+deb10u1</p> <p>Description A heap-based buffer overflow was found in libwebp in versions before 1.0.1 in GetLE16().</p> </div>			
> CVE-2018-250...	libwebp6-0.6.1-2	Yes	No
> CVE-2018-250...	libwebpdemux2-...	Yes	No
> CVE-2018-250...	libwebpmux3-0....	Yes	No
> CVE-2018-250...	libwebp-dev-0.6....	Yes	No

- To view all vulnerabilities of this container, click the link  icon in the **Vulnerabilities** section. See [Investigate Container Image Vulnerabilities](#).
- To view additional details about the deployed image, click **View more** in the **Image Details** section. The **Overview** tab of the Image Scan Report page opens.

← Back to Container images

DOCKER.IO/OCTARINESEC/IMAGE-SCANNING-DEMO-IMAGES:MALWARE-CRITICAL-FILES-27-01-2023 Copy URL

Overview | Layers | Packages | Suspicious Files | Vulnerabilities | KBs Workloads | Scan Log

General Information

Image docker.io/octarinesec/image-scanning-demo-image-malware-critical-files-27-01-2023

Registry docker.io

Repository octarinesec/image-scanning-demo-images

Image layers 9

Manifest digest sha256:2930ad942b3232c95d28039a...

Repo digests octarinesec/image-scanning-demo-l...

OS rhel


OS version 8.1


Architecture amd64

Size 236 MB

Last scan 1:17:26 pm Aug 23, 2023

User --

Labels 22 

Environment variables 2 


Command tail -f /dev/null

Volumes 0


Entry point --

Exposed port --

Violations

RULE	ITEMS
 No records found	


Vulnerability Summary




TOTAL 682


● Critical 13 ● High 71 ● Medium 462
● Low 133 ● Unknown 3

Malware and Secret Detection


 Malware detected


 Secrets detected

● Critical content 3 files
● Secrets 1 layers

- You can view additional information about secrets by:
 - Clicking the **Labels**  icon. For example:

Labels

LABEL

maintainer="Kong Docker Maintainers <docker@konghq.com> (@team-gateway-bot)"

org.opencontainers.image.ref.name="ubuntu"

org.opencontainers.image.version="22.04"

- Clicking the **Environmental variables**  icon. For example:

Environment Variables

VARIABLE ▾

= "ASSET=ce"

KONG_VERSION="3.3.0"

PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"

View Container Image Repositories

To view image repositories, perform the following procedure.

Procedure

- 1 On the left navigation pane, do one of the following depending on your system configuration and role:
 - If you have the Kubernetes Security DevOps or SecOps role and your system has only the Container security feature, click **Inventory > Container Images**.
 - If you have any other role and your system has Container security and other Carbon Black Cloud features, click **Inventory > Kubernetes > Container Images**.
- 2 Click the **Images Repos** tab.

A list of repositories and their registries is displayed. You can sort the list and you can search for a particular repository or registry.

3 To view more information about a repository, click its name in the **Repository** column.

SCAN STATUS	LAST SCAN	IMAGE TAG	VULNERABILITIES/FIXES	WORKLOADS	EXCEPTIONS
Scanned	Apr 12, 2023	docker.io/octarinesec/hunttime-kubernetes-resolver-test	10/15 37/37 5/5 2/2	0	—
Scanned	Apr 12, 2023	docker.io/octarinesec/hunttime-kubernetes-resolver-switch	10/15 5/5 1/1	0	—
Scanned	Apr 12, 2023	docker.io/octarinesec/hunttime-kubernetes-resolver-debug	10/15 5/5 1/1	0	—
Scanned	Jun 23, 2022	docker.io/octarinesec/hunttime-kubernetes-resolver-ghatom	No vulnerabilities	0	—
Scanned	Sep 22, 2022	docker.io/octarinesec/hunttime-kubernetes-resolver-ws-david-test	No vulnerabilities	0	—
Scanned	Sep 22, 2022	docker.io/octarinesec/hunttime-kubernetes-resolver-ws-test-david	No vulnerabilities	0	—

- a To open the Container Images page for an image, click its name in the **Image Tag** column.
- b To view details about an image, click the arrow icon at the right of the row. See [View Deployed Container Image Details](#).

View Image Scan Report - Scan Log Details

To view the scan logs of all image scans in the Carbon Black Cloud console, perform the following procedure.


Procedure

- 1 On the left navigation pane, do one of the following depending on your system configuration and role:
 - If you have the Kubernetes Security DevOps or SecOps role and your system has only the Container security feature, click **Inventory > Container Images**.
 - If you have any other role and your system has Container security and other Carbon Black Cloud features, click **Inventory > Kubernetes > Container Images**.
- 2 Click the **Scan Log** tab.

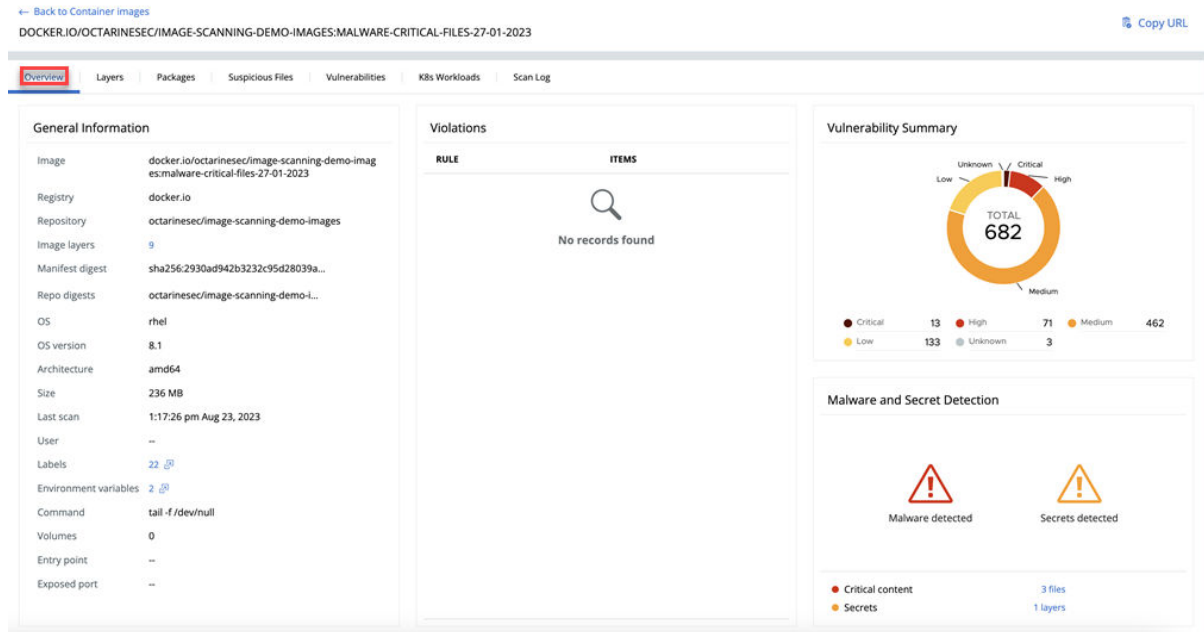
SCAN TIME	SOURCE	IMAGE TAG	NEW VULNERABILITIES
5:17:26 am Aug 23, 2023	malware secret CLI	docker.io/octarinesec/image-scanning-demo-images/malware-critical-files-27-01-2023	13 71 133 462 3
4:55:16 am Aug 23, 2023	secret CLI	docker.io/octarinesec/image-scanning-demo-images/malware-critical-cctest-27-01-2023	No new vulnerabilities
4:53:52 am Aug 23, 2023	CLI	docker.io/octarinesec/image-scanning-demo-images/malware-critical-cctest-27-01-2023	No new vulnerabilities
4:49:57 am Aug 23, 2023	CLI	docker.io/octarinesec/image-scanning-demo-images/malware-critical-cctest-27-01-2023	13 71 133 462 3

The **Source** column defines the reason that the scan was initiated:

Source Column	Description
CLI	Scan triggered by the CI/CD pipeline or a manual scan.
Cluster rescan	Kubernetes sensor version update.
Cluster scan	Initial cluster scan of container images located in the Kubernetes cluster that you set up in the Carbon Black Cloud console.
Feed update	Image scanning based on new vulnerabilities in the Carbon Black Cloud vulnerabilities database.
Reputation Update	Updated file reputation.

3 For more image details, click the **Image Tag**  icon.

This action opens the **Overview** tab on the **Image Scan Report**.



← Back to Container images Copy URL

DOCKER.IO/OCTARINESEC/IMAGE-SCANNING-DEMO-IMAGES:MALWARE-CRITICAL-FILES-27-01-2023

Overview Layers Packages Suspicious Files Vulnerabilities K8s Workloads Scan Log

General Information

Image: docker.io/octarinesec/image-scanning-demo-images:malware-critical-files-27-01-2023

Registry: docker.io

Repository: octarinesec/image-scanning-demo-images

Image layers: 9

Manifest digest: sha256:2930ad942b3232c95d28039a...

Repo digests: octarinesec/image-scanning-demo-I...

OS: rhel


OS version: 8.1


Architecture: amd64

Size: 236 MB

Last scan: 1:17:26 pm Aug 23, 2023

User: --

Labels: 22 

Environment variables: 2 

Command: tail -f /dev/null

Volumes: 0

Entry point: --

Exposed port: --

Violations

RULE ITEMS


No records found


Vulnerability Summary

TOTAL 682

Critical: 13, High: 71, Medium: 462, Low: 133, Unknown: 3

Malware and Secret Detection

Malware detected:  3 files

Secrets detected:  1 layers

See [View Container Image Scan Report](#).

View Container Image Scan Report

You can review the scan report for a container image and plan your next actions. The **Image Scan Report** presents complete information on all aspects of the image scan.

Prerequisites

See [Chapter 4 Scanning Images](#).

Procedure

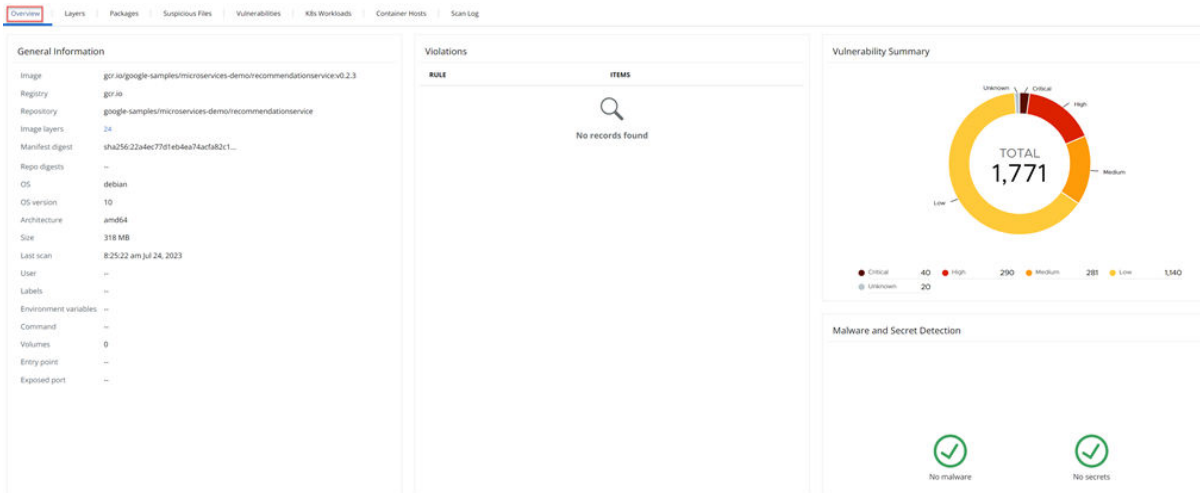
- On the left navigation pane, do one of the following depending on your system configuration and role:
 - If you have the Kubernetes Security DevOps or SecOps role and your system has only the Container security feature, click **Inventory > Container Images**.
 - If you have any other role and your system has Container security and other Carbon Black Cloud features, click **Inventory > Kubernetes > Container Images**.
- Click the **Deployed Images** tab.
- Click the name of an image in the **Image Tag** column to open the **Image Scan Report**. The **Overview** tab is opened by default.

View a Container Image Scan Report - Overview

You can review the scan report for a container image and plan your next actions. The **Image Scan Report** presents complete information on all aspects of the image scan.

Procedure

- 1 On the left navigation pane, do one of the following depending on your system configuration and role:
 - If you have the Kubernetes Security DevOps or SecOps role and your system has only the Container security feature, click **Inventory > Container Images**.
 - If you have any other role and your system has Container security and other Carbon Black Cloud features, click **Inventory > Kubernetes > Container Images**.
- 2 Click the **Deployed Images** tab.
- 3 Click the name of an image in the **Image Tag** column to open the **Image Scan Report**. The **Overview** tab is opened by default.



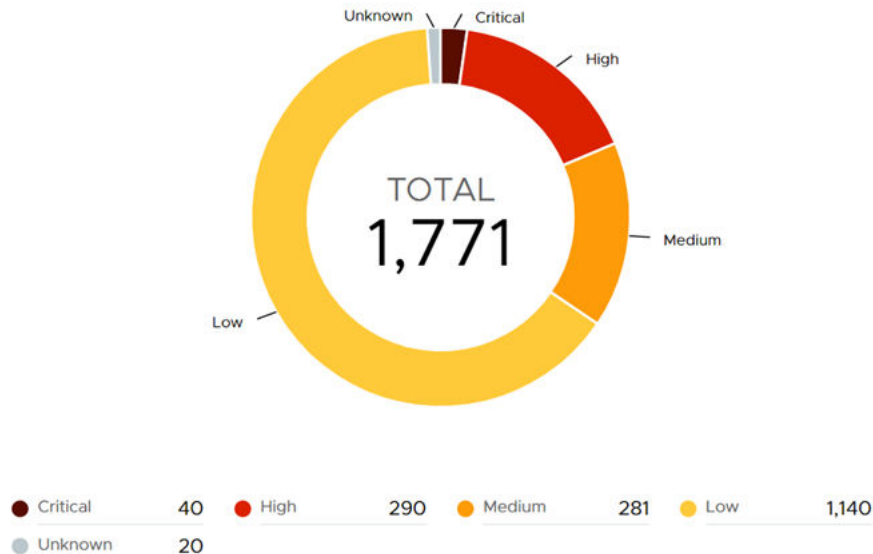
The **General Information** section lists basic container image data:

Image name	Registry	Repository
Image layers; the layers number links to the Layers tab of this report. See View a Container Image Scan Report - Layers .	Manifest digest	Repo digests
Operating system	Operating system version	Architecture
Size	Last scan date and time	User
Labels	Environmental variables	Command
Volumes	Entry point	Exposed port

The **Violations** section displays a count of violations for Kubernetes hardening policy rules, including rules for container images. The number of violations is equal to the number of CVE codes.

The **Vulnerability Summary** section displays a circular chart of discovered vulnerabilities. Hover over any section (low, medium, high, critical, or unknown) to view the number of vulnerabilities in that category. (These numbers are also displayed below the chart.)

Vulnerability Summary



The **Malware and Secret Detection** section displays files that have a suspicious or malevolent reputation, and files that contain secrets.

View a Container Image Scan Report - Layers

To view layers in a container image scan report, perform the following procedure.

Procedure


- 1 On the left navigation pane, do one of the following depending on your system configuration and role:
 - If you have the Kubernetes Security DevOps or SecOps role and your system has only the Container security feature, click **Inventory > Container Images**.
 - If you have any other role and your system has Container security and other Carbon Black Cloud features, click **Inventory > Kubernetes > Container Images**.
- 2 Click the **Deployed Images** tab.
- 3 Click the name of an image in the **Image Tag** column.
- 4 Click the **Layers** tab.

- 5 You can search for a specific layer. You can also limit the layer table results to only those layers that have vulnerabilities: deselect the check box for **Show layers with no vulnerabilities**.


LAYER	PACKAGES	VULNERABILITIES / FIXES	SIZE
CMD ["mongod"]	2	No vulnerabilities	0 B
EXPOSE 27017	2	1/1	0 B
ENTRYPOINT ["docker-entry_lorem ipsum command"]	3	No vulnerabilities	0 B
COPY file.df3353d9b2c25ef8...	1	1/0	13.2 kB
VOLUME [/data/db data/co...]	1	No vulnerabilities	0 B
mkdir -p /data/db /data/configdb...	1	3/1	0 B
set -x && export DEBIAN_FRONTEND...	2	No vulnerabilities	595 MB
ENV MONGO_VERSION=5.0.3	3	3/1	0 B
echo "deb http://\$MONGO_REPO/apt/..."	1	No vulnerabilities	72 B
ENV MONGO_MAJOR=5.0	1	No vulnerabilities	0 B
ENV MONGO_PACKAGE=mongodb...	1	No vulnerabilities	0 B

The **Layers** tab shows the following information:

- Layer name
- A `secret` or `malware` tag, if applicable
- Number of packages in the layer
- Vulnerabilities and applicable fixes
- Layer size

- 6 For more details about a layer, click the arrow  icon at the right of the layer row.

LAYER DETAILS

Layer  [ADD file:a5ec219cbfc4e0c31e7df48cc51abd9a5b92...](#)
 Layer digest sha256:ce8168f123378f7e04b085c9672717013d1d28b2aa726361bb132c...
 Packages 114
 Size 109 MB

MALWARE DETECTION

No malware found

VULNERABILITIES

	CVE	PACKAGE	FIX	EXCEPTION
>	CVE-2021-202...	libgnutls30	Yes	No
>	CVE-2021-202...	libgnutlsxx28	Yes	No
>	CVE-2021-202...	libgnutls30	Yes	No
>	CVE-2021-202...	libgnutlsxx28	Yes	No
>	CVE-2021-335...	libc-bin	Yes	No

[Show all \(329\)](#)

SECRET DETECTION

No secrets found

PACKAGES

NAME	TYPE	VERSION
adduser	deb	3.118
apt	deb	1.8.2
base-files	deb	10.3+deb10u2

In the **Layer Details** panel, you can:

- Copy the command that was used to create the image layer from the **Layer** field.
- View the layer's unique identifier in the **Layer digest** field.
- View malware.

- Show all vulnerabilities in this layer. Click **Show all** in the **Vulnerabilities** section to be directed to the **Vulnerabilities** tab. See [View a Container Image Scan Report - Vulnerabilities](#).
- View a vulnerability summary. Click the carat > icon at the left of the CVE.

CVE	PACKAGE	FIX	EXCEPTION
▼ CVE-2018-100...	multiarch-support	Yes	No
Severity 7.8 Package multiarch-support Fix 2.26-0ubuntu2.1 Description In glibc 2.26 and earlier there is confusion in the usage of getcwd() by realpath() which can be used to write before the destination buffer leading to a buffer underflow and potential code execution.			

- View secrets.
- Show all packages in this layer. Click **Show all** in the **Packages** section to be directed to the **Packages** tab. See [View a Container Image Scan Report - Packages](#).

View a Container Image Scan Report - Packages

To view packages in a container image scan report, perform the following procedure.

Procedure

- 1 On the left navigation pane, do one of the following depending on your system configuration and role:
 - If you have the Kubernetes Security DevOps or SecOps role and your system has only the Container security feature, click **Inventory > Container Images**.
 - If you have any other role and your system has Container security and other Carbon Black Cloud features, click **Inventory > Kubernetes > Container Images**.
- 2 Click the **Deployed Images** tab.
- 3 Click the name of an image in the **Image Tag** column to open the **Image Scan Report**.

4 Click the **Packages** tab.

DOCKER.IO/OCTARINESEC/E2E-HTTP:LATEST

PACKAGE/ LIBRARY ▲	TYPE	VERSION
Flask	python	0.12.2
Jinja2	python	2.10
MarkupSafe	python	1.0
PyGObject	python	3.24.1
PyYAML	python	3.12
Pygments	python	2.2.0
SecretStorage	python	2.3.1

The **Packages** tab shows the following information:

- Package and library
- Package type
- Package version

You can filter the list of displayed packages by **Type** and by **Layer**. For example, when selecting the `74fbdd4b6d6206a97532d4156e0` layer, the search result contains only the packages that belong to that layer.

View a Container Image Scan Report - Suspicious Files

To view suspicious files in a container image scan report, perform the following procedure.

Procedure

- 1 On the left navigation pane, do one of the following depending on your system configuration and role:
 - If you have the Kubernetes Security DevOps or SecOps role and your system has only the Container security feature, click **Inventory > Container Images**.
 - If you have any other role and your system has Container security and other Carbon Black Cloud features, click **Inventory > Kubernetes > Container Images**.
- 2 Click the **Deployed Images** tab.
- 3 Click the name of an image in the **Image Tag** column to open the **Image Scan Report**.

4 Click the **Suspicious Files** tab.

FILE ▾	REPUTATION ▾	HASH	TYPE ▾	EXCEPTION ▾
.eslintignore	Company Approved	fbfb5a708099dd85b8968c94b9bf68855ec4e2fefa6b101d69fe33e85add6f2a	ELF	No >
.estlintrc.js malware secret	Company Banned	643ec58e82e0272c97c2a59f6020970d881af19c0ad5029db9c958c13b6558c7	ELF	No >
.gitignore secret	Suspicious	643ec58e82e0172c43c2a59f6020970d881af19c0ad5029db9c958c13b6558c7	Script	No >
.nrcrc	Critical	3211ec58e82e0172c43c2a59f6020970d881af19c0ad5029db9c958c13b6558c7	ELF	No >
.npmignore	Company Banned	321ec58e82e0172c43c2a59f6020970d881af19c0ad5029db9c958c13b6558c7	ELF	Yes >


You can sort the list of displayed files by **File**, **Reputation**, **Type**, and **Exception**.

- 5 For more information about a file, click the arrow  icon at the right of the row.

FILE DETAILS

File	.estlintrc.js
Hash	643ec58e82e0272c97c2a59f 6020970d881af19c0ad5029d b9c958c13b6558c7
Image layer	MONGO_VERSION-5.0.3
File type	ELF

FILE REPUTATION

Reputation	Company Banned
Source	Signature feed
Description	Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore.
Techniques 	<div style="background-color: black; color: white; border-radius: 10px; padding: 2px 5px; display: inline-block;">policy_deny</div> <div style="background-color: red; color: white; border-radius: 10px; padding: 2px 5px; display: inline-block;">run_blacklist_app</div>
Exploit via CVE	--
	Find in VirusTotal

SECRET

TYPE	FILE	EXCEPTION
File	.estlintrc.js	No

View a Container Image Scan Report - Vulnerabilities

To view vulnerabilities in a container image scan report, perform the following procedure.

Procedure

- On the left navigation pane, do one of the following depending on your system configuration and role:
 - If you have the Kubernetes Security DevOps or SecOps role and your system has only the Container security feature, click **Inventory > Container Images**.

- If you have any other role and your system has Container security and other Carbon Black Cloud features, click **Inventory > Kubernetes > Container Images**.
- 2 Click the **Deployed Images** tab.
 - 3 Click the name of an image in the **Image Tag** column to open the **Image Scan Report**.
 - 4 Click the **Vulnerabilities** tab.

SEVERITY	VULNERABILITY	TYPE	PACKAGE/LIBRARY	VERSION	FIX	EXCEPTION	NOTE
Critical	CVE-2013-7459	python	pycrypto	2.6.1	No	<input checked="" type="checkbox"/>	Add Note
Critical	CVE-2016-0718	binary	python	2.7.14	No	<input checked="" type="checkbox"/>	Add Note
Critical	CVE-2016-9983	binary	python	2.7.14	No	<input checked="" type="checkbox"/>	Add Note
Critical	CVE-2013-1088158	binary	python	2.7.14	No	<input checked="" type="checkbox"/>	Add Note
Critical	CVE-2017-15266	java-archive	jackson-databind	2.9.1	2.9.4	<input checked="" type="checkbox"/>	Add Note
Critical	CVE-2017-17485	java-archive	jackson-databind	2.9.1	2.9.4	<input checked="" type="checkbox"/>	Add Note
Critical	CVE-2017-18342	python	PyYAML	3.12	4.1	<input checked="" type="checkbox"/>	Add Note
Critical	CVE-2017-7637	java-archive	jetty-continuation	8.1.14+20131031	No	<input checked="" type="checkbox"/>	Add Note
Critical	CVE-2017-7637	java-archive	jetty-http	8.1.14+20131031	No	<input checked="" type="checkbox"/>	Add Note
Critical	CVE-2017-7637	java-archive	jetty-io	8.1.14+20131031	No	<input checked="" type="checkbox"/>	Add Note

You can filter the list of vulnerabilities by severity, available fixes, type, and layer. For example, you can view only those vulnerabilities that have a high severity, available fixes, and of type `deb`:

FILTERS Clear <<

- **Severity (1)**
 - High 5
- **Available fixes (1)**
 - Available fixes 5
- **Type (1)**
 -
 - deb 5
- **Layer (2)**
 -
 - ...54c87f74910bfe9654248e39f 4
 - ...8622768d155349d86517d13e 1

- 5 Perform your search or view all vulnerabilities. The resulting list of vulnerabilities contains the following fields:
 - **Severity level.** Container images can have multiple vulnerabilities, each with a different risk score. Based on this score, vulnerabilities are filtered on the level of severity - critical, high, medium, and low. See [Severity Scoring](#).

- **Vulnerability.** You can click any CVE tag to see more details. See [View a Container Image Scan Report - Vulnerability Details](#).
- **Type.** You can filter vulnerabilities based on the package type. For example, the `dpkg` packages on Debian Linux type.
- **Package / Library**
- **Version**
- **Available fix.** If a fix is available, you can view the package and version.
- **Exception toggle.** See [Allow an Exception for a Vulnerability](#).
- **Note.** Click **Add Note** to include a note about this vulnerability; for example, if you create an exclusion, it is useful to note the reason for the exclusion.

6 To export the vulnerability data into a CSV file, click **Export**.

View a Container Image Scan Report - Vulnerability Details

You can review details about a container vulnerability in the **Container Details** panel.

Prerequisites

Perform steps 1 through 5 in [View a Container Image Scan Report - Vulnerabilities](#).

Procedure

- 1 Select the vulnerability for which to view details. Click the CVE tag in the **Vulnerability** column.

CVE-2018-25009

Overview | Affected Images | Affected K8s Workloads | Exceptions

CVE CVE-2018-25009

Description A heap-based buffer overflow was found in libwebp in versions before 1.0.1 in getle16().

[National Vulnerability Database](#)

CVSS Vector Details		CVSS Score	
Attack complexity	Low	V3 score	9.1
Attack vector	Network	V3 exploit score	3.9
Availability impact	High	V3 impact score	5.2
Confidentiality impact	High	Vector	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:H
Integrity impact	None	V2 exploit subscore	10
Privileges required	None	V2 impact subscore	4.9
Scope	Unchanged		
User interaction	None		

The **Overview** tab opens by default. On this tab, you can view the following details:

- CVE identifier
- Description
- CVSS Vector
- CVSS Score

To view the CVE in greater detail on an external web site, click **National Vulnerability Database**. For example:

CVE-2018-25009

Name	CVE-2018-25009
Description	A heap-based buffer overflow was found in libwebp in versions before 1.0.1 in GetLE16().
Source	CVE (at NVD ; CERT , LWN , oss-sec , fuldisc , bugtraq , EDB , Metasploit , Red Hat , Ubuntu , Gentoo , SUSE bugzilla/CVE , Mageia , GitHub advisories/code/issues , web search , more)
References	DLA-2677-1 , DSA-4930-1

Vulnerable and fixed packages

The table below lists information on source packages.

Source Package	Release	Version	Status
libwebp (PTS)	buster, buster (security)	0.6.1-2+deb10u1	fixed
	bullseye	0.6.1-2.1	fixed
	bookworm, sid	1.2.4-0.1	fixed

The information below is based on the following data on fixed versions.

Package	Type	Release	Fixed Version	Urgency	Origin	Debian Bugs
libwebp	source	stretch	0.5.2-1+deb9u1		DLA-2677-1	
libwebp	source	buster	0.6.1-2+deb10u1		DSA-4930-1	
libwebp	source	(unstable)	0.6.1-2.1			

Notes

<https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=9100>
<https://chromium.googlesource.com/webm/libwebp/+95fd65970662e01cc9170c4444f5c0859a710097%5E%21/>

2 To view the images that are affected by this vulnerability, click the **Affected Images** tab.

The following information is displayed:

CVE-2018-25009
✕

Overview
Affected Images
Affected K8s Workloads
Exceptions

TAG ▼	SCAN STATUS	INITIAL SCAN ▼	VULNERABILITIES/ FIXES ▼	WORKLOADS ▼	EXCEPTIONS
docker.io/vmwareallspark/acme-load-ge n:latest	Error	Apr 18, 2023	151/150 755/748 783/30 856/836 38/36	1	0

- To view the Kubernetes workloads that are affected by this vulnerability, click the **Affected K8s Workloads** tab.

CVE-2018-25009 X

Overview | Affected Images | **Affected K8s Workloads** | Exceptions

Search

NAME ▾	RESOURCE KIND ▾	SCOPES	CLUSTER ▾	NAMESPACE ▾	HARDENING POLICY
loadgenerator	Deployment	(+3)		acme-fe	magic

On the **Affected K8s Workloads** tab, you can:

- Click the workload name to open the **Kubernetes Workloads** panel. See [View a Kubernetes Workload - Overview](#).
 - Click **Scopes** to view summary information about the associated scope.
- If the vulnerability has any exceptions, they are listed on the **Exceptions** tab. See [Allow an Exception for a Vulnerability](#).

View a Container Image Scan Report - K8s Workloads

To view Kubernetes workloads in a container image scan report, perform the following procedure.

Procedure

- On the left navigation pane, do one of the following depending on your system configuration and role:
 - If you have the Kubernetes Security DevOps or SecOps role and your system has only the Container security feature, click **Inventory > Container Images**.
 - If you have any other role and your system has Container security and other Carbon Black Cloud features, click **Inventory > Kubernetes > Container Images**.
- Click the **Deployed Images** tab.
- Click the name of an image in the **Image Tag** column to open the **Image Scan Report**.
- Click the **K8s Workloads** tab.

The Kubernetes workloads associated with the container image are listed and include the following fields:

- **Workload Name**. Click the name to open the **Workload Details** panel. See [Monitoring Kubernetes Workloads](#).
- **Resource Kind** such as DaemonSet
- **Scope**
- **Cluster** that contains the workload
- **Namespace**

- **Hardening Policy.** Click the name of the policy to view its summary.

View a Container Image Scan Report - Scan Log

To view a scan log in a container image scan report, perform the following procedure.

Procedure

- 1 On the left navigation pane, do one of the following depending on your system configuration and role:
 - If you have the Kubernetes Security DevOps or SecOps role and your system has only the Container security feature, click **Inventory > Container Images**.
 - If you have any other role and your system has Container security and other Carbon Black Cloud features, click **Inventory > Kubernetes > Container Images**.
- 2 Click the **Deployed Images** tab.
- 3 Click the name of an image in the **Image Tag** column to open the **Image Scan Report**.
- 4 Click the **Scan Log** tab.

SCAN TIME ▾	SOURCE ⓘ	IMAGE TAG ▾	NEW VULNERABILITIES ▾
5:41:23 am Aug 3, 2023	Cluster scan	docker.io/cbartfactory/kube-rbac-proxyv0.14.2 ⓘ	No new vulnerabilities
5:40:18 am Aug 3, 2023	Cluster scan	k8s.gcr.io/etcd3.5.3-0 ⓘ	No new vulnerabilities
5:40:05 am Aug 3, 2023	Cluster scan	docker.io/kinddest/kindnetdv20221004-446545d1 ⓘ	2
5:40:01 am Aug 3, 2023	Cluster scan	docker.io/kindest/local-path-provisionerv0.0.22-kind.0 ⓘ	2
5:18:35 am Aug 3, 2023	Cluster scan	k8s.gcr.io/kube-controller-managerv1.24.7 ⓘ	2
5:18:31 am Aug 3, 2023	Cluster scan	docker.io/octarinesec/octarine-operatorsam ⓘ	No new vulnerabilities
5:18:27 am Aug 3, 2023	Cluster scan	k8s.gcr.io/kube-apiserverv1.24.7 ⓘ	2
5:18:22 am Aug 3, 2023	Cluster scan	k8s.gcr.io/coredns/corednsv1.8.6 ⓘ	No new vulnerabilities
5:18:19 am Aug 3, 2023	Cluster scan	k8s.gcr.io/kube-proxyv1.24.7 ⓘ	8 23 162 11 1
5:18:12 am Aug 3, 2023	Cluster scan	k8s.gcr.io/kube-schedulerv1.24.7 ⓘ	2
4:22:13 am Aug 3, 2023	CLI	docker.io/library/alpine:3.17 ⓘ	No new vulnerabilities
4:16:38 am Aug 3, 2023	secret CLI	docker.io/library/alpine:3.17 ⓘ	8 10

- 5 Optionally specify the timeframe for the list of scan logs. The default timeframe is **All available**. The resulting list of vulnerabilities contains the following fields:

- **Scan Time**
- **Source** — what triggered the scan
- **Workloads**
- **New Vulnerabilities**

Investigate Container Image Vulnerabilities


The container image is matched against known vulnerabilities in the National Vulnerability Database. Based on your configured Kubernetes policy, you can view security vulnerabilities, discover the availability of a fix for that vulnerability, and schedule patches or updates.

Procedure

- 1 On the left navigation pane, click **Harden > Vulnerabilities**.
- 2 Click the **Container Images** tab.

The default severity filter is **Critical**. To view all vulnerabilities regardless of their severity, click **All**.

By default, you can see vulnerabilities for all the containers images that are scanned using the CLI Client. To filter vulnerabilities that are only running in the Kubernetes environment, select the **Running in Kubernetes** checkbox on the top right.




- 3 Double-click a row or click the arrow  icon at the right of the row to view the **Vulnerability Details** panel.

VULNERABILITY DETAILS

[ALAS-2021-1722](#)




Description

NSS (Network Security Services) versions prior to 3.73 or 3.68.1 ESR are vulnerable to a heap overflow when handling DER-encoded DSA or RSA-PSS signatures. Applications using NSS for handling signatures encoded within CMS, S/MIME, PKCS \#7, or PKCS \#12 are likely to be impacted. Applications using NSS for certificate validation or other TLS, X.509, OCSP or CRL functionality may be impacted, depending on how they configure NSS. *Note: This vulnerability does NOT impact Mozilla Firefox.* However, email clients and PDF viewers that use NSS for signature verification, such as Thunderbird, LibreOffice, Evolution and Evince are believed to be impacted. This vulnerability affects NSS < 3.73 and NSS < 3.68.1.

Images 4 
 Workloads 1 
 Risk [Critical \(9.8\)](#) 
 Fix 4.32.0-1.amzn2

[National Vulnerability Database](#)

In this panel, you can:

- Click the link  icon next to **Images** to open the **Affected Images** tab of the **Vulnerability** panel.
- Click the link  icon next to **Workloads** to open the **Affected K8s Workloads** tab of the **Vulnerability** panel.
- Click the link  icon next to the **Risk** category to open the **Overview** tab on the **Vulnerability** panel.

ALAS-2021-1722
✕

Overview
Affected Images
Affected K8s Workloads
Exceptions

CVE

Description

ALAS-2021-1722

Nss (network security services) versions prior to 3.73 or 3.68.1 esr are vulnerable to a heap overflow when handling der-encoded dsa or rsa-pss signatures. applications using nss for handling signatures encoded within cms, s/mime, pkcs \#7, or pkcs \#12 are likely to be impacted. applications using nss for certificate validation or other tls, x.509, ocsf or crl functionality may be impacted, depending on how they configure nss. *note: this vulnerability does not impact mozilla firefox.* however, email clients and pdf viewers that use nss for signature verification, such as thunderbird, libreoffice, evolution and evince are believed to be impacted. this vulnerability affects nss < 3.73 and nss < 3.68.1.

[National Vulnerability Database](#)

CVSS Vector Details		CVSS Score	
Attack complexity	Low	V3 score	9.8
Attack vector	Network	V3 exploit score	3.9
Availability impact	High	V3 impact score	5.9
Confidentiality impact	High	Vector	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H
Integrity impact	High	V2 exploit subscore	10
Privileges required	None	V2 impact subscore	6.4
Scope	Unchanged		
User interaction	None		

[Evaluating risk](#)

- Click the vulnerability reference tag or **National Vulnerability Database** to open a relevant external web page.

See [View a Container Image Scan Report - Vulnerability Details](#).


Allow an Exception for a Vulnerability

You can create an exception for a vulnerability for an image. The exception will be skipped by Kubernetes hardening policies.

An image can have many vulnerabilities. If you consider some of them to not incur risk for your environment, you can enable an exception for those vulnerabilities for a specific image only.

Procedure

- 1 On the left navigation pane, do one of the following depending on your system configuration and role:
 - If you have the Kubernetes Security DevOps or SecOps role and your system has only the Container security feature, click **Inventory > Container Images**.
 - If you have any other role and your system has Container security and other Carbon Black Cloud features, click **Inventory > Kubernetes > Container Images**.
- 2 Click the **Deployed Images** tab.
- 3 Click the name of an image in the **Image Tag** column to open the **Image Scan Report**.
- 4 Click the **Vulnerabilities** tab.
- 5 In the **Exception** column, toggle ON to enable the exception. Any Kubernetes hardening policy capturing this vulnerability for this image will not restrict further action.

- 6 Click **Add Note** (or if there is already a note for this vulnerability, click the **Edit**  icon to edit it). Enter the reason for the exclusion and click **Save**. This is an optional but recommended step.

Results

The rule validation for a Kubernetes hardening policy with container image rules skips the images that have exceptions.

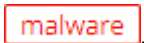
Managing and Viewing File Reputations in Container Images

A reputation is the level of trust or distrust that is given to an application. Reputations are based on multiple sources of known good and known bad reputations. There are various ways to view file reputations in your system.

If a file is suspicious or matches known malware, the file reputation service labels it as such in the Carbon Black Cloud console. Any binaries that are added to the company banned or company approved list through the SHA-256 hash are also detected and labeled as either malicious or trusted.

Important Carbon Black is replacing the terms *blacklist* and *whitelist* with *banned list* and *approved list*. Notice will be provided in advance of terminology updates to APIs, TTPs, and Reputations.

Note

- The Carbon Black Cloud console indicates images that have company banned or critical files with a malware badge .

The malware badge displays only when the Carbon Black Cloud considers the image file to be partially or fully malicious. For example, a malware badge displays for a malware hash that has been added to the company banned list. A malware badge does not display for a hash that has been blocked through a company policy.

- MD5 is not supported. The hash must be in SHA-256 format.
-

Detect Malware in a Container Image

Cluster image scanning helps you identify and classify discovered software by comparing it to an extensive database of known files.

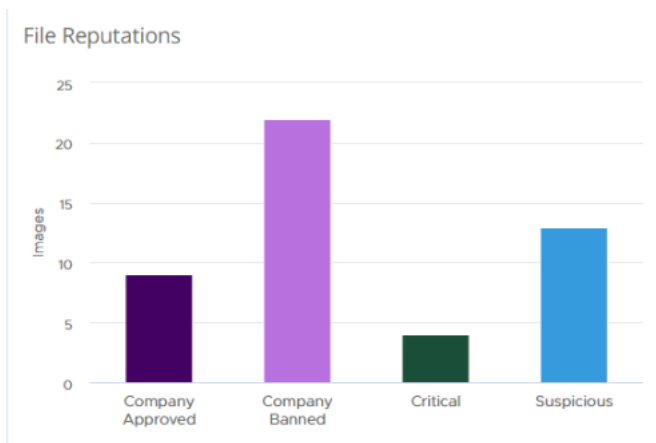
As a security admin, you can use image scanning file reputation functionality to analyze all Linux ELF files of a specific container image against a list of known malicious files.

As a DevSecOps, you can view the suspicious/malicious file reputation for all deployed container images in your cluster.

Procedure

- 1 On the left navigation pane, do one of the following depending on your system configuration and role:
 - If you have the Kubernetes Security DevOps or SecOps role and your system has only the Container security feature, click **Inventory > Container Images**.
 - If you have any other role and your system has Container security and other Carbon Black Cloud features, click **Inventory > Kubernetes > Container Images**.

The Container Images page contains a general summary of what is currently happening in your Kubernetes environment. The **File Reputations** widget summarizes all reputations for deployed container images in a bar chart.



This visualization reveals the number of images running with suspicious files and their distribution by reputation.

- **Company Approved** — indicates the file is added to the Company Approved List through the SHA-256 hash.
 - **Company Banned** — indicates the file is added to the Company Banned List through the SHA-256 hash.
 - **Critical** — indicates the file is a known malware. Cloud analytics and threat intelligence feeds determine the known malware reputation.
 - **Suspicious** — indicates the image file is a suspected malware. Cloud analytics and threat intelligence feeds determine the suspect malware reputation. The analysis cannot determine if the file is good or a malware.
- 2 To further investigate the level of trust or distrust that the file reputation assigns to the files in container images:
 - Click the **Deployed Images** tab and then click the arrow **>** icon to the right of the image row.

The **File Reputations** section in the **Image Details** panel lists all interesting files in the container image and their assigned reputations.

- In the **Deployed Images** tab, click the link under the **Image Tag** column for that container image.


The **File Reputations** widget displays in the **Overview** tab of the **Container Image** page. It shows the distribution of suspicious and malicious files for that image in a pie chart.

- In the **Container Image** page, click the **Layers** tab and double-click a layer row.

In the **File Reputations** section, you can view the filename and reputation.

Override a File Reputation in a Container Image

If there are suspicious or critical (malicious) files running in your container images, you can override their Cloud reputation by adding them either to the company approved list or to the company banned list of reputations.

Note The malware badge  displays only when the Carbon Black Cloud considers the image file to be partially or fully malicious.

You can also use the **Enforce > Reputation** page to remove or add a suspicious file's hash to the list of company approved or company banned reputations.

MD5 is not supported. The hash must be in SHA-256 format.

Procedure

- 1 On the left navigation pane, do one of the following depending on your system configuration and role:
 - If you have the Kubernetes Security DevOps or SecOps role and your system has only the Container security feature, click **Inventory > Container Images**.
 - If you have any other role and your system has Container security and other Carbon Black Cloud features, click **Inventory > Kubernetes > Container Images**.
- 2 Click the **Deployed Images** tab.
- 3 Locate a container image and click its link under the **Image Tag** column.
- 4 On the **Container Image** page, click the **Suspicious files** tab.

Only the suspicious or malicious files within the deployed container image display.

5 Double-click a file of interest.

If the file has a suspicious or critical reputation, you can add it to the list of company approved or banned hashes.

Note When the file runs within a container image and an endpoint, when you override the file reputation, it applies to the endpoint as well.

- From the **Action** drop-down menu, select either **Add Hash to approved list** or **Add Hash to banned list**.
- Optional: Enter a comment.
- Click **Add**.

It takes up to ten minutes for the feed to update.

If the file is already assigned with the Company Approved or the Company Banned reputation, you have the option to remove it from that list.

- From the **Action** dropdown menu, select **Remove hash from list**.
- Optional: Enter a comment.
- Click **Remove**.

For more information about the suspicious file using its hash, use the VirusTotal service.

- In the **File Details** panel, select **Find in VirusTotal** from the **Action** dropdown menu. You are redirected to the web site of the service.
- Observe the basic results and use them to improve your system.

Manage File Reputations for Container Images

There are multiple ways to manage file reputations. This topic describes how to perform reputation management tasks by using the **Enforce > Reputation** page.

Procedure

- On the left navigation pane, click **Enforce > Reputation**.

REPUTATION
Define the level of trust applied to specified hashes, tools, and certificates

Upload Revoke Add

Search

List All Banned List Approved List Type All Export

DATE CREATED	LIST	TYPE	VALUE	NOTE	ADDED BY
10:56:58 am Mar 2, 2023	Banned	SHA256	61617973e225a08a2ca6795a4117505a7380246d2897274076005a544836 Application name: wfs.exe	Changes generated for hash 61617973e225a08a2ca6795a4117505a7380246d2897274076005a544836	Management console user
3:31:51 am Feb 15, 2023	Approved	SHA256	99f1a4c27096275201064a285ca230f5aaf741f6d40a8afdc2044866cc0ff Application name: powershell.exe		Management console user
9:19:31 pm Jan 30, 2023	Banned	SHA256	203ae07228412654e5504622ee767e838095972e1c110a4e44a4c3958c20f5623 Application name: copy.exe	Changes generated for hash 203ae07228412654e5504622ee767e838095972e1c110a4e44a4c3958c20f5623	Management console user
4:59:47 pm Jan 30, 2023	Banned	SHA256	979f2e0a48020071a3f922113a47612a1881a7a6515281a4c2d77118172d9f Application name: setup.exe	Changes generated for hash 979f2e0a48020071a3f922113a47612a1881a7a6515281a4c2d77118172d9f	Management console user
4:50:17 pm Jan 30, 2023	Banned	SHA256	69ee487766620c15e144cc0489814379a23a6c3634e360354863733ab71304 Application name: csplayer.exe	Changes generated for hash 69ee487766620c15e144cc0489814379a23a6c3634e360354863733ab71304	Management console user

On this page, you can:

- Filter the list by **All**, **Hash**, **IT Tools**, or **Certs**.
- Upload a CSV file with a list of hashes, certificates, or IT tools. Click the **Upload** button at the top right of the page and follow the onscreen instructions.

- Add a reputation to a file. See [Adding File Reputations in Container Images](#).
- Export the reputation data to a CSV file. Click the **Export** button at the top right of the page.
- Remove a hash. Select the check box to the left of the hash and click **Remove**.
- Investigate the occurrences of a file. Click the hash value in the **Value** column.

Adding File Reputations in Container Images

This topic provides conceptual information about adding reputations to the approved list or banned list.

Using Wildcards in Paths

When adding a path, you can use wildcards to target certain files or directories.

Note Be as specific as possible when approving certs because using wildcards can lead to incidentally approving malicious software that appears to be signed by a trusted certificate authority.

Wildcard	Description	Example
*	Matches 0 or more consecutive characters up to a single subdirectory level.	C:\program files*\custom application*.exe Executable files in C:\program files\custom application\ or C:\program files(x86)\custom application\.
**	Matches a partial path across all subdirectory levels and is recursive.	C:\Python27\Lib\site-packages** Files in that directory and all its subdirectories.
?	Matches 0 or 1 character in that position.	C:\Program Files\Microsoft Visual Studio 1?.0** Files in the MS Visual Studio version 1 or versions 10-19.

Approving Files

Adding to the approved list approves the presence and actions of specified applications. Adding to the approved list is global in its effects and applies to all policies attached to a particular version of an application.

Use adding to the approved list for use cases such as: software deployment tools, executable installers, IDEs, compilers, script editors, and so on.

Carbon Black recommends that you routinely update your approved applications to account for new versions.

Benefits of Approving IT Tools and Certificates

- Minimized performance impact when IT tools drop large amounts of new code that are immediately executed.
- For IT tools, there will be no interference with new code execution. The dropped code is not blocked.

- For certs, there will be no blocking on initial execution of files that are signed with specific certificates.
- Adding to the approved list is not absolute to prevent exploitation. Deferred analysis of new code occurs in the background as it executes.

Reputations that Supersede Approved IT Tools and Certificates

- Company Black
- Company White
- Known Malware
- PUP Malware
- Suspect Malware
- Trusted White

Banning Files

Adding to the banned list prohibits the presence and actions of specified applications. Adding to the banned list is global in its effects.

Add a File to the Banned List

To add a file to the Banned List, perform the following procedure.

Note MD5 is not supported. The hash must be in SHA-256 format.

Procedure

- 1 On the left navigation pane, click **Enforce > Reputation**.
- 2 Click the **Add** button at the top right of the page.
- 3 Click **Hash** for the type.
- 4 Click **Banned List**.

- 5 Enter the SHA-256 hash of the file, the file name, and a note explaining why you are banning the file.

Add Reputation ✕

Type Hash IT Tools Certs List Approved List Banned List

* SHA-256

Blocking occurs by hash

* Name

Note

Save Cancel

- 6 Click **Save**.

Add a Reputation to the Approved List

To add a file, trusted IT tools, or certificates to the Approved List, perform the following procedure.

Procedure

- 1 On the left navigation pane, click **Enforce > Reputation**.
- 2 Click the **Add** button at the top right of the page.
- 3 Click **Approved List**.

4 Select the **Type**.

Type	Fields to Enter	Notes
Hash	<ul style="list-style-type: none"> ■ SHA-256 hash ■ Name of the file ■ Note (optional) 	<p>MD5 is not supported. The hash must be in SHA-256 format.</p> <p>Any hash added to the approved list is assigned to the <code>COMPANY_WHITE_LIST</code> with the highest priority in the reputation hierarchy. No other reputation takes precedence over this status.</p>
IT Tools	<ul style="list-style-type: none"> ■ Path of trusted IT tool ■ Select the check box next to Include all child processes to enable this option. ■ Note (optional) 	<p>Tip You can use wildcards in the path. See Using Wildcards in Paths.</p> <p>If selected, files dropped by child processes of the newly defined trusted IT tool also receive the initial trust. This option is useful when IT tools create a child process to which to delegate work, and the child process represents a generic executable such as a copy command.</p> <p>Applications added to the approved list are assigned the <code>LOCAL_WHITE</code> reputation and are not stalled for static analysis or cloud reputation when they are executed.</p>
Certs	<ul style="list-style-type: none"> ■ Certificate (Signed by) ■ Name of the Certificate Authority ■ Note (optional) 	<p>To use this functionality, a file must be signed and verified by a valid certificate.</p> <p>Certs added to the approved list are assigned the <code>LOCAL_WHITE</code> reputation and are not stalled for static analysis or cloud reputation when they are executed.</p>

5 Click **Save**.

Expiration of Approved Certificates

All certificates have a validity range that defines the time range for when the certificate is considered valid.

Background

Most digitally signed files carry both content signatures that verify that the content has not been tampered with, and a separate counter signature to verify when the file was signed.

For these files, even if the code signing certificate has expired, files signed within the validity range of the code signing certificate remains valid in terms of expiration because the counter signature timestamp allows verification that the file was signed during the certificate's valid lifetime.

Rare files that lack a counter signature/timestamp are no longer be considered valid after the certificate expires because you can no longer determine whether the file was signed during the certificate's validity period.

Certificate Revocation is a separate concept from expiration. Revocation is used to state that a previously valid certificate is no longer trustworthy, and is not trusted even if the validity time range has not expired.

How Expired Certs are Handled in Carbon Black Cloud

Carbon Black Cloud examines the file signature validity only when Carbon Black Cloud first discovers the hash. This methodology can lead to the following edge cases:

- If a non-timestamped hash was found on Machine 1 when its certificate was valid, and found by Machine 2 when it was expired, machine 1 continues to treat the file as eligible for certificate approval. Machine 2 does not treat the file as eligible, because Machine 2 first detected it as invalid/expired; Machine 1 initially saw it as valid.

Note This does not apply for timestamped files because you can verify if the file was signed during the validity range.

- If a hash was discovered before a certificate was known to be revoked, it could be approved and remains approved on that machine even if the certificate is found to be revoked later. New hashes signed by the revoked certificate that appear after sensor has realized the certificate is revoked are not approved by certificate approvals but can still be approved by other reputations.

In summary, certificate expiration and revocation can affect the reputation of new hashes that appear on a system but do not affect the hash reputation of existing hashes that are already on the asset. Machines can enforce certificate approval rules differently based on whether the certificate is expired, whether there is a counter signature, when the sensor determined that the certificate was revoked, or if different sensors have different trusted root certificate stores.

Detecting and Preventing Secrets

A secret is an object that contains a small amount of sensitive data such as a password, a token, or a key. It is often used to authenticate users or services to control access to sensitive information or external services. Secret management is an essential tool to help control and enforce the distribution of secrets across workloads. This section describes how to detect and prevent statically defined secrets that are deployed in Kubernetes environments.

Carbon Black Cloud secret management can help you detect and prevent static secrets that are injected into the workload. You can use a policy rule to detect and prevent secrets.

Note Secrets detection is disabled by default. You can enable this feature when you create or edit a cluster. See [Add a Cluster and Install the Kubernetes Sensor](#).

Data regarding secrets is available on the following pages in the Carbon Black Cloud console:

- [View Container Images - Overview](#)
- [View Deployed Container Image Details](#)
- [Detect Secrets in Containers on the Scan Log Page](#)
- [View a Container Image Scan Report - Suspicious Files](#)
- [View a Container Image Scan Report - Layers](#)
- [Prevent Secrets in Containers](#)

Roles

The following roles can use Carbon Black Cloud secret management.

DevOps and DevSecOps

- Detect and prevent statically defined secrets in containers at the image build phase.
- Inspect image information to help detect potential security or compliance violations.
- Inspect workload information to help detect potential security compliance violations.
- Deny workloads that use images with static secrets through policy to help enforce security and compliance.
- Explore and mitigate static secret policy violations.
- Include secrets in the existing explore, prioritize, and mitigate risk process.
- Detect files that contain statically defined secrets.
- Scan all deployed images for secrets.

DevOps and Developer

- Inspect image information to help detect potential security compliance violations.
- Explore and mitigate static secret policy violations.

Secret Detection

Secrets are detected in the following ways:

SECRET DETECTION

IN FILES

AWS Access Key ID (AKIA...4DM2)

in "/usr/local/sbin/acme/.ignoreme"

AWS Secret Key (bE45...dd3Q)

in "/usr/local/sbin/acme/.should_have_been_deleted"

IN ENVIRONMENT VARIABLES

Github authentication (ghs_...8hXy)

in "gitlab_auth"

Slack token (xapp-...me09)

in "hook"

IN COMMAND PARAMETERS

AWS Access Key ID (AKIA...4DM2)

in "CMD["acme-app", "be45..."]"

IN LABELS

AWS Access Key ID (AKIA...4DM2)

in "secret_id"

AWS Secret Key (bE45...dd3Q)

in "secret_key"

Data Types

The following table provides an example of the captured secret data types.

Table 5-1. Example of Captured Secrets Data

Source	Category	Secret Type	Secret Key	Secret Value
/.aws	FILE	Keyword Detector	aws_access_key_id	JKSN...3E3Q
RUN /bin/sh -c eco hi --password "pddj...f837" # buildkit	COMMAND	Keyword Detector	password	pdhj...f837
azure	LABEL	Azure Storage Account access key	azure	abcd...uv==
GITHUB_KEY	ENVIRONMENT_VARIABLE	Github authentication	GITHUB_KEY	ghu...UKpr

Secret Types

The following table lists the types of secrets that Carbon Black Cloud detects.

Azure Storage Account access key	JFrog Artifactory credentials	AWS Client ID
AWS Secret Key	Amazon Marketplace Web Service (MWS) Key	HTTP Bearer Authentication

URL with password	Github authentication	JSON Web Token
Mailchimp API Key	npm auth token	Private Key
Sendgrid API key	Slack token	Square authentication
Stripe API key	Twilio authentication	

Detect Secrets in Containers on the Scan Log Page

To detect secrets in Containers on the Scan Log page in the **Image Scan Report**, perform the following procedure.

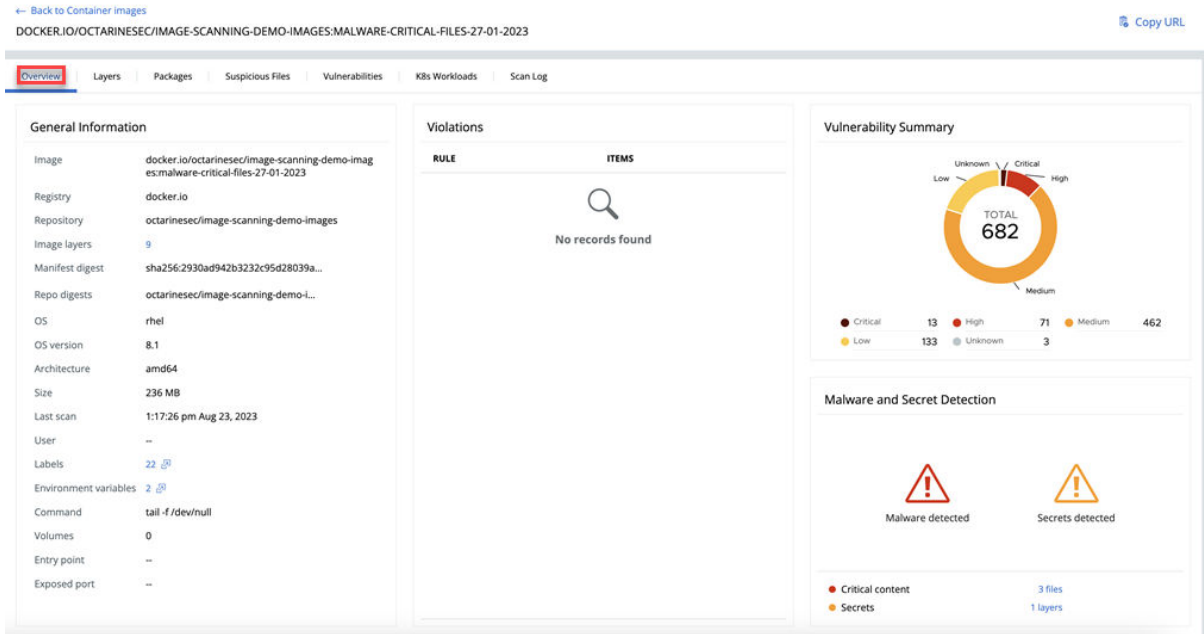
Note This topic is offered as an example of one way to view secrets in Containers. For a list of alternate pages in the Carbon Black Cloud console that present secrets data, see [Detecting and Preventing Secrets](#).

Procedure

- On the left navigation pane, do one of the following depending on your system configuration and role:
 - If you have the Kubernetes Security DevOps or SecOps role and your system has only the Container security feature, click **Inventory > Container Images**.
 - If you have any other role and your system has Container security and other Carbon Black Cloud features, click **Inventory > Kubernetes > Container Images**.
- Click the **Scan Log** tab.

SCAN TIME	SOURCE	IMAGE TAG	NEW VULNERABILITIES
5:17:26 am Aug 23, 2023	CLI	docker.io/octarinesec/image-scanning-demo-images/malware-critical-files-27-01-2023	13 71 133 462 3
4:55:16 am Aug 23, 2023	CLI	docker.io/octarinesec/image-scanning-demo-images/malware-critical-cctest-27-01-2023	No new vulnerabilities
4:53:52 am Aug 23, 2023	CLI	docker.io/octarinesec/image-scanning-demo-images/malware-critical-cctest-27-01-2023	No new vulnerabilities
4:49:57 am Aug 23, 2023	CLI	docker.io/octarinesec/image-scanning-demo-images/malware-critical-cctest-27-01-2023	13 71 133 462 3

- For any image that contains secrets, click the **Image Tag** icon. This action opens the **Overview** tab on the **Image Scan Report**.



4 You can view additional information about secrets by:

- Clicking the **Labels** icon. For example:

Labels

LABEL

maintainer="Kong Docker Maintainers <docker@konghq.com> (@team-gateway-bot)"

org.opencontainers.image.ref.name="ubuntu"

org.opencontainers.image.version="22.04"

- Clicking the **Environmental variables** icon. For example:

Environment Variables

VARIABLE

= "ASSET=ce"

KONG_VERSION="3.3.0"

PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"

Prevent Secrets in Containers

Set a policy rule to prevent secrets in Containers.


Procedure

1 On the left navigation pane, click **Enforce > K8s Policies**.

2 Click the **Hardening Policies** tab.

3 Select or create a policy to which to add the secret prevention rule.

To edit an existing policy, see [Edit a Kubernetes Hardening Policy](#). To create a new policy, see [Create a Kubernetes Hardening Policy](#).

4 On the Available rules page, scroll down to the **Secret found** rule in the **Container Images** category. This rule prevents the deployment of images that have secrets. Select **Alert** or **Block** and click the arrow  icon at the right of the rule.

The rule is added to the policy.

5 Click **Next**.

Review Violations

RULE ▲	ACTION	VIOLATIONS ▼	EXCEPTIONS
Secret found 	Block	0	No <input checked="" type="checkbox"/>

6 Click **Next**.

7 If you are creating a new policy, click **Enable Policy** or **Save as Draft**. If you are editing an existing policy, click **Save**.

Monitoring Kubernetes Workloads

You can review the risk exposure and related information for your Kubernetes workloads in the Carbon Black Cloud console.

To remediate risks and fix issues at a workload level in your Kubernetes environment, you can view the following:

- Risk severity details
- Details on applied Kubernetes hardening and runtime policies
- Policy violations for the Kubernetes hardening policy
- Alerts for the Kubernetes runtime policy
- Network connections to ingress or egress traffic

Note

- For more information on the risk severity, see [Kubernetes Risk Severity Scoring](#).
 - For more information on investigating the alerts related to a workload, see [Triaging Kubernetes Alerts](#).
-

View Kubernetes Workloads

To view and assess Kubernetes workloads, perform the following procedure.

Procedure

- On the left navigation pane, do one of the following depending on your system configuration and role:
 - If you have the Kubernetes Security DevOps or SecOps role and your system has only the Container security feature, click **Inventory > Workloads**.
 - If you have any other role and your system has Container security and other Carbon Black Cloud features, click **Inventory > Kubernetes > Workloads**.

The Kubernetes Workloads page opens.

Note If you modified a workload by enforcing values through the rule enforcement presets, that workload is shown with a `mutated` label next to its name. See [Mutate Hardening Rules](#) and [Mutate a Rule Outcome](#).

The remaining steps describe your options on this page.


- View a specific workload page — click the workload name. See [View a Kubernetes Workload - Overview](#).
- View the runtime policy that is assigned to a workload — click the runtime policy name. The **Policy Details** panel displays a summary of the runtime policy.

Policy Details
✕

Status	Enabled
Name	eks runtime policy
Scope	eks scope
Last modified	5:41:49 am Feb 14, 2023
Last modified by	

RULE ▼	ACTION
Medium risk malicious destinations ⓘ	Alert
Medium or low risk internal connections ⓘ	Alert
Medium or low risk ingress connections ⓘ	Alert
Medium or low risk egress connections ⓘ	Alert


Close

- 4 View the hardening policy that is assigned to a workload — click the hardening policy name. The **Policy Details** panel displays a summary of the hardening policy.
- 5 View the workload details — click the arrow  icon at the right of the row.

▼ **WORKLOAD DETAILS** [View more](#)

Name	aws-node
Kind	DaemonSet
Cluster	
Namespace	kube-system

▼ **RISK**



Configuration risks 11
Vulnerabilities 127

▼ **RUNTIME**

Policy	Any runtime policy
Scope	Any

ALERTS (0) [View all](#)

▼ **HARDENING**

Policy	Any hardening policy
Scope	Any

VIOLATIONS (4)

- custom** Custom - for container image
- medium** Require hash tags
- medium** Vulnerabilities with fixes
- medium** Critical vulnerabilities

ENFORCEMENTS (0)

▼ **NETWORK CONNECTIONS**

No connections within the last 2 hours

▼ **CONTAINER IMAGES (2)**

-
-

From the **Workload Details** panel, you can view:

- A specific workload page — click **View more** in the **Workload Details** section. See [View a Kubernetes Workload - Overview](#).

- The workload's configuration risks in order of severity — click the number next to **Configuration risks** in the **Risk** section.

Workload Configuration (11)


- > **High** Allow privileged container
- > **Medium** Access to host path
- > **Medium** Additional capabilities
- > **Medium** Host port
- > **Low** Memory limits

- The workload's vulnerabilities in order of severity — click the number next to **Vulnerabilities** in the **Risk** section.
- The runtime policy, the hardening policy, and associated scopes with either policy by clicking the name of the policy or scope in the **Runtime** and **Hardening** sections.
- The number of alerts that have arisen from policy violations. To view all such alerts, click **View all** in the **Runtime** section. The Alerts page opens and lists the relevant alerts. See [Triaging Kubernetes Alerts](#).
- A list of hardening policy violations and enforcements.
- Network connections within the past 2 hours.
- Container images in this workload. You can click any hyperlinked container image name to view information about that container image.

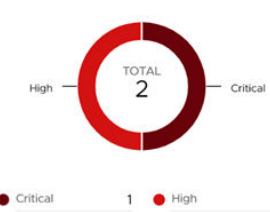
[docker://bartfactory/runtime-kubernetes-sensor-main](#) ✕

Overview | Layers | Packages | Suspicious Files | Vulnerabilities | K8s Workloads | Scan Log

General Information	
Image	docker://bartfactory/runtime-kubernetes-sensor-main
Registry	docker.io
Repository	cbartifactory/runtime-kubernetes-sensor
Image layers	25
Manifest digest	sha256:abb0c260a3ff16d0d9c1646b66aa635ccaf1d1697a7e6ba4a2a29853a473035
Repo digests	sha256:2844dd11a3503900a15ede61a511b5d571b084140e231ce649be3f26b180dfe5
OS	photon
OS version	4.0
Architecture	amd64
Size	219 MB
Last scan	8:36:34 am Mar 9, 2023

Violations	
RULE	ITEMS
 No records found	


Vulnerability Summary



TOTAL 2

● Critical 1 ● High 1

File Reputations



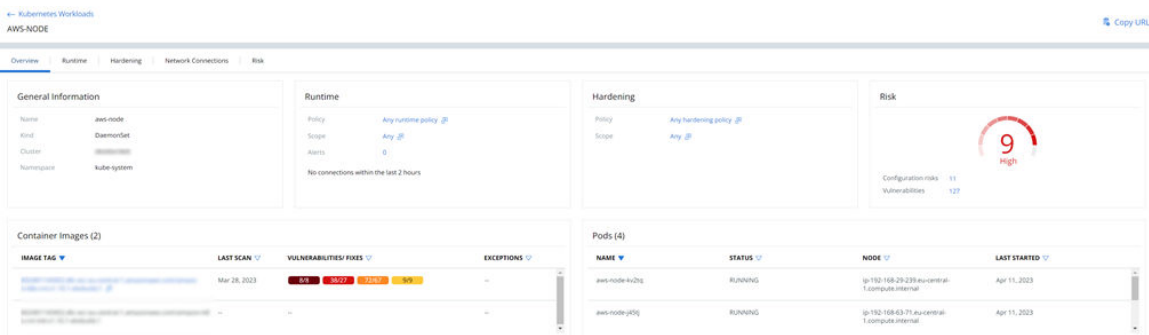
TOTAL 0

View a Kubernetes Workload - Overview

To see an overview of a Kubernetes workload, perform the following procedure.

Procedure

- On the left navigation pane, do one of the following depending on your system configuration and role:
 - If you have the Kubernetes Security DevOps or SecOps role and your system has only the Container security feature, click **Inventory > Workloads**.
 - If you have any other role and your system has Container security and other Carbon Black Cloud features, click **Inventory > Kubernetes > Workloads**.
- Click the name of the Workload in the second column.
 - The **Overview** tab shows the following details:



- General information** — name, kind, cluster, and namespace.
- Runtime** — The assigned runtime policy and scope. You can click the policy or scope name for additional details. This section lists any alerts associated with the runtime policy, and shows network connections within the last 2 hours.
- Hardening** — The assigned hardening policy and scope. You can click the policy or scope name for additional details.
- Risk** — This section shows the overall risk severity, configuration risks, and vulnerabilities. To go to the **Risk** tab for more information, click the number next to **Configuration risks** or **Vulnerabilities**. See [View a Kubernetes Workload - Risks](#).
- Container Images** — Lists the container images in the workload. You can click any hyperlinked container name to go to its Container Image page. See [View Container Images - Overview](#).
- Pods** — Lists the pod name, status, node, and last started date for the associated pods.

View a Kubernetes Workload - Runtime Policy

For information about the runtime policy for a Kubernetes workload, perform the following procedure.

See also [Kubernetes Runtime Policies](#).

Procedure

- 1 On the left navigation pane, do one of the following depending on your system configuration and role:
 - If you have the Kubernetes Security DevOps or SecOps role and your system has only the Container security feature, click **Inventory > Workloads**.
 - If you have any other role and your system has Container security and other Carbon Black Cloud features, click **Inventory > Kubernetes > Workloads**.
- 2 Click the hyperlinked name of the Workload in the second column.
- 3 Click the **Runtime** tab.

The **Runtime** tab shows the following runtime policy information for this workload.

- Name
- Scope
- Alerts
- Workload Baseline

The **Workload Baseline** section includes the following data:

- Remote connection
- Protocol
- Port
- Connection type
- Who added the baseline behavior
- Actions

To reset the baseline, click **Reset**. See [Kubernetes Scope Baselines for Runtime Policies](#).

View a Kubernetes Workload - Hardening Policy

For information about the hardening policy for a Kubernetes workload, perform the following procedure.

See also [Kubernetes Hardening Policies](#).

Procedure

- 1 On the left navigation pane, do one of the following depending on your system configuration and role:
 - If you have the Kubernetes Security DevOps or SecOps role and your system has only the Container security feature, click **Inventory > Workloads**.

- If you have any other role and your system has Container security and other Carbon Black Cloud features, click **Inventory > Kubernetes > Workloads**.
- 2 Click the hyperlinked name of the Workload in the second column.
 - 3 Click the **Hardening** tab.

The **Hardening** tab shows the following hardening policy information for this workload.

- Name
- Scope
- Rule Compliance

STATUS ▾	RULE ▾	CATEGORY
Violation	Require hash tags ⓘ	Container images
Violation	Custom - for container image ⓘ	Container images
Violation	Vulnerabilities with fixes ⓘ	Container images
Violation	Critical vulnerabilities ⓘ	Container images
Compliant	Image not scanned ⓘ	Container images
Compliant	Exec to container ⓘ	Command
Compliant	Deny latest tag ⓘ	Container images
Compliant	Port forward ⓘ	Command

Category Select ^
 Custom
 Container images
 Workload security
 Network
 Quota
 RBAC
 Volume
 Command
 CRD

In the **Rule Compliance** section, you can select specific categories to view or you can view all categories.

View a Kubernetes Workload - Network Connections

For network connection information related to a Kubernetes workload, perform the following procedure.

See also [Analyzing Network Activity](#) and [Egress Groups](#).

Procedure

- 1 On the left navigation pane, do one of the following depending on your system configuration and role:
 - If you have the Kubernetes Security DevOps or SecOps role and your system has only the Container security feature, click **Inventory > Workloads**.
 - If you have any other role and your system has Container security and other Carbon Black Cloud features, click **Inventory > Kubernetes > Workloads**.
- 2 Click the hyperlinked name of the Workload in the second column.
- 3 Click the **Network Connections** tab.

You can filter the list of connections by:

- Egress
- Ingress
- Internal

- Outbound cross-namespace
- Inbound cross-namespace

You can also specify whether to show **Public destinations**, **Private destinations**, or both.

The following fields display for the selected connections:

- Destination
- Egress Group
- Port
- Protocol

View a Kubernetes Workload - Risks

To see the risks associated with a Kubernetes workload, perform the following procedure.

See also [Kubernetes Risk Severity Scoring](#) and [Investigate Container Image Vulnerabilities](#).

Procedure

- 1 On the left navigation pane, do one of the following depending on your system configuration and role:
 - If you have the Kubernetes Security DevOps or SecOps role and your system has only the Container security feature, click **Inventory > Workloads**.
 - If you have any other role and your system has Container security and other Carbon Black Cloud features, click **Inventory > Kubernetes > Workloads**.
- 2 Click the hyperlinked name of the Workload in the second column.
- 3 Click the **Risk** tab.

RISK	VULNERABILITY	TYPE	PACKAGE/LIBRARY	VERSION	FIX	AFFECTED IMAGES
Critical	ALAS-2021-1722	rpm	nginx	4.25.0-2.amzn2	4.32.0-1.amzn2	
Critical	ALAS-2021-1722	rpm	msi	3.53.1-7.amzn2	3.67.0-4.amzn2.0.1	
Critical	ALAS-2021-1722	rpm	msi-softhsm-free64	3.53.1-6.amzn2	3.67.0-3.amzn2	

The following sections provide risk assessments and related information.

- **Risk Severity** — Summarizes the risk severity associated with this workload.
- **Workload Configuration** — Lists the workload configuration risks in order of risk severity.

- **Vulnerabilities** — Lists the following details for vulnerabilities of this workload. You can search for a particular package or CVE to display in the table, and you can filter the list by severity.
 - Risk severity
 - Vulnerability name. Click on this hyperlink to view an overview of the vulnerability. In this panel, you can view all affected images, workloads, and exceptions.

ALAS-2021-1722
✕

Overview
Affected Images
Affected K8s Workloads
Exceptions

CVE ALAS-2021-1722

Description Nss (network security services) versions prior to 3.73 or 3.68.1 esr are vulnerable to a heap overflow when handling der-encoded dsa or rsa-pss signatures. applications using nss for handling signatures encoded within cms, s/mime, pkcs \#7, or pkcs \#12 are likely to be impacted. applications using nss for certificate validation or other tls, x.509, ocsf or crl functionality may be impacted, depending on how they configure nss. *note: this vulnerability does not impact mozilla firefox.* however, email clients and pdf viewers that use nss for signature verification, such as thunderbird, libreoffice, evolution and evince are believed to be impacted. this vulnerability affects nss < 3.73 and nss < 3.68.1.

[National Vulnerability Database](#)

CVSS Vector Details		CVSS Score	
Attack complexity	Low	V3 score	9.8
Attack vector	Network	V3 exploit score	3.9
Availability impact	High	V3 impact score	5.9
Confidentiality impact	High	Vector	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H
Integrity impact	High	V2 exploit subscore	10
Privileges required	None	V2 impact subscore	6.4
Scope	Unchanged		
User interaction	None		

[Evaluating risk](#)

- Type
- Package or library
- Fix, if available
- Affected images. Click any image name to open the related Container Image page.

View a Kubernetes Workload - Behavior Models

To see the behavior models for a Kubernetes workload, perform the following procedure.

Procedure

- 1 On the left navigation pane, do one of the following depending on your system configuration and role:
 - If you have the Kubernetes Security DevOps or SecOps role and your system has only the Container security feature, click **Inventory > Workloads**.
 - If you have any other role and your system has Container security and other Carbon Black Cloud features, click **Inventory > Kubernetes > Workloads**.
- 2 Click the hyperlinked name of the Workload in the second column.

3 Click **Behavior Models**.

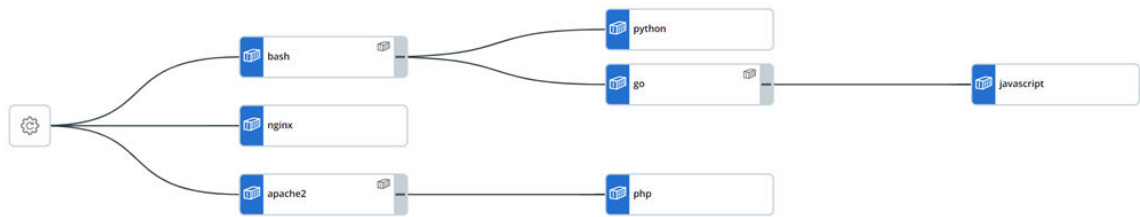
The Behavior Models page contains four tabs: **Network model**, **Process activity model**, **File access model**, and **Resource usage model**.

- **Network model**

This tab displays the process name, traffic direction, remote host, remote group, port, and protocol of the transmission.

- **Process activity model**

This tab displays a process tree. For example:




- **File access model**

Displays process, file, and access type.

- **Resource usage model**

Kubernetes Virtual Workloads

Kubernetes provides workload resources that manage a set of pods on your behalf. These resources configure controllers for running the right number and kind of pods to match a desired cluster state.

Some applications do not use the workload controllers in Kubernetes. They overload the Carbon Black Cloud backend and intensify your user experience with a high volume of objects that are otherwise hidden. To manage the desired state of your cluster, Carbon Black Cloud automatically applies a virtual workload logic by grouping pods that are not spawned through the native Kubernetes controllers. A virtual workload behaves like any native workload. If there are virtual workloads in your system, they are labeled as such in the **Inventory > Kubernetes > Workloads** page by the  icon in their names.

Analyzing Network Activity

You can view and analyze the network activity of your Kubernetes clusters in the Carbon Black Cloud console. The network map is a graphic representation of all the namespaces and workloads running in the cluster with their network traffic.

The Network map helps identify alerts that originated from workload and network activity — these are highlighted on the map for easy usage. The network map displays a high-level overview of the cluster ingress and egress connections as well as narrowing the focus to individual namespaces and workloads. The map lets you select a namespace, workload, or ingress or egress group, and shows traffic and related details including network security violations of a workload. The map focuses on data collected over the last 24 hours.

Note You can view tabular network data for a workload as well as seeing this activity in the network map. See [View a Kubernetes Workload - Network Connections](#).

In the Carbon Black Cloud console, you can see how Kubernetes workloads are exposed to the Internet through either NodePort services or Load Balancer services ingress types. For more information about ingress, see [Ingress](#) (external link).

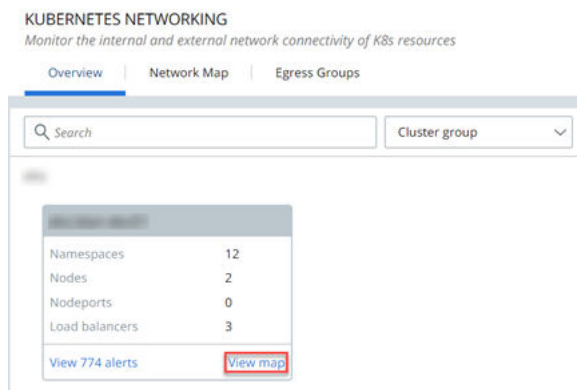
Egress traffic is the traffic going from the cluster to another network (public or private). In the Carbon Black Cloud console, you can see the outgoing traffic from the cluster to egress groups. The default egress groups are **public** and **private**. You can create additional egress groups. See [Egress Groups](#).

Investigate Cluster Activity in the Network Map

You can observe your Kubernetes clusters activity by using the interactive network map. You can select the map's focus — ingress channel, egress group, namespace, or workload.

Procedure

- On the left navigation pane, do one of the following depending on your system configuration and role:
 - If you have the Kubernetes Security DevOps or SecOps role and your system has only the Container security feature, click **Inventory > Network**.
 - If you have any other role and your system has Container security and other Carbon Black Cloud features, click **Inventory > Kubernetes > Network**.
- On the **Overview** tab, select the cluster to monitor and click **View map**.



- The **Network Map** tab becomes active and loads the data for the selected cluster.



- The left side of the map shows the ingress resources that are available for the cluster — **NodePort** services, **Load Balancer** services, or all. To filter the map for a specific ingress resource, select the graphical element on the left of the page for that ingress resource; for example, **LoadBalancers**.
- The right side of the map shows the egress groups. To filter the map for a specific egress group, select the graphical element for that group; for example, **Public**.
- To review the cluster details, the Carbon Black Cloud Kubernetes sensor version, and the resources allocated to the cluster, see the cluster details panel to the right of the map.

▼ **CLUSTER DETAILS**

Name	[REDACTED]
Cluster group	eks
Sensor version	2.3.0-rc2
Last updated	9:00:12 am Mar 22, 2023
CNI	AWS EKS
API server IP	10.100.0.1

▼ **RESOURCES**

Nodes	2
Workloads	45
Load balancers	3
Node ports	0

- Connection colors in the map indicate whether the connection is ingress, egress, between namespaces, or internal for a namespace. If you click a connection, its network connection details display to the right of the map. A color legend at the bottom left of the map defines each color connection.
- 3 To change the default map settings, click **Manage map settings** and toggle settings **ON** or **OFF**.

For example, to better analyze your Kubernetes network exposure to risk, you can filter out the encrypted connections and observe only the unencrypted ones.

- Toggle **View encrypted connections** **OFF**.
- Toggle **View unencrypted connections** **ON**.

Only the unencrypted connections stay visible on the network map for easier investigation.

Visualizing Namespace Data on the Network Map

The Kubernetes interactive network map displays the namespaces in the cluster and their network connections.

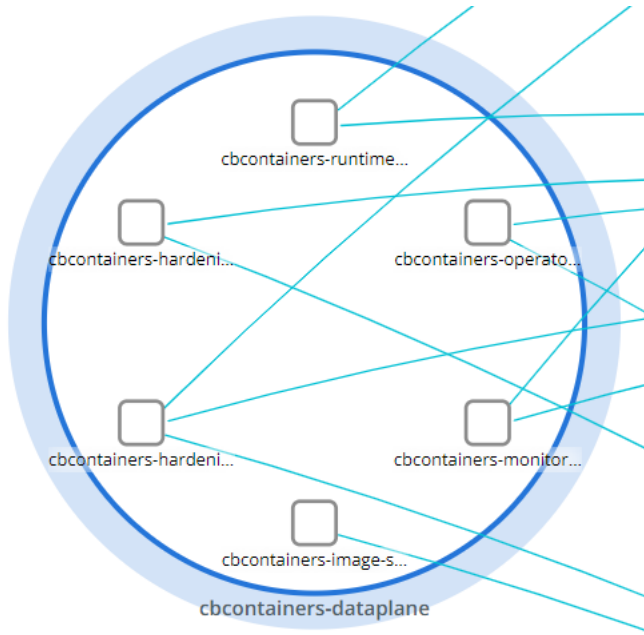
Note System namespaces are filtered out by default. To see system namespaces in the map, click **Manage map settings** and toggle **View system namespaces** **ON**.

System namespaces are:

- kube-system
- kube-public
- cbcontainers-dataplane
- vmware-system
- gatekeeper-system
- tanzu-system
- tanzu-observability-saas

To view more information for a namespace, click its visual representation in the map.

The map graphically displays the selected namespace and shows all the workloads running in it. For example:



Within the map, you can:

- Click anywhere in the white space of the map to view cluster details and resources.
- Click any line to view alerted network connections.
- Click a workload to view its data. See [Visualizing Workloads Data on the Network Map](#).

Namespace Details Panel

The panel to the right of the map provides detailed information on all egress and ingress connections for that namespace, inbound and outbound cross namespace traffic going to and from the namespace, and internal traffic inside the namespace.

▼ RUNTIME

Policy [eks runtime policy](#) 

Scope [eks scope](#) 

ALERTS (530)

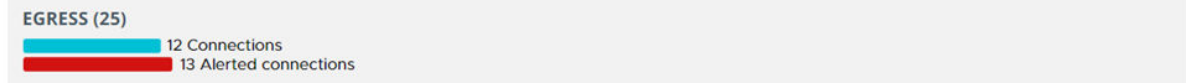
[View all](#)

medium	224	Allowed public destinations
medium	155	Allowed private destinations
medium	60	Medium or low risk internal connections
low	90	Medium or low risk egress connections

[Show all >](#)











▼ NETWORK CONNECTIONS

Data from the last 24 hours



TOP CONNECTIONS

[View all](#) 

▸ cbcontainers-hardening-enforcer	 Private/ 192.168.129.230
▸ cbcontainers-monitor	 Private/ 192.168.115.174
▸ cbcontainers-runtime-resolver	 Private/ 192.168.129.230
▸ cbcontainers-operator	 Private/ 192.168.115.174
▸ cbcontainers-image-scanning-reporter	 Public/ defense-dev01.cbctest.io
▸ cbcontainers-hardening-state-reporter	 Public/ defense-dev01.cbctest.io
▸ cbcontainers-hardening-enforcer	 Public/ defense-dev01.cbctest.io
▸ cbcontainers-runtime-resolver	 Carbon Black/ runtime.events.octarine-cp.dev.containers.carbonblack.io
▸ cbcontainers-monitor	 Carbon Black/ events.octarine-cp.dev.containers.carbonblack.io
▸ cbcontainers-hardening-state-reporter	 Carbon Black/ events.octarine-cp.dev.containers.carbonblack.io

Alerts are indicated in the following ways:

- In the **Runtime** section.
- In the bar chart in the **Network Connections** section. Alert results from the last 24 hours are included.
- In the map, an alerted connection is indicated by an exclamation mark icon on its edge.

The panel offers the following views:

- To view the associated runtime policy, click the hyperlinked policy name. Similarly, you can view scope summary details by clicking the hyperlinked scope name.
- Clicking **View all** in the **Runtime** section of the panel opens the Alerts page, which shows the network connection alerts for this namespace.

- To view additional network data, click **View all** in the **Network Connections** section of the panel:

Network Connections - cbcontainers-dataplane ✕

Ingress (0)
0 Connections
0 Alerted connections

Egress (12)
0 Connections
12 Alerted connections

Inbound (0)
0 Connections
0 Alerted connections

Outbound (0)
0 Connections
0 Alerted connections

Internal (0)
0 Connections
0 Alerted connections

Public Private Alerts only [Export](#)

SOURCE	DESTINATION	EGRESS GROUP	PORT	PROTOCOL	ALERTS
cbcontainers-hardening-enforcer	192.168.176.126	Private	443	TLS 1.3	<div style="display: flex; gap: 5px;"> 5 Allowed private destinations 4 Medium or low risk egress connections </div>
cbcontainers-monitor	192.168.144.162	Private	443	TLS 1.3	<div style="display: flex; gap: 5px;"> 5 Allowed private destinations 4 Medium or low risk egress connections </div>
cbcontainers-runtime-resolver	192.168.144.162	Private	443	TLS 1.3	<div style="display: flex; gap: 5px;"> 5 Allowed private destinations 4 Medium or low risk egress connections </div>
cbcontainers-operator	192.168.144.162	Private	443	TLS 1.3	<div style="display: flex; gap: 5px;"> 5 Allowed private destinations </div>
cbcontainers-image-scanning-reporter	defense-dev01.cbctest.io	Public	443	TLS 1.2	<div style="display: flex; gap: 5px;"> 5 Allowed public destinations </div>

Showing 1-12 of 12 Items per page Jump to page < 1 >

In this panel, you can:

- View ingress, egress, inbound, outbound, and internal network connections.
- Search for specific network connections
- Filter table results. For example, in the **Egress** tab, you can filter results by **Public**, **Private**, or **Alerts only**.
- Export the network connection data into a CSV file; for example:

```
{
  "num_found": 12,
  "results": [
    {
      "alerts": [
        {
          "action": "ALERT",
          "rule_id": "ae7b3c4e-4b6b-47fc-847c-8d0c8929c23f",
          "rule_name": "Medium or low risk egress connections",
          "severity": 4
        },
        {
          "action": "ALERT",
          "rule_id": "ec912f66-57c1-466b-b597-1d4a4ce1429c",
          "rule_name": "Allowed private destinations",
          "severity": 5
        }
      ],
      "destination_name": "192.168.176.126",
      "destination_parent_name": "Private",
      "is_private": true,
      "port": 443,
      "protocol": "TLS 1.3",
      "source_name": "cbcontainers-hardening-enforcer",
      "source_parent_name": "cbcontainers-dataplane",
      "type": "EGRESS"
    }
  ]
}
```

Visualizing Workloads Data on the Network Map

The Kubernetes interactive network map displays workloads in the cluster.

To view detailed information for a workload, click the respective visual element in the map. This will be a workload element in a namespace.

Tip You can also select a workload by clicking its name in the **Workload** dropdown menu above the map.

The map displays only the specified workload. The panel to the right of the map provides a summary of the workload data.

∨ **K8S WORKLOAD**

Name	loadgenerator
Kind	Deployment
Cluster	██████████
Namespace	acme-fe

[View more](#)

∨ **RUNTIME**

Policy	eks runtime policy
Scope	eks scope

ALERTS (4)

medium	3	Medium or low risk internal connections
<hr/>		
low	1	Medium or low risk ingress connections

[View all](#)

∨ **HARDENING**

Policy	██████████
Scope	██████████

VIOLATIONS (1)

low	Allow privilege escalation
--	----------------------------

ENFORCEMENTS (0)

∨ **NETWORK CONNECTIONS**

Data from the last 24 hours

∨ **LoadBalancers/** loadgen ↻ loadgenerator

Protocol	HTTP
Port	8089

The panel offers the following views:

- To view all data for this workload, exit the network map and go to the specific workloads summary page by clicking **View more** next to **Workload details**.
- Clicking **View all** in the **Runtime** section of the panel opens the Alerts page, which shows alerts for this workload.
- Under **Hardening**, view the associated runtime policy summary by clicking the hyperlinked policy name. Similarly, you can view scope summary details by clicking the hyperlinked scope name.

Investigating and Remediating Container Security Issues

6

During your review of container images and vulnerabilities, you can remediate any discovered container security issues. This section describes the steps you can take to identify and remediate security issues.

Read the following topics next:

- [Exploring Kubernetes Events \(Hardening\)](#)
- [Investigating Container Events on the Investigate Page](#)
- [Investigate Containers Events on the Process Analysis Page](#)
- [Triaging Kubernetes Alerts](#)
- [Identify Available Fixes and Patches](#)

Exploring Kubernetes Events (Hardening)

Kubernetes events are reported each time a resource violates a policy. They can be grouped by policy or rule and filtered by scope, cluster, and other criteria.

You can reduce violations by:

- Resolving issues in your environment
- Creating exceptions for selected rules
- Modifying policy rules as appropriate

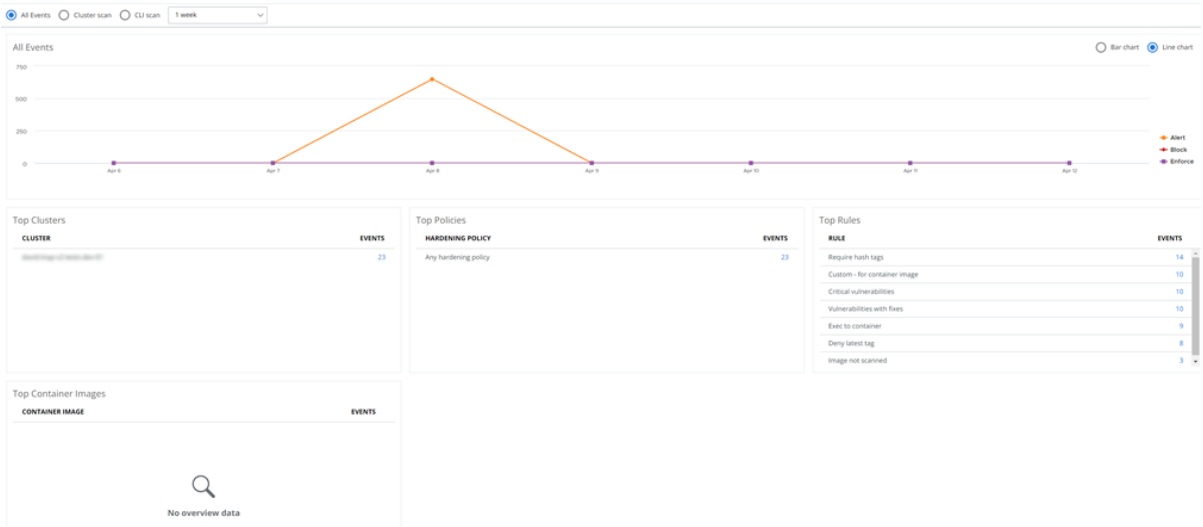
Explore Kubernetes Events - Overview

For an overview of Kubernetes events, perform the following procedure.

Procedure

- 1 On the left navigation pane, click **Harden > K8s Events**.

2 Click the **Overview** tab.



On the **Overview** tab, you can select what data you want to view and the way in which that data is presented. Your options are to view:

- All events, cluster scan events, or CLI scan events
- Events that have occurred within one week, two weeks, or one month
- Event data in a bar chart or in a line chart

You can view event details on the **Events** tab (see [Explore Kubernetes Events - Details](#)). To specify events for which to retrieve detailed information:

- To view the events for a scanned cluster, in the **Top Clusters** table, click the number in the **Events** column. The **Events** tab will open with the focus on that cluster's events.
- To view the events associated with a policy, in the **Top Policies** table, click the number in the **Events** column. The **Events** tab will open with the focus on that policy's events.
- To view the events associated with a policy rule, in the **Top Rules** table, click the number in the **Events** column. For example:

RULE	EVENTS
Require hash tags	14
Custom - for container image	10
Critical vulnerabilities	10
Vulnerabilities with fixes	10
Exec to container	9
Deny latest tag	8
Image not scanned	3

The **Events** tab will open with the focus on that rule.

- To view the events associated with a container image, in the **Top Container Images** table, click the number in the **Events** column. The **Events** tab will open with the focus on that container image's events.

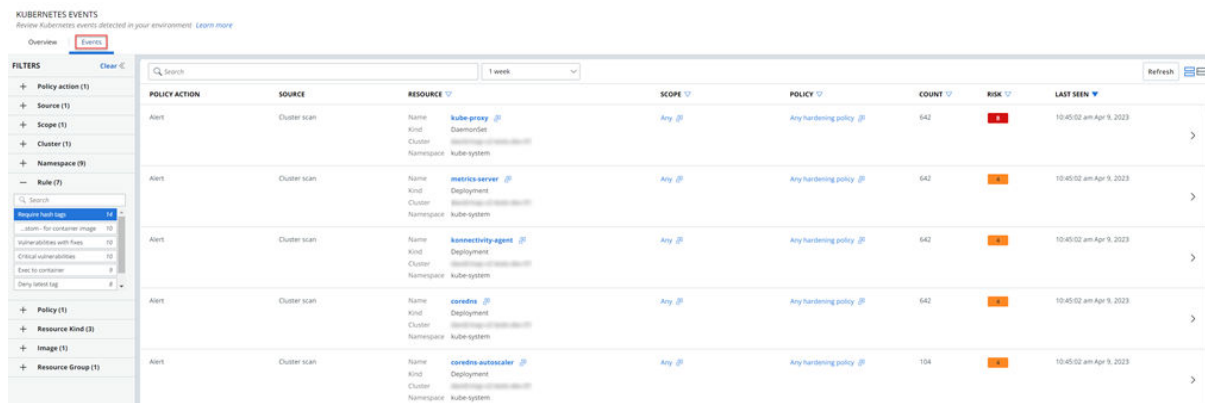
Explore Kubernetes Events - Details

To view Kubernetes event details, perform the following procedure.

Procedure

- 1 On the left navigation pane, click **Harden > K8s Events**.
- 2 Click the **Events** tab.

This page provides a list of Kubernetes events that you can review and act upon.





You can refine the list of rules in multiple ways:


- Focus rules on certain aspects (such as policy rules or container images). See [Explore Kubernetes Events - Overview](#).
- Search for events using the **Search** bar.
- Use a filter in the left panel. You can filter on multiple facets at the same time. For example, you can set your filter to only include events that match a rule and a policy.

The **Events Results** table includes the following columns by default.

Column	Description
Policy Action	The policy action that initiated the event. This is Alert, Block, or Enforce.
Source	How this event was discovered. This can be either cluster scan or CLI scan.
Resource	The Kubernetes resource type. This column includes the Name, Kind, Cluster, and Namespace data for this workload. To open a detail panel about this resource, click the link 🔗 icon next to Name .

Column	Description
Scope	The scope associated with this resource and event. To display a summary of the scope, click the link  icon next to the scope name.
Policy	The policy associated with this resource and event. To display policy details, click the link  icon next to the policy name.
Count	The number of identical events that surfaced due to the policy action.
Risk	The risk severity for this event.
Last Seen	The last time this event was discovered.

To customize these columns, click **Configure Table** in the bottom left of the page.

To view more event details, click the carat  icon at the right of the row.


▼ **EVENT DETAILS**


Policy action Alert

Type Violation

Source Cluster scan

Resource group Workload

Scope Any 

Hardening policy Any hardening policy 


User aksService

Count 642

First seen 6:21:22 am Mar 30, 2023

Last seen 10:45:02 am Apr 9, 2023




> **RESOURCE** [View more](#)

▼ **VIOLATIONS (2)** [View JSON](#) 

medium Require hash tags

medium Image not scanned

The following links are available to open additional pages:

- To display a summary of the scope, click the link  icon next to the scope name.
- To display policy details, click the link  icon next to the policy name.
- To open the Workloads page, in the **Resource** section, click **View more**.
- To view the violating resource details in JSON format, in the **Violations** section, click the link  icon next to **View JSON**. For example:

Violations JSON



```
1 {
2   "apiVersion": "apps/v1",
3   "kind": "DaemonSet",
4   "metadata": {
5     "annotations": {
6       "deprecated.daemonset.template.generation": "13",
7       "kubectrl.kubernetes.io/last-applied-configuration": "{
8         \"apiVersion\": \"apps/v1\", \"kind\": \"DaemonSet\", \"metadata\": {
9           \"annotations\": {}, \"labels\": {
10            \"addonmanager.kubernetes.io/mode\": \"Reconcile\", \"component\": \"kube-
11            proxy\", \"tier\": \"node\"}, \"name\": \"kube-proxy\", \"namespace\": \"kube-system\"}, \"spec\": {
12            \"selector\": {
13              \"matchLabels\": {
14                \"component\": \"kube-proxy\", \"tier\": \"node\"},
15              \"template\": {
16                \"metadata\": {
17                  \"annotations\": {
18                    \"nodeAffinity\": {
19                      \"requiredDuringSchedulingIgnoredDuringExecution\": {
20                        \"nodeSelectorTerms\": {
21                          [
22                            {
23                              \"matchExpressions\": [
24                                {
25                                  \"key\": \"kubernetes.azure.com/cluster\", \"operator\": \"Exists\"},
26                                  {
27                                    \"key\": \"type\", \"operator\": \"NotIn\", \"values\": [\"virtual-kubelet\"],
28                                  {
29                                    \"key\": \"kubernetes.io/os\", \"operator\": \"In\", \"values\": [\"linux\"]}]}]}],
30                                \"containers\": [
31                                  {
32                                    \"command\": [
33                                      [\"kube-proxy\", \"--contrack-max-per-core=0\", \"--metrics-bind-address=0.0.0:10249\", \"--
34                                      kubeconfig=/var/lib/kubelet/kubeconfig\", \"--cluster-cidr=10.244.0.0/16\", \"--detect-local-
35                                      mode=ClusterCIDR\", \"--pod-interface-name-prefix=\", \"--
36                                      v=3\"],
37                                    \"image\": \"mcr.microsoft.com/oss/kubernetes/kube-proxy:v1.24.9-hotfix.20230208.1\", \"name\": \"kube-
38                                    proxy\", \"resources\": {
39                                      \"requests\": {
40                                        \"cpu\": \"100m\"}, \"securityContext\": {
41                                        \"privileged\": true, \"volumeMounts\": [
42                                          [
43                                            {
44                                              \"mountPath\": \"/var/lib/kubelet\", \"name\": \"kubeconfig\", \"readOnly\": true,
45                                              {
46                                                \"mountPath\": \"/etc/kubernetes/certs\", \"name\": \"certificates\", \"readOnly\": true,
47                                              {
48                                                \"mountPath\": \"/run/xtables.lock\", \"name\": \"iptableslock\"},
49                                              {
50                                                \"mountPath\": \"/lib/modules\", \"name\": \"modules\"}], \"hostNetwork\": true, \"initContainers\": [
51                                                {
52                                                  \"command\": [
53                                                    [\"/bin/sh\", \"-c\", \"SYSCTL=/proc/sys/net/netfilter/nf_contrack_max\\necho \\\"Current
54                                                    net.netfilter.nf_contrack_max: $(cat $SYSCTL)\\\"\\nDESIRED=$(awk -F= '/net.netfilter.nf_contrack_max/
55                                                    {print $2} /etc/sysctl.d/999-sysctl-aks.conf)\\nif [ -z \\\"$DESIRED\\\" ]; then\\n
56                                                    DESIRED=$(32768*(nproc))\\n if [ $DESIRED -lt 131072 ]; then\\n DESIRED=131072\\n fi\\n\\n echo
57                                                    \\\"AKS custom config for net.netfilter.nf_contrack_max not set.\\\"\\n echo \\\"Setting nf_contrack_max to
58                                                    $DESIRED (32768 * $(nproc) cores, minimum 131072).\\\"\\n\\n echo $DESIRED \\u003e $SYSCTL\\nelse\\n echo
59                                                    \\\"AKS custom config for net.netfilter.nf_contrack_max set to $DESIRED.\\\"\\n echo \\\"Setting
60                                                    nf_contrack_max to $DESIRED.\\\"\\n\\n echo $DESIRED \\u003e
61                                                    $SYSCTL\\nfi\\n\"], \"image\": \"mcr.microsoft.com/oss/kubernetes/kube-proxy:v1.24.9-
62                                                    hotfix.20230208.1\", \"name\": \"kube-proxy-bootstrap\", \"resources\": {
63                                                    \"requests\": {
64                                                      \"cpu\": \"100m\"}, \"securityContext\": {
65                                                      \"privileged\": true, \"volumeMounts\": [
66                                                        [
67                                                          {
68                                                            \"mountPath\": \"/etc/sysctl.d\", \"name\": \"sysctl\"},
69                                                          {
70                                                            \"mountPath\": \"/lib/modules\", \"name\": \"modules\"}], \"priorityClassName\": \"system-node-
71                                                            critical\", \"serviceAccountName\": \"kube-proxy\", \"tolerations\": [
72                                                            {
73                                                              \"key\": \"CriticalAddonsOnly\", \"operator\": \"Exists\"}, {
74                                                              \"effect\": \"NoExecute\", \"operator\": \"Exists\"},
75                                                            {
76                                                              \"effect\": \"NoSchedule\", \"operator\": \"Exists\"}, \"volumes\": {
77                                                              \"hostPath\": {
78                                                                \"path\": \"/var/lib/kubelet\", \"name\": \"kubeconfig\"}, {
79                                                                \"path\": \"/etc/kubernetes/certs\", \"name\": \"certificates\"}, {
80                                                                \"path\": \"/run/xtables.lock\", \"type\": \"FileOrCreate\"}, {
81                                                                \"name\": \"iptableslock\"}, {
82                                                                \"path\": \"/etc/sysctl.d\", \"type\": \"Directory\"}, {
83                                                                \"name\": \"sysctl\"}, {
84                                                                \"path\": \"/lib/modules\", \"type\": \"Directory\"}, {
85                                                                \"name\": \"modules\"}], \"updateStrategy\": {
86                                                                \"rollingUpdate\": {
87                                                                  \"maxUnavailable\": 1, \"type\": \"RollingUpdate\"}}]}]}]
88          },
89          \"resourceVersion\": \"56993028\",
90          \"name\": \"kube-proxy\",
91          \"uid\": \"3ea49e41-fa7d-4ae2-8678-eb4433517574\",
92          \"creationTimestamp\": \"2022-07-06T11:25:00Z\",
93          \"generation\": 13,
94          \"managedFields\": [
95            {
96              \"apiVersion\": \"apps/v1\",
97              \"fieldsType\": \"FieldsV1\",
98              \"fieldsV1\": {
99                \"metadata\": {
100                  \"f:annotations\": {
101                    \":\": {},
102                    \"f:deprecated.daemonset.template.generation\": {},
103                    \"f:kubectrl.kubernetes.io/last-applied-configuration\": {}
104                  },
105                },
106              },
107            ],
108          },
109        },
110      ],
111    },
112  },
113}
```

Investigating Container Events on the Investigate Page

This section describes how to investigate Container and Kubernetes events on the Investigate page in the Carbon Black Cloud console.

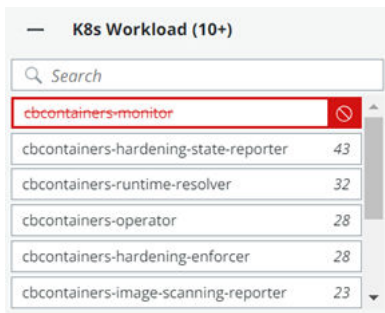
Note This content is specific to Containers and Kubernetes. For additional documentation that more broadly describes the Investigate page in the Carbon Black Cloud console, see [Investigating Events](#) in the main *Investigate* section of the *VMware Carbon Black Cloud User Guide*.

The Investigate page offers five ways to filter events for Containers and Kubernetes:

- **Container**
- **Container Image**
- **Kubernetes Cluster**
- **Kubernetes Namespace**
- **Kubernetes Workload**

You can combine filters to achieve a particular result.

- Click the vertical 3-dot **Configuration** menu to configure the filters that display in the console.
- You can exclude search results by clicking the **Exclude** icon to the right of a filter value. For example:



Note

- For a list of Container and Kubernetes event and search fields, see the following tables.
- For a full list of all available Search fields, open the in-product *Search Guide* in the upper right corner of the Investigate page.

Container Fields

Table 6-1. Container Fields in Alphabetical Order

Field Name	Description	Searchable?	Example
Container Annotations	A key-value list of arbitrary metadata that is assigned to the container by the container admin.	No	"com.example.gpu-cores": "2"
Container Engine	The engine that runs the container: Containerd, Docker, or CRI-O.	No	Docker
Container Engine Version	The version of the container engine.	No	1
Container ID	ID of the container .	Yes	f78375b1c487e03c9438c729 345e54db9d20cfa2ac1fc349 4b6eb60872e74778

Table 6-1. Container Fields in Alphabetical Order (continued)

Field Name	Description	Searchable?	Example
Container Image Hash	SHA-256 hash of the container image.	Yes	sha256:83d3456789b9a85b98bd162f1ec4d7bc1942f0035caed0f80b3b98a3eab225a7dc
Container Image Name	Name of the container image. Images are static files with executable code that can create containers.	Yes	docker.io/alpine:latest
Container IP Address	IP address assigned to the container.	No	192.168.23.100
Container Name	Name of the container; names are typically generated by runtime engines or by platforms. For example, Kubernetes.	Yes	cbcontainers-node-agent
Container Process PID	Container process identifier that is assigned by the operating system; can be multi-valued in case of fork() or exec() process operations on Linux.	Yes	2134
Container Root Path	The host's path of the container image.	No	root@someworkloadname-67cf888bcd-gk4jl
Entry Point	The command that is executed when the container is started.	No	/bin/nginx -c /etc/nginx/config.json
Host Name	Container's host name.	No	
Host Process PID	Host's process PID.	Yes	2345
Mount List	List of the container's mounted volumes.	No	
Mount Name	Name of the container's mount.	No	mylib
Mount Read/Write	Type of access to the mounted file or directory. Write access allows modifying files on the node.	No	RW
Mount Source Path	A device name, file, or directory name at the container's host.	No	/var/lib/somedirectory
Mount Target Path	Destination of mount point: the path inside container.	No	/lib/somedirectory

Table 6-1. Container Fields in Alphabetical Order (continued)

Field Name	Description	Searchable?	Example
Mount Type	Container's mount type, which can be bind, volume, or tempfs.	No	tempfs
Privileged Container	Defines whether privileged capability is enabled for the running container. https://github.com/opencontainers/runtime-spec/blob/main/config.md .	No	True
Start Time	Container start time.	No	

Kubernetes Fields

Table 6-2. Kubernetes Fields in Alphabetical Order

Field Name	Description	Searchable?	Example
Cluster Name	Name of the Kubernetes cluster that is associated with the alert.	Yes	ross:aks-test
Namespace	Namespace within the Kubernetes cluster that is associated with the alert.	Yes	Default, kube-system
Replica Name	Name of the pod within a workload.	Yes	example-workload-1643104800-b2t7f
Workload ID	ID of the workload within a specific cluster_name/namespace pair.	Yes	example-workload
Workload Kind	Type of workload: Pod, Deployment, Job, etc.	Yes	CronJob, Deployment, Demon Set
Workload Name	Name of the workload within a specific cluster_name/namespace pair.	Yes	example-workload

Kubernetes Network Security Fields

Table 6-3. Kubernetes Network Security Fields in Alphabetical Order

Field Name	Description	Searchable?	Example
Connection Type	Type of connection: INGRESS, EGRESS, INTERNAL_INBOUND, etc.	Yes	EGRESS
Egress Group Name	Name of the egress group.	Yes	null

Table 6-3. Kubernetes Network Security Fields in Alphabetical Order (continued)

Field Name	Description	Searchable?	Example
IP Reputation	Reputation assigned by Carbon Black Cloud; ranges 1-100 where 100 is trustworthy.	Yes	74
Port	Listening port: remote or local.	Yes	80
Protocol	Name of the protocol.	Yes	HTTP
Remote Domain	Name of the remote domain.	Yes	archive.ubuntu.com
Remote IP	IP address of the remote side of the communication.	Yes	91.189.88.152

Investigate Container Events


To investigate events associated with Containers, perform the following procedure.

Procedure

- 1 On the left navigation pane, click **Investigate > Processes**.
- 2 In the left pane, filter by **Container** or **Container Image**.
- 3 Optionally define any additional query criteria in the **Search** bar and press **Enter** to run the query.


Note

- For a list of Container and Kubernetes event and search fields, see [Investigating Container Events on the Investigate Page](#).
 - For a list of all available Search fields, open the in-product *Search Guide* in the upper right corner of the page.
-

- 4 For details about a specific event in the results table, click the arrow  icon at the right of the row.

The **Container** section in the right panel shows the following details:

CONTAINER

Name	de88728ddb5e357d7ec5865f81ce4fe3b34cee0f7c038ff8844e2ec9025d5a...
Container ID	de88728ddb5e357d7ec5865f81ce4fe3b34cee0f7c038ff8844e2ec9025d5ae3
Start time	6:07:07 am Jul 24, 2023
Stop time	7:18:24 pm Jul 25, 2023
Status	Stopped
Image	docker.io/octarinesec/cndr:skostov
Mounts	21 
Root path	rootfs

Note For more information about the **Event Details** panel, see [Investigate - Processes](#) in the main *Investigate* section of the *VMware Carbon Black Cloud User Guide*.

Investigate Kubernetes Clusters


To investigate events associated with Kubernetes clusters, perform the following procedure.

Procedure

- 1 On the left navigation pane, click **Investigate > Process**.
- 2 In the left pane, filter by **K8s Cluster**.
- 3 Optionally define any additional query criteria in the **Search** bar and press **Enter** to run the query.

Note

- For a list of Container and Kubernetes event and search fields, see [Investigating Container Events on the Investigate Page](#).
 - For a list of all available Search fields, open the in-product *Search Guide* in the upper right corner of the page.
-

- For details about a specific event in the results table, click the arrow  icon at the right of the row.

K8S WORKLOAD

[View more](#)

Name	csi-azuredisk-node
Kind	DaemonSet
Cluster	tomer:azure
Namespace	kube-system
Pod name	csi-azuredisk-node-vbmdm

K8S WORKLOAD RISK



Configuration risks **9**
 Vulnerabilities **235**

Note For more information about the **Event Details** panel, see [Investigate - Processes](#) in the main *Investigate* section of the *VMware Carbon Black Cloud User Guide*.

Click **View more** to view the Kubernetes Workloads page. See [View Kubernetes Workloads](#).

To investigate Kubernetes cluster configuration issues, click the number associated with **Configuration risks**. See [View a Kubernetes Workload - Risks](#).

To investigate Kubernetes cluster vulnerabilities, click the number associated with **Vulnerabilities**. See [View a Kubernetes Workload - Risks](#) and click any link in the **Vulnerability** column for more information about that vulnerability. For example:

CVE-2022-37454
✕

Overview
Images
K8s Workloads
Exceptions

CVE	CVE-2022-37454
Description	The keccak xkcp sha-3 reference implementation before fdc6fef has an integer overflow and resultant buffer overflow that allows attackers to execute arbitrary code or eliminate expected cryptographic properties. this occurs in the sponge function interface.
	National Vulnerability Database

CVSS Vector Details		CVSS Score	
Attack complexity	Low	V3 score	9.8
Attack vector	Network	V3 exploit score	3.9
Availability impact	High	V3 impact score	5.9
Confidentiality impact	High	Vector	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H
Integrity impact	High	V2 exploit subscore	--
Privileges required	None	V2 impact subscore	--
Scope	Unchanged		
User interaction	None		

[Evaluating risk](#)

Investigate Kubernetes Namespaces


To investigate events associated with Kubernetes namespaces, perform the following procedure.

Procedure


- 1 On the left navigation pane, click **Investigate > Process**.
- 2 In the left pane, filter by **K8s Namespace**.
- 3 Optionally define any additional query criteria in the **Search** bar and press **Enter** to run the query.

Note

- For a list of Container and Kubernetes event and search fields, see [Investigating Container Events on the Investigate Page](#).
 - For a list of all available Search fields, open the in-product *Search Guide* in the upper right corner of the page.
-

- 4 For details about a specific event in the results table, click the arrow  icon at the right of the row.

CONTAINER

Name	d2167b79b3cb9f2971abdf6dc85c5d6219151faa540a686ee120820711fdf...
Container ID	d2167b79b3cb9f2971abdf6dc85c5d6219151faa540a686ee120820711fdfdf6
Start time	10:24:45 am Jul 31, 2023
Stop time	10:25:21 am Jul 31, 2023
Status	Stopped
Image	docker.io/cbartifactory/cluster-scanner:main
Mounts	20 
Root path	rootfs

K8S WORKLOAD

[View more](#)

Name	cbcontainers-node-agent
Kind	DaemonSet
Cluster	meori:meori-aks-test
Namespace	cbcontainers-dataplane
Pod name	cbcontainers-node-agent-cw8wk

K8S WORKLOAD RISK



Configuration risks [7](#)
Vulnerabilities [34](#)

Note For more information about the **Event Details** panel, see [Investigate - Processes](#) in the main *Investigate* section of the *VMware Carbon Black Cloud User Guide*.

Click **View more** to view the Kubernetes Workloads page. See [View Kubernetes Workloads](#).

To investigate Kubernetes cluster configuration issues, click the number associated with **Configuration risks**. See [View a Kubernetes Workload - Risks](#).

To investigate Kubernetes cluster vulnerabilities, click the number associated with **Vulnerabilities**. See [View a Kubernetes Workload - Risks](#) and click any link in the **Vulnerability** column for more information about that vulnerability.

Investigate Kubernetes Workloads

To investigate events associated with Kubernetes workloads, perform the following procedure.


Procedure

- 1 On the left navigation pane, click **Investigate > Process**.

- 2 In the left pane, filter by **K8s Workload**.
- 3 Optionally define any additional query criteria in the **Search** bar and press **Enter** to run the query.

Note

- For a list of Container and Kubernetes event and search fields, see [Investigating Container Events on the Investigate Page](#).
 - For a list of all available Search fields, open the in-product *Search Guide* in the upper right corner of the page.
-

- 4 For details about a specific event in the results table, click the arrow  icon at the right of the row.

The right panel shows the following details:

K8S WORKLOAD

[View more](#)

Name	cbcontainers-node-agent
Kind	DaemonSet
Cluster	skostov:cnr
Namespace	cbcontainers-dataplane
Pod name	cbcontainers-node-agent-8hb47


Note For more information about the **Event Details** panel, see [Investigate - Processes](#) in the main *Investigate* section of the *VMware Carbon Black Cloud User Guide*.

Click **View more** to view the Kubernetes Workloads page. See [View Kubernetes Workloads](#).

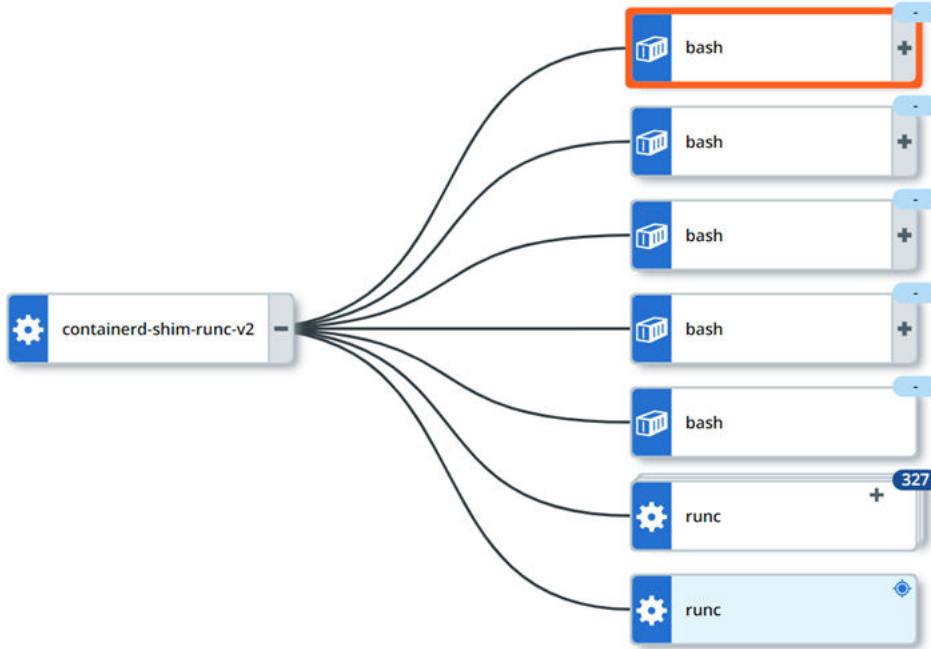
Investigate Containers Events on the Process Analysis Page

Note This content is specific to Containers and Kubernetes. For additional documentation that more broadly describes the Process Analysis page in the Carbon Black Cloud console, see [Process Analysis](#) in the main *Investigate* section of the *VMware Carbon Black Cloud User Guide*.

Procedure

- 1 On the left navigation pane, click **Investigate > Processes**.
- 2 Perform a search query for events associated with Containers or Kubernetes.
- 3 In the results table, click the **Process Analysis**  icon at the right of the row.

The Process Analysis tree displays. For example:



The information in the right panel depends on the type of event searched for and selected. The following topics describe each selection by filtered event type. For example:

CONTAINER PROCESSCMD [/cluster-scanner](#)Path [/cluster-scanner](#)

PID 796939

CONTAINER DETAILS

Name ff58772a08a38e0924ceab0693...


Container ID ff58772a08a38e0924ceab06933b26832e2fc12990ee6d8915922ad6e2d55a69

Start time 5:18:40 am Aug 1, 2023

Stop time 5:19:19 am Aug 1, 2023

Status Stopped

Image docker.io/cbartifactory/cluster-scanner:main

Mounts [20](#) 

Root path rootfs

K8S WORKLOAD[View more](#)

Name cbcontainers-node-agent

Kind DaemonSet

Cluster meori:meori-aks-test

Namespace cbcontainers-dataplane

Pod name cbcontainers-node-agent-qjgtg

K8S WORKLOAD RISKConfiguration risks [7](#)Vulnerabilities [34](#)

- Click **View more** to view the Kubernetes Workloads page. See [View Kubernetes Workloads](#).
- To investigate Kubernetes cluster configuration issues, click the number associated with **Configuration risks**. See [View a Kubernetes Workload - Risks](#).
- To investigate Kubernetes cluster vulnerabilities, click the number associated with **Vulnerabilities**. See [View a Kubernetes Workload - Risks](#) and click any link in the **Vulnerability** column for more information about that vulnerability.

Triaging Kubernetes Alerts

This section describes how to triage Kubernetes alerts in the Carbon Black Cloud console.

Note This content is specific to Kubernetes alerts. For additional documentation that more broadly describes triaging any kind of alert in the Carbon Black Cloud console, see [Alerts](#) and [Alert Triage](#) in the main *Alerts* section of the user guide.

Search for Kubernetes Alerts

To search for Kubernetes policy rule violations (alerts), perform the following procedure.

Procedure

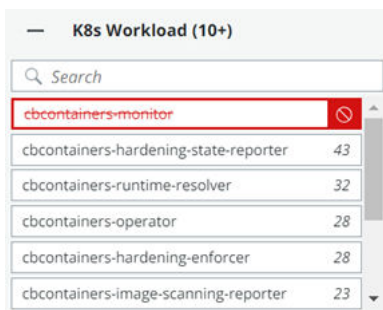
- 1 On the left navigation pane, click **Alerts**.
- 2 Search and filter for Kubernetes violations using the filters in the left pane and the **Search** text box. For help constructing a query, see the in-product *Search Guide*.

Note

- You can define search results by time.
- The Alerts page offers four ways to filter alerts for Containers and Kubernetes:
 - **K8s Cluster**
 - **K8s Namespace**
 - **K8s Workload**
 - **K8s Policy**



You can combine filters to achieve a particular result.

- Click the vertical 3-dot **Configuration** menu to configure the filters that display in the console.
- Alerts with Monitor action rules are not visible by default. They are part of the **Other Activity > Observed** filter category.
- You can exclude search results by clicking the **Exclude** icon to the right of a filter value. For example:



Example search results table:

295 results				Group by: None			
STATUS	SEVERITY	TYPE/REASON	CREATED	ASSET	POLICY	WORKFLOW	ACTIONS
🔍 Ran	5	Containers Runtime Detected an abnormal internal connection with medium or low risk	7:07:24 pm Apr 16, 2023	eks-cluster-1-frontend	eks runtime policy	Open	⌵ >
🔍 Ran	4	Containers Runtime Detected an abnormal egress connection with medium or low risk	7:06:23 pm Apr 16, 2023	eks-cluster-1-loadgenerator	eks runtime policy	Open	⌵ >
🔍 Ran	5	Containers Runtime Detected a connection to a public destination that isn't allowed for this scope	7:03:00 pm Apr 16, 2023	eks-cluster-1-cscontainers-operator	eks runtime policy	Open	⌵ >
🔍 Ran	5	Containers Runtime Detected a connection to a private network that isn't allowed for this scope	7:03:00 pm Apr 16, 2023	eks-cluster-1-cscontainers-hardening-state-reporter	eks runtime policy	Open	⌵ >

- To view details about a workload, click the workload name in the **Asset** column. See [View a Kubernetes Workload - Overview](#).
- To view a summary of the policy assigned to a workload, click the policy name.
- To view the Process Analysis tree and details for this alert, click the Process Analysis  icon. See [Investigate Containers Events on the Process Analysis Page](#).
- To investigate the alert on the Investigate page, click the Investigate  icon. See [Investigating Container Events on the Investigate Page](#).
- Click the **Actions** dropdown menu for actions you can perform on the alert:
 - Close the alert.

Important Closing alerts is only recommended for excluding specific workloads that exhibit known behaviors from the alerts list.

- Mark the alert as being in progress.
- View the notifications that have been sent out about the alert.
- Add the alert behavior to the baseline. See [Kubernetes Scope Baselines for Runtime Policies](#).
- To view more alert details, click the arrow > icon at the right of the row. See [View Kubernetes Alert Details](#).

View Kubernetes Alert Details

To investigate Kubernetes alert details in the Carbon Black Cloud console, perform the following procedure.

This page only describes Kubernetes alert details. For more information about the Alerts page, see [View Alert Details](#) in the main *Alerts* section of the *VMware Carbon Black Cloud User Guide*.

Prerequisites

This topic assumes you have conducted a search for Kubernetes alerts and are viewing the search results. Before you proceed, see [Search for Kubernetes Alerts](#).

Procedure

- 1 For details about a specific alert, click the arrow > icon at the right of the alert row. The **Alert Details** panel includes the following Kubernetes information:

The screenshot shows a panel with two sections. The top section is titled 'K8S WORKLOAD' and includes a 'View more' link. Below the title is a table with the following information:

Name	loadgenerator
Kind	Deployment
Cluster	aws:prod
Namespace	boutique
Pod name	loadgenerator-76556f89bf-jr545

The bottom section is titled 'K8S WORKLOAD RISK' and features a gauge showing a score of 7 with the label 'High'. To the right of the gauge, the following metrics are listed:

Configuration risks	3
Vulnerabilities	522

- 2 To open a specific workload page, click **View more** next to **K8s Workload**. See [View Kubernetes Workloads](#).
- 3 To access more information about any aspect of the alert or workload, click the relevant hyperlink in the panel. For example, you can view the associated risks and vulnerabilities by clicking the numbers next to **Configuration risks** or **Vulnerabilities** in the **K8s Workload Risk** section.

Identify Available Fixes and Patches

You can identify the available fixes and patches for known vulnerabilities in container images.

Each vulnerability is characterized by the following:

- CVE code
- List of impacted packages or libraries
- Package version
- Available fix or patch and version

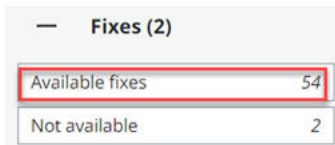
Important You can only identify the available fixes or patches in the Carbon Black Cloud console. To apply them, proceed to your Kubernetes environment.

Prerequisites

Become familiar with the [Common Vulnerabilities and Exposures \(CVE\) list](#) (external link).

Procedure

- 1 On the left navigation pane, do one of the following depending on your system configuration and role:
 - If you have the Kubernetes Security DevOps or SecOps role and your system has only the Container security feature, click **Inventory > Container Images**.
 - If you have any other role and your system has Container security and other Carbon Black Cloud features, click **Inventory > Kubernetes > Container Images**.
- 2 Click the **Deployed Images** tab.
- 3 In the **Fixes** filter in the left pane, select **Available Fixes**.



— Fixes (2)	
Available fixes	54
Not available	2

The table only displays images for which there are fixes. The **Vulnerabilities/Fixes** column indicates the number of fixes per vulnerability severity category inside associated color bars.

- 4 To expand the **Image Details** panel, click the arrow  icon at the right of the row.

IMAGE DETAILS

Rescan
[View more](#)


Name	gke.gcr.io/kube-proxy-amd64:v1.20.8		
Registry	gke.gcr.io		
Repository	kube		
Image digest	sha256:51130e63b8158e7b3b3507e4dec916aa7e07a6cb7ce9279f6afe4803036e4128		
Scan status	Completed		
Last scan	06:00 am on Dec 27, 2020		

SECRETS (1)

TYPE	FILE	EXCEPTION
File	.eslintignore	No

VULNERABILITIES (110)

CVE	PACKAGE	FIX	EXCEPTION
> CVE-lorem-ips...	libgnutls30-3.6.7-4...	Yes	No
> CVE-2020-3453	libhogweed4-3.4.1-1	--	No
> CVE-2020-3453	passwd-1:4.5-1.1	--	No
> CVE-2020-3453	libgnutls30-3.6.7	--	No
> CVE-2020-3453	debian3942.1	--	No

- 5 To view a short description of the CVE code and the package where the vulnerability is identified, click the arrow  icon to the left of the **CVE**.

	CVE	PACKAGE	FIX	EXCEPTION
∨	CVE-2018-250...	libwebp-dev-0.6....	Yes	Yes
<div style="background-color: #f0f0f0; padding: 5px;"> <p>Severity 9.1</p> <p>Package libwebp-dev-0.6.1-2</p> <p>Fix 0.6.1-2+deb10u1</p> <p>Description A heap-based buffer overflow was found in libwebp in versions before 1.0.1 in GetLE16().</p> </div>				
>	CVE-2018-250...	libwebp6-0.6.1-2	Yes	No
>	CVE-2018-250...	libwebpdemux2-...	Yes	No
>	CVE-2018-250...	libwebpmux3-0....	Yes	No
>	CVE-2018-250...	libwebp-dev-0.6....	Yes	No

What to do next

Apply the fix or patch accordingly.

Managing Clusters and Kubernetes Sensors

7

This section describes management tasks on clusters and Kubernetes Sensors after your security environment is up and running.

For instructions on setting up your Kubernetes clusters and sensors in the Carbon Black Cloud, see [Adding Clusters and Installing Kubernetes Sensors](#).

Read the following topics next:

- [View Clusters](#)
- [Edit a Cluster](#)
- [Delete a Cluster and its Sensor](#)
- [Upgrading or Downgrading the Kubernetes Sensor](#)
- [Delete a CLI Client](#)

View Clusters

After you've added clusters to the Carbon Black Cloud console, you can view details about the clusters.

Prerequisites

Add clusters to the console. See [Adding Clusters and Installing Kubernetes Sensors](#).

Procedure

- 1 On the left navigation pane of the console, do one of the following depending on your system configuration and role:
 - If you are assigned Kubernetes Security DevOps role and your system has only the Container Security feature, select **Inventory > Clusters**.
 - If you are assigned any other role and your system has Container security and other Carbon Black Cloud features, select **Inventory > Kubernetes > Clusters**.
- 2 Click the **Clusters** tab and then click the **General** tab.

3 In the left pane, you can filter the list of displayed clusters by:

- Status
- Sensor Version
- Operator Version
- Cluster Label Key
- Cluster Label Value

4 In the Clusters panel, you can search for a cluster, and you can select a displayed cluster name to view both general information and sensor health data. For more about sensor health, see [Check the Kubernetes Sensor Status and Health](#).

The right pane shows the following information:

General	Sensor health
Cluster	[REDACTED]
Cluster group	[REDACTED]
Status	Running
Sensor version	main
Nodes	4
Sensors	4
Kubernetes version	v1.24.9
Last check-in	4:05:32 am Apr 26, 2023
Labels	--

Edit a Cluster

You can edit a Kubernetes cluster in the Carbon Black Cloud console to enable features of the Kubernetes Sensor that were not included during the cluster setup.

Prerequisites

Before you begin, open both the Carbon Black Cloud console and a terminal window.

Procedure

- 1 On the left navigation pane of the console, do one of the following depending on your system configuration and role:
 - If you are assigned Kubernetes Security DevOps role and your system has only Containers Security feature, select **Inventory > Clusters**.
 - If you are assigned any other role and your system has Container security and other Carbon Black Cloud features,

select **Inventory > Kubernetes > Clusters**.

- 2 Locate the cluster to edit and in the **Options** dropdown menu, click **Edit** and then click **Next**.
- 3 Select the features to include. For example, **Runtime protection** or **Cluster image scanning**. Click **Next**.
- 4 To run the update, copy the command from the **Finish Setup** page, and run it in the terminal window.

Delete a Cluster and its Sensor

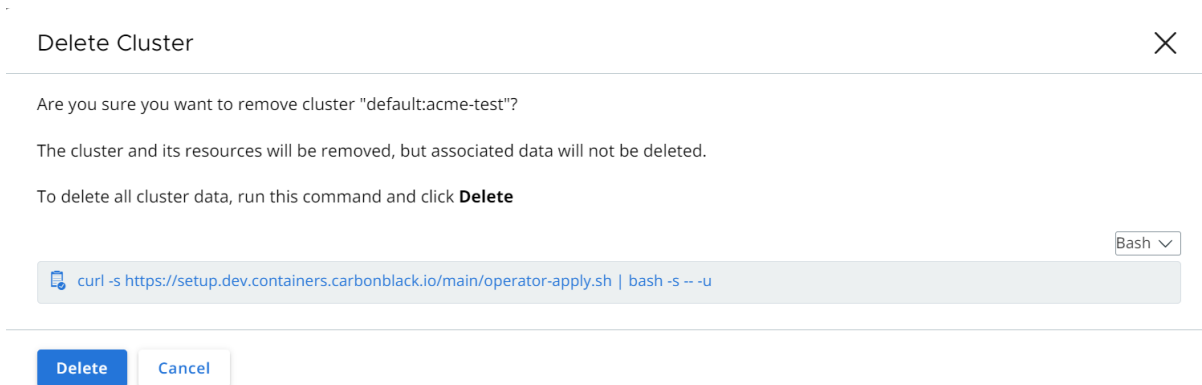
To delete the Kubernetes Sensor from a cluster, you must delete the cluster from the Carbon Black Cloud console.

Prerequisites

Before you begin, open both the Carbon Black Cloud console and a terminal window.

Procedure

- 1 On the left navigation pane of the console, do one of the following depending on your system configuration and role:
 - If you are assigned Kubernetes Security DevOps role and your system has only Containers Security feature,
 - Click **Inventory > Clusters**.
 - If you are assigned any other role and your system has Container security and other Carbon Black Cloud features,
 - click **Inventory > Kubernetes > Clusters**.
- 2 Locate the cluster to delete from the console.
- 3 In the **Options** dropdown menu, click **Delete**.



- 4 Select **Bash** or **PowerShell** from the dropdown menu.

- 5 Copy the command into your terminal window and run it.

This step deletes the Kubernetes Sensor and the Carbon Black Cloud operator from your cluster.

Important If you execute the command without removing the cluster from the console in the next step, the cluster status becomes **Critical** after a certain time. In this case, you can re-add or remove the cluster.

- 6 Click **Delete**.

Important If you click **Delete** without executing the command from the previous step, the Kubernetes Sensor and the Carbon Black Cloud operator remain on your cluster without any activity.

Upgrading or Downgrading the Kubernetes Sensor

Carbon Black recommends that you use the latest Kubernetes Sensor version.

You can upgrade or downgrade the Kubernetes Sensor in the following ways:

- Direct access to the cluster through a command-line interface.
- Direct access to the cluster through the Carbon Black Cloud console.
- Remotely through the Carbon Black Cloud console without directly accessing the cluster.

Upgrade or Downgrade the Kubernetes Sensor through the Command Line

You can upgrade or downgrade the Kubernetes Sensor through the Command Line.

This method:

- Requires direct user access to the cluster.
- Simply patches the sensor version.
- Does not enable or disable features.
- Does not override customized values with default values.
- Does not change the cluster or sensor configuration.

For a more thorough upgrade experience, upgrade the sensor through the console instead. See [Upgrade or Downgrade the Kubernetes Sensor through the Console](#).

Procedure

- 1 Open a terminal window.

- 2 Run the following command, where the `value` definition is the version of the sensor.

```
kubectl patch cbcontainersagent.operator.containers.carbonblack.io/cbcontainers-agent --
type='json' -p='[{"op": "replace", "path": "/spec/version", "value": "3.0.2"}]
```

Note `cbcontainers-agent` refers to the Kubernetes Sensor. In the preceding code block, 3.0.2 is the latest Kubernetes Sensor version. Substitute this value with the appropriate version.

You can use the [Setup API](#) to list the supported Kubernetes Sensor versions. See `/deploy/sensors` and `/deploy/compatibility`. The former API usage lists available sensor versions; the latter API usage defines operator and sensor version compatibility.

What to do next

[Check the Kubernetes Sensor Status and Health](#)

Upgrade or Downgrade the Kubernetes Sensor through the Console

You can upgrade or downgrade the Kubernetes Sensor through the Carbon Black Cloud console.

This method:

- Requires direct admin access to the cluster.
- Generates a new cluster configuration (custom resource).
- Automatically enables or disables supported features depending on the sensor version that you select.
- Overwrites customized values with default values when the custom resource is applied.

Procedure

- 1 Open a terminal window.
- 2 On the left navigation pane of the console, do one of the following depending on your system configuration and role:
 - If you are assigned Kubernetes Security DevOps role and your system has only Containers Security feature, Click **Inventory** > **Clusters**.
 - If you are assigned any other role and your system has Container security and other Carbon Black Cloud features, click **Inventory** > **Kubernetes** > **Clusters**.
- 3 Locate the cluster to update.
- 4 In the **Options** dropdown menu, click **Edit**.
- 5 Optionally add any new labels for the cluster. Click **Next**.

- 6 Select the **Sensor version** from the dropdown list. This should usually be the latest version, which is the first version listed.

Add Cluster
✕

SENSOR [Cluster setup guide](#)

The latest sensor will be installed unless a different version is selected

Version 3.0.2

Included Features

- K8s security posture management for workload risk
- Runtime workload and container protection with threat detection and prevention
- Automated scanning of running image for vulnerability, malware and secrets

Next

Back

Cancel

- 7 Included features for the selected sensor version are displayed. Click **Next**.
- 8 Copy the commands from the **Finish Edits** page and run them in the terminal window.

Edit Cluster
✕

FINISH EDITS [Cluster setup guide](#)

Run these commands in this order in your terminal and click **Done**

1 — Apply secret to cluster, or add to secrets management tool

```
kubectl create secret generic cbcontainers-company-code --namespace cbcontainers-dataplane --from-literal=companyCode=UO2IJJFUHSNEBSNEF8#M1SGNWG#JV
```

2 — Apply updated cluster configuration [View YAML details](#)

```
kubectl apply -f https://setup.dev.containers.carbonblack.io/cr-20803b16-2e99-4c82-987d-979d343c30eb
```

Apply changes directly to the cluster

Done

Back

Cancel

Important Do not select **Apply changes directly to the cluster** for this method of upgrade or downgrade. That option is relevant only if you are not directly accessing the cluster, which is covered in [Upgrade or Downgrade the Kubernetes Sensor Remotely through the Console](#).

9 Click **Done**.

Results

It takes between two and three minutes for the new sensor to become effective. If the upgraded sensor version supports new functionality, such as CNDR, that new functionality is automatically enabled by the Operator. This capability assumes that you are running the latest version of the Operator.

If you are downgrading the sensor version and the downgraded sensor does not support a particular functionality, the Operator disables that functionality upon the downgrade action.

What to do next

[Check the Kubernetes Sensor Status and Health](#)

Upgrade or Downgrade the Kubernetes Sensor Remotely through the Console

You can upgrade or downgrade the Kubernetes Sensor remotely through the Carbon Black Cloud console.

This method:

- Does not require direct admin access to the cluster because the operator handles the upgrade or downgrade.
- Automatically enables or disables supported features depending on the sensor version that you select.
- Does not override customized values with default values.
- Does not change the cluster or sensor configuration.
- Requires an Operator version 6.1.0 or higher.

Procedure

- 1 On the left navigation pane of the console, do one of the following depending on your system configuration and role:
 - If you are assigned Kubernetes Security DevOps role and your system has only Containers Security feature, Click **Inventory > Clusters**.
 - If you are assigned any other role and your system has Container security and other Carbon Black Cloud features, click **Inventory > Kubernetes > Clusters**.
- 2 Locate the cluster to update.
- 3 In the **Options** dropdown menu, click **Edit**.
- 4 Optionally add any new labels for the cluster. Click **Next**.

- 5 Select the **Sensor version** from the dropdown list. This should usually be the latest version, which is the first version listed.

Add Cluster ✕

✓

CLUSTER DETAIL

✓

AUTHENTICATION

3

SENSOR

4

FINISH SETUP

SENSOR [Cluster setup guide](#)

The latest sensor will be installed unless a different version is selected

Version 3.0.2 ▼

Included Features

- K8s security posture management for workload risk
- Runtime workload and container protection with threat detection and prevention
- Automated scanning of running image for vulnerability, malware and secrets

Next

Back

Cancel

- 6 Included features for the selected sensor version are displayed. Click **Next**.

7 Select the check box for **Apply changes directly to the cluster**.

Edit Cluster
✕

FINISH EDITS [Cluster setup guide](#)

Run these commands in this order in your terminal and click **Done**

1 — Apply secret to cluster, or add to secrets management tool

```
kubectl create secret generic cbcontainers-company-code --namespace cbcontainers-dataplane --from-literal=companyCode=UO2JJFUHSNEBSNEF8#M1SGNWG#JV
```

2 — Apply updated cluster configuration [View YAML details](#)

```
kubectl apply -f https://setup.dev.containers.carbonblack.io/cr-c0a83ed5-1863-4df6-b1d8-e2faa6ad9547
```

Apply changes directly to the cluster

⚠ The cluster may be managed by automated CI/CD infrastructure and modifications may be overwritten unless you take manual measures to prevent that.

Save
Back
Cancel

Note

- When using the option to **Apply changes directly to the cluster**, you do not need to copy and run commands in a terminal window.
- If you manage the cluster using Helm or other infrastructure-as-code, that system will override the settings you establish using this procedure.

8 Click **Done**.

Results

It takes between two and three minutes for the new sensor to become effective. If the upgraded sensor version supports new functionality, such as CNDR, that new functionality is automatically enabled by the Operator. This capability assumes that you are running the latest version of the Operator.

If you are downgrading the sensor version and the downgraded sensor does not support a particular functionality, the Operator disables that functionality upon the downgrade action.

Any remote upgrade or downgrade of the Kubernetes Sensor is noted in the Audit Log.

What to do next

[Check the Kubernetes Sensor Status and Health](#)

Delete a CLI Client

You can delete a CLI instances that is no longer in use.

Procedure

- 1 On the left navigation pane, click **Inventory > Kubernetes > Clusters**.
- 2 Click the **CLI Config** tab.
- 3 Under **Actions**, click the delete icon next to the CLI Client that you want to remove.

Results

Deleting a CLI Client removes the instance and the generated API-key from Carbon Black Cloud. It does not remove the instance from your environment.

Carbon Black Container Operator Technical Reference



The Carbon Black Container Operator runs within a Kubernetes cluster. The Container Operator is a set of controllers that deploy and manage the Carbon Black Container components.

The Operator handles the following actions:

- Deploys and manages the Carbon Black Container product, including the configuration and the image scanning for Kubernetes security.
- Automatically fetches and deploys the Carbon Black Container private image registry secret.
- Automatically registers the Carbon Black Container cluster.
- Manages the Carbon Black Container validating webhook and dynamically manages the admission control webhook to avoid possible downtime.
- Monitors and reports agent availability to the Carbon Black Cloud console.

The Carbon Black Container Operator uses the operator-framework to create a GO operator that is responsible for managing and monitoring the Carbon Black Container components deployment.

To review the Operator compatibility matrix, see [Kubernetes Sensor Operator Distributions and Kubernetes Version](#).

Note We recommend that you deploy the Operator by using the **Add Cluster** wizard (see [Add a Cluster and Install the Kubernetes Sensor](#)). However, this technical reference section of the *User Guide* also includes manual Operator and Agent installation instructions.

Read the following topics next:

- [Manually Deploy the Container Operator](#)
- [Manually Deploy the Container Agent](#)
- [Openshift](#)
- [Reading Metrics by using Prometheus](#)
- [Custom Resources Definitions](#)
- [Changing Components Resources](#)
- [Configuring Container Services to use HTTP Proxy](#)
- [Changing the Image Source](#)

- [Operator Role-based Access Control](#)
- [Container Operator Developer Instructions](#)
- [Helm Charts](#)

Manually Deploy the Container Operator

To manually deploy the Carbon Black Container Operator, perform the following procedure.

These instructions use an Operator image. To deploy the Operator without using an image, see [Container Operator Developer Instructions](#).

Prerequisites

Your cluster must be running Kubernetes 1.18+.

Procedure

- ◆ You can initiate the Operator deployment in two ways:

- **Script:**

```
export OPERATOR_VERSION=v6.0.2
export OPERATOR_SCRIPT_URL=https://setup.containers.carbonblack.io/$OPERATOR_VERSION/operator-apply.sh
curl -s $OPERATOR_SCRIPT_URL | bash
```

{OPERATOR_VERSION} is of the format "v{VERSION}".

- **Source code:**

- a Clone the GIT project and deploy the operator from the source code.

By default, the Operator uses `CustomResourceDefinitions v1`, which requires Kubernetes 1.16+. You can also deploy an Operator by using `CustomResourceDefinitions v1beta1` (deprecated in Kubernetes 1.16, removed in Kubernetes 1.22).

- b Create the Operator image:

```
make docker-build docker-push IMG={IMAGE_NAME}
```

- c Deploy the Operator resources:

```
make deploy IMG={IMAGE_NAME}
```

What to do next

[Manually Deploy the Container Agent](#)

Uninstall the Container Operator

To uninstall the Carbon Black Container Operator, perform the following procedure.

Procedure

- ◆ To uninstall the Carbon Black Container Operator, run the following command:

```
export OPERATOR_VERSION=v6.0.2
export OPERATOR_SCRIPT_URL=https://setup.containers.carbonblack.io/$OPERATOR_VERSION/
operator-apply.sh
curl -s $OPERATOR_SCRIPT_URL | bash -s -- -u
```

This command deletes the Carbon Black Container custom resource definitions (CRDs) and instances.

Manually Deploy the Container Agent

To manually deploy the Carbon Black Container Agent, perform the following procedure.

Prerequisites

Manually Deploy the Container Operator

Procedure

- 1 Apply the Carbon Black Container API token secret:

```
kubectl create secret generic cbcontainers-access-token \
--namespace cbcontainers-dataplane --from-literal=accessToken=\
{API_Secret_Key}/{API_ID}
kubectl create secret generic cbcontainers-company-code --namespace cbcontainers-dataplane
--from-literal=companyCode=RXXXXXXXXXXG!XXXX
```

- 2 Apply the Carbon Black Container Agent custom resource:

Deploy `cbcontainersagents.operator.containers.carbonblack.io` to prompt the Operator to deploy the dataplane components:

```
apiVersion: operator.containers.carbonblack.io/v1
kind: CBContainersAgent
metadata:
  name: cbcontainers-agent
spec:
  account: {ORG_KEY}
  clusterName: {CLUSTER_GROUP}:{CLUSTER_NAME}
  version: {AGENT_VERSION}
  gateways:
    apiGateway:
      host: {API_HOST}
    coreEventsGateway:
      host: {CORE_EVENTS_HOST}
```

```

hardeningEventsGateway:
  host: {HARDENING_EVENTS_HOST}
runtimeEventsGateway:
  host: {RUNTIME_EVENTS_HOST}

```

Note See also [Custom Resources Definitions](#).

OpenShift

The Carbon Black Container Operator and Agent require elevated permissions to operate properly. However, this requirement violates the default `SecurityContextConstraints` on most OpenShift clusters, thereby causing the components to fail to start.

You can resolve this issue by applying the following custom security constraint configurations on the cluster. This action requires cluster administrator privileges.

```

kind: SecurityContextConstraints
apiVersion: security.openshift.io/v1
metadata:
  name: scc-anyuid
runAsUser:
  type: MustRunAsNonRoot
allowHostPID: false
allowHostPorts: false
allowHostNetwork: false
allowHostDirVolumePlugin: false
allowHostIPC: false
allowPrivilegedContainer: false
readOnlyRootFilesystem: true
seLinuxContext:
  type: RunAsAny
fsGroup:
  type: RunAsAny
supplementalGroups:
  type: RunAsAny
users:
- system:serviceaccount:cbcontainers-dataplane:cbcontainers-operator
- system:serviceaccount:cbcontainers-dataplane:cbcontainers-enforcer
- system:serviceaccount:cbcontainers-dataplane:cbcontainers-state-reporter
- system:serviceaccount:cbcontainers-dataplane:cbcontainers-monitor
- system:serviceaccount:cbcontainers-dataplane:cbcontainers-runtime-resolver
---
kind: SecurityContextConstraints
apiVersion: security.openshift.io/v1
metadata:
  name: scc-image-scanning # This probably needs to be fixed in the actual deployment
runAsUser:
  type: RunAsAny
allowHostPID: false
allowHostPorts: false
allowHostNetwork: false
allowHostDirVolumePlugin: false
allowHostIPC: false

```

```

allowPrivilegedContainer: false
readOnlyRootFilesystem: false
seLinuxContext:
  type: RunAsAny
fsGroup:
  type: RunAsAny
supplementalGroups:
  type: RunAsAny
allowedCapabilities:
- 'NET_BIND_SERVICE'
users:
- system:serviceaccount:cbcontainers-dataplane:cbcontainers-image-scanning
---
kind: SecurityContextConstraints
apiVersion: security.openshift.io/v1
metadata:
  name: scc-node-agent
runAsUser:
  type: RunAsAny
allowHostPID: true
allowHostPorts: false
allowHostNetwork: true
allowHostDirVolumePlugin: true
allowHostIPC: false
allowPrivilegedContainer: true
readOnlyRootFilesystem: false
seLinuxContext:
  type: RunAsAny
fsGroup:
  type: RunAsAny
supplementalGroups:
  type: RunAsAny
volumes:
- configMap
- downwardAPI
- emptyDir
- hostPath
- persistentVolumeClaim
- projected
- secret
users:
- system:serviceaccount:cbcontainers-dataplane:cbcontainers-agent-node

```

Uninstalling the Operator on Openshift

Add this SecurityContextConstraints before running the operator uninstall command:

```

kind: SecurityContextConstraints
apiVersion: security.openshift.io/v1
metadata:
  name: scc-edr-cleaner
runAsUser:
  type: RunAsAny
allowHostPID: true

```

```

allowHostPorts: false
allowHostNetwork: true
allowHostDirVolumePlugin: true
allowHostIPC: false
allowPrivilegedContainer: true
readOnlyRootFilesystem: false
seLinuxContext:
  type: RunAsAny
fsGroup:
  type: RunAsAny
supplementalGroups:
  type: RunAsAny
volumes:
- configMap
- downwardAPI
- emptyDir
- hostPath
- persistentVolumeClaim
- projected
- secret
users:
- system:serviceaccount:cbcontainers-edr-sensor-cleaners:cbcontainers-edr-sensor-cleaner

```

Reading Metrics by using Prometheus

Operator metrics are protected by `kube-auth-proxy`. You must grant permissions to a Prometheus server before it can scrape the protected metrics.

You can create a `ClusterRole` and bind it with `ClusterRoleBinding` to the service account that your Prometheus server uses.

If you have not configured this cluster role and cluster role binding, you can use the following configuration:

Cluster Role

```

apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
name: cbcontainers-metrics-reader
rules:
- nonResourceURLs:
  - /metrics
  verbs:
  - get

```

Cluster Role Binding

```

kubectl create clusterrolebinding metrics --clusterrole=cbcontainers-metrics-reader --
serviceaccount=<prometheus-namespace>:<prometheus-service-account-name>

```


Use the following `ServiceMonitor` to scrape metrics from the Carbon Black Container Operator. Your Prometheus custom resource service monitor selectors must match this configuration.

```

apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  labels:
    control-plane: operator
  name: cbcontainers-operator-metrics-monitor
  namespace: cbcontainers-dataplane
spec:
  endpoints:
  - bearerTokenFile: /var/run/secrets/kubernetes.io/serviceaccount/token
    path: /metrics
    port: https
    scheme: https
    tlsConfig:
      insecureSkipVerify: true
  selector:
    matchLabels:
      control-plane: operator

```

Custom Resources Definitions

The Carbon Black Container Operator implements controllers for Carbon Black Container custom resources definitions (CRDs).

Carbon Black Container Agent Custom Resource

Deploy `cbcontainersagents.operator.containers.carbonblack.io` to prompt the Operator to deploy the dataplane components.

Table 8-1. Required Parameters

Parameter	Description
<code>spec.account</code>	Carbon Black Container org key
<code>spec.clusterName</code>	Carbon Black Container cluster name (<code><cluster_group:cluster_name></code>)
<code>spec.version</code>	Carbon Black Container Agent version
<code>spec.gateways.apiGateway.host</code>	Carbon Black Container API host
<code>spec.gateways.coreEventsGateway.host</code>	Carbon Black Container core events host (for example, health checks)
<code>spec.gateways.hardeningEventsGateway.host</code>	Carbon Black Container hardening events host (for example, deleted, validated, and blocked resources)
<code>spec.gateways.runtimeEventsGateway.host</code>	Carbon Black Container runtime events host (for example, traffic events)

Table 8-2. Optional Parameters

Parameter	Description	Default Value
<code>spec.apiGateway.port</code>	Carbon Black Container API port	443
<code>spec.accessTokenSecretName</code>	Carbon Black Container API access token secret name	cbcontainers-access-token
<code>spec.gateways.coreEventsGateway.port</code>	Carbon Black Container core events port	443
<code>spec.gateways.hardeningEventsGateway.port</code>	Carbon Black Container hardening events port	443
<code>spec.gateways.runtimeEventsGateway.port</code>	Carbon Black Container runtime events port	443

Table 8-3. Basic Components Optional Parameters

Parameter	Description	Default Value
<code>spec.components.basic.enforcer.replicasCount</code>	Carbon Black Container Hardening Enforcer number of replicas	1
<code>spec.components.basic.monitor.image.repository</code>	Carbon Black Container Monitor image repository	cbartifactory/monitor
<code>spec.components.basic.enforcer.image.repository</code>	Carbon Black Container Hardening Enforcer image repository	cbartifactory/guardrails-enforcer
<code>spec.components.basic.stateReporter.image.repository</code>	Carbon Black Container Hardening State Reporter image repository	cbartifactory/guardrails-state-reporter
<code>spec.components.basic.monitor.resources</code>	Carbon Black Container Monitor resources	{requests: {memory: "64Mi", cpu: "30m"}, limits: {memory: "256Mi", cpu: "200m"}}
<code>spec.components.basic.enforcer.resources</code>	Carbon Black Container Hardening Enforcer resources	{requests: {memory: "64Mi", cpu: "30m"}, limits: {memory: "256Mi", cpu: "200m"}}
<code>spec.components.basic.stateReporter.resources</code>	Carbon Black Container Hardening State Reporter resources	{requests: {memory: "64Mi", cpu: "30m"}, limits: {memory: "256Mi", cpu: "200m"}}

Table 8-4. Runtime Components Optional Parameters

Parameter	Description	Default Value
<code>spec.components.runtimeProtection.enabled</code>	Carbon Black Container flag to control Runtime components deployment	True
<code>spec.components.runtimeProtection.resolver.image.repository</code>	Carbon Black Container Runtime Resolver image repository	cbartifactory/runtime-kubernetes-resolver
<code>spec.components.runtimeProtection.sensor.image.repository</code>	Carbon Black Container Runtime Sensor image repository	cbartifactory/runtime-kubernetes-sensor

Table 8-4. Runtime Components Optional Parameters (continued)

Parameter	Description	Default Value
<code>spec.components.runtimeProtection.internalGrpcPort</code>	Carbon Black Container Runtime gRPC port that the resolver exposes for the sensor	443
<code>spec.components.runtimeProtection.resolver.logLevel</code>	Carbon Black Container Runtime Resolver log level	"panic", "fatal", "error", "warn", "info", "debug", "trace" (default info)
<code>spec.components.runtimeProtection.resolver.resources</code>	Carbon Black Container Runtime Resolver resources	{requests: {memory: "64Mi", cpu: "200m"}, limits: {memory: "1024Mi", cpu: "900m"}}
<code>spec.components.runtimeProtection.sensor.logLevel</code>	Carbon Black Container Runtime Sensor log level	"panic", "fatal", "error", "warn", "info", "debug", "trace" (default info)
<code>spec.components.runtimeProtection.sensor.resources</code>	Carbon Black Container Runtime Sensor resources	{requests: {memory: "64Mi", cpu: "30m"}, limits: {memory: "1024Mi", cpu: "500m"}}

Table 8-5. Cluster Scanning Components Optional Parameters

Parameter	Description	Default Value
<code>spec.components.clusterScanning.enabled</code>	Carbon Black Container flag to control Cluster Scanning components deployment	True
<code>spec.components.clusterScanning.imageScanningReporter.image.repository</code>	Carbon Black Container Image Scanning Reporter image repository	cbartifactory/image-scanning-reporter
<code>spec.components.clusterScanning.clusterScanner.image.repository</code>	Carbon Black Container Scanner Agent image repository	cbartifactory/cluster-scanner
<code>spec.components.clusterScanning.imageScanningReporter.resources</code>	Carbon Black Container Image Scanning Reporter resources	{requests: {memory: "64Mi", cpu: "200m"}, limits: {memory: "1024Mi", cpu: "900m"}}
<code>spec.components.clusterScanning.clusterScanner.resources</code>	Carbon Black Container Cluster Scanner resources	{requests: {memory: "64Mi", cpu: "30m"}, limits: {memory: "1024Mi", cpu: "500m"}}
<code>spec.components.clusterScanning.clusterScanner.k8sContainerEngine.engineType</code>	Carbon Black Container Cluster Scanner Kubernetes container engine type. One of these options: <code>containerd</code> / <code>docker-daemon</code> / <code>cri-o</code>	N/A
<code>spec.components.clusterScanning.clusterScanner.k8sContainerEngine.endpoint</code>	Carbon Black Container Cluster Scanner Kubernetes container engine endpoint path	N/A
<code>spec.components.clusterScanning.clusterScanner.k8sContainerEngine.CRIO.storagePath</code>	Carbon Black Container Cluster Scanner override default image storage path (CRI-O only)	N/A
<code>spec.components.clusterScanning.clusterScanner.k8sContainerEngine.CRIO.storageConfigPath</code>	Carbon Black Container Cluster Scanner override default image storage config path (CRI-O only)	N/A

Table 8-5. Cluster Scanning Components Optional Parameters (continued)

Parameter	Description	Default Value
<code>spec.components.clusterScanning.clusterScanner.k8sContainerEngine.CRIO.configPath</code>	Carbon Black Container Cluster Scanner override default CRI-O config path (CRI-O only)	N/A
<code>spec.components.clusterScanning.clusterScanner.cliFlags.enableSecretDetection</code>	Carbon Black Container Cluster Scanner flag of whether the scan should scan for secrets	False
<code>spec.components.clusterScanning.clusterScanner.cliFlags.skipDirsOrFiles</code>	Carbon Black Container Cluster Scanner flag of files or directories to not scan for secrets	N/A
<code>spec.components.clusterScanning.clusterScanner.cliFlags.scanBaseLayers</code>	Carbon Black Container Cluster Scanner flag of whether the scan should include the base layers scan for secrets	False
<code>spec.components.clusterScanning.clusterScanner.cliFlags.ignoreBuildInRegex</code>	Carbon Black Container Cluster Scanner flag of whether the scan should ignore the built-in regexes of files to skip secret detection	False

Table 8-6. Components Common Optional Parameters

Parameter	Description	Default Value
<code>labels</code>	Carbon Black Container component deployment and pod labels	Empty map
<code>deploymentAnnotations</code>	Carbon Black Container component deployment annotations	Empty map
<code>podTemplateAnnotations</code>	Carbon Black Container component pod annotations	{}
<code>env</code>	Carbon Black Container component pod environment variables	Empty map
<code>image.tag</code>	Carbon Black Container component image tag	Agent version
<code>image.pullPolicy</code>	Carbon Black Container component pull policy	IfNotPresent
<code>probes.port</code>	Carbon Black Container component probes port	8181
<code>probes.scheme</code>	Carbon Black Container component probes scheme	HTTP
<code>probes.initialDelaySeconds</code>	Carbon Black Container component probes initial delay seconds	3
<code>probes.timeoutSeconds</code>	Carbon Black Container component probes timeout seconds	1
<code>probes.periodSeconds</code>	Carbon Black Container component probes period seconds	30

Table 8-6. Components Common Optional Parameters (continued)

Parameter	Description	Default Value
<code>probes.successThreshold</code>	Carbon Black Container component probes success threshold	1
<code>probes.failureThreshold</code>	Carbon Black Container component probes failure threshold	3
<code>prometheus.enabled</code>	Carbon Black Container component enable Prometheus scraping	False
<code>prometheus.port</code>	Carbon Black Container component Prometheus server port	7071
<code>nodeSelector</code>	Carbon Black Container component node selector	<code>{}</code>
<code>affinity</code>	Carbon Black Container component affinity	<code>{}</code>

Table 8-7. Centralized Proxy Parameters

Parameter	Description	Default Value
<code>spec.components.settings.proxy.enabled</code>	Enables applying the centralized proxy settings to all components	False
<code>spec.components.settings.proxy.httpProxy</code>	HTTP proxy server address to use	Empty string
<code>spec.components.settings.proxy.httpsProxy</code>	HTTPS proxy server address to use	Empty string
<code>spec.components.settings.proxy.noProxy</code>	A comma-separated list of hosts to which to connect without using a proxy	Empty string
<code>spec.components.settings.proxy.noProxySuffix</code>	A comma-separated list of hosts to which to append the <code>noProxy</code> list of values	The API server IP addresses followed by <code>cbcontainers-dataplane.svc.cluster.local</code>

Table 8-8. Other Components Optional Parameters

Parameter	Description	Default Value
<code>spec.components.settings.daemonSetsTolerations</code>	Carbon Black DaemonSet component tolerances	Empty array

Changing Components Resources

Needs description/intro.

```
spec:
  components:
    basic:
      monitor:
```

```

resources:
  limits:
    cpu: 200m
    memory: 256Mi
  requests:
    cpu: 30m
    memory: 64Mi
enforcer:
  resources:
    ##### DESIRED RESOURCES SPEC - for hardening enforcer container
stateReporter:
  resources:
    ##### DESIRED RESOURCES SPEC - for hardening state reporter container
runtimeProtection:
resolver:
  resources:
    ##### DESIRED RESOURCES SPEC - for runtime resolver container
sensor:
  resources:
    ##### DESIRED RESOURCES SPEC - for node-agent runtime container
clusterScanning:
imageScanningReporter:
  resources:
    ##### DESIRED RESOURCES SPEC - for image scanning reporter pod
clusterScanner:
  resources:
    ##### DESIRED RESOURCES SPEC - for node-agent cluster-scanner container

```

Cluster Scanner Component Memory

By default, the `clusterScanning.clusterScanner` component attempts to scan images of sizes up to 1GB. Its recommended resources are:

```

resources:
  requests:
    cpu: 100m
    memory: 1Gi
  limits:
    cpu: 2000m
    memory: 6Gi

```

To scan images larger than 1GB, allocate higher memory resources in the component's `requests.memory` and `limits.memory`, and add an environment variable `MAX_COMPRESSED_IMAGE_SIZE_MB` to override the maximum images size in MB that the scanner tries to scan.

For example, to set the cluster scanner to scan images up to 1.5 GB. the configuration is:

```

spec:
  components:
    clusterScanning:
      clusterScanner:
        env:

```

```

MAX_COMPRESSED_IMAGE_SIZE_MB: "1536" // 1536 MB == 1.5 GB
resources:
  requests:
    cpu: 100m
    memory: 2Gi
  limits:
    cpu: 2000m
    memory: 5Gi

```

If your nodes have low memory and you want the cluster scanner to consume less memory, you must reduce the component's `requests.memory` and `limits.memory`, and override the `MAX_COMPRESSED_IMAGE_SIZE_MB` parameter to be less than 1GB (1024MB).

For example, assign lower memory resources and set the cluster-scanner to scan images up to 250MB:

```

spec:
  components:
    clusterScanning:
      clusterScanner:
        env:
          MAX_COMPRESSED_IMAGE_SIZE_MB: "250" // 250 MB
        resources:
          requests:
            cpu: 100m
            memory: 250Mi
          limits:
            cpu: 2000m
            memory: 1Gi

```

Configuring Container Services to use HTTP Proxy

You can configure the Carbon Black Container to use an HTTP proxy by enabling the centralized proxy settings or by manually setting `HTTP_PROXY`, `HTTPS_PROXY`, and `NO_PROXY` environment variables.

The centralized proxy settings apply an HTTP proxy configuration for all components. The manual setting of environment variables allows you to set the configuration parameters on a per component basis. If both HTTP proxy environment variables and centralized proxy settings are provided, the environment variables take precedence. The Operator does not use the centralized proxy settings, so you must use the environment variables for it instead.

Configure Centralized Proxy Settings

To configure the proxy environment variables in the Operator, use the following command to patch the Operator deployment:

```

kubectl set env -n cbcontainers-dataplane deployment cbcontainers-operator HTTP_PROXY="<<proxy-url>" HTTPS_PROXY="<<proxy-url>" NO_PROXY="<<kubernetes-api-server-ip>/<range>"

```

Update the `CBContainersAgent` CR to use the centralized proxy settings (`kubectl edit cbcontainersagents.operator.containers.carbonblack.io cbcontainers-agent`):

```
spec:
  components:
    settings:
      proxy:
        enabled: true
        httpProxy: "<proxy-url>"
        httpsProxy: "<proxy-url>"
        noProxy: "<exclusion1>,<exclusion2>"
```

You can disable the centralized proxy settings without deleting them by setting the `enabled` key to `false`.

By default, the centralized proxy settings determine the API server IP address(es) and the necessary proxy exclusions for the `cbcontainers-dataplane` namespace. These determined values are automatically appended to the `noProxy` values or the specified `NO_PROXY` environment variable for a particular component. To change those pre-determined values, you can specify the `noProxySuffix` key at the same level as the `noProxy` key. It has the same format as the `noProxy` key and its values are treated as if they were pre-determined. You can also force nothing to be appended to `noProxy` or `NO_PROXY` by setting `noProxySuffix` to an empty string.

Configure HTTP Proxy Per-Component Environment Variables

To configure environment variables for the `basic`, `Runtime`, and `Image Scanning` components, update the `CBContainersAgent` CR using the proxy environment variables (`kubectl edit cbcontainersagents.operator.containers.carbonblack.io cbcontainers-agent`):

```
spec:
  components:
    basic:
      enforcer:
        env:
          HTTP_PROXY: "<proxy-url>"
          HTTPS_PROXY: "<proxy-url>"
          NO_PROXY: "<kubernetes-api-server-ip>/<range>"
      stateReporter:
        env:
          HTTP_PROXY: "<proxy-url>"
          HTTPS_PROXY: "<proxy-url>"
          NO_PROXY: "<kubernetes-api-server-ip>/<range>"
    runtimeProtection:
      resolver:
        env:
          HTTP_PROXY: "<proxy-url>"
          HTTPS_PROXY: "<proxy-url>"
          NO_PROXY: "<kubernetes-api-server-ip>/<range>"
    sensor:
      env:
          HTTP_PROXY: "<proxy-url>"
          HTTPS_PROXY: "<proxy-url>"
```



```

    NO_PROXY: "<kubernetes-api-server-ip>/<range>,cbcontainers-runtime-
resolver.cbcontainers-dataplane.svc.cluster.local"
  clusterScanning:
    clusterScanner:
      env:
        HTTP_PROXY: "<proxy-url>"
        HTTPS_PROXY: "<proxy-url>"
        NO_PROXY: "<kubernetes-api-server-ip>/<range>,cbcontainers-image-scanning-
reporter.cbcontainers-dataplane.svc.cluster.local"
    imageScanningReporter:
      env:
        HTTP_PROXY: "<proxy-url>"
        HTTPS_PROXY: "<proxy-url>"
        NO_PROXY: "<kubernetes-api-server-ip>/<range>"

```

Important You must configure the `NO-PROXY` environment variable to use the value of the Kubernetes API server IP address. To find the API-server IP address, run the following command:

```
kubectl -n default get service kubernetes -o=jsonpath='{..clusterIP}'
```

Additional Proxy Considerations

When using a non-transparent HTTPS proxy, you must configure the agent to use the proxy certificate authority:

```

spec:
  gateways:
    gatewayTLS:
      rootCAsBundle: <Base64 encoded proxy CA>

```

Alternatively, you can allow the agent to communicate without verifying the certificate. We do not recommend this option because it exposes the agent to an MITM attack.

```

spec:
  gateways:
    gatewayTLS:
      insecureSkipVerify: true

```

Changing the Image Source

By default, all images for the Operator and Agent deployments are pulled from Docker Hub. If you prefer to mirror the images in your internal repositories, you can specify the image by modifying the `CBContainersAgent` resource that you apply to your cluster.

Modify the following properties to specify the image for each service:

- `monitor` - `spec.components.basic.monitor.image`
- `enforcer` - `spec.components.basic.enforcer.image`
- `state-reporter` - `spec.components.basic.stateReporter.image`

- `runtime-resolver` - `spec.components.runtimeProtection.resolver.image`
- `runtime-sensor` - `spec.components.runtimeProtection.sensor.image`
- `image-scanning-reporter` - `spec.components.clusterScanning.imageScanningReporter.image`
- `cluster-scanner` - `spec.components.clusterScanning.clusterScanner.image`

The image object consists of four properties:

- `repository` - the repository of the image; for example, `docker.io/my-org/monitor`
- `tag` - the version tag of the image; for example, `1.0.0`, `latest`, and so forth.
- `pullPolicy` - the pull policy for that image; for example, `IfNotPresent`, `Always`, or `Never`. See [Image pull policy](#) (external link).
- `pullSecrets` - the image pull secrets that are going to be used to pull the container images. The secrets must already exist in the cluster. See [Pull an Image from a Private Registry](#) (external link).

Sample configuration:

```
spec:
  monitor:
    image:
      repository: docker.io/my-org/monitor
      tag: 1.0.0
      pullPolicy: Always
      pullSecrets:
        - my-pull-secret
```

In this case, the operator attempts to run the monitor service from the `docker.io/my-org/monitor:1.0.0` container image and the kubelet is instructed to always pull the image by using the `my-pull-secret` secret.

Using a Shared Secret for all Images

To use just one pull secret to pull all the custom images, specify it under `spec.settings.imagePullSecrets`.

The secret is added to the `imagePullSecrets` list of all Agent workloads.

Operator Role-based Access Control

This section describes how to configure and use Carbon Black Container Operator Role-based Access Control (RBAC).

RBAC Definition and Design

Following the principle of least-privilege, any permission given to the Operator should have good reason and be scoped as tightly as possible.

In practice, this means:

- If the resource is namespaced and part of the agent, use a `Role` to give permissions in the agent's namespace only.
- If the resource is namespaced and not part of the agent:
 - To read it, use a `ClusterRole` unless you are sure what the namespace will be.
 - To modify it, examine whether this is absolutely necessary.
- If the resource is non-namespaced, use a `ClusterRole` and restrict `delete`, `get`, `update`, and `patch` through `resourceNames`. `Create`, `list`, and `watch` either do not support this restriction or require extra care.

Changing the Operator Access Levels

Operator access level permissions are generated by controller-gen and controlled by using +kubebuilder directives. See [controller definitions](#) (external link). Any change to those directives requires running make manifests to update the respective `role.yaml` file. You must also propagate changes to the helm charts.

Changing the Agent Component Access Levels

Agent component access levels, service accounts, and role bindings are manually maintained in `dataplane_roles.yaml` and the helm equivalent. You must apply changes in both locations.

The roles should follow the least-privilege principle. Agent components often need more permissions than the Operator to work as expected.

Container Operator Developer Instructions

This topic describes instructions using the SDK version 1.29.0 for the Operator.

Deploy the Operator without using an Image

To install dependencies to verify the `kubeconfig` context:

```
make deploy OPERATOR_REPLICAS=0
```

To run the Operator from the terminal to verify the `kubeconfig` context:

```
make run
```

From your editor, run and debug `main.go` to verify the `KUBECONFIG` environment variable.

Install the Dataplane on your own Control Plane

Under the Carbon Black Container Cluster CR:

```
spec:
  apiGatewaySpec:
    adapter: {MY-ADAPTER-NAME}
```

where `{MY-ADAPTER-NAME}` is your control plane adapter name. The default value is `containers`.

Uninstall the Container Operator

From a terminal, run the following command:

```
make undeploy
```

Note This command does not clean up the Carbon Black directory on the dataplane nodes.

Changing Security Context Settings

Hardening enforcer/state_reporter security context settings:

You can change the values under `cbcontainers/state/hardening/objects` for `enforcer_deployment.go` or `state_reporter_deployment.go`.

Using defaults:

Defaults in the `OpenAPISchema` is a feature in `apiextensions/v1` version of `CustomResourceDefinitions`. These default values are supported by `kubebuilder` by using tags; for example, `kubebuilder:default=something`. For backwards compatibility, all defaults should also be implemented and set in the controllers to make sure that they work on clusters v1.15 and below.

Note `kubebuilder` does not support an empty object as a default value. See [related issue](#) (external link). The root issue is in regard to maps, but the same code causes issues with objects.

Therefore, the following specification will not apply the default for `test` unless the user specifies `bar`.

```
spec:
  properties:
    bar:
      properties:
        test:
          default: 10
          type: integer
```

Applying this YAML will save an empty object for `bar`: `spec: {}`.

Instead, applying `spec: { bar: {} }` works as expected and saves the following object:

```
spec: { bar: { test: 10 } }
```

For example:

```
spec:
  properties:
    bar:
      default: {}
      properties:
        test:
          default: 10
          type: integer
```

`kubebuilder` cannot currently produce that output. Therefore, replacing all instance of `<>` with `{}` so that using `kubebuilder:default=<>` produces the correct output.

Defaulting is not supported by `v1beta1` versions of CRD.

Local Debugging

To debug locally, run `make run-delve`. This command builds and starts a delve debugger in headless mode. Then use an editor to start a remote session and connect to the delve instance.

For `goland`, the built-in go remote configuration works.

Custom Namespace

If the Operator is not deployed in the default namespace (`cbcontainers-dataplane`), you must set the `OPERATOR_NAMESPACE` environment variable when using `make run` or `make run-delve`.

Helm Charts

This topic describes the official Helm charts for installing the Carbon Black Container Agent (Operator, CRD, and Agent components).

cbcontainers-operator

The [cbcontainer-operator](#) chart (external link) is the official Helm chart for installing the Carbon Black Container Operator and CRD. Helm 3 is supported.

You can install the chart without any customizations or modifications, and you can create the Helm release in any namespace. You can customize the namespace in which the Operator is installed.

To install the Helm chart from the source:

```
cd charts/cbcontainers-operator
helm install cbcontainers-operator ./cbcontainers-operator-chart
```

Table 8-9. Customization

Parameter	Description	Default Value
<code>spec.operator.image.repository</code>	Repository of the Operator image	<code>cbartifactory/octarine-operator</code>
<code>spec.operator.image.version</code>	Version of the Operator image	The latest version of the Operator image
<code>spec.operator.resources</code>	Carbon Black Container Operator resources	<code>{requests: {memory: "64Mi", cpu: "30m"}, limits: {memory: "256Mi", cpu: "200m"}}</code>
<code>spec.rbacProxy.resources</code>	Kube RBAC proxy resources	<code>{requests: {memory: "64Mi", cpu: "30m"}, limits: {memory: "256Mi", cpu: "200m"}}</code>
<code>spec.operator.environment</code>	Environment variables to be set to the Operator pod	<code>[]</code>

Namespace

By default, the Carbon Black Container Operator is installed in the `cbcontainers-dataplane` namespace.

To change the namespace, set the `operatorNamespace` field in your `values.yaml` file.

The chart automatically creates the namespace. If you do not want to do that (because you have already created the namespace), set the `createOperatorNamespace` field in your `values.yaml` file to `false`.

If the namespace is pre-created, then it must also be labeled properly or the Operator and Agent might not reconcile successfully. The following commands show an example of creating a custom namespace and labeling and installing the operator inside.

```

NAMESPACE=<your_value>
kubectl create namespace $NAMESPACE
kubectl label namespace $NAMESPACE control-plane=operator octarine=ignore
helm install cbcontainers-operator ./cbcontainers-operator-chart --set
createOperatorNamespace=false,operatorNamespace=$NAMESPACE

```

CRD Installation

By default, installing the chart will also create the `CBContainersAgent` CRD.

To manage the CRD in a different way and not install it together with the chart, set the `installCRD` field in your `values.yaml` file to `false`.

HTTP Proxy

To use an HTTP proxy for the communication with the Carbon Black Cloud backend, you must set 3 environment variables. These variables are exposed through the `Values.operator.proxy` parameters in the `values.yaml` file:

- `Values.operator.proxy.http`

- `Values.operator.proxy.https`
- `Values.operator.proxy.noProxy`

See also [Configuring Container Services to use HTTP Proxy](#) .

Templates

The `cbcontainers-operator` chart consists of four [templates](#) (external link).

The `operator.yaml` file (external link) contains all resources except for the Operator deployment. It is generated by `kustomize`. For more info see [config/default_chart](#) (external link).

The `deployment.yaml` file contains the Operator `Deployment` resource. It is derived from [this Kustomize configuration](#). Because it must be configurable through Helm, it is heavily templated. Therefore, it cannot be generated automatically, so it must be maintained by hand. If any changes are made to the [Kustomize configuration](#), they must also be reflected in the `deployment.yaml` file.

The `dataplane_rbac.yaml` and `dataplane_service_accounts` files contain necessary RBAC objects for the Agent to work as expected.

cbcontainers-agent

The `cbcontainer-agent` chart (external link) is the official Helm chart for installing the Carbon Black Container Agent components. Helm 3 is supported.

Note Before installing the Agent components, you must install the Operator and the CRD.

Installation

Before you can install the chart, you must configure it. You must provide the following eight required fields:

Parameter	Description
<code>spec.orgKey</code>	Org key of the organization using Carbon Black Cloud
<code>spec.clusterName</code>	Name of the cluster that will be added to Carbon Black Cloud
<code>spec.clusterGroup</code>	The group that the cluster belongs to in Carbon Black Cloud
<code>spec.version</code>	Version of the Agent images
<code>spec.gateways.apiGatewayHost</code>	URL of the Carbon Black Cloud API gateway
<code>spec.gateways.coreEventsGatewayHost</code>	URL of the Carbon Black Cloud core events gateway
<code>spec.gateways.hardeningEventsGatewayHost</code>	URL of the Carbon Black Cloud hardening events gateway
<code>spec.gateways.runtimeEventsGatewayHost</code>	URL of the Carbon Black Cloud runtime events gateway

After setting these required fields in a `values.yaml` file, you can install the chart from source:

```
cd charts/cbcontainers-agent
helm install cbcontainers-agent ./cbcontainers-agent-chart -n cbcontainers-dataplane
```

Customization

The way in which the Carbon Black Container components are installed is highly customizable.

You can set different properties for the components or enable and disable components by using the `spec.components` section of your `values.yaml` file.

For a list of all possible values, see [Custom Resources Definitions](#).

Namespace

The Carbon Black Cloud Containers Agent will run in the same namespace as the deployed Operator. This is by design because only one running agent per cluster is supported. To customize that namespace, see [operator-chart](#) (external link).

The actual namespace where Helm tracks the release (see `--namespace` flag, external link) is not important to the Agent chart, but the recommended approach is to also use the same namespace as the Operator chart.

The `agentNamespace` value is only required if the Agent chart is responsible for deploying the Agent's secret as well. If the secret is pre-created before deploying the agent, then `agentNamespace` has no effect.

Secret Creation

Carbon Black API Key

For the Agent components to function correctly and communicate with the Carbon Black Cloud backend, an access token is required. This token is located in a *secret*. By default, the secret is named `cbcontainers-access-token`, but that name is configurable through the `accessTokenSecretName` property. If that secret does not exist, the Operator will not start any of the Agent components.

To create the secret as part of the chart installation, provide the `accessToken` value to the chart.

Inject this value as part of your pipeline in a secure way: store the secret as plain text in your `values.yaml` file.

To create the secret in an alternative and more secure way, do not set the `accessToken` value: the chart will not create the secret objects.

Important Do not store the token in your source code.

Carbon Black Company Codes

For the agent CNDR component to function correctly and communicate with the Carbon Black Cloud backend, a company code is required. This code is located in a secret. By default, the secret is named `cbcontainers-company-code`, but that name is configurable through the `components.cndr.companyCodeSecretName` property.

If that secret does not exist, the CNDR component will fail.

If you want to create the secret as part of the chart installation, provide the `companyCode` value to the chart.

Inject this value as part of your pipeline in a secure way: store the secret as plain text in your `values.yaml` file.

To create the secret in an alternative and more secure way, do not set the `companyCode` value: the chart will not create the secret objects.

Important Do not store the code in your source code.
