Illumio Core for Kubernetes

Published: 2024

This document provides an overview of how you can use Illumio Core with Kubernetes or OpenShift.

Table of Contents

Kubernetes and Openshift

Overview of Containers in Illumio Core

This section describes the architecture, key concepts, and the integration requirements to use Illumio Core with Kubernetes or OpenShift.

Before You Begin

- Prepare your environment
- Create a container cluster in the PCE
- Deploy Kubelink and C-VENs in your cluster
- Configure labels for namespaces, pods, and services
- Configure security policies for containerized environments
- Upgrade and uninstall the C-VEN in your containerized environments
- Migrate to a Helm Chart deployment from a previously-installed C-VEN deployment

Recommended Skills

- Illumio Core
- Linux shell (bash)
- TCP/IP networks, including protocols and well-known ports and a familiarity with PKI certificates
- Docker concepts, such as containers, container images, and docker commands.
  See Get Started with Docker.
- Red Hat OpenShift Container Platform.
  See OpenShift Documentation.
- Kubernetes concepts, such as clusters, Pod, and services.
  See Kubernetes Documentation.

Architecture

With the increased adoption of containers, the threat of unauthorized lateral movement from vulnerabilities and exploits increases considerably in the east-west attack surface. In addition, **Destinations** and **Sources** may be other containers, bare-metal servers, or virtual machines running on-premises or in the cloud. Multiple disparate solutions create complexity in management and operational workflow, leaving your organization more open to attack.

Illumio Core provides a homogenous segmentation solution for your applications regardless of where they are running - bare-metal servers, virtual machines, or containers. It is a single unified solution with many points of integration, including how you can easily and quickly secure your applications regardless of their location or form.

A container is a loosely defined construct that abstracts a group of processes into an addressable entity, which can run application instances inside it. Containers are implemented using Linux namespaces and cgroups, allowing you to virtualize and limit system resources.

Since containers operate at a process-level and share the host OS, they require fewer resources than virtual machines. The isolation mechanism provided through Linux namespaces allows containers to have unique IP addresses. Illumio Core uses these mechanisms to program iptables in the network namespace.

Kubernetes-based orchestration platforms such as native Kubernetes and Red Hat OpenShift integrate with Core by using the following two components in the cluster:

- Kubelink - An Illumio software component that listens to events stream on the Kubernetes API server.

  CLAS - (Cluster Local Actor Store) A new architecture introduced in Core for Kubernetes 5.0.0, When Kubelink is enabled with CLAS, it tracks Pods at the Kubernetes Workload level, and dispenses any existing policy for them, reducing the load on and interaction with the Policy Compute Engine (PCE), which improves scalability, responsiveness, and overall system performance.
- Containerized VEN (C-VEN) - An Illumio software component that provides visibility and enforcement on the nodes and the Pods.

The following sections describe some key concepts of the Illumio Core for Kubernetes solution, including more details about its main components, the C-VEN and Kubelink.

Containerized VEN (C-VEN)

The C-VEN provides visibility and enforcement on nodes and Pods. In a standard Illumio deployment the Virtual Enforcement Node (VEN) is installed on the host as a package. In contrast, the C-VEN is not installed on the host but runs as a Pod on the Kubernetes nodes. The C-VEN functions in the same manner as a standard Illumio VEN. However, in order to program iptables on the node and Pods namespaces, the C-VEN requires privileged access to the host. For details on the privileges required by the C-VEN, see Privileges [12].

The C-VENs are delivered as a DaemonSet, with one replica per host in the Kubernetes cluster. A C-VEN Pod instance is required on each node in the cluster to ensure proper segmentation in your environment. In self-managed deployments, C-VENs are deployed on all nodes in the cluster. In cloud-managed deployments, C-VENs are deployed only on the Worker nodes and not on the Master nodes (Master nodes are not managed by Cloud customers).

Kubelink

Kubelink is a software component provided by Illumio to make the integration between the PCE and Kubernetes easier. Starting in Illumio Core for Kubernetes 5.0.0, Kubelink is enhanced with a Cluster Local Actor Store (CLAS) module, that handles the workload-to-Pod relationship via C-VEN communication. See Cluster Local Actor Store (CLAS) below for details on how Kubelink in CLAS mode operates. The remainder of this description of Kubelink describes its basic, non-CLAS behavior.

Kubelink queries Kubernetes APIs to discover nodes, networking details, and services and synchronizes them between the Kubernetes cluster and the PCE. Kubelink reports network information to the PCE, enabling the PCE to understand the cluster network for both the hosts and the Pods in the cluster. This enables the PCE to both accurately visualize the communication flow and create the correct policies for the C-VENs to implement in the iptables of the host and the Pods. It provides flexibility in the type of networking used with the cluster. Kubelink also associates C-VENs with the particular container cluster by matching a unique identifier of the underlying OS called machine-id reported by each C-VEN with the one reported by the Kubernetes cluster.

Kubelink is delivered as a Deployment with only one replica within the Kubernetes cluster. One Kubelink Pod instance is required per cluster. There is no node affinity required for Kubelink, so the Kubelink Pod can be spun up on either a Master or Worker node.

Cluster Local Actor Store (CLAS)

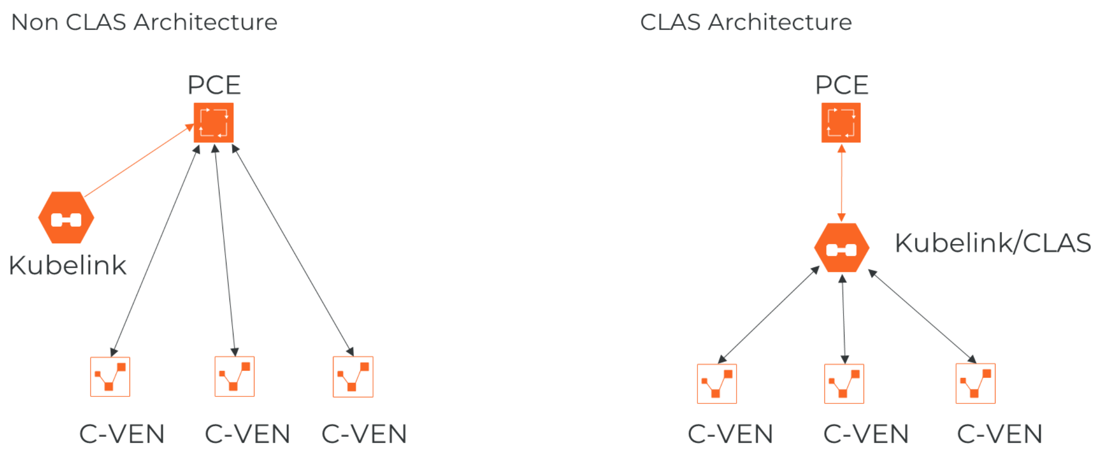
A Cluster Local Actor Store (CLAS) mode is introduced into the architecture of Illumio Core for Kubernetes 5.0.0. When this mode is enabled, Kubelink still interacts with the Kubernetes API to track and manage Kubernetes components, and their interaction with PCE and C-VENs. This includes policy flowing from PCE to C-VENs, and traffic flowing from C-VENs to PCE.

Within the CLAS architecture, Kubelink provides greater scalability, faster responsiveness, and streamlined policy convergence with several key improvements. For example:

- Kubelink now discovers that a new Pod is being created directly from a Kubernetes API event. While Kubernetes (via Kubelet) continues with the process of downloading the proper images, and starting the Pod, Kubelink in CLAS mode is in parallel delivering policy for the emerging Pod to the proper C-VEN to apply.

  Because CLAS stores (caches) all existing policies that have been calculated, C-VENs can get matching policies directly from the CLAS cache without needing to communicate with the PCE, which also improves convergence times.
- With Kubelink now a full intermediary between the PCE and the C-VENs, and maintaining a store of workload data, the C-VENs report traffic flow not to the PCE directly, but now to Kubelink, which "decorates" the flows with the proper Workload IDs based on IP addresses on either end, and then sends this information to the PCE.

The following graphic illustrates the basic difference between the new CLAS architecture and the legacy non-CLAS architecture:



CLAS Degraded Mode

To ensure robustness of policy enforcement and traffic flow in the CLAS architecture, Kubelink and C-VEN can operate in *degraded mode*. If a CLAS-enabled Kubelink detects that its connection with the PCE becomes unavailable (for example, due to connectivity problems or an upgrade), Kubelink by default enters this degraded mode.

In degraded mode, new Pods of existing Kubernetes Workloads get the latest policy version cached in CLAS storage. When Kubelink detects a new Kubernetes Workload labeled the same way and in the same namespace as an existing Kubernetes Workload, Kubelink delivers the existing, cached policy to Pods of this new Workload.

If Kubelink cannot find a cached policy (that is, when labels of a new Workload do not match those of any existing Workload in the same namespace), Kubelink delivers a "fail open" or "fail closed" policy to the new Workload based on the Helm Chart parameter `degradedModePolicyFail`. The degraded mode can also be turned on or off by Helm Chart parameter as well -- `disableDegradedMode`. For more details on degraded mode, see the section on "disableDegradedMode and degradedModePolicyFail" in Deploy with Helm Chart [22].

Kubernetes Workloads

Starting in Illumio Core for Kubernetes 5.0.0, the concept of Kubernetes Workloads is introduced in CLAS-enabled environments as the front-end for the Deployment of an application or service. In contrast to the Container Workload concept used previously (and still used in non-CLAS environments), Kubernetes Workloads now closely match the typical definition of workloads in Kubernetes and similar container orchestration platforms.

Therefore, Kubernetes Workloads as shown in the PCE Web UI are any workloads that have Pods, including but not limited to Deployment or DaemonSet workloads. StatefulSet, DeploymentConfig, ReplicationControler, ReplicaSet, CronJob, Job, Pod, and ClusterIP are also modeled as Kubernetes Workloads in CLAS mode. Kubernetes Workloads replace Container Workloads in the non-CLAS mode.

Container Workloads

Container Workloads are reported only in non-CLAS environments. In these environments, Container Workloads are basic containers (as with Docker), or the smallest resource that can be assimilated within a container in an orchestration system (as with Kubernetes). In the context of Kubernetes and OpenShift, a Pod is a container workload. Similar to workloads reported in Illumio Core, these container workloads (managed Pods) can have labels assigned to them. Container workloads with their associated Illumio labels are also displayed in Illumination. In Illumio Core non-CLAS environments, containers are differentiated based on whether they are on the Pod network or the host network:

- Containers on the Pod network are considered container workloads and can be managed similarly to workloads.
- Containers sharing the host network stack (Pods that are host networked) are not considered as container workloads and therefore inherit the labels and policies of the host.

To manage container workloads, you can define the Policy Enforcement mode (Full, Selective, or Visibility Only) in container workload profiles.

> **NOTE**
> Container Workloads are relevant only in non-CLAS environments. CLAS-enabled environments instead use the concept of Kubernetes Workloads in Illumio Core, which more closely maps to the standard Kubernetes workload concept of an application that is run on any number of dynamically-created (or destroyed) Pods.

Workloads

A workload is commonly referred to as a host OS in Illumio Core. In the context of container clusters, a workload is referred to as a node in a container cluster. Usually, a Kubernetes cluster is composed of two types of nodes:

- One or more Master Node(s) - In the control plane of the cluster, these nodes control and manage the cluster.
- One or more Worker Node(s) - In the data plane of the cluster, these nodes run the application (containers).

In Illumio Core, Master and Worker nodes are called workloads and are part of a container cluster. Labels and policies can be applied to these workloads, similar to any other workload that does not run containers. For a managed Kubernetes solution, only the Worker nodes are visible to the administrator and the Master nodes are not displayed in the list of Workloads.

Virtual Services

Virtual services are labeled objects and can be utilized to write policies for the respective services and the member Pods they represent.

Kubernetes services are represented as virtual services in the Illumio policy model. Kubelink creates a virtual service in the PCE for services in the Kubernetes cluster. Kubelink reports the list of Replication Controllers, DaemonSets, and ReplicaSets that are responsible for managing the Pods supporting that service.

In CLAS mode, only NodePort and LoadBalancer services are reported in the PCE UI as virtual services. Replication Controllers, DaemonSets, and ReplicaSets are no longer reported as virtual service backends in CLAS.

Container Cluster

A container cluster object is used to store all the information about a Kubernetes cluster in the PCE by collecting telemetry from Kubelink. Each Kubernetes cluster maps to one container cluster object in the PCE. Each Pod network(s) that exists on a container cluster is uniquely identified on the PCE in order to handle overlapping subnets. This helps the PCE in differentiating between container workloads that may have the same IP address but are running on two different container clusters. This differentiation is required both for Illumination and for policy enforcement.

You can see the workloads that belong to a container cluster in the PCE Web Console. This mapping between the host workload and the container cluster is done using machine-ids reported by Kubelink and C-VEN.

Container Workload Profiles

A Container Workload Profile maps to a Kubernetes namespace, and defines:

- Policy Enforcement state (Full, Selective, or Visibility Only) for the Pods and services that belong to the namespace.
- Labels assigned to the Pods and services. Standard predefined label types were Role, Application, Environment, and Location. Newer releases of Core allow you to define your own custom label types and label values for these types.

Once Illumio Core is installed on a container cluster, all namespaces that exist on the clusters are reported by Kubelink to the PCE and made visible via Container Workload Profiles. Each time Kubelink detects the creation of a namespace from Kubernetes, a corresponding Container Workload Profile object gets dynamically created in the PCE.

After creating a Container Workload Profile, be sure to copy the pairing key that is automatically generated, and save it. Use this key for the `cluster_code` Helm Chart parameter value when installing.

Each profile can either be in a managed or unmanaged state. The default state for a profile is unmanaged. The main difference between both states:

- Unmanaged: no policy applied to Pods by the PCE, and no visibility
- Managed: policy is controlled by the PCE, and full visibility through Illumination and traffic explorer

In a CLAS environment, Kubernetes Workloads are displayed only for managed Container Workload Profiles.

A Container Workload Profile is a convenient way to dynamically secure new applications with Illumio Core just by inheriting security policies associated with the scope of that profile.

For more information about Container Workload Profiles, see Use Container Workload Profiles [52].

Deployment with Helm Chart (Core for Kubernetes 3.0.0 and Later)

After you set up your clusters, make sure you do the steps in the order provided in this section.

NOTE

Illumio Core for Kubernetes 3.0.0 and later is a combined release of C-VEN and Kubelink. Starting with C-VEN 21.5.17 and Kubelink 3.0, C-VEN and Kubelink 3.0 will be only used through the combined release. A Helm Chart (via quay.io) is used to deploy all necessary product components. If you are deploying C-VEN 21.5.15 or earlier, instead follow the deployment instructions in Deployment for C-VEN Versions 21.5.15 or Earlier [28].

The installation process is mostly the same for Kubernetes and OpenShift, except a few steps differ. A dedicated section is created for Kubernetes or OpenShift wherever required.

You also have the option to manually deploy components with YAML manifests that are first generated by Helm, but are not actually deployed with a Helm chart. See Generating YAML Manifests for Manual Deployment [26] for details.

Helm Chart Deployment Overview

Starting with the Illumio Core for Kubernetes 3.0.0 release and later, the product (including C-VEN and Kubelink) is now deployed by using a Helm Chart. The product components and the Helm Chart are downloaded from a public container repository: https://quay.io/repository/illumio/illumio.

Use these steps to deploy Helm Chart:

1. Deploy and configure your PCE. See the PCE Installation and Upgrade Guide.
2. Create a container cluster. See Create a Container Cluster in the PCE [18].
3. Create a pairing profile. See Create a Pairing Profile for Your Cluster Nodes [20].
4. Deploy Helm Chart. See Deploy with Helm Chart [22]. At this stage you can optionally map existing Kubernetes labels to Illumio labels.

Follow the sections in this order, including the requirements and environment preparations described next.

Host and Cluster Requirements

To deploy Illumio containers into your environment, you must meet the following requirements.

Supported Configurations for On-premises and IaaS

For full details on all supported configurations for Illumio Core for Kubernetes version 3.0.0 and later, see the Kubernetes Operator OS Support and Dependencies page on the Illumio Support Portal (under **Software** > **OS Support**).

Privileges

The Helm Chart deployment process automatically sets all necessary privileges. The privileges listed below must be provided on host-level and cluster-level for the respective components. They are listed here for reference.

Host-Level

C-VEN

C-VEN requires the following privileges on the host:

- C-VEN is a privileged container and requires access to the following system calls:
  - `NET_ADMIN`
  - `SYS_MODULE`
  - `SYS_ADMIN`
- C-VEN requires persistent storage on the host to write iptables rules and logs.
- C-VEN mounts volumes on the local host to be able to operate (mount points may differ depending on the orchestration platform).

Kubelink

Kubelink does not require specific privileges on the host because Kubelink:

- is not a privileged container
- is a stateless container
- does not require persistent storage

Cluster-Level

Namespace

C-VENs and Kubelink are deployed in the `illumio-system` namespace.

C-VEN

C-VEN requires the following privileges on the cluster:

- C-VEN uses the `illumio-ven` ServiceAccount.

Kubelink

Kubelink requires the following privileges on the cluster:

- Kubelink creates a new Cluster Role to list and watch events occurring on the Kubernetes API server for the following elements:
  - `nodes`
  - `hostsubnets`
  - `replicationcontrollers`
  - `services`
  - `replicasets`
  - `daemonsets`
  - `namespaces`
  - `statefulsets`
- Kubelink uses the `illumio-kubelink` ServiceAccount.

Prepare Your Environment

You need to do these steps before creating clusters or pairing profiles in the PCE, or subsequent deployment.

> **CAUTION**
> If the prerequisite steps are not done before deployment, then containerized environments and Kubelink can get disrupted.

Unique Machine ID

Some of the functionality and services provided by the Illumio C-VEN and Kubelink depend on the Linux machine ID of each Kubernetes cluster node. Each machine ID must be unique in order to take advantage of the functionality. By default, the Linux operating system generates a random machine IDto give each Linux host uniqueness. However, there are cases when machine IDs can be duplicated across machines. This is common across deployments that clone machines from a golden image, for example, spinning up virtual machines from VMware templates, creating compute instances from a reference image, or from a template from a Public Cloud provider.

> **IMPORTANT**
> Illumio Core requires a unique machine ID on all nodes. This issue is more likely to occur with on-premises or IaaS deployments, rather than with Managed Kubernetes Services (from Cloud Service Providers). For more information, see "Troubleshooting".

Create Labels

For details on creating labels, see "Labels and Label Groups" in Security Policy Guide. The labels shown below are used in examples throughout this document. You are not required to use the same labels

Name	Label Type
Kubernetes Cluster	Application
OpenShift Cluster	Application
Production	Environment
Development	Environment
Data Center	Location
Cloud	Location

Name	Label Type
Kubelink	Role
Node	Role
Control Plane Node (formerly Master)	Role
Worker	Role

NOTE
Starting in Illumio Core for Kubernetes 4.2.0, you can map Kubernetes labels to Illumio labels by using a Container Resource Definition in your **`illumio-values.yaml`** with the Helm Chart deployment. See Map Kubernetes Labels to Illumio Labels [21] for details.

Create a ConfigMap to Store Your Root CA Certificate

This section describes how to implement Kubelink with a PCE using a certificate signed by a private PKI. It describes how to configure Kubelink and C-VEN to accept the certificate from the PCE signed by a private root or intermediate Certificate Authority (CA), and ensure that Kubelink can communicate in a secure way with the PCE.

Prerequisites

- Access to the root CA to download the root CA certificate
- Access to your Kubernetes cluster and can run `kubectl` commands
- Correct privileges in your Kubernetes cluster to create resources like ConfigMaps, secrets, and Pods
- Access to the PCE web console as a Global Organization Owner

Download the Root CA Certificate

Before you begin, ensure that you have access to the root CA certificate. The root CA certificate is a file that can be exported from the root CA without compromising the security of the company. It is usually made available to external entities to ensure a proper SSL handshake between a server and its clients.

You can download the root CA certificate in the CRT format on your local machine. Below is an example of a root CA certificate:

```
$ cat root.democa.illumio-demo.com.crt
-----BEGIN CERTIFICATE-----
MIIGSzCCBDOgAwIBAgIUAPw0NfPAivJW4YmKZ499eHZH3S8wDQYJKoZIhvcNAQEL
---output suppressed---
wPG0lug46K1EPQqMA7YshmrwOd6ESy6RGNFFZdhk9Q==
-----END CERTIFICATE-----
```

You can also get the content of your root CA certificate in a readable output format by using the following command:

```
$ openssl x509 -text -noout -in ./root.democa.illumio-demo.com.crt
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            fc:34:35:f3:c0:8a:f2:56:e1:89:8a:67:8f:7d:78:76:47:dd:2f
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=US, ST=California, L=Sunnyvale, O=Illumio,
         OU=Technical Marketing, CN=Illumio Demo Root
         CA 1/emailAddress=tme-team@illumio.com
        Validity
            Not Before: Jan 20 00:05:36 2020 GMT
            Not After : Jan 17 00:05:36 2030 GMT
        Subject: C=US, ST=California, L=Sunnyvale, O=Illumio,
        OU=Technical Marketing, CN=Illumio Demo Root
        CA 1/emailAddress=tme-team@illumio.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (4096 bit)
                Modulus:
                    00:c0:e5:48:7d:97:f8:5b:8c:ef:ac:16:a8:8c:aa:
                    68:b8:48:af:28:cd:17:8f:02:c8:82:e9:69:62:e2:
                    89:2b:be:bd:34:fc:e3:4d:3f:86:5e:d7:e6:89:34:
                    71:60:e6:54:61:ac:0f:26:1c:99:6f:80:89:3f:36:
                    b3:ad:78:d1:6c:3f:d7:23:1e:ea:51:14:48:74:c3:
                    e8:6e:a2:79:b1:60:4c:65:14:2a:f1:a0:97:6c:97:
                    50:43:67:07:b7:51:5d:2c:12:49:81:dc:01:c9:d1:
                    57:48:32:2e:87:a8:d2:c0:b9:f8:43:b2:58:10:af:
                    54:59:09:05:cb:3e:f0:d7:ef:70:cc:fc:53:48:ee:
                    a4:a4:61:f1:d7:5b:7c:a9:a8:92:dc:77:74:f4:4a:
                    c0:4a:90:71:0f:6d:9e:e7:4f:11:ab:a5:3d:cd:4b:
                    8b:79:fe:82:1b:16:27:94:8e:35:37:db:dd:b8:fe:
                    fa:6d:d9:be:57:f3:ca:f3:56:aa:be:c8:57:a1:a8:
                    c9:83:dd:5a:96:5a:6b:32:2d:5e:ae:da:fc:85:76:
                    bb:77:d5:c2:53:f3:5b:61:74:e7:f3:3e:4e:ad:10:
                    7d:4f:ff:90:69:7c:1c:41:2f:67:e4:13:5b:e6:3a:
                    a3:2f:93:61:3b:07:56:59:5a:d9:bc:34:4d:b3:54:
                    b5:c6:e5:0a:88:e9:62:7b:4b:85:d2:9e:4c:ee:0b:
                    0d:f4:72:b1:1b:44:04:93:cf:cc:bb:18:31:3a:d4:
                    83:4a:ff:15:42:2d:91:ca:d0:cb:36:d9:8d:62:c0:
                    41:59:1a:93:c7:27:79:08:94:b2:a2:50:3c:57:27:
                    33:af:f0:b6:92:44:49:c5:09:15:a7:43:2a:0f:a9:
                    02:61:b3:66:4f:c3:de:d3:63:1e:08:b1:23:ea:69:
                    90:db:e8:e9:1e:21:84:e0:56:e1:8e:a1:fa:3f:7a:
                    08:0f:54:0a:82:41:08:6b:6e:bb:cf:d6:5b:80:c6:
                    ea:0c:80:92:96:ab:95:5d:38:6d:4d:da:38:6b:42:
                    ef:7c:88:58:83:88:6d:da:28:62:62:1f:e5:a7:0d:
                    04:9f:0d:d9:52:39:46:ba:56:7c:1d:77:38:26:7c:
                    86:69:58:4d:b0:47:3a:e2:be:ee:1a:fc:4c:de:67:
                    f3:d5:fe:e6:27:a2:ef:26:86:19:5b:05:85:9c:4c:
                    02:24:76:58:42:1a:f8:e0:e0:ed:78:f2:8f:c8:5a:
                    20:a9:2d:0b:d4:01:fa:57:d4:6f:1c:0a:31:30:8c:
                    32:7f:b0:01:1e:fe:94:96:03:ee:01:d7:f4:4a:83:
                    f5:06:fa:60:43:15:05:9a:ca:88:59:5c:f5:13:09:
                    82:69:7f
```

```
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                3D:3D:3D:61:E6:88:09:FE:34:0F:1D:5E:5E:52:72:71:C7:DE:15:92
            X509v3 Authority Key Identifier:
                keyid:3D:3D:3D:61:E6:88:09:FE:34:0F:1D:5E:5E:52:72:71:C7:
                DE:15:92

            X509v3 Basic Constraints: critical
                CA:TRUE
            X509v3 Key Usage: critical
                Digital Signature, Certificate Sign, CRL Sign
    Signature Algorithm: sha256WithRSAEncryption
         28:24:86:91:a6:4a:88:e4:8d:6b:fc:67:2a:68:08:67:35:e5:
         a6:77:ff:07:4b:89:53:99:2e:6d:95:df:12:81:28:6a:8e:6f:
         5a:98:95:5b:4a:21:ae:f0:20:a4:4e:06:b2:4e:5a:67:c1:6a:
         06:f1:0f:c1:f7:7e:f2:e0:b3:9d:d8:54:26:6a:b2:1c:19:b8:
         b5:5c:c7:03:6b:f7:70:9e:72:85:c9:29:55:f9:f4:a4:f2:b4:
         3b:3d:ce:25:96:67:32:1e:8d:e2:00:22:55:4b:05:4f:ee:0e:
         67:ac:db:1b:61:da:5f:9c:10:1c:0c:05:66:c0:5b:5f:b9:95:
         59:a9:58:5b:e7:69:ac:b0:bd:b3:c2:a3:35:58:01:a4:ff:c0:
         8d:ac:1c:19:21:41:50:fb:8e:e0:f5:a9:ad:ec:de:cb:53:04:
         a9:d8:ac:76:8a:09:0d:7c:c6:1a:bc:06:74:bb:10:1c:aa:07:
         f6:cb:b2:1b:0c:0c:65:03:45:2b:51:d5:6e:a0:4d:91:ce:c5:
         ed:8d:a9:e7:f6:37:7d:ab:1b:a4:a2:a3:3b:76:17:5b:d9:3a:
         9c:c1:df:cc:cd:a0:b0:a9:5c:74:61:d7:a0:1d:04:67:68:ee:
         a6:7b:1e:41:a4:02:fc:65:9e:e3:c1:c2:57:b2:2e:b0:ff:a9:
         86:82:35:4d:29:b2:fe:74:2e:b8:37:5d:2b:e8:69:f2:80:29:
         19:f1:1e:7a:5d:e3:d2:51:50:46:30:54:7e:b8:ad:59:61:24:
         45:a8:5a:fe:19:ff:09:31:d0:50:8b:e2:15:c0:a2:f1:20:95:
         63:55:18:a7:a2:ad:16:25:c7:a3:d1:f2:e5:be:6d:c0:50:4b:
         15:ac:e0:10:5e:f3:7b:90:9c:75:1a:6b:e3:fb:39:88:e4:e6:
         9f:4c:85:60:67:e8:7d:2e:85:3d:87:ed:06:1d:13:0b:76:d7:
         97:a5:b8:05:76:67:d6:41:06:c5:c0:7a:bd:f4:c6:5b:b2:fd:
         23:6f:1f:57:2e:df:95:3f:26:a5:13:4d:6d:96:12:56:98:db:
         2e:7d:fd:56:f5:71:b7:19:2b:c9:de:2d:b9:c8:17:cc:20:de:
         7c:19:7a:aa:12:97:1c:80:b7:d3:67:d3:b7:a7:96:f0:c9:4d:
         f5:8b:0e:10:3b:b9:4e:09:90:5a:3b:51:c9:48:a2:ca:9f:db:
         72:44:87:59:db:49:fa:75:44:b5:f6:7f:c5:26:e1:01:ae:7b:
         6f:4a:75:d1:b5:b3:68:c0:31:48:f8:5c:06:c0:f1:b4:96:e8:
         38:e8:ad:44:3d:0a:8c:03:b6:2c:86:6a:f0:39:de:84:4b:2e:
         91:18:d1:45:65:d8:64:f5
```

Create a ConfigMap in the Kubernetes Cluster

After downloading the certificate locally on your machine, create a ConfigMap in the Kubernetes cluster that will copy the root CA certificate on your local machine into the Kubernetes cluster.

To create a ConfigMap, use the following command:

```
$ kubectl -n illumio-system create configmap root-ca-config \
    --from-file=./certs/root.democa.illumio-demo.com.crt
```

The `--from-file` option points to the path where the root CA certificate is stored on your local machine.

To verify that ConfigMap was created correctly, use the following command:

```
$ kubectl -n illumio-system create configmap root-ca-config \
> --from-file=./certs/root.democa.illumio-demo.com.crt
configmap/root-ca-config created
$
$ kubectl -n illumio-system get configmap
NAME                                 DATA   AGE
root-ca-config                       1      12s
$
$ kubectl -n illumio-system describe configmap root-ca-config
Name:         root-ca-config
Namespace:    illumio-system
Labels:       <none>
Annotations:  <none>

Data
====
root.democa.illumio-demo.com.crt:
----
-----BEGIN CERTIFICATE-----
MIIGSzCCBDOgAwIBAgIUAPw0NfPAivJW4YmKZ499eHZH3S8wDQYJKoZIhvcNAQEL
---output suppressed---
wPG0lug46K1EPQqMA7YshmrwOd6ESy6RGNFFZdhk9Q==
-----END CERTIFICATE-----

Events:  <none>
$
```

`root-ca-config` is the name used to designate the ConfigMap. You can modify it according to your naming convention.

Configure Calico in Append Mode

In case your cluster is configured with Calico as the network plugin (usually for Kubernetes and not for OpenShift), both Calico and Illumio Core will write iptables rules on the cluster nodes.

- Calico - Needs to write iptables rules to instruct the host how to forward packets (overlay, IPIP, NAT, and so on).
- Illumio Core - Needs to write iptables rules to secure communications between nodes and/or Pods.

You should establish a hierarchy to make the firewall coexistence work smoothly because Illumio Core and Calico will write rules at the same time. By default, both solutions are configured to insert rules first in the iptables chains/tables and Illumio Core will remove other rules added by a third-party software (in the Exclusive mode).

To allow Calico to write rules along with Illumio without flushing rules from one another, you should:

- Configure Illumio to work in Firewall Coexistence mode (default for workloads that are part of a container cluster).

- Configure Calico to work in Append mode (default is Insert mode).

To configure Calico to work in Append mode with iptables:

1. Edit the Calico DaemonSet:

   ```
   kubectl -n kube-system edit ds calico-node
   ```
2. Locate the `spec: > template: > spec: > containers:` section inside the `YAML` file and change ChainInsertMode by adding the following code block:

   ```
   - name: FELIX_CHAININSERTMODE
     value: Append
   ```
3. Save your changes and exit.
4. Kubernetes will restart all Calico Pods in a rolling update.

For more information on changing Calico ChainInsertMode, see Calico documentation.

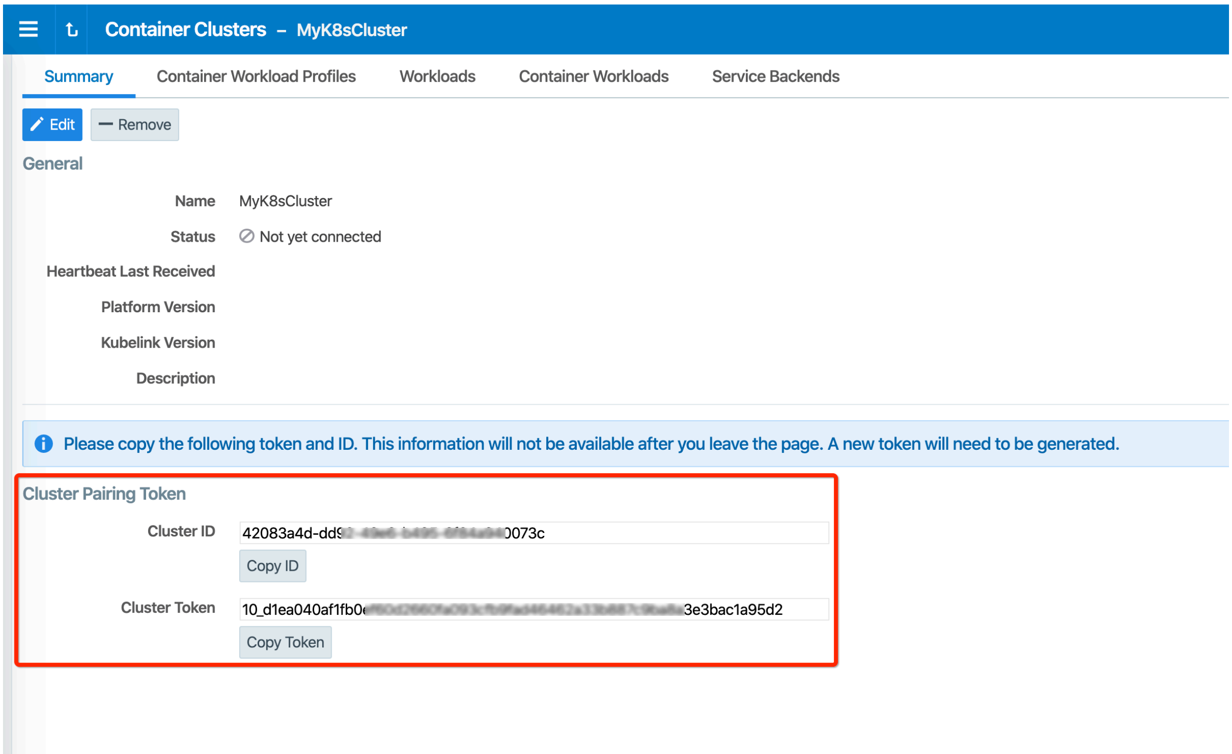
Create a Container Cluster in the PCE

To provide visibility and enforcement to your containerized environment, you first need to create a container cluster in the PCE. Each container cluster maps to an existing Kubernetes or OpenShift cluster.

Create a Container Cluster

To create a new container cluster:

1. Log into the PCE web console as a user with Global Organization Owner privileges.
2. From the PCE web console menu, navigate to **Infrastructure** > **Container Clusters**.
3. Click **Add**.
   a. Add a Name.
   b. **Save** the Container Cluster.
4. You will see a summary page of the new Container Cluster. From the Cluster Pairing Token section, copy the values of the Cluster ID and Cluster Token.
5. After copying and saving the values (in a text editor or similar tool), open the Container Workload Profiles page.



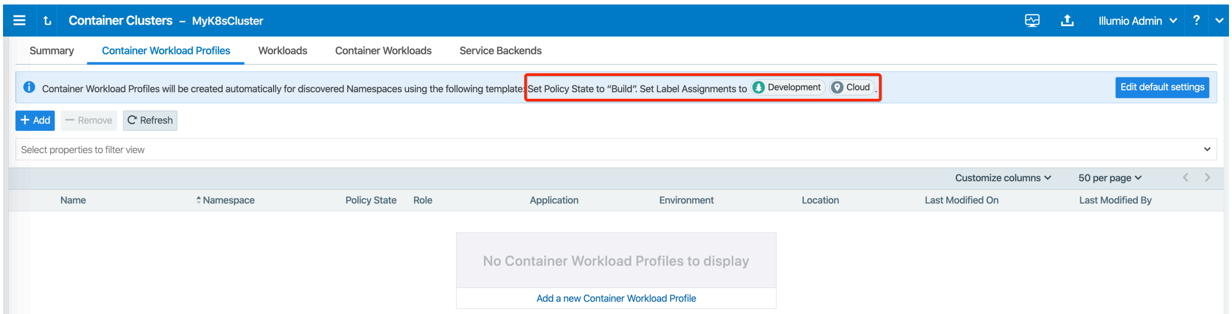
Configure a Container Workload Profile Template

When configuring a new Container Cluster, it is recommended to set the default settings shared by all the Container Workload Profiles. Illumio provides a Container Workload Profile template that can be used for that purpose. By defining the default Policy State and minimum set of labels common to all namespaces in the cluster, you will save time later on when new namespaces are discovered by Kubelink. Each new profile created will inherit what was defined in the template.

> **IMPORTANT**
>
> Illumio does not provide a method to redefine at once all the labels associated with each profile. Hence, it is **strongly recommended** to use the provided template to define the default values for all profiles that are part of the same cluster.

To define the default parameters for all profiles using a template, under *Container Workload Profiles*, click **Edit default settings** and select values for all the fields.

After you click OK, the following information is displayed:



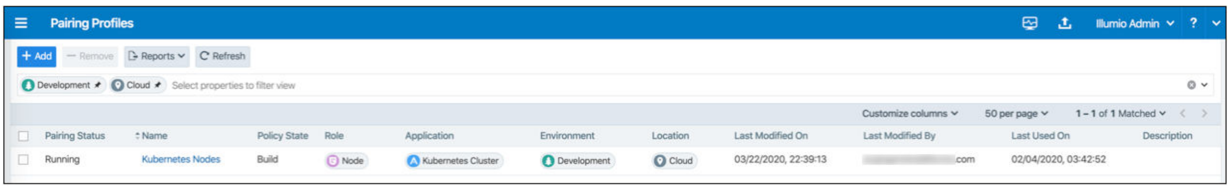
Create a Pairing Profile for Your Cluster Nodes

> **IMPORTANT**
>
> Before deploying the C-VEN, ensure that either of the following two requirements has been met:
>
> - Kubelink is deployed on the Kubernetes cluster and is in sync with the PCE, or
> - Firewall coexistence is enabled.

Before deploying, you should create a pairing profile to pair the cluster nodes with the PCE. You only need to create one pairing profile for all your nodes.

> **NOTE**
>
> You only need to create pairing profiles for Kubernetes or OpenShift nodes and not for container workloads.

For ease of configuration and management, consider applying the same Application, Environment, and Location labels across all nodes of the same Kubernetes or OpenShift cluster. The screenshot below shows an example of a pairing profile for a Kubernetes cluster.



TIP

Illumio recommends all pairing profiles for Kubernetes nodes to *not* use Full Enforcement policy state. Use Idle or Visibility Only mode for initial configuration.

You should only move them into Full enforcement state after you have completed all other configuration steps in this guide.

Map Kubernetes Node Labels to Illumio Labels

Label mapping is a method of mapping some or all existing Kubernetes node labels to Illumio labels. Label maps are a new way to assign Illumio labels to container host workloads in addition to existing methods (such as with container workload profiles and pairing profiles). Labels assigned through label maps take precedence over these other methods -- that is, they overwrite any labels assigned with these other methods.

A label map is defined by a Kubernetes *Custom Resource Definition* (CRD) within a yaml file that is typically installed via a Helm Chart. Installing the Helm Chart then applies the defined labels.

Label Mapping CRD

The CRD is defined in the yaml file with a `kind: LabelMap` declaration, which in turn contains a `nodeLabelMap` section that applies to nodes (host workloads).

Within the `nodeLabelMap` section, Illumio label types are mapped with `fromKey` and `toKey` key-value pairs, where the `fromKey` value specifies a source Kubernetes label, and the `toKey` value paired with it defines the destination Illumio label type.

If an optional `allowCreate: true` is within a `fromKey` and `toKey` pair, the Illumio label type defined in that mapping is created if it does not already exist on the PCE.

An optional `valuesMap:` within a `fromKey` and `toKey` pair specifies one or more label value mappings for that label type, with `from:` value identifying the source Kubernetes label and the `to:` value following it specifying the destination Illumio label value. If no `valuesMap:` is specified, then label values for the mapped label type are not changed. Only the label type is changed in the PCE.

Example Label Map

Note these points about the following example label map:

- The first `nodeLabelMap` item creates a new Illumio `location` label of `Amazon` (if it does not exist, per the `allowCreate: true` declaration) and maps this label to all nodes with the Kubernetes label `topology.kubernetes.io/region` with either value of `eu-west-1` or `eu-west-2`.

- With the second item under `nodeLabelMap`, for every `node-type` Kubernetes label, the map creates Illumio `k8s-node` labels with values based on the existing Kubernetes label values (because there is no associated `valuesMap` mapping definition).

```
kind: LabelMap
apiVersion: ic4k.illumio.com/v1alpha1
metadata:
  name: default
nodeLabelMap:
  - allowCreate: true
    fromKey: topology.kubernetes.io/region
    toKey: loc
    valuesMap:
      - from: eu-west-1
        to: Amazon
      - from: eu-west-2
        to: Amazon
- allowCreate: true
  fromKey: node-type
  toKey: k8s-node
```

The label type has to be created and exist in PCE first before new labels can be created through label mapping.

Deploy with Helm Chart

To deploy via Helm Chart:

1. Install Helm. Refer to https://helm.sh/docs/ for a quick start guide and other relevant information.
   According to official Helm documentation, if your version of Helm is lower than 3.8.0, the following command must be executed in the installation environment:
   ```
   $ export HELM_EXPERIMENTAL_OCI=1
   ```
2. Prepare an **`illumio-values.yaml`** file with the following mandatory parameters set with values that describe this deployment:
   ```
   pce_url: URL_PORT # PCE URL with port, e.g. mypce.example.com:8443
   cluster_id: ILO_CLUSTER_UUID # Cluster ID from PCE, e.g. cc4997c1-40...
   cluster_token: ILO_CLUSTER_TOKEN # Cluster Token from PCE, e.g.
   1_170b...
   cluster_code: ILO_CODE # Pairing Profile key from PCE, e.g. 1391c...
   containerRuntime: containerd # Container runtime engine used in cluster,
   allowed values are [containerd, crio, k3s_containerd]
   containerManager: kubernetes # Container manager used in cluster,
   allowed values are [kubernetes, openshift]
   ```
   where **URL_PORT**, **ILO_CLUSTER_UUID**, **ILO_CLUSTER_TOKEN**, and **ILO_CODE** are placeholders for customer provided variables.
   If you are using a private PKI, you need to add these additional lines to your `illumio_values.yaml`:
   ```
   extraVolumeMounts:
       - name: root-ca
   ```

```
    mountPath: /etc/pki/tls/ilo_certs/
    readOnly: false
extraVolumes:
    - name: root-ca
    configMap:
        name: root-ca-config
ignore_cert: true
```

You may also want to include selected optional parameters when installing, for example, with `clusterMode: clas` to deploy with a CLAS-enhanced Kubelink component. For more information, see Important Optional Parameters [23].

1. **IMPORTANT**

   If you want to deploy with CLAS enabled, you must explicitly set the `clusterMode` Helm Chart parameter. The default is to deploy in legacy (non-CLAS) mode

2. Optionally map existing Kubernetes labels to desired Illumio labels by adding a Kubernetes *Custom Resource Definition* (CRD) label map to your **`illumio-values.yaml`** file. For details on using a label map, see the "Map Kubernetes Labels to Illumio Labels" topic.
3. Install the Helm Chart:

```
$ helm install illumio -f illumio-values.yaml oci://quay.io/illumio/
illumio
--version <ver#> --namespace illumio-system --create-namespace
```

IMPORTANT

Be sure to explicitly specify the version to install with the `--version` <ver#> option (for example, `--version 5.1.0`), after confirming that the product version you want to install is supported with your PCE version. Verify which PCE versions support the Illumio Core for Kubernetes version you want to deploy at the Kubernetes Operator OS Support and Dependencies page on the Illumio Support Portal.

In case the illumio-system namespace already exists, omit the **`--create-namespace`** flag.

NOTE

Kubelink version labeling has changed. Prior to version 3.3.0, Kubelink used a 6-hexit suffix for its release version, like 3.2.1.445a83. In Kubelink 3.3.0 and later, the version suffix is now changed to a numeric build number, like 3.3.0-56.

Important Optional Parameters

Refer to the **README** file included with the Helm Chart for important deployment information, including additional parameters you can specify in the Helm Chart before installing it.

The following list describes a few important optional parameters to consider using in your `illumio-values.yaml` file.

Flat Networks: networkType

To add support for flat network CNIs in addition to the default (where pods run on an overlay network), an optional **`networkType`** parameter is now available in the Helm Chart where you can specify **`flat`** or **`overlay`** type. The default value is **`overlay`**.

CLAS Mode: clusterMode

Starting in Illumio Core for Kubernetes versions 5.0.0 and later, a Cluster Local Actor Store (CLAS) mode is introduced into the Kubelink architecture. Use the optional `clusterMode` parameter to configure Kubelink when first installing a new cluster, or when migrating an existing cluster.

When installing, set `clusterMode` to `clas` or `legacy` in your **`illumio-values.yaml`** file to turn on (or leave off) CLAS mode in the cluster, respectively. The default setting for `clusterMode` is `legacy` (non-CLAS). To enable CLAS in a new cluster, you must explicitly include `clusterMode:clas` in the **`illumio-values.yaml`** file when installing.

When upgrading an existing non-CLAS cluster to CLAS, set `clusterMode` to `migrateLegacyToClas`. When reverting (or downgrading) CLAS to non-CLAS, set `clusterMode` to `migrateClasToLegacy`. For more information about upgrading to a CLAS-enabled cluster, see Upgrade to CLAS Architecture [91].

IMPORTANT

To properly upgrade to CLAS, you must follow the procedure described in Upgrade to CLAS Architecture [91].

Illumio recommends enabling (or migrating to) CLAS-enabled clusters to take advantage of this architecture's benefits. For more information about CLAS, see the "Cluster Local Actor Store (CLAS)" section in the Architecture [7] topic.

CLAS Degraded Mode: disableDegradedMode and degradedModePolicyFail

If the connection between Kubelink and the PCE becomes unavailable, a CLAS-enabled Kubelink can still serve policies to C-VEN (and therefore to its Kubernetes Workloads and pods). When a PCE interruption is detected, a CLAS-enabled Kubelink enters a **degraded mode**.

By default, degraded mode is enabled in CLAS clusters. You can disable degraded mode by explicitly setting the parameter/value pair `disableDegradedMode: true` in **`illumio-values.yaml`**, and performing a **`helm upgrade`**.

In degraded mode, new Pods of existing Kubernetes Workloads get the latest policy version cached in CLAS storage. When Kubelink detects a new Kubernetes Workload labeled the same way and in the same namespace as an existing Kubernetes Workload, Kubelink delivers the existing, cached policy to Pods of this new Workload.

If Kubelink cannot find a cached policy (that is, when labels of a new Workload do not match the labels of any existing Workload in the same namespace), Kubelink delivers a "fail open" or "fail closed" policy based on the Helm Chart parameter `degradedModePolicyFail` setting, as specified in the **`illumio-values.yaml`** file when installing (or upgrading).

The default parameter value of `degradedModePolicyFail` is `open`, which opens the firewall of new Pods. The `closed` value means the firewall of new Pods is programmed to block all network connectivity.

The precise behavior of `closed` depends on the Cluster Workload Profile's Enforcement setting: all connectivity is blocked only if the Enforcement of the namespace is set to Full.

By default, degraded mode is enabled in CLAS clusters. You can disable degraded mode by explicitly setting the parameter/value pair `disableDegradedMode: true` in **`illumio-values.yaml`**, and performing a **`helm upgrade`**.

When degraded mode is disabled, Kubelink/CLAS does not deliver policy based on matching labels. Kubelink continues to run, and delivers the cached policy to existing Kubernetes Workloads, but does not deliver policy to new Workloads. Kubelink continues to attempt re-establishing communication with the PCE.

After the PCE becomes available again, it restarts, synchronizes policy and labels, and then continues normal operation.

> **NOTE**
> If the PCE becomes inaccessible due to database restoration or maintenance, and Kubelink has disabled degraded mode, you are advised to restart Kubelink by deleting its Pod to synchronize the current state.

CLAS etcd Internal Storage Size: sizeGi

Kubelink in CLAS mode uses etcd as a local cache for policy and runtime data. The Helm Chart parameter `storage.sizeGi` sets the size in GB of this ephemeral storage. Set the parameter under `storage` in the **`illumio-values.yaml`** for a cluster, as shown in the following example:

```
storage:
  registry: "docker.io/bitnami"
  repo: "etcd"
  imageTag: "3.5.7"
  imagePullPolicy: "IfNotPresent"
  sizeGi: 1
```

The default value is 1, for 1 GB, which should be enough for a cluster with under 1000 Kubernetes workloads. If a cluster is bigger and you increase memory limits for C-VEN and Kubelink, then increase the etcd internal storage size with this parameter.

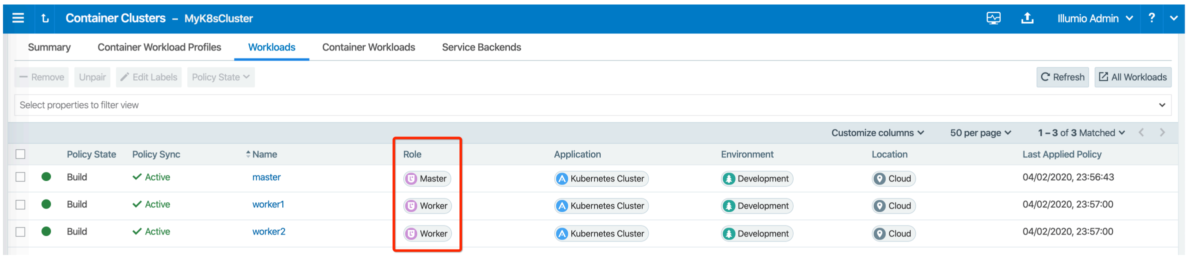
Re-Label Your Cluster Nodes

> **NOTE**
> Re-labeling the cluster nodes is optional.

In the case of self-managed deployments in which both Master and Worker nodes are managed, you may want to re-label your nodes to differentiate Master nodes from Worker nodes. Doing this helps when you are writing different policies for the Worker and Master nodes, or if you want to segment these nodes differently.

To re-label your cluster nodes:

1. In the PCE UI, go to **Infrastructure > Container Clusters >** ***YourClusterName*** **> Workloads**.
2. Select the workloads you want to re-label.
3. Click **Edit Labels** to assign the new labels (for example, Master and Worker).



4. After re-labeling your cluster nodes, the nodes part of the cluster reflect the updated label(s).

Generating YAML Manifests for Manual Deployment

In addition to the typical deployment with a Helm Chart, alternatively you can manually deploy Illumio Core for Kubernetes and OpenShift using customized YAML manifests that you have changed to suit your specific needs.

The procedure consists of the following steps, which are described in the following sections:

1. Install Helm tool.
2. Generate files.
3. Remove unpair DaemonSet and Job commands.

Install Helm Tool

There are several options for installing the Helm tool, depending on the operating system you are running. For complete details on all options, see https://helm.sh/docs/intro/install/. A few common installation commands are shown below:

```
brew install helm
```

```
sudo snap install helm --classic
```

```
export HELM_LATEST=$(curl -s https://api.github.com/repos/helm/helm/
releases/latest |
grep tag_name | cut -d '"' -f 4)
curl -LJO https://get.helm.sh/helm-$HELM_LATEST-linux-amd64.tar.gz
tar -zxvf helm-$HELM_LATEST-linux-amd64.tar.gz
mv linux-amd64/helm /usr/local/bin/helm
```

Generate Files

Prepare **`values.yaml`** in advance. The file must set at least the following minimally required parameters:

```
pce_url: URL_PORT
cluster_id: ILO_CLUSTER_UUID
cluster_token: ILO_CLUSTER_TOKEN
cluster_code: ILO_CODE
containerRuntime: RUNTIME # supported values: [containerd (default),
docker, crio, k3s_containerd]
containerManager: MANAGER # supported values: [kubernetes, openshift]
networkType: flat # CNI type, allowed values are [overlay, flat]
clusterMode: clas #
```

Generate templates and redirect output into a file, for example, into **`illumio.yaml`**:

```
helm template oci://quay.io/illumio/illumio -f values.yaml
--version <ver#> > illumio.yaml
```

IMPORTANT

Be sure to explicitly specify the version you want to install with the `--version <ver#>` option (for example, `--version 5.1.0`), after confirming that the product version you want to install is supported with your PCE version. Verify which PCE versions support the Illumio Core for Kubernetes version you want to deploy at the Kubernetes Operator OS Support and Dependencies page on the Illumio Support Portal.

Remove Unpair DaemonSet and Job Objects

In the generated YAML file **`illumio.yaml`**, search for and remove the DaemonSet and Job objects. Remove only these two objects; they are only used for the removal of Illumio product:

```
. . .
kind: Job
metadata:
name: illumio-ven-unpair-job
...
kind: DaemonSet
metadata:
```

```
name: illumio-ven-unpair
...
```

Deployment for C-VEN Versions 21.5.15 or Earlier

After you set up your clusters, make sure you perform the steps in the order provided in this section.

> **NOTE**
>
> Follow these instructions if you are deploying Illumio Core for Kubernetes (C-VEN) versions 21.5.15 or earlier.
>
> If you are deploying the Illumio Core for Kubernetes 3.0.0 release (or later), do not follow these instructions, but instead refer to Deployment with Helm Chart (Core for Kubernetes 3.0.0 and Higher) [10], which describes how to use a Helm Chart to deploy all necessary product components.
>
> The installation process is mostly the same for Kubernetes and OpenShift, except a few steps differ. A dedicated section is created for Kubernetes or OpenShift wherever required.

Host and Cluster Requirements

To deploy Illumio containers into your environment, you must meet the following requirements.

Supported Configurations for On-premises and IaaS

For full details on all supported configurations for Containerized VEN release 21.5.15 and earlier, see the C-VEN/Kubelink OS Support and Dependencies page on the Illumio Support Portal (under Software > OS Support).

Privileges

The privileges listed below should be provided on host-level and cluster-level for the respective components.

Host-Level

C-VEN

C-VEN requires the following privileges on the host:

- C-VEN is a privileged container and requires access to the following system calls:
  - `NET_ADMIN`
  - `SYS_MODULE`
  - `SYS_ADMIN`

- C-VEN requires persistent storage on the host to write iptables rules and logs.
- C-VEN mounts volumes on the local host to be able to operate (mount points may differ depending on the orchestration platform).

Optionally, you can set the Priority Class to `system-node-critical`. This option is only supported in Kubernetes 1.17 and later, in a namespace other than `kube-system`. For more details, see the Kubernetes documentation.

Kubelink

Kubelink does not require specific privileges on the host because Kubelink:

- Is not a privileged container.
- Is a stateless container.
- Does not require persistent storage.

Cluster-Level

Namespace

C-VENs and Kubelink are deployed in the `illumio-system` namespace. You can modify this namespace name according to your deployment (manifest file modification).

C-VEN

C-VEN requires the following privileges on the cluster:

- C-VEN uses the `illumio-ven` ServiceAccount.

Kubelink

Kubelink requires the following privileges on the cluster:

- Kubelink creates a new Cluster Role to list and watch events occurring on the Kubernetes API server for the following elements:
  - `nodes`
  - `hostsubnets`
  - `replicationcontrollers`
  - `services`
  - `replicasets`
  - `daemonsets`
  - `namespaces`
  - `statefulsets`
- Kubelink uses the `illumio-kubelink` ServiceAccount.

Optionally, you can set the Priority Class to `system-cluster-critical`. This option is only supported in Kubernetes 1.17 and later, in a namespace other than `kube-system`. For more details, see the Kubernetes documentation.

Prepare Your Environment

> **IMPORTANT**
> The following steps for preparing your environment are no longer needed when deploying Illumio Core for Kubernetes version 3.0.0 and beyond, which now uses Helm Chart for deploying C-VEN and Kubelink. This section is included here for backwards compatibility and historical purposes. If you are deploying using Helm Chart, skip this section and now follow the instructions in Create a Container Cluster in the PCE [38].

You need to do these steps before C-VEN installation and pairing.

> **CAUTION**
> If the prerequisite steps are not done before C-VEN and Kubelink installation, then containerized environments and Kubelink can get disrupted.

Unique Machine ID

Some of the functionality and services provided by the Illumio C-VEN and Kubelink depend on the Linux machine-id of each Kubernetes cluster node. Each machine-id must be unique in order to take advantage of the functionality. By default, the Linux operating system generates a random machine-id to give each Linux host uniqueness. However, there are cases when machine-id's can be duplicated across machines. This is common across deployments that clone machines from a golden image, for example, spinning up virtual machines from VMware templates, creating compute instances from a reference image, or from a template from a Public Cloud provider.

> **IMPORTANT**
> Illumio Core requires a unique machine-id on all nodes. This issue is more likely to occur with on-premises or IaaS deployments, rather than with Managed Kubernetes Services (from Cloud Service Providers). For more information on how to create a new unique machine-id, see Troubleshooting [95].

Create Labels

For details on creating labels, see "Labels and Label Groups" in Security Policy Guide. The labels shown below are used in examples throughout this document. You are not required to use the same labels

Name	Label Type
Kubernetes Cluster	Application
OpenShift Cluster	Application
Production	Environment
Development	Environment
Data Center	Location
Cloud	Location
Kubelink	Role
Node	Role
Control Plane Node (formerly Master)	Role
Worker	Role

Push Kubelink and C-VEN Images to Your Container Registry

In order to install Illumio Core for containers, you first need to upload (or push) Kubelink and C-VEN container images to your container registry. The files in the C-VEN and Kubelink packages you've downloaded are as follows:

C-VEN `illumio-ven-xx.x.x-xxxx.k8s.x86_64.tgz` package includes:

- A Docker image
  - `illumio-ven-xx.x.x-xxxx.tgz`
- Configuration files:
  - `illumio-ven-secret.yml`
  - `illumio-ven-kubernetes.yml`
  - `illumio-ven-openshift.yml`

Kubelink `illumio-kubelink-x.x.x.tar.gz` package includes:

- A docker image
  - `kubelink-image.tar.gz`
- Configuration files in kube-yaml
  - `illumio-kubelink-secret.yml`
  - `illumio-kubelink-kubernetes.yml`
  - `illumio-kubelink-openshift.yml`
  - `illumio-kubelink-namespace.yml`

CAUTION

These images are not publicly available and should **not** be posted on a publicly open container registry without Illumio's consent.

In a self-managed deployment, Kubelink and C-VEN images can be pushed to a private container registry. In OpenShift, a container registry is provided as part of the platform, and images can be pushed to this registry for simplicity and better authentication. In the case of Kubernetes, there is no container registry provided by default and must be provided as an external component.

In a cloud-managed deployment, Cloud Service Providers (CSPs) provide integration of private container registries such as, Amazon ECR, Microsoft ACR, and so on. These registries can securely be used to host Illumio's container images for Kubelink and C-VEN. Refer to the documentation provided by the respective CSPs to learn how to push images to those registries.

To push Kubelink and C-VEN container images to your private container registry, use the following commands (based on docker):

1. Log in to your private container registry.

   ```
   docker login <docker-registry>
   ```
2. Load Kubelink and C-VEN container images on your local computer.

   ```
   docker load -i kubelink-image.tar.gz
   docker load -i illumio-ven-21.5.x-xxxx.tgz
   ```

   Verify that docker images are loaded on your computer.

   ```
   docker image ls
   ```
3. Tag the Kubelink and C-VEN container image IDs with the name of your container registry.

   ```
   docker tag <illumio-kubelink-image-id> <docker-registry>/
   illumio-kubelink:2.1.x.xxxxxx
   docker tag <illumio-ven-image-id> <docker-registry>/illumio-ven:21.5.x-
   xxxx
   ```

   Verify that images are tagged on your computer and ready to be pushed to your private container registry.

   ```
   docker image ls
   ```
4. Push Kubelink and C-VEN container images on your private container registry.

   ```
   docker push <docker-registry>/illumio-kubelink:2.1.x.xxxxxx
   docker push <docker-registry>/illumio-ven:21.5.x-xxxx
   ```

After pushing images to your private container registry, proceed to the next section.

Create Illumio Namespace

Create Illumio Namespace

Illumio Core for containers is deployed in a dedicated namespace `illumio-system`, by default. This namespace has the minimum privileges in the cluster required to run Illumio Core and can tie into the Kubernetes and OpenShift RBAC models.

To create the `illumio-system` namespace for Kubernetes, use the following command:

```
kubectl create namespace illumio-system
```

> **NOTE**
> Illumio provides a yaml manifest file to create the namespace in the Kubelink tarball `illumio-kubelink-namespace.yml`. You can create this namespace by applying this manifest file to your Kubernetes cluster, using the following command:
>
> ```
> kubectl apply -f illumio-kubelink-namespace.yml
> ```

To create the `illumio-system` project for OpenShift, use the following command:

```
oc new-project illumio-system
```

Authenticate Kubernetes Cluster with Container Registry

> **NOTE**
> Depending on your deployment, the steps in the Authenticate Kubernetes Cluster with Container Registry [33], Create a ConfigMap to Store Your Root CA Certificate [34], and Configure Calico in Append Mode [37] topics are optional.

When storing container images in a private container registry, it is often required and strongly recommended to authenticate against the registry to be able to pull an image from it. In order to do this, the Kubernetes or OpenShift cluster must have the credentials configured and stored in a secret file to be able to pull container images.

To configure a secret to store your container registry credentials, use the following command:

```
kubectl create secret docker-registry <container-registry-secret-name>
-n illumio-system --docker-server=<container-registry>
--docker-username=<username> --docker-password=<password>
```

To verify that the secret has been created, use the following command:

```
kubectl get secret -n illumio-system | grep <container-registry-secret-name>
```

> **IMPORTANT**
> The above commands are valid for deployments with your own private container registry, but may not be valid for a cloud-managed private container registry. For more information, refer to your Cloud Service Provider documentation.

Create a ConfigMap to Store Your Root CA Certificate

This section describes how to implement Kubelink with a PCE using a certificate signed by a private PKI. It describes how to configure Kubelink and C-VEN to accept the certificate from the PCE signed by a private root or intermediate Certificate Authority (CA) and ensure that Kubelink can communicate in a secure way with the PCE.

Prerequisites

- Access to the root CA to download the root CA certificate
- Access to your Kubernetes cluster and can run `kubectl` commands
- Correct privileges in your Kubernetes cluster to create resources like a ConfigMaps, secrets, and Pods
- Access to the PCE web console as a Global Organization Owner

Download the Root CA Certificate

Before you begin, ensure that you have access to the root CA certificate. The root CA certificate is a file that can be exported from the root CA without compromising the security of the company. It is usually made available to external entities to ensure a proper SSL handshake between a server and its clients.

You can download the root CA certificate in the CRT format on your local machine. Below is an example of a root CA certificate:

```
$ cat root.democa.illumio-demo.com.crt
-----BEGIN CERTIFICATE-----
MIIGSzCCBDOgAwIBAgIUAPw0NfPAivJW4YmKZ499eHZH3S8wDQYJKoZIhvcNAQEL
---output suppressed---
wPG0lug46K1EPQqMA7YshmrwOd6ESy6RGNFFZdhk9Q==
-----END CERTIFICATE-----
```

You can also get the content of your root CA certificate in a readable output format by using the following command:

```
$ openssl x509 -text -noout -in ./root.democa.illumio-demo.com.crt
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            fc:34:35:f3:c0:8a:f2:56:e1:89:8a:67:8f:7d:78:76:47:dd:2f
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=US, ST=California, L=Sunnyvale, O=Illumio,
        OU=Technical Marketing, CN=Illumio Demo Root CA
        1/emailAddress=tme-team@illumio.com
        Validity
            Not Before: Jan 20 00:05:36 2020 GMT
            Not After : Jan 17 00:05:36 2030 GMT
        Subject: C=US, ST=California, L=Sunnyvale, O=Illumio,
        OU=Technical Marketing, CN=Illumio Demo Root CA
        1/emailAddress=tme-team@illumio.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (4096 bit)
                Modulus:
```

```
                    00:c0:e5:48:7d:97:f8:5b:8c:ef:ac:16:a8:8c:aa:
                    68:b8:48:af:28:cd:17:8f:02:c8:82:e9:69:62:e2:
                    89:2b:be:bd:34:fc:e3:4d:3f:86:5e:d7:e6:89:34:
                    71:60:e6:54:61:ac:0f:26:1c:99:6f:80:89:3f:36:
                    b3:ad:78:d1:6c:3f:d7:23:1e:ea:51:14:48:74:c3:
                    e8:6e:a2:79:b1:60:4c:65:14:2a:f1:a0:97:6c:97:
                    50:43:67:07:b7:51:5d:2c:12:49:81:dc:01:c9:d1:
                    57:48:32:2e:87:a8:d2:c0:b9:f8:43:b2:58:10:af:
                    54:59:09:05:cb:3e:f0:d7:ef:70:cc:fc:53:48:ee:
                    a4:a4:61:f1:d7:5b:7c:a9:a8:92:dc:77:74:f4:4a:
                    c0:4a:90:71:0f:6d:9e:e7:4f:11:ab:a5:3d:cd:4b:
                    8b:79:fe:82:1b:16:27:94:8e:35:37:db:dd:b8:fe:
                    fa:6d:d9:be:57:f3:ca:f3:56:aa:be:c8:57:a1:a8:
                    c9:83:dd:5a:96:5a:6b:32:2d:5e:ae:da:fc:85:76:
                    bb:77:d5:c2:53:f3:5b:61:74:e7:f3:3e:4e:ad:10:
                    7d:4f:ff:90:69:7c:1c:41:2f:67:e4:13:5b:e6:3a:
                    a3:2f:93:61:3b:07:56:59:5a:d9:bc:34:4d:b3:54:
                    b5:c6:e5:0a:88:e9:62:7b:4b:85:d2:9e:4c:ee:0b:
                    0d:f4:72:b1:1b:44:04:93:cf:cc:bb:18:31:3a:d4:
                    83:4a:ff:15:42:2d:91:ca:d0:cb:36:d9:8d:62:c0:
                    41:59:1a:93:c7:27:79:08:94:b2:a2:50:3c:57:27:
                    33:af:f0:b6:92:44:49:c5:09:15:a7:43:2a:0f:a9:
                    02:61:b3:66:4f:c3:de:d3:63:1e:08:b1:23:ea:69:
                    90:db:e8:e9:1e:21:84:e0:56:e1:8e:a1:fa:3f:7a:
                    08:0f:54:0a:82:41:08:6b:6e:bb:cf:d6:5b:80:c6:
                    ea:0c:80:92:96:ab:95:5d:38:6d:4d:da:38:6b:42:
                    ef:7c:88:58:83:88:6d:da:28:62:62:1f:e5:a7:0d:
                    04:9f:0d:d9:52:39:46:ba:56:7c:1d:77:38:26:7c:
                    86:69:58:4d:b0:47:3a:e2:be:ee:1a:fc:4c:de:67:
                    f3:d5:fe:e6:27:a2:ef:26:86:19:5b:05:85:9c:4c:
                    02:24:76:58:42:1a:f8:e0:e0:ed:78:f2:8f:c8:5a:
                    20:a9:2d:0b:d4:01:fa:57:d4:6f:1c:0a:31:30:8c:
                    32:7f:b0:01:1e:fe:94:96:03:ee:01:d7:f4:4a:83:
                    f5:06:fa:60:43:15:05:9a:ca:88:59:5c:f5:13:09:
                    82:69:7f
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                3D:3D:3D:61:E6:88:09:FE:34:0F:1D:5E:5E:52:72:71:C7:DE:15:92
            X509v3 Authority Key Identifier:
                keyid:3D:3D:3D:61:E6:88:09:FE:34:0F:1D:5E:5E:52:72:71:C7:DE:
                15:92

            X509v3 Basic Constraints: critical
                CA:TRUE
            X509v3 Key Usage: critical
                Digital Signature, Certificate Sign, CRL Sign
    Signature Algorithm: sha256WithRSAEncryption
         28:24:86:91:a6:4a:88:e4:8d:6b:fc:67:2a:68:08:67:35:e5:
         a6:77:ff:07:4b:89:53:99:2e:6d:95:df:12:81:28:6a:8e:6f:
         5a:98:95:5b:4a:21:ae:f0:20:a4:4e:06:b2:4e:5a:67:c1:6a:
         06:f1:0f:c1:f7:7e:f2:e0:b3:9d:d8:54:26:6a:b2:1c:19:b8:
         b5:5c:c7:03:6b:f7:70:9e:72:85:c9:29:55:f9:f4:a4:f2:b4:
         3b:3d:ce:25:96:67:32:1e:8d:e2:00:22:55:4b:05:4f:ee:0e:
         67:ac:db:1b:61:da:5f:9c:10:1c:0c:05:66:c0:5b:5f:b9:95:
```

```
         59:a9:58:5b:e7:69:ac:b0:bd:b3:c2:a3:35:58:01:a4:ff:c0:
         8d:ac:1c:19:21:41:50:fb:8e:e0:f5:a9:ad:ec:de:cb:53:04:
         a9:d8:ac:76:8a:09:0d:7c:c6:1a:bc:06:74:bb:10:1c:aa:07:
         f6:cb:b2:1b:0c:0c:65:03:45:2b:51:d5:6e:a0:4d:91:ce:c5:
         ed:8d:a9:e7:f6:37:7d:ab:1b:a4:a2:a3:3b:76:17:5b:d9:3a:
         9c:c1:df:cc:cd:a0:b0:a9:5c:74:61:d7:a0:1d:04:67:68:ee:
         a6:7b:1e:41:a4:02:fc:65:9e:e3:c1:c2:57:b2:2e:b0:ff:a9:
         86:82:35:4d:29:b2:fe:74:2e:b8:37:5d:2b:e8:69:f2:80:29:
         19:f1:1e:7a:5d:e3:d2:51:50:46:30:54:7e:b8:ad:59:61:24:
         45:a8:5a:fe:19:ff:09:31:d0:50:8b:e2:15:c0:a2:f1:20:95:
         63:55:18:a7:a2:ad:16:25:c7:a3:d1:f2:e5:be:6d:c0:50:4b:
         15:ac:e0:10:5e:f3:7b:90:9c:75:1a:6b:e3:fb:39:88:e4:e6:
         9f:4c:85:60:67:e8:7d:2e:85:3d:87:ed:06:1d:13:0b:76:d7:
         97:a5:b8:05:76:67:d6:41:06:c5:c0:7a:bd:f4:c6:5b:b2:fd:
         23:6f:1f:57:2e:df:95:3f:26:a5:13:4d:6d:96:12:56:98:db:
         2e:7d:fd:56:f5:71:b7:19:2b:c9:de:2d:b9:c8:17:cc:20:de:
         7c:19:7a:aa:12:97:1c:80:b7:d3:67:d3:b7:a7:96:f0:c9:4d:
         f5:8b:0e:10:3b:b9:4e:09:90:5a:3b:51:c9:48:a2:ca:9f:db:
         72:44:87:59:db:49:fa:75:44:b5:f6:7f:c5:26:e1:01:ae:7b:
         6f:4a:75:d1:b5:b3:68:c0:31:48:f8:5c:06:c0:f1:b4:96:e8:
         38:e8:ad:44:3d:0a:8c:03:b6:2c:86:6a:f0:39:de:84:4b:2e:
         91:18:d1:45:65:d8:64:f5
```

Create a ConfigMap in Kubernetes Cluster

After downloading the certificate locally on your machine, create a ConfigMap in the Kubernetes cluster that will copy the root CA certificate on your local machine into the Kubernetes cluster.

To create the ConfigMap, use the following command:

```
$ kubectl -n illumio-system create configmap root-ca-config \
    --from-file=./certs/root.democa.illumio-demo.com.crt
```

The `--from-file` option points to the path where the root CA certificate is stored on your local machine.

To verify that the ConfigMap was created correctly, use the following command:

```
$ kubectl -n illumio-system create configmap root-ca-config \
> --from-file=./certs/root.democa.illumio-demo.com.crt
configmap/root-ca-config created
$
$ kubectl -n illumio-system get configmap
NAME                                 DATA   AGE
root-ca-config                       1      12s
$
$ kubectl -n illumio-system describe configmap root-ca-config
Name:         root-ca-config
Namespace:    illumio-system
Labels:       <none>
Annotations:  <none>

Data
====
```

```
root.democa.illumio-demo.com.crt:
----
-----BEGIN CERTIFICATE-----
MIIGSzCCBDOgAwIBAgIUAPw0NfPAivJW4YmKZ499eHZH3S8wDQYJKoZIhvcNAQEL
---output suppressed---
wPG0lug46K1EPQqMA7Yshmrw0d6ESy6RGNFFZdhk9Q==
-----END CERTIFICATE-----

Events:  <none>
$
```

`root-ca-config` is the name used to designate the ConfigMap. You can modify it according to your naming convention.

Configure Calico in Append Mode

In case your cluster is configured with Calico as the network plugin (usually for Kubernetes and not for OpenShift), both Calico and Illumio Core will write iptables rules on the cluster nodes.

- Calico - Needs to write iptables rules to instruct the host how to forward packets (overlay, IPIP, NAT, and so on).
- Illumio Core - Needs to write iptables rules to secure communications between nodes and/or Pods.

You should establish a hierarchy to make the firewall coexistence work smoothly because Illumio Core and Calico will write rules at the same time. By default, both solutions are configured to insert rules first in the iptables chains/tables and Illumio Core will remove other rules added by a third-party software (in the Exclusive mode).

To allow Calico to write rules along with Illumio without flushing rules from one another, you should:

- Configure Illumio to work in Firewall Coexistence mode (default for workloads that are part of a container cluster).
- Configure Calico to work in Append mode (default is Insert mode).

To configure Calico to work in Append mode with iptables:

1. Edit the Calico DaemonSet.

   ```
   kubectl -n kube-system edit ds calico-node
   ```
2. Locate the `spec: > template: > spec: > containers:` section inside the `YAML` file and change ChainInsertMode by adding the following code block:

   ```
   - name: FELIX_CHAININSERTMODE
     value: Append
   ```
3. Save your changes and exit.
4. Kubernetes will restart all Calico Pods in a rolling update.

For more information on changing Calico ChainInsertMode, see Calico documentation.

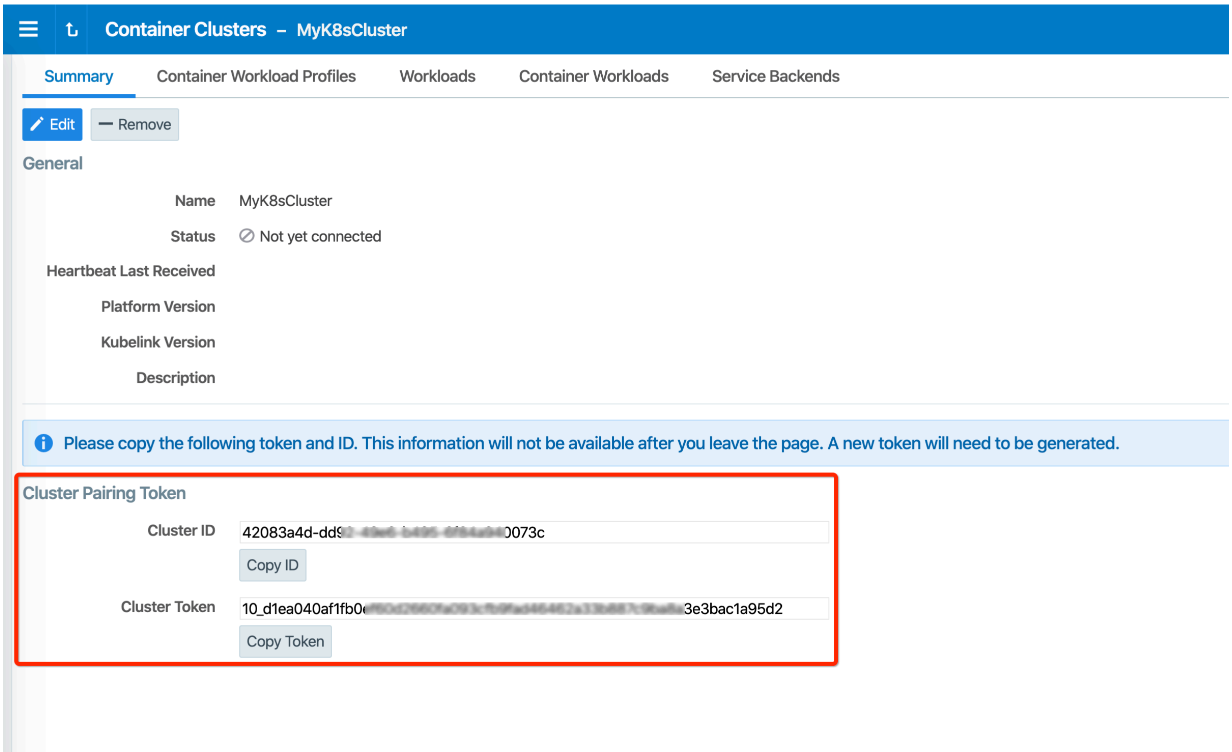
Create a Container Cluster in the PCE

To provide visibility and enforcement to your containerized environment, you first need to create a container cluster in the PCE. Each container cluster maps to an existing Kubernetes or OpenShift cluster.

Create a Container Cluster

To create a new container cluster:

1. Log into the PCE web console as a user with Global Organization Owner privileges.
2. From the PCE web console menu, navigate to **Infrastructure** > **Container Clusters**.
3. Click **Add**.
   a. Add a Name.
   b. **Save** the Container Cluster.
4. You will see a summary page of the new Container Cluster. From the Cluster Pairing Token section, copy the values of the Cluster ID and Cluster Token.
5. After copying and saving the values (in a text editor or similar tool), open the Container Workload Profiles page.



Configure a Container Workload Profile Template

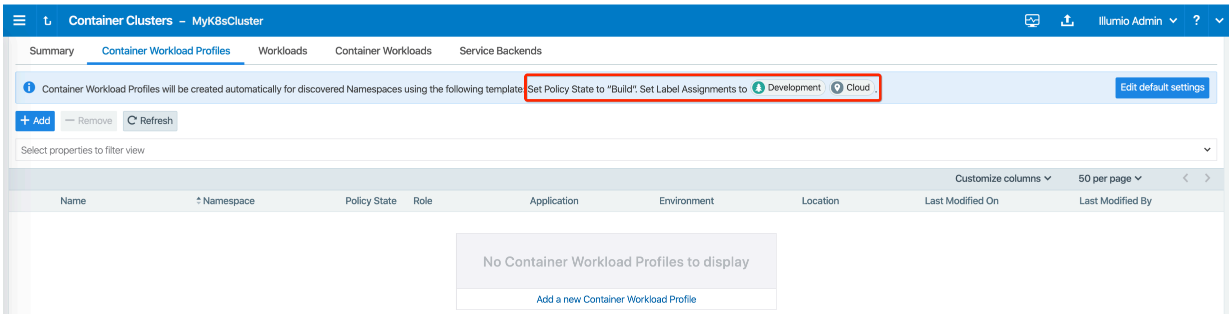
When configuring a new Container Cluster, it is recommended to set the default settings shared by all the Container Workload Profiles. Illumio provides a Container Workload Profile template that can be used for that purpose. By defining the default Policy State and minimum set of labels common to all namespaces in the cluster, you will save time later on when new namespaces are discovered by Kubelink. Each new profile created will inherit what was defined in the template.

> **IMPORTANT**
> Illumio does not provide a method to redefine at once all the labels associated with each profile. Hence, it is **strongly recommended** to use the provided template to define the default values for all profiles that are part of the same cluster.

To define the default parameters for all profiles using a template, under *Container Workload Profiles*, click **Edit** default settings and select values for all the fields.

For information about assigning default labels in the template, see the "Labels Restrictions for Kubernetes Namespaces" topic.

After you click OK, the following information is displayed:



Deploy C-VENs in Your Cluster

> **IMPORTANT**
> Before deploying the C-VEN, ensure that either of the following two requirements has been met:
>
> - Kubelink is deployed on the Kubernetes cluster and is in sync with the PCE, or
> - Firewall coexistence is enabled.

Prerequisites

- VEN deployment file provided by Illumio.
- VEN secret file provided by Illumio.
- Illumio's C-VEN docker image uploaded to a private container registry.
- In OpenShift, create the 'illumio-ven' service account in the 'illumio-system' project and add this account to the privileged Security Context Constraint (SCC):
  - `oc create sa illumio-ven`

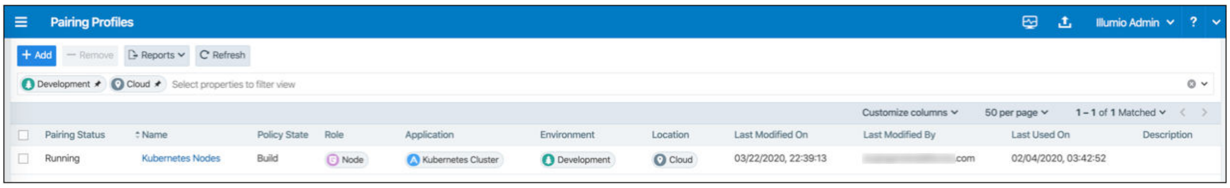
- `oc adm policy add-scc-to-user privileged -z illumio-ven -n illumio-system`

Create a Pairing Profile for Your Cluster Nodes

Before deploying the C-VEN in your cluster, you should create a pairing profile to pair the cluster nodes with the PCE. You only need to create one pairing profile for all your nodes.

> **NOTE**
> You only need to create pairing profiles for Kubernetes or OpenShift nodes and not for container workloads.

For ease of configuration and management, consider applying the same Application, Environment, and Location labels across all nodes of the same Kubernetes or OpenShift cluster. The screenshot below shows an example of a pairing profile for a Kubernetes cluster.

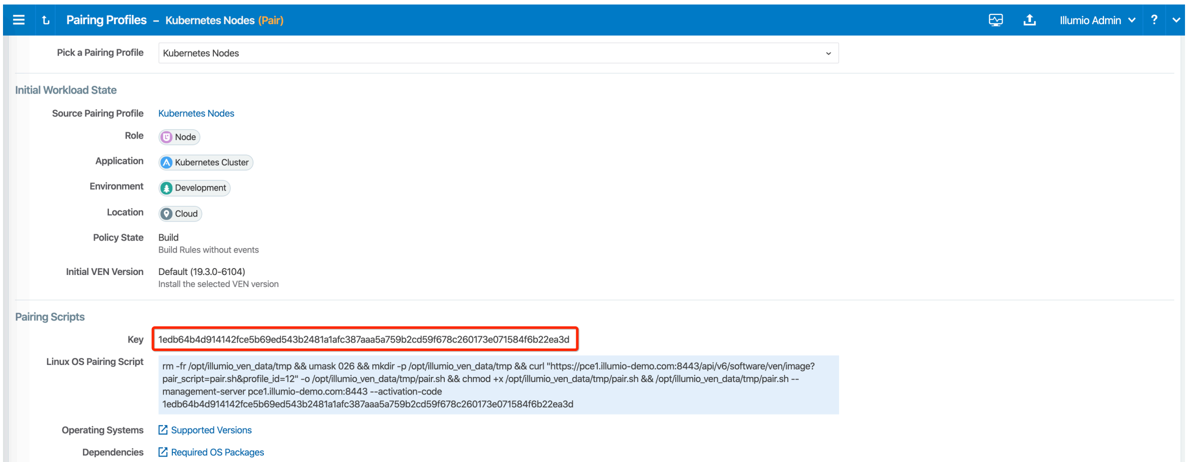


> **TIP**
> Illumio recommends all pairing profiles for Kubernetes nodes to *not* use the Full enforcement policy state. Use Idle or Visibility Only mode for initial configuration.
>
> You should only move them into Full enforcement state after you have completed all other configuration steps in this guide, such as setup Kubelink, discover services, and write rules.

Configure C-VEN Secret

This section assumes that you have already created a Pairing Profile in the PCE. You will need the activation code for the C-VEN secret.

1. To retrieve the activation code from the pairing profile, go to **Policy Objects > Pairing Profiles**, open the pairing profile created for your cluster nodes, and click **Generate Key**.



2. After copying and saving the **Key** (in a text editor or similar tool), you can exit the page.
3. Open the C-VEN secret YAML file and modify the following keys (under `stringData`):
   - `ilo_server` = PCE URL and port. Example: mypce.example.com:8443
   - `ilo_code` = Activation code value from Step 1. Example: 1edb64b4d914142fce5b69ed543b2481a1afc387aaa5a759b2cd59f678c260173e071584f6b22ea3d

   Contents of a modified `illumio-ven-secret.yml` file are shown below.

```
#
# Copyright 2013-2021 Illumio, Inc. All Rights Reserved.
#
# VEN 21.5.x-xxxx

apiVersion: v1
kind: Secret
metadata:
  name: illumio-ven-config
  namespace: illumio-system
type: Opaque
stringData:
  ilo_server: mypce.example.com:8443 # Example: mypce.example.com:8443
  ilo_code:
1edb64b4d914142fce5b69ed543b2481a1afc387aaa5a759b2cd59f678c260173e071584f
6b22ea3d # activation-code
```

> ⚠ **CAUTION**
>
> Do not use 'https://' for the value associated with the `ilo_server:` key. This is a known issue and will be fixed in a future release.

4. Save the changes.
5. Create the C-VEN secret using the file.
   - To create the secret for Kubernetes:

```
kubectl apply -f illumio-ven-secret.yml
```

   - To create the secret for OpenShift:

```
oc apply -f illumio-ven-secret.yml
```

6. Verify the C-VEN secret creation in your cluster.
   - To verify the creation of the secret for Kubernetes:

```
kubectl get secret -n illumio-system
```

```
oc get secret -n illumio-system
```

Deploy C-VENs

Modify the C-VEN configuration file to point to the correct Docker image. The example in this document has `illumio-ven:21.5.x-xxxx` uploaded to `registry.example.com:443`, so the image link in this example is: `registry.example.com:443/illumio-ven:21.5.x-xxxx`

1. Edit the C-VEN configuration YAML file. The file name is `illumio-ven-kubernetes.yml` for a Kubernetes cluster and `illumio-ven-openshift.yml` for an OpenShift cluster.
   - Locate the `spec: > template: > spec: > containers:` section inside the YAML file. Modify the image link in the `image:` attribute.
2. Save the changes.

   Below is a snippet from an example of the C-VEN configuration for Kubernetes or OpenShift to illustrate the image location.

```
#
# Copyright 2013-2021 Illumio, Inc. All Rights Reserved.
#
# VEN 21.5.x-xxxx

---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: illumio-ven
  namespace: illumio-system
---
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: illumio-ven
  namespace: illumio-system
  labels:
    k8s-app: illumio-ven
spec:
  selector:
    matchLabels:
      name: illumio-ven
  template:
    metadata:
      labels:
        name: illumio-ven
    spec:
      priorityClassName: system-node-critical
      serviceAccountName: illumio-ven
      hostNetwork: true
      hostPID: true
      tolerations:
      - key: node-role.kubernetes.io/master
        effect: NoSchedule
      containers:
        - name: illumio-ven
          env:
```

```
            - name: ILO_SERVER
              valueFrom:
                secretKeyRef:
                  name: illumio-ven-config
                  key: ilo_server
            - name: ILO_CODE
              valueFrom:
                secretKeyRef:
                  name: illumio-ven-config
                  key: ilo_code
          command: [ "/ven-init", "activate" ]
          image: registry.example.com/illumio-ven:21.5.x-xxxx
          imagePullPolicy: IfNotPresent
          <...>
```

3. (Optional) Reference your root CA certificate.

   If you are using a private PKI to sign your PCE certificate, make sure you add the references to the root CA certificate that signed the PCE certificate. If Kubelink is already deployed in the cluster, the ConfigMap used to store the root CA certificate should already be created in the cluster.

   Add the following sections to the C-VEN manifest file to reference the ConfigMap containing the root CA certificate:

   - `volumeMounts` (under `spec.template.spec.containers`)
   - `volumes` (under `spec.template.spec`)

   `root-ca` is the name used to designate the new volume mounted in the container. You can modify it according to your naming convention.

```
        volumeMounts:
        - name: root-ca
          mountPath: /etc/pki/tls/ilo_certs/
          readOnly: false
      volumes:
      - name: root-ca
        configMap:
          name: root-ca-config
```

4. (Optional) Reference your container registry secret. See the Authenticate Kubernetes Cluster with Container Registry [33] section.

   In case you need to authenticate against your container registry when you pull an image from your cluster, you must make reference to the secret previously created for the container registry. Locate the `spec: > template: > spec:` section inside the YAML file and add the following lines:

```
      imagePullSecrets:
      - name: <container-registry-secret-name>
```

IMPORTANT

Indentation matters in a YAML file. Make sure there are 6 spaces to the left before inserting the 'imagePullSecrets' keyword and align the '-' character below it with the 'i' of the 'imagePullSecrets' keyword.

5. **NOTE**

   From the 20.2.0 and later releases, the container runtime detection is done automatically. You do not need to manually modify the container runtime socket path. You should do this 'Modify the container runtime socket path' step only if you are using a customized configuration for your container runtime.

   (Optional) Modify the container runtime socket path.

   In some cases, you have to modify the default socket path the C-VEN relies on to get information about the containers due to the following reasons:

   - A non-conventional or customized container runtime socket path
   - Two concurrent container runtimes

   In this case, you may have to modify the default mount path for the `unixsocks` volume in the C-VEN configuration file.

   For example, you want to listen on the 'containerd' container runtime, however, docker is also used on the nodes. You should modify the file as shown below, so that the C-VEN listens to events on 'containerd':

```
          volumeMounts:
          - name: unixsocks
            mountPath: /var/run/containerd/
            <...>
      volumes:
      - name: unixsocks
        hostPath:
          path: /var/run/containerd/
          type: Directory
          <...>
```

6. Save the changes.
7. Deploy C-VEN.
   - For Kubernetes:

```
kubectl apply -f illumio-ven-kubernetes.yml
```

   - For OpenShift:

```
oc apply -f illumio-ven-openshift.yml
```

8. Verify the deployment.
   - For Kubernetes:

```
kubectl get pods -n illumio-system
```

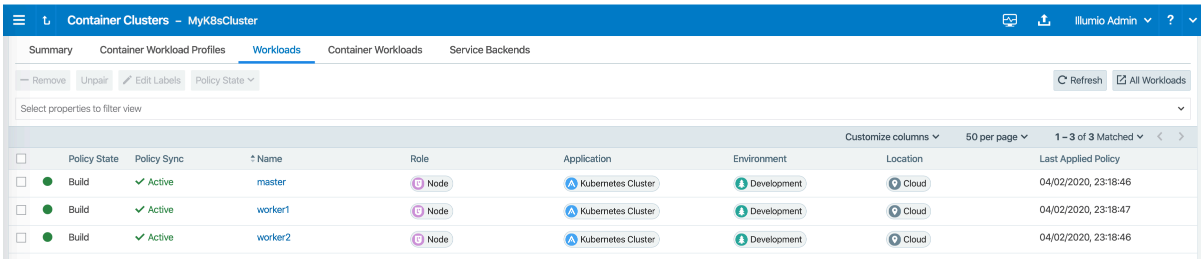
```
oc get pods -n illumio-system
```

The `illumio-ven-xxxxxxxxxx-xxxxx` Pods should be in the "Running" state.

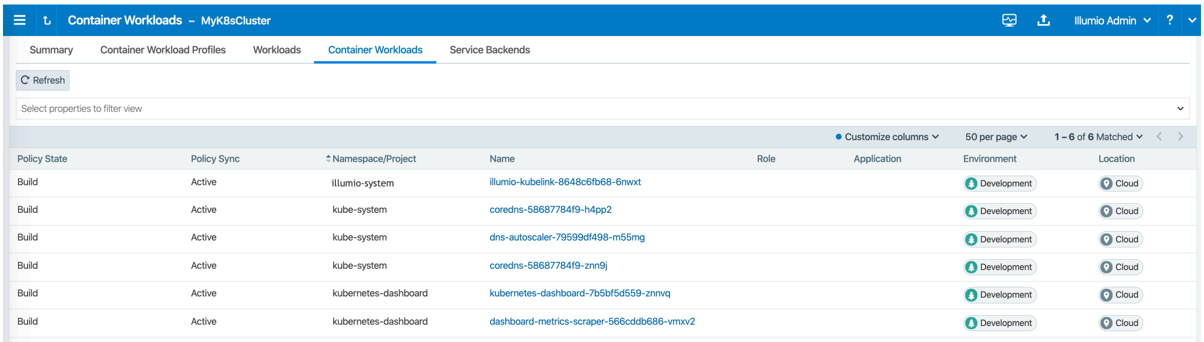
After C-VENs are successfully deployed, you can check the cluster information in the Illumio PCE UI. From the main menu, navigate to **Infrastructure > Container Clusters**.

You can also verify in the PCE UI that the C-VENs were successfully deployed by checking the following:

- Under the **Workload** tab, nodes that are part of your Kubernetes or OpenShift cluster should be listed. An example is shown below.



- Under the **Container Workloads** tab, Pods deployed in your Kubernetes or OpenShift cluster should be listed. An example is shown below.



- Illumination Map now displays system and application Pods running in your cluster.

Deploy C-VENs in Your Cluster

> **IMPORTANT**
>
> Before deploying the C-VEN, ensure that either of the following two requirements has been met:
>
> - Kubelink is deployed on the Kubernetes cluster and is in sync with the PCE, or
> - Firewall coexistence is enabled.

Prerequisites

- VEN deployment file provided by Illumio.
- VEN secret file provided by Illumio.
- Illumio's C-VEN docker image uploaded to a private container registry.
- In OpenShift, create the 'illumio-ven' service account in the 'illumio-system' project and add this account to the privileged Security Context Constraint (SCC):
  - `oc create sa illumio-ven`
  - `oc adm policy add-scc-to-user privileged -z illumio-ven -n illumio-system`

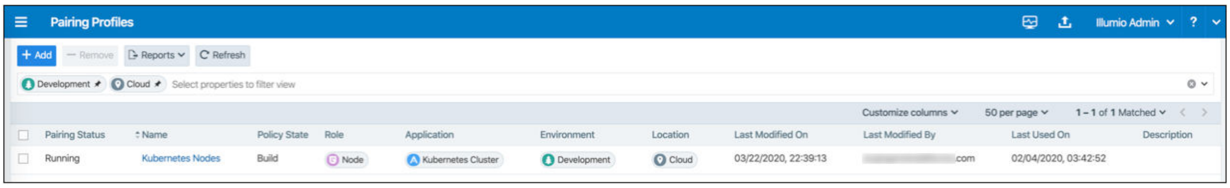
Create a Pairing Profile for Your Cluster Nodes

Before deploying the C-VEN in your cluster, you should create a pairing profile to pair the cluster nodes with the PCE. You only need to create one pairing profile for all your nodes.

NOTE

You only need to create pairing profiles for Kubernetes or OpenShift nodes and not for container workloads.

For ease of configuration and management, consider applying the same Application, Environment, and Location labels across all nodes of the same Kubernetes or OpenShift cluster. The screenshot below shows an example of a pairing profile for a Kubernetes cluster.



TIP

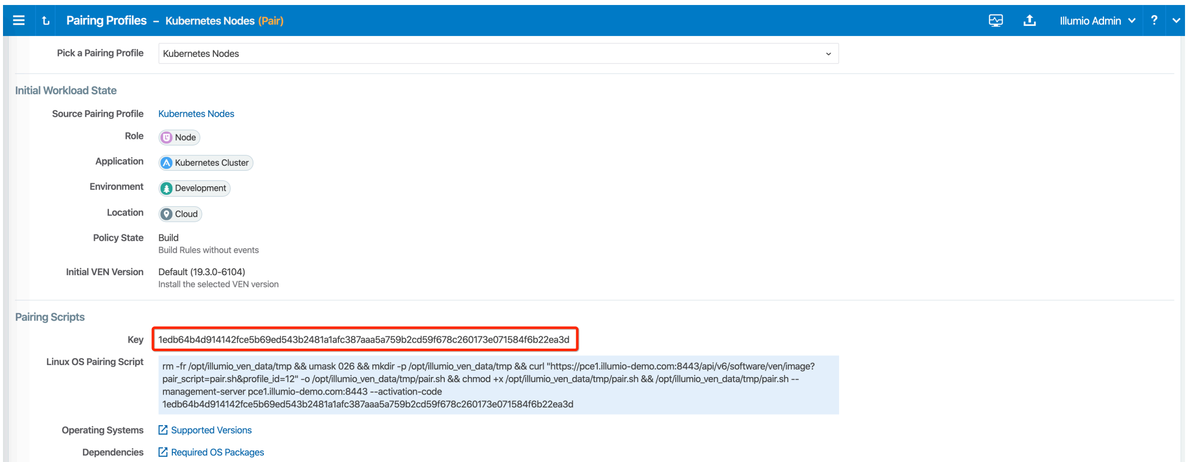
Illumio recommends all pairing profiles for Kubernetes nodes *not* to use the Full enforcement policy state. Use Visibility Only mode for initial configuration.

You should only move them into Full enforcement state after you have completed all other configuration steps in this guide, such as setting up Kubelink, discovering services, and writing rules.

Configure C-VEN Secret

This section assumes that you have already created a Pairing Profile in the PCE. You will need the activation code for the C-VEN secret.

1. To retrieve the activation code from the pairing profile, go to **Policy Objects > Pairing Profiles**, open the pairing profile created for your cluster nodes, and click **Generate Key**.



2. After copying and saving the **Key** (in a text editor or similar tool), you can exit the page.

3. Open the C-VEN secret YAML file and modify the following keys (under `stringData`):
   - `ilo_server` = PCE URL and port. Example: mypce.example.com:8443
   - `ilo_code` = Activation code value from Step 1. Example: 1edb64b4d914142fce5b69ed543b2481a1afc387aaa5a759b2cd59f678c260173e071584f6b22ea3d

   Contents of a modified `illumio-ven-secret.yml` file are shown below.

```
#
# Copyright 2013-2021 Illumio, Inc. All Rights Reserved.
#
# VEN 21.5.x-xxxx

apiVersion: v1
kind: Secret
metadata:
  name: illumio-ven-config
  namespace: illumio-system
type: Opaque
stringData:
  ilo_server: mypce.example.com:8443 # Example: mypce.example.com:8443
  ilo_code: 1edb64b4d914142fce5b69ed543b2481a1afc387aaa5a759b2cd59f678c
  260173e071584f6b22ea3d # activation-code
```

> **CAUTION**
> Do not use 'https://' for the value associated with the `ilo_server:` key. This is a known issue and will be fixed in a future release.

4. Save the changes.
5. Create the C-VEN secret using the file.
   - To create the secret for Kubernetes:

```
kubectl apply -f illumio-ven-secret.yml
```

   - To create the secret for OpenShift:

```
oc apply -f illumio-ven-secret.yml
```

6. Verify the C-VEN secret creation in your cluster.
   - To verify the creation of the secret for Kubernetes:

```
kubectl get secret -n illumio-system
```

```
oc get secret -n illumio-system
```

Deploy C-VENs

Modify the C-VEN configuration file to point to the correct Docker image. The example in this document has `illumio-ven:21.5.x-xxxx` uploaded to `registry.example.com:443`, so the image link in this example is: `registry.example.com:443/illumio-ven:21.5.x-xxxx`

1. Edit the C-VEN configuration YAML file. The file name is `illumio-ven-kubernetes.yml` for a Kubernetes cluster and `illumio-ven-openshift.yml` for an OpenShift cluster.
   - Locate the `spec: > template: > spec: > containers:` section inside the YAML file. Modify the image link in the `image:` attribute.
2. Save the changes.

Below is a snippet from an example of the C-VEN configuration for Kubernetes or Open-Shift to illustrate the image location.

```
#
# Copyright 2013-2021 Illumio, Inc. All Rights Reserved.
#
# VEN 21.5.x-xxxx

---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: illumio-ven
  namespace: illumio-system
---
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: illumio-ven
  namespace: illumio-system
  labels:
    k8s-app: illumio-ven
spec:
  selector:
    matchLabels:
      name: illumio-ven
  template:
    metadata:
      labels:
        name: illumio-ven
    spec:
      priorityClassName: system-node-critical
      serviceAccountName: illumio-ven
      hostNetwork: true
      hostPID: true
      tolerations:
      - key: node-role.kubernetes.io/master
        effect: NoSchedule
      containers:
        - name: illumio-ven
          env:
            - name: ILO_SERVER
              valueFrom:
                secretKeyRef:
                  name: illumio-ven-config
                  key: ilo_server
            - name: ILO_CODE
              valueFrom:
                secretKeyRef:
                  name: illumio-ven-config
                  key: ilo_code
          command: [ "/ven-init", "activate" ]
          image: registry.example.com/illumio-ven:21.5.x-xxxx
          imagePullPolicy: IfNotPresent
          <...>
```

3. (Optional) Reference your root CA certificate.

If you are using a private PKI to sign your PCE certificate, make sure you add the references to the root CA certificate that signed the PCE certificate. If Kubelink is already deployed in the cluster, the ConfigMap used to store the root CA certificate should already be created in the cluster.

Add the following sections to the C-VEN manifest file to reference the ConfigMap containing the root CA certificate:

- `volumeMounts` (under `spec.template.spec.containers`)
- `volumes` (under `spec.template.spec`)

`root-ca` is the name used to designate the new volume mounted in the container. You can modify it according to your naming convention.

```
        volumeMounts:
        - name: root-ca
          mountPath: /etc/pki/tls/ilo_certs/
          readOnly: false
      volumes:
      - name: root-ca
        configMap:
          name: root-ca-config
```

4. (Optional) Reference your container registry secret. See the "Authenticate Kubernetes Cluster with Container Registry" topic.

In case you need to authenticate against your container registry when you pull an image from your cluster, you must make reference to the secret previously created for the container registry. Locate the `spec: > template: > spec:` section inside the YAML file and add the following lines:

```
      imagePullSecrets:
      - name: <container-registry-secret-name>
```

IMPORTANT

Indentation matters in a YAML file. Make sure there are 6 spaces to the left before inserting the 'imagePullSecrets' keyword and align the '-' character below it with the 'i' of the 'imagePullSecrets' keyword.

NOTE

From the 20.2.0 and later releases, the container runtime detection is done automatically. You do not need to manually modify the container runtime socket path. You should do this 'Modify the container runtime socket path' step only if you are using a customized configuration for your container runtime.

5. (Optional) Modify the container runtime socket path.

In some cases, you have to modify the default socket path the C-VEN relies on to get information about the containers due to the following reasons:

- A non-conventional or customized container runtime socket path
- Two concurrent container runtimes

In this case, you may have to modify the default mount path for the `unixsocks` volume in the C-VEN configuration file.

For example, you want to listen on the 'containerd' container runtime, however, docker is also used on the nodes. You should modify the file as shown below, so that the C-VEN listens to events on 'containerd':

```
          volumeMounts:
          - name: unixsocks
            mountPath: /var/run/containerd/
            <...>
      volumes:
      - name: unixsocks
        hostPath:
          path: /var/run/containerd/
          type: Directory
          <...>
```

6. Save the changes.
7. Deploy C-VEN.
   - For Kubernetes:

```
kubectl apply -f illumio-ven-kubernetes.yml
```

   - For OpenShift:

```
oc apply -f illumio-ven-openshift.yml
```

8. Verify the deployment.
   - For Kubernetes:

```
kubectl get pods -n illumio-system
```

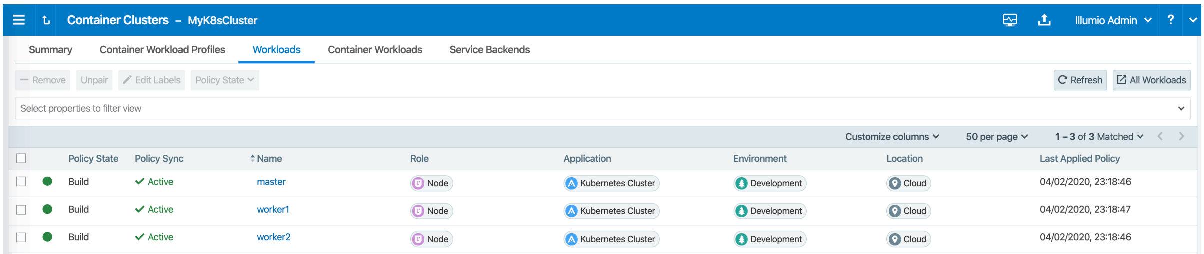
```
oc get pods -n illumio-system
```

The `illumio-ven-xxxxxxxxxx-xxxxx` Pods should be in the "Running" state.

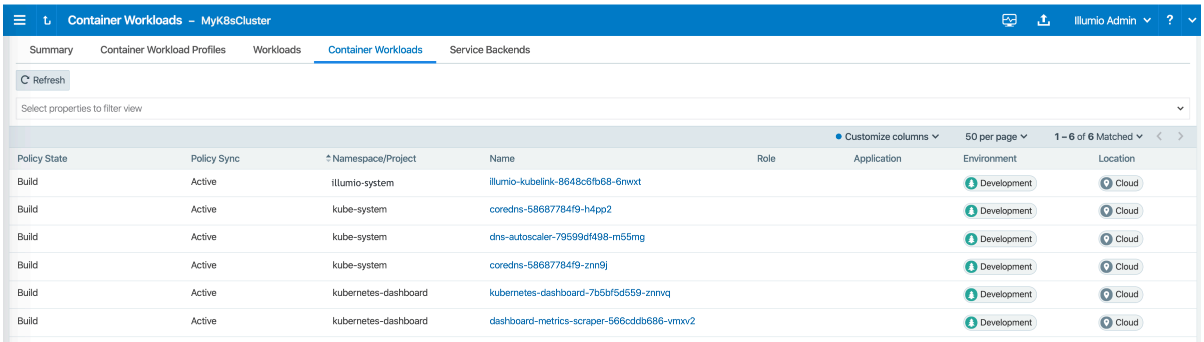
After C-VENs are successfully deployed, you can check the cluster information in the Illumio PCE UI. From the main menu, navigate to **Infrastructure > Container Clusters**.

You can also verify in the PCE UI that the C-VENs were successfully deployed by checking the following:

- Under the **Workload** tab, nodes that are part of your Kubernetes or OpenShift cluster should be listed. An example is shown below.



- Under the **Container Workloads** tab, Pods deployed in your Kubernetes or OpenShift cluster should be listed. An example is shown below.



- Illumination Map now displays system and application Pods running in your cluster.

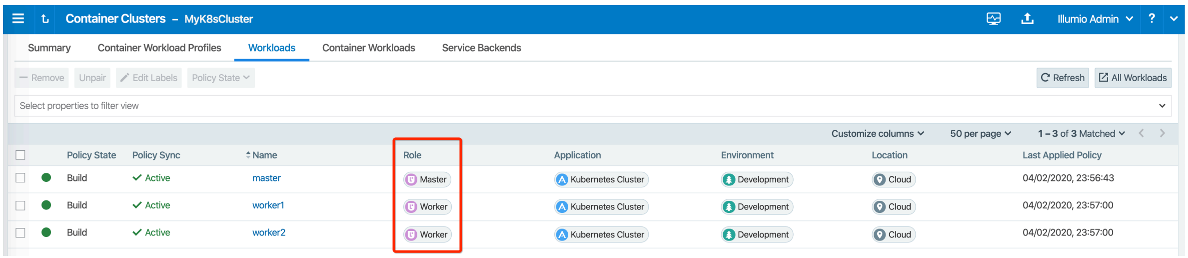
Re-Label Your Cluster Nodes

NOTE
Re-labeling the cluster nodes is optional.

In the case of self-managed deployments in which both Master and Worker nodes are managed, you may want to re-label your nodes to differentiate Master nodes from Worker nodes. Doing this helps when you are writing different policies for the Worker and Master nodes or if you want to segment these nodes differently.

To re-label your cluster nodes:

1. In the PCE UI, go to **Infrastructure > Container Clusters > YourClusterName > Workloads**.
2. Select the workloads you want to re-label.
3. Click **Edit Labels** to assign the new labels (for example, Master and Worker).



4. After re-labeling your cluster nodes, the nodes part of the cluster reflect the updated label(s).

Configure Labels for Namespaces, Pods, and Services

Once Kubelink is deployed onto the Kubernetes cluster and it gets synced with the PCE, the namespaces within the cluster appear as Container Workload Profiles. By default, all namespaces are unmanaged, which means Illumio does not apply any inbound or outbound controls to the Pods within those namespaces. Any Pods or services within unmanaged namespaces do not show up in the PCE inventory or in Illumination.

Use Container Workload Profiles

The Illumio PCE administrator can change a Kubernetes namespace from unmanaged to managed by modifying the Container Workload Profile. Each profile can be modified even if the Illumio C-VEN is not yet installed on the Kubernetes nodes. If the C-VEN is deployed on the cluster nodes and Container Workload Profile is in the managed state, the Pods and services are displayed in Illumination, and they inherit the labels assigned to the Kubernetes namespace.

In non-CLAS environments, the Pods are represented in Illumio Core as Container Workloads. In CLAS-mode environments (available after Illumio Core for Kubernetes and OpenShift 5.0.0), the Kubernetes Workloads are represented independent of what Pods might exist at the time in Kubernetes for these workloads. These Kubernetes Workloads are shown separately in the PCE UI from any non-CLAS Container Workloads. If Kubernetes services exist in the respective namespace, Illumio Core represents each service as an Illumio Core Virtual Service object. In CLAS mode, only NodePort and LoadBalancer service types are shown in the PCE UI as Virtual Service objects.

This section describes how to change a namespace from unmanaged to managed, and how to use labels and custom annotations to add more context to your applications. This section also describes how to set enforcement boundaries for your containerized workloads.

1. Log in to the PCE UI and navigate to **Infrastructure** > **Container Clusters**.
2. Select the **Container Cluster** you want to manage.
3. Select the **Container Workload Profiles** tab.
4. You will see a list of all namespaces in the cluster. Select the namespace you want to manage.
5. Click **Edit**:
   a. Enter a Name (optional).
   b. Select a Management state (any state, except unmanaged).
   c. Select an Enforcement mode for how policy rules will be enforced.
   d. Select a Visibility state.
   e. Assign Labels (optional).
   f. Click **Save**.

Configure New Container Workload Profiles

A Container Workload Profile is beneficial when you want to assign labels to resources that are deployed in a namespace and also define the state of the policy created for the scope of labels assigned. A new Container Workload Profile can be created in either of the following ways:

- Dynamically created through the creation of a new namespace in the Kubernetes or OpenShift cluster. This is a *reactive* option in which the Illumio Core Administrator assigns labels and a policy state after the creation of the namespace.
- Manually pre-created to assign labels and a policy state to a namespace that will be created later on. This is a *proactive* option in which the Illumio Core Administrator assigns labels and a policy state before the creation of the namespace. This option offers the best-in-class security mechanism and authenticates each namespace created in the cluster by leveraging the concept of pairing key (same concept that Illumio Core provides in a pairing profile).

TIP

For a best-in-class security deployment, Illumio recommends to *proactively* create pairing profiles and assign labels and a policy state to them. The pairing key for each profile can be provided to the DevOps team for namespaces deployments later on.

When a Container Cluster is created for the first time in the PCE, Kubelink will report the existing namespaces or projects in the cluster. These namespaces will inherit what was defined as part of the Container Workload Profile Template for that cluster.

Dynamic Creation of a Profile

When the team managing Kubernetes or OpenShift clusters creates a namespace in a cluster, this namespace is reported immediately to the PCE via Kubelink. The new namespace will be listed under Container Workload Profiles and the following scenarios can occur:

- A Container Workload Profile Template exists for this cluster - The new namespace will inherit what was defined in the template, as far as Policy state and labels are concerned.
- A Container Workload Profile Template does not exist for this cluster - The new namespace will remain blank until further edited by an Illumio Core Administrator.

The example below shows a new namespace "namespace1" created in a cluster where a Container Workload Profile Template exists with a policy state set to "Visibility Only" and a partial label assignment as "Development | Cloud":

NOTE

The namespace is created by the Kubernetes or OpenShift administrator (outside the scope of Illumio Core).

For example, to edit the "namespace1" namespace:

1. Click on it and then click **Edit**.
2. Enter a *Name*.
3. Assign missing *Labels* wherever relevant or modify the existing ones.
4. After you are done, click **Save**.

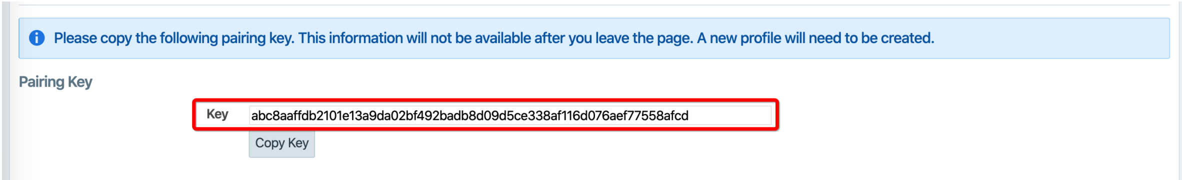
The updates are displayed in the *Container Workload Profiles* list.

Manual Pre-creation of a Profile

To pre-create a profile:

1. In the Container Workload Profiles page, click **Add**.
2. Enter a *Name*.
3. Select the desired *Management* state.

4. Select the Enforcement mode.
5. Choose a **Visibility** state. Note that starting in the Illumio Core release 24.4.0, the Enhanced Data Collection feature is freely available to enable on the PCE if you want. Before that release, it was an optional feature that you must contact Illumio Support to enable. Therefore, the availability of Enhanced Data Collection depends on what version of PCE you are using.
6. Assign *Labels* to the profile.
7. Click **Save**.



8. Click **Copy Key** and provide this key to the DevOps team, which will be used as an annotation in a namespace manifest file to authenticate this resource with the PCE.

You can view the newly-created Container Workload Profile. The status is in "Pending" state with the hourglass icon displayed next to it.

To edit the namespace configuration file to include the pairing key in order to authenticate this namespace with the PCE:

1. Navigate to `metadata: > annotations:`. If `annotations:` does not exist, create an `annotations:` section under `metadata:`.
2. Add the `com.illumio.pairing_key:` Illumio label key field under the `annotations:` section.
   - Enter the pairing key obtained during the new Container Workload Profile creation.
   - Save the file and exit.
3. Apply the change using `kubectl` commands.

An example is shown below.

```
apiVersion: v1
kind: Namespace
metadata:
  name: namespace2
  annotations:
    com.illumio.pairing_key:
    abc8aaffdb2101e13a9da02bf492badb8d09d5ce338af116d076aef77558afcd
```

The updates are displayed in the *Container Workload Profiles* list.

Container workload profile restriction

> **WARNING**
>
> There's a 1:1 mapping between the pairing key of a container workload profile and the Kubernetes namespace that uses this pairing key.
>
> Two namespaces in Kubernetes that use the same pairing key for a given container profile are not supported or accepted. Such use would cause an error in the container cluster status.

Using Annotations

> **NOTE**
>
> Illumio annotations operate differently in CLAS-mode clusters (optionally available starting in Illumio Core for Kubernetes version 5.0.0) than in previous legacy (non-CLAS) environments.
>
> The initial portion of this topic describes how to use annotations in legacy non-CLAS clusters. After this initial portion, in the latter part of this topic, you can find information about using annotations in CLAS-mode clusters, described in the section Using Annotations in CLAS [61].

When assigning labels, you can assign no labels, some labels, or all labels to the namespace. If there is a label that is not assigned, then you can insert annotations in the Deployment configuration (or application configuration) to assign labels. If there is a conflict between a label assigned via the Container Workload Profile and the annotations in the deployment configuration, the label from the Container Workload Profile overrides the Deployment configuration file. This security mechanism ensures that a malicious actor cannot spoof labels and get a preferential security policy based on a different scope. Regardless of how you assign labels, it is not required for Pods or services to have all labels in order for the PCE to manage them.

To manually annotate the different resources created in a Kubernetes namespace or OpenShift project, use the steps described in the sections below.

Deployments

1. Edit the Deployment configuration file:
   a. Navigate to `spec: > template: > metadata: > annotations:`. If `annotations:` does not exist, create an `annotations:` section underneath `metadata:`.
   b. The annotation can support any Illumio label key fields, including user-defined label types, as well as the standard set of predefined Illumio labels:

    - `com.illumio.role:`
    - `com.illumio.app:`
    - `com.illumio.env:`
    - `com.illumio.loc:`
   c. Fill in the appropriate labels.
   d. Save the file and exit.
2. Apply the change using `kubectl` commands.

Services

1. Edit the Deployment configuration file:
   a. Navigate to `metadata: > annotations:`. If `annotations:` does not exist, create an `annotations:` section underneath `metadata:`.
   b. The following Illumio label key fields can be under the `annotations:` section.
    - `com.illumio.role:`
    - `com.illumio.app:`
    - `com.illumio.env:`
    - `com.illumio.loc:`
   c. Fill in the appropriate labels.
   d. Save the file and exit.
2. Apply the change using `kubectl` commands.

> **IMPORTANT**
> When using the annotations method, you should redeploy the Pods or services after saving the changes to the configuration files by using the `kubectl apply` command.

Annotation Examples

Below are examples of namespaces, Pods, and services that use label assignments using either Container Workload Profiles or Container Workload Profiles with annotation insertion.

In the example shown below:

- Kubernetes default services or control plane Pods exist within namespaces such as, `kube-system`. They will inherit the Application, Environment, and Location labels from what has been configured in the Container Workload Profile(s). Kubelink is part of the `illumio-system` namespace, and because the Role label is left blank on the `illumio-system` namespace, you should assign a Role to Kubelink using annotations in the manifest file.
- A new `app1` namespace that contains two different Deployments or a two-tier application (Web and Database) is deployed. To achieve tier-to-tier segmentation across the application they will need different Role labels. Therefore, a Role label should be inserted into the annotations of each Deployment configuration.

A snippet of the `illumio-kubelink` Deployment configuration file is shown below, and the "Kubelink" Role label is inserted under the `spec: > template: > metadata: > annotations:` section:

illumio-kubelink-kubernetes.yml

```
spec:
  replicas: 1
  selector:
    matchLabels:
      app: illumio-kubelink
  template:
    metadata:
      annotations:
        com.illumio.role: Kubelink
      labels:
        app: illumio-kubelink
    spec:
#      nodeSelector:
#        node-role.kubernetes.io/master: ""
      serviceAccountName: illumio-kubelink
      tolerations:
      - key: node-role.kubernetes.io/master
        effect: NoSchedule
```

A snippet of the **app1**'s Web Deployment configuration file is shown below, and the "Web" Role label is inserted under the `spec: > template: > metadata: > annotations:` section:

shopping-cart-web.yml

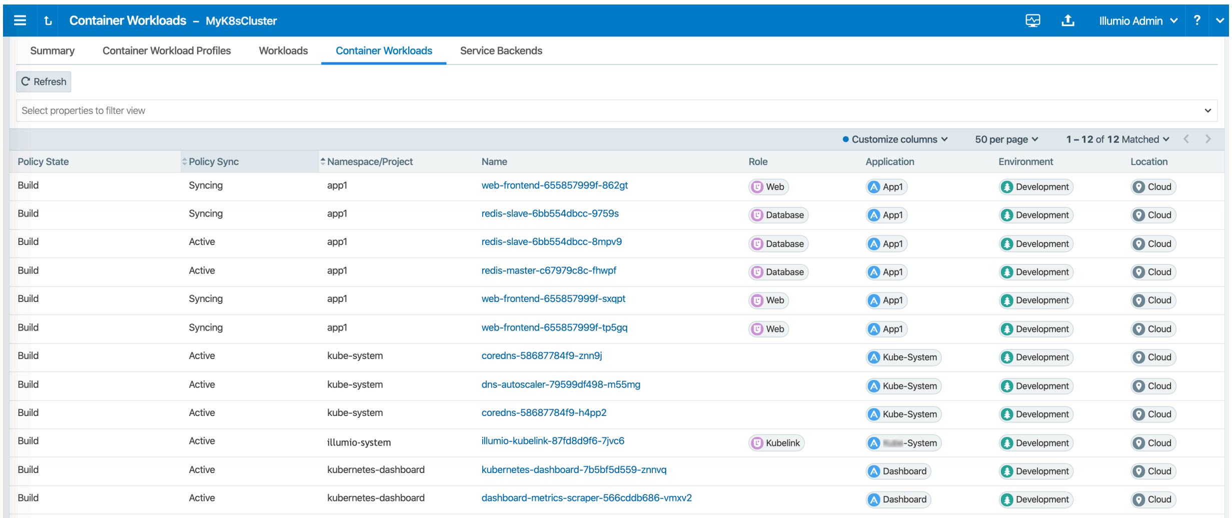
```
spec:
  replicas: 3
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      app: webapp1
      tier: frontend
  strategy:
    activeDeadlineSeconds: 21600
    resources: {}
    rollingParams:
      intervalSeconds: 1
      maxSurge: 25%
      maxUnavailable: 25%
      timeoutSeconds: 600
      updatePeriodSeconds: 1
    type: Rolling
  template:
    metadata:
      annotations:
        com.illumio.role: Web
      creationTimestamp: null
      labels:
```

A snippet of the **app1**'s Database Deployment configuration file is shown below and the "Database" Role label is inserted under the `spec: > template: > metadata: > annotations:` section:

shopping-cart-db.yml

```
spec:
  replicas: 2
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      app: redis
      role: slave
      tier: backend
  strategy:
    activeDeadlineSeconds: 21600
    recreateParams:
      timeoutSeconds: 600
    resources: {}
    type: Recreate
  template:
    metadata:
      annotations:
        com.illumio.role: Database
      creationTimestamp: null
      labels:
```

Below is the final outcome of the label assignment from the example.



Policy State	Policy Sync	Namespace/Project	Name	Role	Application	Environment	Location
Build	Syncing	app1	web-frontend-655857999f-862gt	Web	App1	Development	Cloud
Build	Syncing	app1	redis-slave-6bb554dbcc-9759s	Database	App1	Development	Cloud
Build	Active	app1	redis-slave-6bb554dbcc-8mpv9	Database	App1	Development	Cloud
Build	Active	app1	redis-master-c67979c8c-fhwpf	Database	App1	Development	Cloud
Build	Syncing	app1	web-frontend-655857999f-sxqpt	Web	App1	Development	Cloud
Build	Syncing	app1	web-frontend-655857999f-tp5gq	Web	App1	Development	Cloud
Build	Active	kube-system	coredns-58687784f9-znn9j		Kube-System	Development	Cloud
Build	Active	kube-system	dns-autoscaler-79599df498-m55mg		Kube-System	Development	Cloud
Build	Active	kube-system	coredns-58687784f9-h4pp2		Kube-System	Development	Cloud
Build	Active	illumio-system	illumio-kubelink-87fd8d9f6-7jvc6	Kubelink	[illegible]-System	Development	Cloud
Build	Active	kubernetes-dashboard	kubernetes-dashboard-7b5bf5d559-znnvq		Dashboard	Development	Cloud
Build	Active	kubernetes-dashboard	dashboard-metrics-scraper-566cddb686-vmxv2		Dashboard	Development	Cloud

In Illumination Map, the application groups will appear differently if you've assigned labels on resources in the cluster.

DaemonSets and ReplicaSets

The steps described in the above section apply only to services in Kubernetes and OpenShift which are bound to Deployment or DeploymentConfig (existing deployments). This is because Kubelink depends on the Pod hash templates to map resources together, templates that DaemonSet and ReplicaSet configurations do not have. If you discover Pods derived from DaemonSet or ReplicaSet configurations and also discover services bound to those Pods, then Kubelink will **not** automatically bind the virtual service and service backends for the PCE. The absence of this binding will create limitations with Illumio policies written against the virtual service.

To work around this limitation for DaemonSets and ReplicaSets follow the steps below.

1. Generate a random uuid using the **uuidgen** command (on any Kubernetes or OpenShift node, or your laptop).
2. Copy the output of the **uuidgen** command.
3. Edit the DaemonSet or ReplicaSet YAML configuration file.
4. Locate the **spec: > template: > metadata: > labels:** field in the YAML file and create the **Pod-template-hash:** field under the **labels:** section.
5. Paste the new uuid as the value of the **Pod-template-hash:** field.
6. Save the changes.

Repeat steps 1 through 6 for each DaemonSet or ReplicaSet configuration.

The examples below generate a random **pod-template-hash** value and applies it to a DaemonSet configuration.

```
$ uuidgen
9e6f8753-d8ac-11e8-9999-0050568b6a18
$
```

```
$ cat nginx-ds.yml
apiVersion: extensions/v1beta1
kind: DaemonSet
metadata:
  name: nginx-webserver
spec:
  template:
    metadata:
      labels:
        app: nginx-webserver
        pod-template-hash: 9e6f8753-d8ac-11e8-9999-0050568b6a18
    spec:
      containers:
        - name: webserver
          image: rstarmer/nginx-curl
          imagePullPolicy: IfNotPresent
          ports:
          - containerPort: 80
```

Static Pods

Another way of deploying Pods without Deployments or ReplicaSet is by using 'Static Pods'. In this case, a Pod is spun up by not depending on the API server and is managed by an individual node's Kubelet. Static Pods are used to spin up control-plane components such as kube-apiserver, controller-manager, and scheduler. Static Pods are useful if you want a Pod to be running even if the Kubernetes control-plane components fail. Unlike Naked Pods, if a Static Pod is not functional, kubelet spins up a new Static Pod automatically by looking at the manifest file in the **/etc/kubernetes/manifests** directory.

Services for such Pods can also be created without any selectors. In which case, you need to manually create the **EndPoint** resources for such services without a selector. For example, the default 'kubernetes' service in the default namespace which binds to the API-Server Pod running on HostNetwork.

If you create Static Pods on an overlay network, you need to create a service without selectors and manually create **EndPoint** resource to map the Pod to see the Container Workload and the Virtual Service on the PCE. You will not see any bindings or backends for this Virtual Service. In order to bind the Static Pods to the Virtual Service, use the '`com.illumio.service_uids`' annotation in the Static Pods manifest and configure the service without selectors and manually create the EndPoints. Once the '`com.illumio.service_uids`' annotation is used, you can bind the Container Workloads to its Virtual Service.

Sample code: Place the Static Pod manifest in the `/etc/kubernetes/manifests` directory

```
[root@qvc-k8s-027-master01 manifests]# pwd
/etc/kubernetes/manifests

[root@qvc-k8s-027-master01 manifests]# cat network-tool.yml
apiVersion: v1
kind: Pod
metadata:
  name: nw-tool1
  annotations:
    com.illumio.service_uids: <numerical-value>
spec:
  containers:
  - name: nw-tool1
    image: praqma/network-multitool
    args: [/bin/sh, -c, 'i=0; while true; do echo "$i: $(date)"; i=$
((i+1)); sleep 10; done']
    imagePullPolicy: IfNotPresent
  restartPolicy: Always

[root@qvc-k8s-027-master01 ~]# cat nw-tool-endpoint.yaml
apiVersion: v1
kind: Endpoints
metadata:
  name: nw-tool-svc
  namespace: default
subsets:
- addresses:
  - ip: <ip-value>
  ports:
  - name: http
    port: 80
    protocol: TCP

[root@qvc-k8s-027-master01 ~]# cat nw-tool-svc.yaml
apiVersion: v1
kind: Service
metadata:
  creationTimestamp: "2020-05-18T18:39:19Z"
  labels:
    app: nw-tool
  name: nw-tool-svc
  namespace: default
  resourceVersion: "29308511"
  selfLink: /api/v1/namespaces/default/services/nw-tool-svc
  uid: <numerical-value>
spec:
  clusterIP: <ip-value>
```

```
  ports:
  - name: http
    port: 80
    protocol: TCP
    targetPort: 80
  sessionAffinity: None
  type: ClusterIP
status:
  loadBalancer: {}
[root@qvc-k8s-027-master01 ~]#
```

IMPORTANT

In the above code sample, you need to modify the following two values based on your configuration:

- `uid: <numerical-value>`
- `clusterIP: <ip-value>`

Using Annotations in CLAS

Illumio annotations in CLAS-mode environments are specified on the Kubernetes Workload, and not on a Pod's template, as is done in legacy non-CLAS environments. This distinction follows from the concept of the Kubernetes Workload in the PCE UI introduced with CLAS-mode, which maps directly to the native Kubernetes concept of a workload resource (that is, Deployments, ReplicaSets, and the like).

Therefore, Kubernetes Workloads on the PCE should be labelled based on the corresponding workload annotations in Kubernetes, instead of on individual pod template annotations in Kubernetes.

This labelling distinction prevents confusion, because Pods from a single Deployment can have different annotations:

```
# kubectl get pod azure-vote-front-6fd8b9b657-6pv8t -n voting-app -o
jsonpath='{.metadata.annotations}' | tr ',' '\n' | grep
com.illumio"com.illumio.app":"A-VotingApp""com.illumio.env":"E-
Production""com.illumio.loc":"Azure"# kubectl get pod azure-vote-
front-6fd8b9b657-npppz -n voting-app -o jsonpath='{.metadata.annotations}'
| tr ',' '\n' | grep com.illumio
            "com.illumio.app":"A-
VotingApp""com.illumio.env":"Development""com.illumio.loc":"Amazon""com.illu
mio.role":"R-Frontend"}
```

Migration

Workloads reporting supports both: Pod template annotations and workload annotations. However, the priority is put on workload, if it contains at least one annotation with a `com.illumio.` prefix.

In the following example, annotations are specified in `metadata.annotations:` and `spec.template.metadata.annotations:`. Annotations specified in `metadata.annotations:` are prioritized.

The resulting annotations mapped to labels are: `app=A-VotingApp` and `env=E-Test` (no merging between the sets of annotations occurs).

```
apiVersion: apps/v1
kind: Deployment
metadata:
  annotations:
    com.illumio.app: A-VotingApp
    com.illumio.env: E-Test
  name: test-deployment
  labels:
    app: nginx
  spec:
    replicas: 2
    selector:
      matchLabels:
        app: test-pod
      template:
        metadata:
          annotations:
            com.illumio.loc: Amazon
            com.illumio.env: test-env
          labels:
            app: test-pod
          spec:
            containers:
              - name: test-pod
                image: nginx:1.14.2
                imagePullPolicy: IfNotPresent
                ports:
                  - containerPort: 80
```

Configure Security Policies for Containerized Environment

Security policies are a set of rules that you can configure to secure your Kubernetes or OpenShift environment. You can follow the guidelines and examples described in this section to write rules for your Kubernetes or OpenShift clusters and containerized applications, which you can then modify incrementally.

IP and FQDN Lists

FQDN Services for Kubernetes

There are some basic services that need to be defined as IP lists, such as docker.io or the Kubernetes API server. These FQDNs will be used later in the ring-fence policy for the Kubernetes cluster. The following FQDNs are commonly found to be dependencies for Kubernetes and should be defined inside Illumio Core's IP list policy objects:

- docker.io
- myregistry.example.com

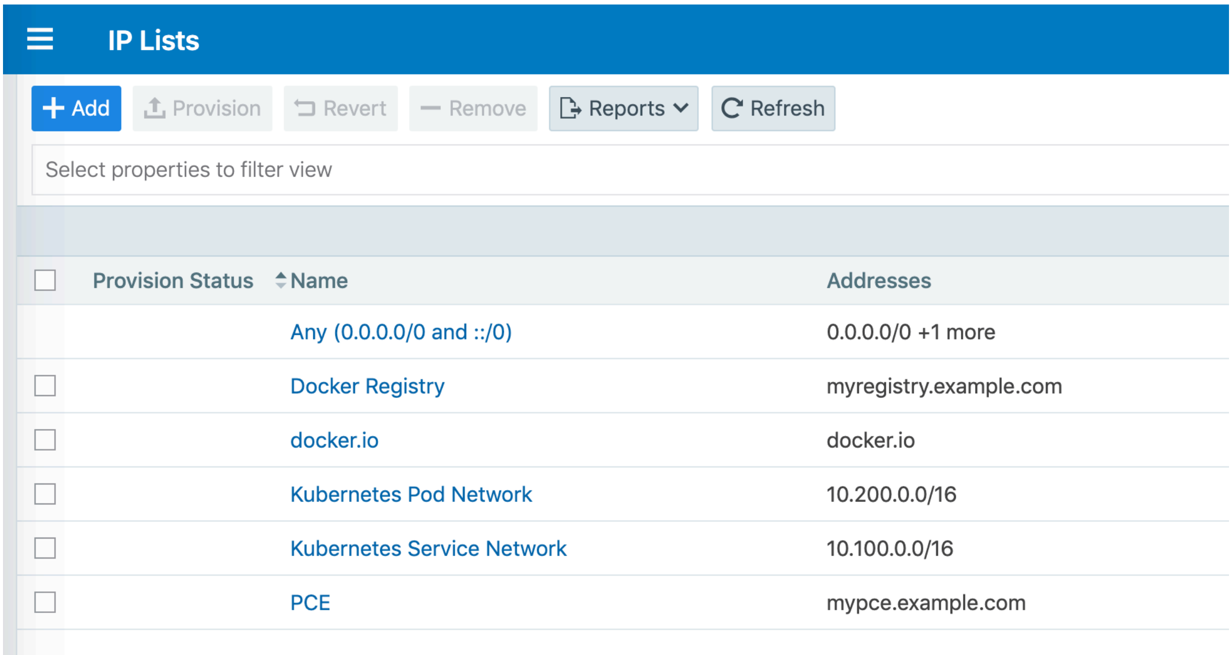
The PCE FQDN is required for Kubelink for example, mypce.example.com.

IP Lists for Kubernetes

Additionally, the following subnets or IP addresses should be defined in the IP list policy objects:

- **Kubernetes Pod Network:** Locate subnet in master node's `/etc/kubernetes/kubeadm-config.yaml` file (Ubuntu) under `networking: > podSubnet:` section, for example, `10.200.0.0/16`
- **Kubernetes Service Network:** Locate subnet in master node's `/etc/kubernetes/kubeadm-config.yaml` file (Ubuntu) under `networking > serviceSubnet` section, for example, `10.100.0.0/16`

The screenshot below displays IP lists created for Kubernetes Infrastructure dependencies.



FQDN Services for OpenShift

There are some basic services that should be defined as IP lists such as docker.io or the Kubernetes API server. These FQDNs will be used later in the ring fence policy for the OpenShift cluster. The following FQDNs are commonly found to be dependencies for OpenShift and should be defined in Illumio IP list policy objects:

- docker.io
- registry.access.redhat.com
- access.redhat.com
- subscription.rhsm.redhat.com
- github.com

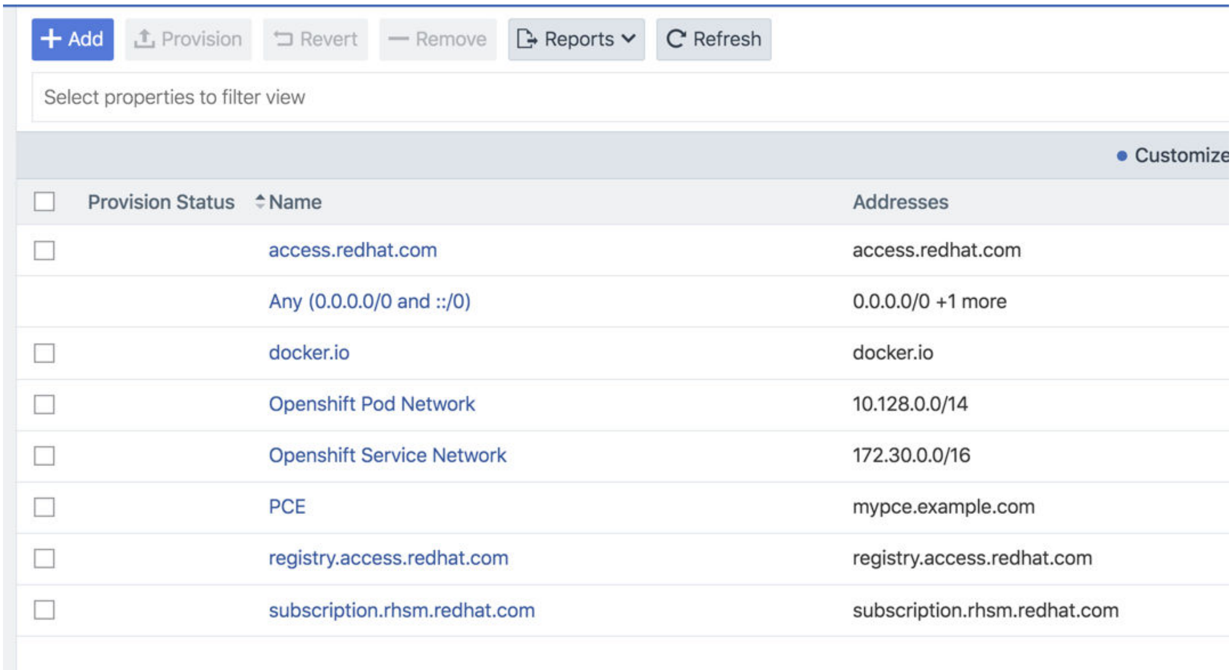
The PCE FQDN is required for Kubelink, for example, mypce.example.com.

IP Lists for OpenShift

Additionally, the following subnets or IP addresses should be defined in IP list policy objects:

- **OpenShift Pod Network:** Find subnet in master node's `/etc/origin/master/master-config.yaml` file under `networkConfig > clusterNetworkCIDR` section, for example, `10.128.0.0/14`
- **OpenShift Service Network:** Find subnet in master node's `/etc/origin/master/master-config.yaml` file under `networkConfig > serviceNetworkCIDR` section, for example, `172.30.0.0/16`

The screenshot below displays IP lists created for OpenShift Infrastructure dependencies. It references the IP lists which automatically come with the Illumio Segmentation Template.



NOTE

The IP lists mentioned above are for FQDNs and IP addresses that Illumio has found to be necessary for basic Kubernetes or OpenShift deployments. Each deployment varies and may have dependencies on additional FQDNs or IP addresses that are not mentioned in this document.

If your Kubernetes or OpenShift infrastructure needs to communicate with external services that are not mentioned here, then make sure you describe those in the IP lists.

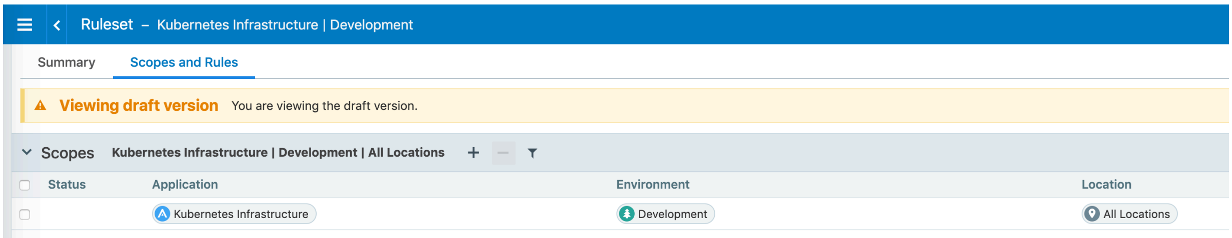
Rules for Kubernetes or OpenShift Cluster

This section assumes the following:

- Kubernetes or OpenShift cluster nodes and infrastructure Pods are activated and managed.
- Labels have been assigned to each workload and container workload.
- All cluster nodes and infrastructure Pods are in the same application group, which means they have been assigned the same application, environment, and location labels.

Kubernetes

Create a ruleset for the Kubernetes cluster and control plane Pods. The labels assigned to all of the Kubernetes nodes and control Pod workloads should fall within the scope.



Add the following lines of policy to the ruleset.

Intra-Scope Rules

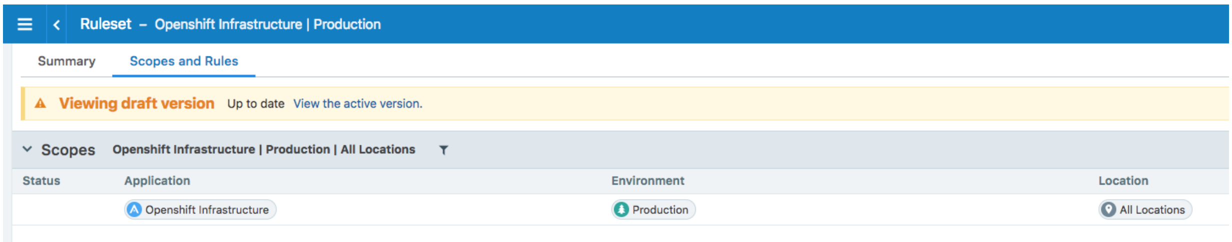
Providers	Services	Consumers	Notes
docker.io (IP List) myregistry.example.com (IP List)	All Services	All Workloads	Containerized environments depend on various external resources to perform basic operations such as pulling a docker image. Illumio has determined that the listed FQDNs are essential to Kubernetes deployments. Each deployment varies and may have dependencies on additional resources. If your container infrastructure has requirements for FQDNs not mentioned in this document, then you should include those FQDNs in this policy line.
Illumio PCE (IP List)	8443 TCP	Kubelink	Kubelink sends context about the Kubernetes cluster to the PCE over TCP 8443 port.
All Workloads	53 TCP 53 UDP	Kubernetes Pod Network (IP List)	The Kubernetes cluster provides internal DNS services to the pods (using coreDNS in this example). This policy enables internal DNS resolution for these tasks.
All Workloads (Uses Virtual Services and Workloads)	All Services	All Workloads	Any communication across all managed Kubernetes nodes or managed infrastructure pods which will be permitted by this policy.
Kubernetes Pod Network (IP List)	All Services	All Workloads	Communications across initiated by any workload which pass through service front ends will be allowed by this policy. It also covers other IP addresses on the Kubernetes pod network which are not discovered by the PCE. Critical for infrastructure functions including but not limited to liveness probes and infrastructure service front ends (Kubernetes).

Extra-scope Rules

Providers	Services	Consumers	Notes
All Workloads	6443 TCP 22 TCP	Any 0.0.0.0/0 (IP List)	Optional: Opens up ports which are purposed for remote management. For example, TCP 22 to provide SSH services to Kubernetes admins. TCP 6443 provides Kubernetes admins with dashboard services. The Dashboard may vary across Kubernetes deployments. The ports can be modified to what is used in your environment and consuming IP list can be changed to corporate subnets or jump servers.
Worker	80 TCP 443 TCP	Any 0.0.0.0/0 (IP List)	This policy assumes Ingress Controllers exist on Worker nodes. If the ingress controllers exist on other nodes, then modify the provider to the host where the Ingress controllers reside. This rule opens default front end ports which are used to access containerized applications from external IP addresses.

OpenShift

Create a ruleset for the OpenShift cluster and control plane Pods. The labels assigned to all of the OpenShift nodes and control Pod workloads should fall within the scope.



Add the following lines of policy to the ruleset.

> **NOTE**
> The IP lists referenced in this ruleset are commonly used public registries (e.g., docker.io) for container environments. If you have confirmed that your OpenShift environment does not depend on a public registry shown below, then it is recommended that you remove the IP lists from the ruleset.

Intra-scope Rules

Providers	Services	Consumers	Notes
docker.io (IP List) registry.access.redhat.com (IP List) registry.webscaleone.info (IP LIst) access.redhat.com (IP List) subscription.rhsm.redhat.com (IP List)	All Services	All Workloads	Containerized environments depend on various external resources to perform basic operations such as pulling a docker image. Illumio has determined that the listed FQDNs are essential to OpenShift deployments. Each deployment varies and may have dependencies on additional resources. If your container infrastructure has requirements for FQDNs not mentioned in this doc, then you should include those FQDNs in this policy line.
Illumio PCE (IP List)	8443 TCP	Kubelink	Kubelink sends context about the OpenShift cluster to the PCE over TCP 8443 port.
All Workloads	53 TCP 53 UDP	OpenShift Pod Network (IP List)	The OpenShift cluster in this example uses DNSmasq meaning each cluster node listens on port 53 and provides internal DNS services to the pods. This policy enables internal DNS resolution for these tasks.
All Workloads (Uses Virtual Services and Workloads)	All Services	All Workloads	Any communication across all managed OpenShift nodes or managed infrastructure pods which will be permitted by this policy.
OpenShift Pod Network (IP List) OpenShift Service Network (IP List)	All Services	All Workloads	Communications across initiated by any workload which pass through service front ends will be allowed by this policy. It also covers other IP addresses on the OpenShift pod network which are not discovered by the PCE. Critical for infrastructure functions including but not limited to liveness probes and infrastructure service front ends (Kubernetes).

Extra-Scope Rules

Providers	Services	Consumers	Notes
All Workloads	8443 TCP 22 TCP	Any 0.0.0.0/0 (IP List)	Optional: Opens up ports which are purposed for remote management. For example, TCP 22 to provide SSH services to OpenShift admins. TCP 8443 provides OpenShift admins with webconsole services. Webconsole may vary across OpenShift deployments. The ports can be modified to server other remote management services and consuming IP list can be changed to corporate subnets or jump servers.
Infra (Role)	TCP 80 TCP 443	Any 0.0.0.0/0 (IP List)	This policy assumes the router exists only on dedicated Infra nodes. If the router exists on other nodes, then modify the provider to the host where the router resides. This rule opens default front end router ports which are used to access containerized applications from external IP addresses. As you start to open up application pods to the outside world, you will need to add the application's exposed port to this policy's list of services. For example, you spin up a httpd server and expose that server on TCP 8080. The first step to allow access to the httpd server from outside is to add TCP 8080 to this line of policy.

> **NOTE**
> The IP lists referenced in the rulesets are commonly used public registries (for example, docker.io) for container environments. If you have confirmed that your Kubernetes or OpenShift environment does not depend on the public registries mentioned above, then it is recommended that you remove the IP lists from the ruleset.

Rules and Traffic Considerations with CLAS

In Container Local Actor Store (CLAS) deployments, be sure to take into account the following special traffic and policy rules considerations that differ from legacy non-CLAS environments that are described in other sections of this chapter.

Mandatory Rules

The CLAS architecture requires mandatory infrastructure rules to be in place for the cluster to work properly. Do not upgrade to the CLAS mode, or move the cluster to Full Enforcement until the following infrastructure rules are configured:

- Node -> Service CIDR IP list (9000/TCP)
- Any (0.0.0.0/0 and ::0) -> Node (2379-2380/TCP, 2379-2380/UDP)

ClusterIP Rules

In the CLAS environment, ClusterIP ports are now represented as Kubernetes Workloads when viewing traffic, and not as Virtual Services, as before.

If you want to migrate from a legacy (non-CLAS) to a CLAS environment, you must make sure that all rules that apply to ClusterIP Services are changed to "Use Workloads" at a specific time within the process of upgrading to CLAS. For complete details, see Upgrade to CLAS Architecture [91].

Because Services are now Kubernetes Workloads, the "All Workloads" flag in a rule will include all Services. Do not use "All Workloads" as a Destination in a rule. Use a more specific label instead that targets the Service.

All rules that include a label of at least one ClusterIP service will have specified ports internally replaced. However - this is not reflected in PCE UI, where the rule still displays ports.

NodePort and LoadBalancer services remain Virtual Services in the PCE. The ClusterIP part of NodePort and LoadBalancer services also exist as a Kubernetes Workload and are linked with the Virtual Service. in the PCE. The ClusterIP part of NodePort and LoadBalancer services also exist as a Kubernetes Workload and are linked with the Virtual Service.

General Traffic View Changes

The following is a summary of general changes to traffic views in a CLAS-enabled cluster:

- Kubernetes workloads (for example, Deployments) are now shown in the UI as Kubernetes Workloads, and not as Container Workloads. Container Workloads (Pods) are still shown in non-CLAS clusters.
- ClusterIP Virtual Services are now shown as Kubernetes Workloads.
- NodePort and LoadBalancer services remain Virtual Services in the PCE.
- Traffic from other Virtual Services to Kubernetes Workloads is not shown.
- Traffic between Kubernetes Workloads within a cluster is shown.

CLAS Traffic Limitations

Consider the following differences and limitations in these scenarios when viewing traffic and writing rules in a CLAS environment:

- **Pod to Host**

  When a Pod is on a different Node than target Node, additional traffic is shown occurring from the Pod's Node to the target Node. This traffic cannot be selectively hidden with filters because it behaves the same way as traffic from host to host that should not be hidden. (This behavior occurs only when Calico is used as the CNI.)
- **Pod to ClusterIP**

  Additional direct traffic occurs from a source Pod to a source Target, regardless whether it is destined for the same node or a different target node.
- **Managed Workload to NodePort**

  Additional traffic is shown for a Client's direct access to a target Pod, and from a Used Node to a target Pod.

  Also, crucial traffic destined for a NodePort is actually showing the Node with the Node-Port's port.
- **Unmanaged Workload (or Internet) to NodePort**

  Additional traffic is shown for a Client's direct access to a target Pod, and from a Used Node to a target Pod.

  Also, in the traffic from the Client to the NodePort virtual service, we are missing the crucial traffic with NodePort's port.
- **Draft Traffic to Virtual Services**

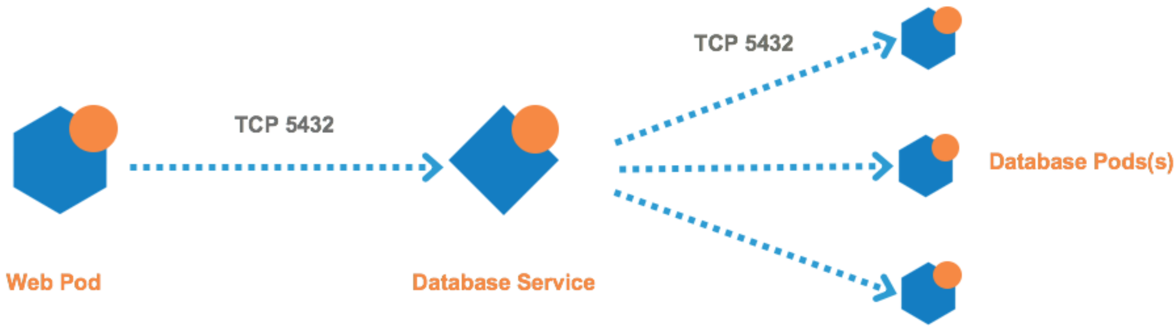
  Traffic in Draft mode where the destination is a Virtual Service is marked as Potentially Blocked.

Rules for Containerized Applications

This section covers different scenarios on writing rules for containerized applications.

Access Services from within the Cluster

For connections to a service from within the cluster, the Pods connect to a Service IP and the connections get distributed to the Pods.



Kubernetes

The rules you need to write are:

Example Ruleset

Scope

Application	Environment	Location
Risk Assessment	Development	Cloud

Intra-Scope Rule

Source	Service	Destination	Notes
Database (Virtual Service Role label for database service) + Use Virtual Services Only	Derived from Provider Virtual Service	Web (Role for Web pods)	Once the database service gets discovered by the PCE it becomes a virtual service object in the PCE - not a container workload. The provider should be the role label of the virtual service plus the "Use Virtual Service Only" option. The Consumer in this example is the Web pod. Use the Web Role label which describes the pod. Leave the Providing Service empty. Once the rule is saved, it will automatically populate with *Derived from Provider Virtual Service*. **NOTE** This does not allow Web pods to directly access Database pods through the pod IP. This only allows traffic through the service.

OpenShift

The rules you need to write are:

Example Ruleset

Scope

Application	Environment	Location
Risk Assessment	Production	HQ

Intra-Scope Rule

Source	Service	Destination	Notes
Database (Virtual Service Role label for database service) + Use Virtual Services Only	Derived from Provider Virtual Service	Web (Role for Web pods)	Once the database service gets discovered by the PCE it becomes a virtual service object in the PCE - not a container workload. The provider should be the role label of the virtual service plus the "Use Virtual Service Only" option. The Consumer in this example is the Web pod. Use the Web Role label which describes the pod. Leave the Providing Service empty. Once the rule is saved, it will automatically populate with *Derived from Provider Virtual Service*. **NOTE** This does not allow Web pods to directly access Database pods through the pod IP. This only allows traffic through the service.

Access Services from Outside the Cluster

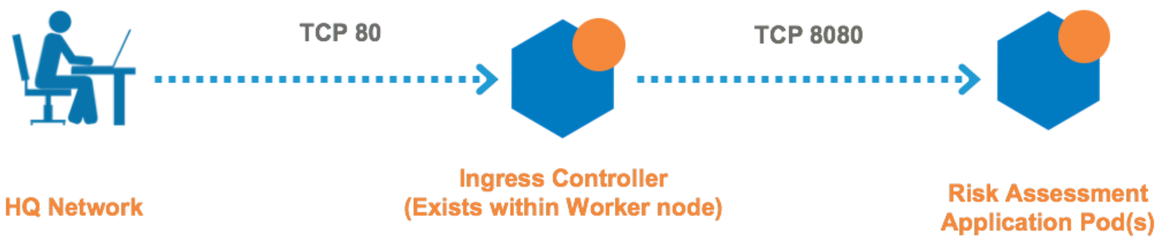
Kubernetes

With Kubernetes, connections to a containerized application from the outside world can be handled in many different ways. In this release, Illumio supports only configurations which expose applications via the Kubernetes NGINX ingress controller (HostNetwork type only). Exposing applications using HostPort are not supported.

In typical Kubernetes deployments, connections to a containerized application from the outside world go through the ingress controllers, then the connection goes directly from controllers to the pods - not the service. Example of scenario and rule coverage are shown below.

Scenario:

- The Kubernetes cluster and containerized applications are in the Development environment
- The containerized application is called RiskAssessment and each Pod within the application listens on TCP 8080
- The RiskAssessment application is exposed to the outside world via the ingress controller. The controller listens on TCP port 80 for the RiskAssessment application
- In Illumio, the RiskAssessment workloads (Pods) provide to the controller on TCP 8080. The controller provides TCP 80 to the outside world.



The rules you need to write are:

Example Ruleset 1

Scope

Application	Environment	Location
Risk Assessment	Development	Cloud

Intra-Scope Rule

Provider	Providing Service	Consumer
All Workloads	All Services	All Workloads

Extra-Scope Rule

Provider	Providing Service	Consumer	Notes
Risk Assessment	TCP 8080	Worker	The consumer should be the role label of the nodes which nest the Ingress controllers.

Example Ruleset 2

The second ruleset opens the ingress controller to the external network. The rule and ruleset below should have been created from the Rules for Kubernetes or OpenShift Cluster [64] section of this guide. You can modify the ruleset as needed.

Scope

Application	Environment	Location	Notes
Kubernetes Infrastructure	Development	Cloud	The scope of the ruleset should match the Kubernetes infrastructure scope.

Intra-Scope Rule

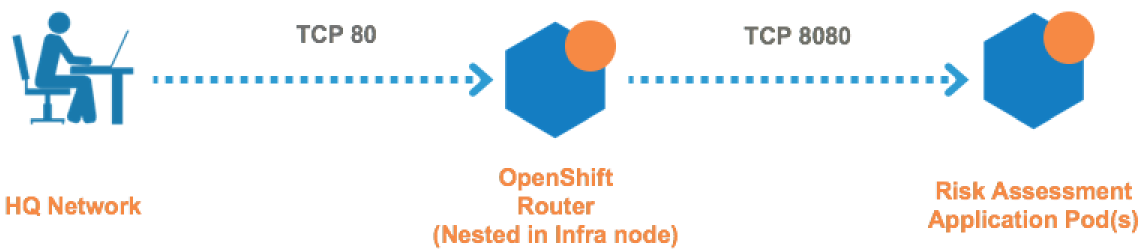
Provider	Providing Service	Consumer	Notes
Worker Node(s)	TCP 80	External Network	This rule should exist from the Rules for Kubernetes or OpenShift Cluster [64] section. The provider should be the Kubernetes node(s) which contain the ingress controller. The consumer can be an IP List such as 0.0.0.0/0 (any), HQ, Corporate, or employee subnet that requires connectivity into the exposed container workloads.

OpenShift

Connections to a containerized application from the outside world go through the OpenShift Router, then the connection goes directly from router to the Pods - not the service. Example of scenario and rule coverage are shown below.

Scenario:

- The OpenShift cluster and containerized applications are in the development environment
- The containerized application is called RiskAssessment and each Pod within the application listens on TCP 8080
- The RiskAssessment application is exposed to the outside world via the router. The router listens on TCP port 80 for the RiskAssessment application
- In Illumio, the RiskAssessment workloads (Pods) provide to the router on TCP 8080. The router provides TCP 80 to the outside world.



The rules you need to write are:

Example Ruleset 1

Scope

Application	Environment	Location
Risk Assessment	Production	HQ

Intra-Scope Rule

Provider	Providing Service	Consumer
All Workloads	All Services	All Workloads

Extra-Scope Rule

Provider	Providing Service	Consumer	Notes
Risk Assessment	TCP 8080	IST Infra (Role)	Consumer refers to the Illumio Segmentation Template. The consumer should be the role label of the node(s) which nest the OpenShift Router.

Example Ruleset 2

The following Ruleset is from the Segmentation Template and you can modify it as needed.

Scope

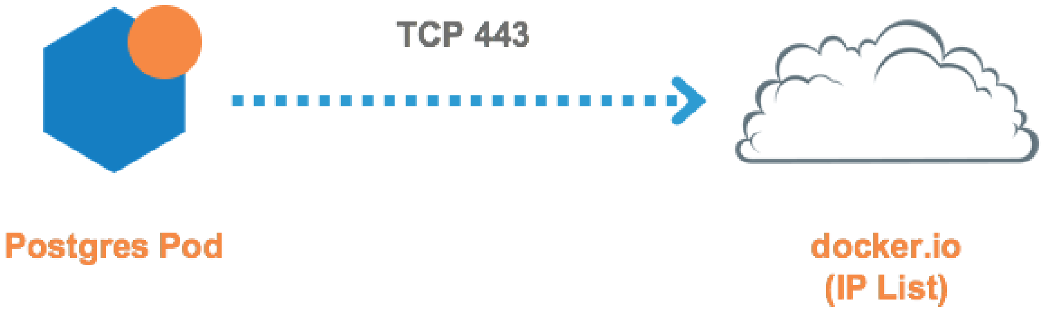
Application	Environment	Location	Notes
IST OpenShift Infrastructure	IST Production	IST HQ	Ruleset is derived from Illumio Segmentation Template. The scope should match the OpenShift cluster.

Intra-Scope Rule

Provider	Providing Service	Consumer	Notes
IST Infra (Role)	TCP 80	External Network	This rule is included in Illumio Segmentation Template. The provider should be the OpenShift cluster node(s) which nest the router. The consumer can be an IP list such as 0.0.0.0/0 (any), HQ, Corporate, or employee subnet. The IST default includes 0.0.0.0/0 (any) IP list.

Outbound Connections

The outbound connections are required to access repositories.



Kubernetes and OpenShift

The rules you need to write are:

Example Ruleset

Scope

Application	Environment	Location
Risk Assessment	Development	Cloud

Intra-Scope Rule

Provid-er	Provid-ing Serv-ice	Consum-er	Notes
docker.io (IP List)	All Services	Database (Role for Postgres Pods)	Once the database service gets discovered by the PCE it becomes a virtual service object in the PCE - not a container workload. The provider should be the role label of the virtual service plus the "Use Virtual Service Only" option. The Consumer in this example is the Web Pod. Use the Web Role label which describes the Pod. Leave the Providing Service empty. Once the rule is saved, it will automatically populate with *Derived from Provider Virtual Service*.

Liveness Probes

Containerized applications may require periodic health checks known as liveness probes and readiness probes. Each application includes a health check YAML file which contains liveness and readiness probe configurations. The health checks between the container node and the local container workload may rely on TCP ports. Illumio has included a consumer object called Container Host for this use case. The Container Host object represents the container node or nodes which host the Pod(s). The example below uses the Container Host object as a consumer for Liveness and Readiness Probes.

> **NOTE**
> The Container Host must always fall under an Extra-Scope rule.

The rules you need to write are:

Kubernetes and OpenShift

The rules you need to write are:

Example Ruleset

Scope

Application	Environment	Location
Risk Assessment	Development	Cloud

Extra-Scope Rule

Provider	Providing Service	Consumer	Notes
All Work-loads	TCP 9090	Container Host (built-in Illumio object)	In this example, the Risk Assessment health check configuration indicates that liveness probe occurs on TCP 9090. Liveness probe ports/protocols may vary across applications. Container Host is an object built into the PCE by default and represents any node which hosts the respective Pod(s).

NodePort Support on Kubernetes and OpenShift

Kubernetes (and OpenShift) provide a mechanism to access cluster services from the outside world, of type NodePort. This service exposes a port on all nodes in the cluster on which traffic will be forwarded to any of the backing pods that match the service's selector.

Scenario:

- The Kubernetes cluster and containerized applications are in the Production environment.
- The containerized application is called RiskAssessment, and each Pod within the application listens on TCP 8080.
- The RiskAssessment application is exposed to the outside world via a FrontEnd service with type NodePort.
- The exact NodePort in use is not specified, but is automatically allocated by Kubernetes.
- There may be clients to the FrontEnd service within the cluster or outside the cluster - in both cases, they are labeled as Client.

The rules you need to write are:

Example Ruleset 1: Internal and External Access to Service

Scope

Application	Environment	Location
Risk Assessment	Production	Cloud

Extra-Scope Rule

Provider	Providing Service	Consumer	Notes
FrontEnd (Virtual Service Role label for Risk Assessment service) + Use Virtual Services Only	Derived from Provider Virtual Service	Client (Role label for Web pods and external workloads)	Once the Risk Assessment service gets discovered by the PCE it becomes a virtual service object in the PCE. The Provider here should be the role label of the virtual service plus the "Use Virtual Service Only" option.

Rules for Persistent Storage

This section only applies to deployments which require communication with external storage nodes over NFS, iSCSI, and others for persistent storage. If the cluster or Pods have external storage dependencies, then you need a policy to allow outbound communications to the storage node. The storage node can be represented as an unmanaged workload or IP list.

The following is an example of outbound policy to a NFS node, which is represented by an IP list.

Kubernetes

The following is an example of an outbound policy to an NFS node, which is represented by an IP list:

Example Ruleset 1

Scope

Application	Environment	Location	Notes
Kubernetes Infrastructure	Development	Cloud	Kubernetes cluster

Intra-Scope Rule

Provider	Providing Serv-ice	Consumer	Notes
NFS Storage (IP List)	TCP 2049	All Workloads	All Kubernetes nodes and infrastructure Pods can communicate outbound to NFS over the NFS TCP port.

Example Ruleset 2

Scope

Application	Environment	Location	Notes
ERP	Development	Cloud	From httpd example

Intra-Scope Rule

Provider	Providing Service	Consumer	Notes
NFS Storage (IP List)	TCP 2049	All Workloads	All Pods can talk outbound to NFS over the NFS TCP port.

OpenShift

The following is an example of an outbound policy to an NFS node, which is represented by an IP list:

Example Ruleset 1

Scope

Application	Environment	Location	Notes
OpenShift Infrastructure	Development	Cloud	OpenShift cluster

Intra-Scope Rule

Provider	Providing Service	Consumer	Notes
NFS Storage (IP List)	TCP 2049	All Workloads	All OpenShift nodes and infrastructure Pods can communicate outbound to NFS over the NFS TCP port.

Example Ruleset 2

Scope

Application	Environment	Location	Notes
ERP	Development	Cloud	From httpd example

Intra-Scope Rule

Provider	Providing Service	Consumer	Notes
NFS Storage (IP List)	TCP 2049	All Workloads	All Pods can talk outbound to NFS over the NFS TCP port.

Local Policy Convergence Controller

The local policy convergence controller provides a deterministic way of setting the readiness state of pods in your cluster after local policy has converged. By controlling the readiness state of pods, you can prevent them from receiving and sending traffic through Kubernetes until they are ready. Using a controller ensures that the network and security infrastructure is ready for a multi-microservice application.

In this release, the Kubernetes Custom Pod Conditions feature introduced in v1.14 is available for containerized VENs.

About the Controller Behavior

By default, the readiness gate is not specified on a pod spec and the C-VEN does not affect the readiness state of the pod regardless of annotations or Illumio managed state.

When the Illumio readiness gate is specified on a pod spec, the PCE completes the following actions when a new pod is created:

1. Sends the C-VEN policy for the new pod P.
2. When pod P is managed, the C-VEN applies local policy for the new pod P.
3. The C-VEN waits for a timer to expire to allow peers to apply policy on their end (such as, updating the new pod P IP address).

   By default, the timer uses the following values:
   - If the pod is managed by Illumio, the timer is set to 15 seconds.

- If the pod is not managed by Illumio, the timer is set to 0 seconds.

> **TIP**
> To configure a custom value for the timer duration, see Timer Customization [80].

4. The C-VEN sets the readiness gate pod condition to "True."
   The pod is now considered "Ready" by Kubernetes.

Configure the Illumio Readiness Gate

To use a local policy convergence controller, specify the Illumio readiness gate under `readinessGates.conditionType` in the pod spec YAML.

See the following example pod spec YAML file:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-deploy
spec:
  selector:
    matchLabels:
      app: my-pod
  replicas: 2
  template:
    metadata:
      labels:
        app: my-pod
    spec:
      readinessGates:
# <----- declare readiness gates
      - conditionType: "com.illumio.policy-ready"
 # <----- Illumio policy convergence readiness gate
      containers:
      - name: my-pod-web
        image: nginx
        ports:
        - containerPort: 80
```

Timer Customization

You can customize the timer cluster-wide or pre-pod.

> **NOTE**
> When configuring a custom timer by using the DaemonSet environment variable or an annotation, you are limited to specifying 0-300 seconds.

Cluster Wide Timer Customization

To customize the timer duration on a cluster-wide basis, set the readiness gate timer variable in the C-VEN DaemonSet YAML.

See the following YAML file:

```
...
containers:
  - name: illumio-ven
    env:
      - name: ILO_SERVER
        valueFrom:
          secretKeyRef:
            name: illumio-ven-config
            key: ilo_server
      - name: ILO_CODE
        valueFrom:
          secretKeyRef:
            name: illumio-ven-config
            key: ilo_code
      - name: ILO_K8S_NODE_NAME
        valueFrom:
          fieldRef:
            fieldPath: spec.nodeName
      - name: ILO_K8S_READINESS_TIMER
 # <--- custom readiness gate timer across the cluster
        value: "20"
 # <--- timer value
...
```

Pre-pod Timer Customization

To customize the timer duration for specific pods, set the Illumio readiness gate timer annotation on the pod spec.

See the following example deployment:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-deploy
spec:
  selector:
    matchLabels:
      app: my-pod
  replicas: 2
  template:
    metadata:
      labels:
        app: my-pod
      annotations:
        com.illumio.readiness-gate-timer: "20"
# <----- custom readiness gate timer for all pods in this deployment
    spec:
```

```
      readinessGates:
      - conditionType: "com.illumio.policy-ready"
      containers:
      - name: my-pod-web
        image: nginx
        ports:
        - containerPort: 80
```

Track the State of the Readiness Gate

You can track the state of the readiness gate by running either of the following commands:

- `kubectl get pod -o wide`
- `kubectl get ep -o wide`

Example: State of the Readiness Gate

This example shows a cluster with Kubelink and the C-VEN deployed and running. When you initially deploy or scaled up the Illumio Readiness Gate, you see the following values:

> **NOTE**
> The state of gate readiness appears in the "READINESS GATES" column.

```
$ kubectl get pod,ep -o wide
NAME                                 READY   STATUS    RESTARTS   AGE
IP           NODE       NOMINATED NODE   READINESS GATES
pod/my-deploy-855dfbf94f-gwz7c       1/1     Running   1          4d20h
172.17.0.7   ubuntu20   <none>           0/1
pod/my-deploy-855dfbf94f-p7czp       1/1     Running   1          4d20h
172.17.0.6   ubuntu20   <none>           0/1

NAME                      ENDPOINTS                           AGE
endpoints/kubernetes      10.0.2.15:8443                      19d
endpoints/my-service                                          4d22h
```

In this example, the readiness gates are marked as 0/1 for both pods and my-service does not have any available endpoints. After the VEN has processed the policy for the new pods and the timer expires, it sets the readiness gate to "True" for each pod and you see the following output:

```
$ kubectl get pod,ep -o wide
NAME                                 READY   STATUS    RESTARTS   AGE
IP           NODE       NOMINATED NODE   READINESS GATES
pod/my-deploy-855dfbf94f-gwz7c       1/1     Running   1          4d20h
172.17.0.7   ubuntu20   <none>           1/1
pod/my-deploy-855dfbf94f-p7czp       1/1     Running   1          4d20h
172.17.0.6   ubuntu20   <none>           1/1

NAME                      ENDPOINTS                           AGE
endpoints/kubernetes      10.0.2.15:8443                      19d
endpoints/my-service      172.17.0.6:9376,172.17.0.7:9376     4d22h
```

To view greater detail about the pod conditions, run the command `kubectl get pod <pod name> -o yaml`:

```
$ kubectl get pod my-deploy-855dfbf94f-gwz7c -o yaml
...
status:
  conditions:
  - lastProbeTime: null                         // <--
    lastTransitionTime: "2021-05-18T20:26:26Z"  // <--
    message: Pod Policy Ready                     // <-- this pod condition
is set by VEN
    reason: PolicyReady                         // <--
    status: "True"                              // <--
    type: illumio.com/policy-ready              // <--
  - lastProbeTime: null
    lastTransitionTime: "2021-05-18T20:25:51Z"
    status: "True"
    type: Initialized
  - lastProbeTime: null
    lastTransitionTime: "2021-05-19T19:56:24Z"
    status: "True"
    type: Ready
// <-- this is only set to True after all readiness gates are set to True
  - lastProbeTime: null
    lastTransitionTime: "2021-05-19T19:56:24Z"
    status: "True"
    type: ContainersReady
  - lastProbeTime: null
    lastTransitionTime: "2021-05-18T20:25:51Z"
    status: "True"
    type: PodScheduled
...
```

Firewall Coexistence on Pods

The Illumio C-VEN configures iptables on each host and each Pod (in a managed namespace). By default, Illumio Core coexistence mode is set to **Exclusive** meaning the C-VEN will take full control of iptables and flush any rules or chains that are not created by Illumio Core. In containerized environments, this may affect communications to/from container components (Docker, Kubernetes, Illumio Kubelink). Therefore, Illumio Core must allow firewall coexistence in order to achieve non-disruptive installation and deployment.

NOTE

For Workloads part of a Container cluster (Kubernetes or OpenShift nodes), firewall coexistence is enabled by default if Kubelink was deployed and is "In Sync" with the PCE (prior to the C-VEN installation).

In some cases, there may be some Pods that implement iptables rules inside the Pod namespace for the containerized application to work (VPN, NAT, and others). In order to support such requirements from containerized applications, you should enable firewall coexistence for these Pods.

In order to allow firewall coexistence, you must set a scope of Illumio labels in the firewall coexistence configuration. Once you provision a firewall coexistence scope, the PCE will enable firewall coexistence configuration on all the Pods whose labels fall within the scope.

> **NOTE**
> Labels assigned to Kubernetes cluster nodes must fall within the firewall coexistence scope. This is not a requirement for the labels assigned to container workloads.

To configure firewall coexistence:

1. In the PCE UI, navigate to **Settings** > **Security**.
2. On the Security page, select the **Manage Firewall Coexistence** tab.
3. Click **Edit**.
4. In the edit wizard, click **Add**. The **Add Firewall Coexistence Labels and Policy State** wizard will pop-up.
5. Select a scope of Illumio labels. The scope must include the labels you intend to use for your Kubernetes cluster nodes.
   a. Select **All** for Policy State.
   b. Illumio Core is Primary Firewall - Select your preference.
      i. **Yes** = (Recommended) Illumio iptable chains will be at the top of iptables at all times. Non-Illumio iptable chains can coexist, but will follow after Illumio chains.
      ii. **No** = (Not Recommended) Non-Illumio iptable chains may coexist and can be placed before Illumio chains.

      > **NOTE**
      > For deployments using Calico, Illumio recommends setting the Calico ChainInsertMode to **Append** and set Illumio Core as Primary Firewall value to **Yes**. If the Kubernetes cluster requires Calico Insert mode, then set Illumio Core as Primary Firewall value to **No**.

   c. Click **OK**.
6. Click **Save**.
7. Provision the changes.

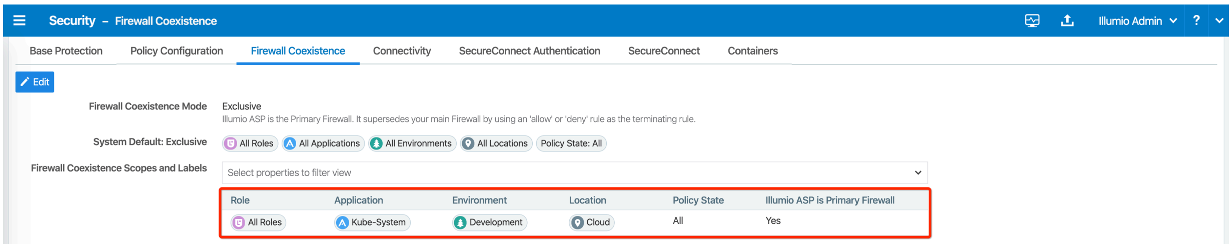
Be sure to provision the saved changes or else firewall coexistence will not take effect.

The following example is of a firewall coexistence scope for a Kubernetes or OpenShift cluster which has the following labels:

- Role: All
- Application: Kube-System
- Environment: Development

- Location: Cloud

The firewall coexistence scope in the example uses the 'All Roles' objects to cover future Pods spun up in the kube-system namespace that may require additional iptables rules to forward packets.



Upgrade and Uninstallation

Follow the steps and sequence described in this section to upgrade or uninstall Illumio Core for Kubernetes components. This section also describes the procedure for migrating from a deployment of C-VEN version 21.5.15 or earlier (which did not use Helm Charts) to a current Helm Chart deployment.

This chapter also describes how to upgrade a non-CLAS deployment to a CLAS-enabled one (which is the default mode starting in version 5.0.0 of Illumio Core for Kubernetes).

> **IMPORTANT**
> Use the proper upgrade and uninstallation procedures according to the method that was first used to deploy the product. For deployments made with a Helm Chart (typically with Illumio Core for Kubernetes 3.0.0 or later), follow the steps in Upgrade and Uninstall Helm Chart Deployments [88]. For deployments made without using a Helm Chart (for installations of C-VEN 21.5.15 or earlier), follow the steps in Upgrade and Uninstall Non-Helm Chart Deployments. [89]

Migrate from Previous C-VEN Versions (21.5.15 or Earlier)

This section describes the steps to migrate a manually-deployed Illumio installation to a Helm-managed deployment. Manually-deployed (or, non-Helm deployments) were used to configure and deploy C-VEN versions 21.5.15 and earlier, and Kubelink versions earlier than 3.0.

To upgrade an existing Helm installation to a newer version, follow standard Helm practice with **`helm upgrade`** command.

Follow these general steps to migrate from a manually-deployed Illumio Core for Kubernetes to a Helm Chart deployment:

1. Annotate and label resources.
2. Delete C-VEN DaemonSet.
3. Install Helm and the Helm Chart.

Annotate and Label Resources

From Helm version 3.0.0 on, Helm supports adopting already-deployed resources with the correct name, annotations, and labels.

Required annotations and labels are:

```
annotations:
  meta.helm.sh/release-name: illumio
  meta.helm.sh/release-namespace: illumio-system
labels:
  app.kubernetes.io/managed-by: Helm
```

To annotate and label all Illumio resources, use the commands below (provided the names of resources match your deployment). Note the **`--overwrite`** flag which replaces any existing ownership annotations that might be already assigned.

```
kubectl -n illumio-system annotate secret illumio-ven-config meta.helm.sh/
release-name=illumio --overwrite
kubectl -n illumio-system annotate secret illumio-ven-config meta.helm.sh/
release-namespace=illumio-system --overwrite
kubectl -n illumio-system label secret illumio-ven-config app.kubernetes.io/
managed-by=Helm --overwrite
kubectl -n illumio-system annotate secret illumio-kubelink-config
meta.helm.sh/
release-name=illumio --overwrite
kubectl -n illumio-system annotate secret illumio-kubelink-config
meta.helm.sh/
release-namespace=illumio-system --overwrite
kubectl -n illumio-system label secret illumio-kubelink-config
app.kubernetes.io/
managed-by=Helm --overwrite
kubectl -n illumio-system annotate serviceaccount illumio-ven meta.helm.sh/
release-name=illumio --overwrite
kubectl -n illumio-system annotate serviceaccount illumio-ven meta.helm.sh/
release-namespace=illumio-system --overwrite
kubectl -n illumio-system label serviceaccount illumio-ven
app.kubernetes.io/
managed-by=Helm --overwrite
kubectl -n illumio-system annotate clusterrole illumio-kubelink
meta.helm.sh/
release-name=illumio --overwrite
kubectl -n illumio-system annotate clusterrole illumio-kubelink
meta.helm.sh/
release-namespace=illumio-system --overwrite
kubectl -n illumio-system label clusterrole illumio-kubelink
app.kubernetes.io/
managed-by=Helm --overwrite
kubectl -n illumio-system annotate clusterrolebinding illumio-ven
meta.helm.sh/
release-name=illumio --overwrite
```

```
kubectl -n illumio-system annotate clusterrolebinding illumio-ven
meta.helm.sh/
release-namespace=illumio-system --overwrite
kubectl -n illumio-system label clusterrolebinding illumio-ven
app.kubernetes.io/
managed-by=Helm --overwrite
kubectl -n illumio-system annotate clusterrole illumio-ven meta.helm.sh/
release-name=illumio --overwrite
kubectl -n illumio-system annotate clusterrole illumio-ven meta.helm.sh/
release-namespace=illumio-system --overwrite
kubectl -n illumio-system label clusterrole illumio-ven app.kubernetes.io/
managed-by=Helm --overwrite
kubectl -n illumio-system annotate serviceaccount illumio-kubelink
meta.helm.sh/
release-name=illumio --overwrite
kubectl -n illumio-system annotate serviceaccount illumio-kubelink
meta.helm.sh/
release-namespace=illumio-system --overwrite
kubectl -n illumio-system label serviceaccount illumio-kubelink
app.kubernetes.io/
managed-by=Helm --overwrite
kubectl -n illumio-system annotate deployment illumio-kubelink meta.helm.sh/
release-name=illumio --overwrite
kubectl -n illumio-system annotate deployment illumio-kubelink meta.helm.sh/
release-namespace=illumio-system --overwrite
kubectl -n illumio-system label deployment illumio-kubelink
app.kubernetes.io/
managed-by=Helm --overwrite
kubectl -n illumio-system annotate clusterrolebinding
illumio-kubelink meta.helm.sh/release-name=illumio --overwrite
kubectl -n illumio-system annotate clusterrolebinding
illumio-kubelink meta.helm.sh/release-namespace=illumio-system --overwrite
kubectl -n illumio-system label clusterrolebinding
illumio-kubelink app.kubernetes.io/managed-by=Helm --overwrite
```

The output should look similar to this:

```
...
clusterrolebinding.rbac.authorization.k8s.io/illumio-kubelink annotated
clusterrolebinding.rbac.authorization.k8s.io/illumio-kubelink annotated
clusterrolebinding.rbac.authorization.k8s.io/illumio-kubelink labeled
```

Delete C-VEN DaemonSet

The next step is removing the C-VEN DaemonSet. Save any custom labels and validations included in the DaemonSet and reapply them later.

```
kubectl delete daemonset illumio-ven -n illumio-system
```

Install Helm

The last remaining step is installing Helm and the Helm Chart for Illumio Core for Kubernetes. Follow the steps in Deploy with Helm Chart [22]. Filling in the fields in **`illumio-values.yaml`** is still mandatory.

Upgrade and Uninstall Helm Chart Deployments

Deployments of Illumio Core for Kubernetes 3.0.0 or later are performed with Helm Charts. Upgrades and uninstallations are also performed with Helm commands.

Upgrade Helm Chart Deployments

To upgrade an existing installation to a newer version after it had been initially deployed with a Helm Chart, follow standard Helm practice with the **`helm upgrade`** command.

For example, if you install the Helm Chart for Core for Kubernetes 4.2.0 initially with this command:

```
helm install illumio -f values.yaml oci://quay.io/illumio/illumio
--version 4.2.0
--namespace illumio-system
```

Then use the following command to upgrade to version 4.3.0:

```
helm upgrade illumio -f values.yaml oci://quay.io/illumio/illumio
--version 4.3.0
```

Use the same **`values.yaml`** file for the upgrade that was used for the original install command.

> **IMPORTANT**
> Be sure to explicitly specify the version to upgrade to with the `--version <ver#>` option (for example, `--version 4.3.0`), after confirming that the product version you want to install is supported with your PCE version. Verify which PCE versions support the Illumio Core for Kubernetes version you want to deploy at the Kubernetes Operator OS Support and Dependencies page on the Illumio Support Portal.

Uninstall Helm Chart Deployments

To completely uninstall an existing installation that had been initially deployed with a Helm Chart:

```
$ helm uninstall illumio --namespace illumio-system

$ kubectl delete namespace illumio-system
```

The uninstallation process also unpairs the C-VENs from the PCE.

Uninstalling the Helm Chart release takes around two minutes to complete.

Upgrade and Uninstall Non-Helm Chart Deployments

This section describes how deployments that were not installed with Helm can be upgraded or uninstalled.

Upgrade Illumio Components

Illumio Core for Kubernetes and OpenShift is a flexible and modular solution that can be upgraded piece by piece.

For minor upgrades, Kubelink can be upgraded independently from the C-VEN and vice versa unless explicitly mentioned in the release notes.

For major upgrades, including PCE, Kubelink, and C-VEN, Illumio recommends the following process:

- Upgrade the PCE to the new desired version.
- Review the compatibility matrix between PCE, Kubelink, and C-VEN on the Illumio support website.
- Upgrade Kubelink.
- Upgrade C-VEN.

Upgrade Kubelink

The supported process to upgrade Kubelink is as follows:

1. Upload the new image to your private container registry.
2. Change the manifest file to point to the latest Kubelink image in the registry. You do not need to change the previously created secret for Kubelink.
3. Apply this new manifest file to the cluster. `illumio-kubelink` follows the default update behavior of Kubernetes. For more information, see Kubernetes Documentation.

You can verify that the upgrade was successful in the PCE UI on the **Container Clusters > Summary** page and checking for the new Kubelink version.

Upgrade C-VEN

The supported process to upgrade C-VENs is as follows:

1. Upload the new image to your private container registry.
2. Change the manifest file to point to the latest C-VEN image in the registry. You do not need to change the previously created secret for C-VEN.
3. Apply this new manifest file to the cluster. `illumio-ven daemonset` follows the default rolling update behavior of Kubernetes. For more information, see Kubernetes Documentation.

You can verify that the upgrade was successful in the PCE UI on the **Container Clusters > Workloads** page and clicking on any workload and checking for the new C-VEN version.

Uninstall Illumio from Your Cluster

To uninstall the Illumio components, you need to contact Illumio Professional Services to unpair the C-VENs and then delete the Illumio resources from your cluster.

Unpair C-VENs

> **IMPORTANT**
> Contact Illumio Professional Services to unpair the C-VENs in your Kubernetes or OpenShift clusters.

Deleting C-VENs or DaemonSet will not properly unpair them from the PCE and can cause the following issues:

- Workloads will go offline in the PCE UI after 5 minutes (defined by the default Offline Timers configured in the PCE).
- Workloads will be left in the PCE UI as offline with the button to unpair them grayed out (this action is not supported by Illumio).
- Firewall rules configured on the Host and Pods namespaces will remain untouched and active.

The current way to properly delete these workloads created in the PCE UI by C-VENs is by deleting the entire cluster in the PCE UI.

> **IMPORTANT**
> Unpairing an individual C-VEN is not supported. It has to be done at the cluster level (through the DaemonSet), because the cluster is considered as a single entity from a security point of view.

If a node unjoins the cluster for any reason or due to the `kubectl delete node <node_name>` command, the PCE automatically unpairs the C-VEN and deletes the workload and Container workloads associated with the C-VEN that was running on the deleted node.

Delete Illumio Resources

To delete the existing Illumio resources created in your Kubernetes or OpenShift cluster, follow these steps:

Delete C-VEN Resources

1. Contact Illumio Professional Services to unpair the C-VENs and clean up existing iptables rules created by Illumio.
2. Check the Workloads and Container Workloads tabs under **Infrastructure > Container Clusters > YourClusterName** and validate that your nodes and Pods are no longer visible.
3. Delete the resources created during the C-VEN installation by using the following command:

```
kubectl delete -f illumio-ven-kubernetes.yml
kubectl delete -f illumio-ven-secret.yml
```

```
oc delete -f illumio-ven-openshift.yml
oc delete -f illumio-ven-secret.yml
```

Delete Kubelink Resources

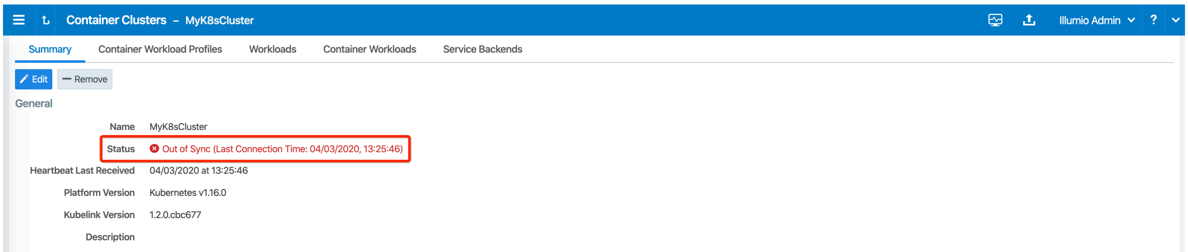
1. Delete the resources created during the Kubelink installation.
2. Delete Kubelink resources from Kubernetes:

```
kubectl delete -f illumio-kubelink-kubernetes.yml
kubectl delete -f illumio-kubelink-secret.yml
```

3. Delete Kubelink resources from OpenShift:

```
oc delete -f illumio-kubelink-openshift.yml
oc delete -f illumio-kubelink-secret.yml
```

4. Check the Summary tab under **Infrastructure > Container Clusters > YourClusterName** and validate that your cluster is "Out of Sync". It takes approximately 10 minutes for the cluster Status to change from "In Sync" to "Out-of-Sync".



5. Finally, delete the container cluster from the PCE UI and verify that there are no resources left in your cluster such as, ConfigMap, Secrets, and others.

Delete Illumio Namespace

- To delete the Illumio namespace in Kubernetes, use the following command:

```
kubectl delete ns illumio-system
```

- To delete the Illumio namespace in OpenShift, use the following command:

```
oc delete project illumio-system
```

Upgrade to CLAS Architecture

A Cluster Local Actor Store (CLAS) mode is introduced into the architecture of Illumio Core for Kubernetes 5.0.0.

> **IMPORTANT**
> To use CLAS, your PCE must be upgraded to Core 23.5.10 or later.

The CLAS architecture brings two major changes to the typical Illumio Core policy model:

- The definition of a workload is fundamentally changed. In a legacy, non-CLAS environment, a container workload is a Pod. In CLAS, a workload is now the Kubernetes Workload resource (such as Deployment, StatefulSet, ReplicaSet, DaemonSet, and so on), which

typically includes multiple Pods that can change in amount during the lifetime of the workload. As such, CLAS workloads are called *Kubernetes Workloads*, to distinguish them from non-CLAS *Container Workloads*.
- ClusterIP services change from Virtual Services to workloads. NodePort and LoadBalancer services remain as Virtual Services in the PCE. The ClusterIP part of a NodePort or LoadBalancer service also exists as a Kubernetes Workload and is linked with the Virtual Service.

Illumio recommends writing a policy using labels. In addition to being impractical, it is not even possible to write a policy for individual Pods. It was (and still is possible) to use Virtual Services in the policy explicitly in rule writing, but Illumio still recommends using labels.

> **IMPORTANT**
> The CLAS architecture is supported only in Illumio Core for Kubernetes versions 5.0.0 and later

Pre-upgrade Policy Check

All policies for your Kubernetes environments must be expressed using labels. In the rare case that policies are using Virtual Service objects, those policies must be changed to label-based policies.

ClusterIP Services as Kubernetes Workloads

ClusterIP services are modeled as workloads in the CLAS environment. If you had a policy written with "Virtual Services Only," that policy will not apply to Kubernetes Workloads (including ClusterIP Services) after the upgrade to CLAS. All rules that apply to ClusterIP Services must be changed to "Use Workloads" before upgrading to CLAS, which needs Destination Services to be specified. This setting also causes ports to be populated from Virtual Services to the rule. So at least one port number must be filled in when writing this rule.

To keep the old functionality of PCE synchronizing the ports of ClusterIP Service, the CLAS now performs this operation. When the rule arrives at Kubelink/CLAS, ports will be replaced by the current ports of the ClusterIP Services. The port replacement includes all ports from the Service. If the Service has two ports, it is not possible to include one and not include the other.

Because Services are now Kubernetes Workloads, the "All Workloads" flag in a rule will include all Services. Do not use "All Workloads" as a Destination in a rule. Use a more specific label instead that targets the Service.

All rules that include a label of at least one ClusterIP service will have specified ports internally replaced. However - this is not reflected in PCE UI, where the rule still displays ports.

Upgrade Strategy

Illumio Core for Kubernetes 5.x is backward-compatible and supports both CLAS and legacy non-CLAS mode operation.

This is controlled by the **clusterMode** parameter specified in the Helm Chart installation yaml file. The default value is **legacy**, meaning that after the upgrade, the software operates in the legacy, non-CLAS mode.

The PCE supporting Illumio Core for Kubernetes 5.0.0 and later (PCE version 23.5.0+A1 and later) also supports both CLAS and non-CLAS modes of operation. Illumio recommends that after the software upgrade, the migration to CLAS is performed one cluster at a time.

The CLAS implementation uses the configuration parameter `clusterMode` set to `clas` or `legacy` to turn on (or off) CLAS mode in the cluster, respectively, when installing. When upgrading an existing non-CLAS cluster to CLAS, set `clusterMode`, to `migrateLegacyToClas`. When reverting (or downgrading) CLAS to non-CLAS, set `clusterMode` to `migrateClasToLegacy`.

Upgrade Steps (on Each Kubernetes Cluster)

Be sure to perform all steps in this procedure on each existing Kubernetes cluster that you want to upgrade to CLAS mode.

1. Prepare the **values.yaml** file with all required parameters. Refer to Deploy with Helm Chart [22].
2. Upgrade Illumio Core for Kubernetes to version 5.1.0 or later. Refer to Upgrade and Uninstall Helm Chart Deployments [88].
3. Verify the upgrade was successful.
4. Perform a pre-upgrade policy check (see Pre-upgrade Policy Check [92] above).
5. Set the "illumio-system" namespace into Visibility Only enforcement mode.
6. Consider setting all cluster nodes into Visibility Only enforcement mode. This step compromises security and will open traffic to/from your protected applications. On the other hand, any policy errors will not result in an application outage.
7. Migrate the cluster to CLAS mode:
   a. Add `clusterMode: migrateLegacyToClas` parameter-value pair to your **values.yaml**.
   b. Perform **helm upgrade** command:

   ```
   helm upgrade <nameofyourhelmdeployment> -n illumio-system -f
   <yourvalues.yaml> oci://quay.io/illumio/illumio --version 5.1.0
   ```

   > **IMPORTANT**
   >
   > Be sure to explicitly specify the version to upgrade to with the `--version <ver#>` option (for example, `--version 5.1.0`), after confirming that the product version you want to upgrade to is supported with your PCE version. Verify which PCE versions support the Illumio Core for Kubernetes version you want to deploy at the Kubernetes Operator OS Support and Dependencies page on the Illumio Support Portal.

8. Refresh the Container Cluster page so that the Kubernetes Workloads tab now appears along with Container Workloads tab
9. Check that all C-VEN pods and the Kubelink pod restarted.
10. This cluster is now running in migration mode, Container Workloads are still present, and new Kubernetes Workloads (CLAS-enabled) are populated.
11. Check if policy sync status of all Kubernetes Workloads and Peer Workloads are "Active." Some Container Workloads might be in Active state, while others in Syncing state -- this

is expected. Check if traffic still works. If something goes wrong, revert the cluster to non-CLAS mode with the following procedure, otherwise go to the next Step:

   a. Specify `clusterMode: migrateClasToLegacy` parameter-value pair in your **`values.yaml`**.
   b. Perform **`helm upgrade`** command:

   ```
   helm upgrade <nameofyourhelmdeployment> -n illumio-system -f
   <yourvalues.yaml> oci://quay.io/illumio/illumio --version 5.1.0
   ```
   c. Refresh the Container Cluster page so the Container Workloads tab appears along with the Kubernetes Workloads tab
   d. Wait until Container Workloads and peers are Active, and traffic is working as expected.
   e. Change the `clusterMode` parameter to `legacy` in **`values.yaml`** -- or delete the variable (because the default parameter value is `legacy`).
   f. Perform the **`helm upgrade`** command:

   ```
   helm upgrade <nameofyourhelmdeployment> -n illumio-system -f
   <yourvalues.yaml> oci://quay.io/illumio/illumio --version 5.1.0
   ```
   g. Verify that Kubernetes Workloads were deleted, that Container Workloads are in a Synced state, and that traffic is working as expected.

12. Set the cluster to CLAS mode:
   a. Change `clusterMode: clas` in **`values.yaml`**.
   b. Perform the **`helm upgrade`** command:

   ```
   helm upgrade <nameofyourhelmdeployment> -n illumio-system -f
   <yourvalues.yaml> oci://quay.io/illumio/illumio --version 5.1.0
   ```
13. Check that the Kubelink Pod restarted.
14. The cluster is now running in the CLAS mode. All Container Workloads from this cluster will no longer be visible on the PCE. Instead, the PCE will display only a list of Kubernetes Workloads (Deployments, etc.).
15. Set all nodes into original enforcement mode if those were previously changed to visibility only.

> **IMPORTANT**
>
> Before using your CLAS cluster, make sure you write mandatory infrastructure rules to enable proper operation. See Rules and Traffic Considerations with CLAS [68] for details on these mandatory rules.

Reference: General

This section lists a few known limitation of this release and how to troubleshoot issues that may occur during the installation process.

For more information see these additional topics.

- Troubleshooting
- Troubleshooting CLAS Mode Architecture
- Known Limitations

- Kubelink Monitoring and Troubleshooting
- Aggregating Logs from Kubelink and C-VEN Pods

Troubleshooting

This section describes how to troubleshoot common issues when installing Illumio on Kubernetes or OpenShift deployments.

Helm deployment (and uninstall) fails with C-VEN stuck in ContainerCreating state

During a deployment with Helm, if C-VEN pods do not start, and instead continually show a status of `ContainerCreating`, check that you have the correct runtime set in your illumio-values.yaml file. If, for example, the `containerRuntime` value is set to containerd but you are now using a Docker runtime (parameter value of docker), then the C-VEN will become stuck in a `ContainerCreating` state. If you later attempt to uninstall, the unpair action for the C-VEN will also become stuck in a `ContainerCreating` state.

Confirm that a C-VEN exhibiting these persistent `ContainerCreating` symptoms is set to the proper `containerRuntime` value in its illumio-values.yaml. Another clue when troubleshooting is to check output of the **`describe`** command for the affected pod:

```
# kubectl -n illumio-system describe pod/<your_pod_name>
```

Check the output under the `Containers` section, and, within that section, under the `Mounts` section, to confirm the pod is attempting to mount to a location appropriate for your container runtime.

```
# kubectl -n illumio-system describe pod/illumio-ven-unpair-cwj2f
Name:         illumio-ven-unpair-cwj2f
Namespace:    illumio-system

[. . .]

Mounts:
    /var/run/containerd/containerd.sock from unixsocks (rw)
```

This problem is also shown under the `Events` section of this output, with a `Warning` event for that mount location due to the mismatched container runtime values.

```
Events:
  Type     Reason       Age                           From                Message
  ----     ------       ----                          ----                -------
[. . .]
  Warning  FailedMount  96s (x11 over 7m48s)  kubelet
MountVolume.SetUp failed for volume “unixsocks” : hostPath type check
failed: /var/run/containerd/containerd.sock is not a socket file
```

Failed Authentication with the Container Registry

In some cases, your Pods are in `ImagePullBackOff` state after the deployment:

```
$ kubectl -n kube-system get Pods
NAME                                READY   STATUS             RESTARTS   AGE
coredns-58687784f9-h4pp2            1/1     Running            8
175d
coredns-58687784f9-znn9j            1/1     Running            9
175d
dns-autoscaler-79599df498-m55mg     1/1     Running            9
175d
illumio-kubelink-87fd8d9f6-nmh25    0/1     ImagePullBackOff   0          28s
```

In this case, check the description of your Pods using the following command:

```
$ kubectl -n kube-system describe Pods illumio-kubelink-87fd8d9f6-nmh25
Name:           illumio-kubelink-87fd8d9f6-nmh25
Namespace:      kube-system
Priority:       0
Node:           node2/10.0.0.12
Start Time:     Fri, 03 Apr 2020 21:05:07 +0000
Labels:         app=illumio-kubelink
                Pod-template-hash=87fd8d9f6
Annotations:    com.illumio.role: Kubelink
Status:         Pending
IP:             10.10.65.55
IPs:
  IP:             10.10.65.55
Controlled By:  ReplicaSet/illumio-kubelink-87fd8d9f6
Containers:
  illumio-kubelink:
    Container ID:
    Image:          registry.poc.segmentationpov.com/illumio-
kubelink:2.0.x.xxxxxx
    Image ID:
    Port:           <none>
    Host Port:      <none>
    State:          Waiting
      Reason:       ImagePullBackOff
    Ready:          False
    Restart Count:  0
    Environment:
      ILO_SERVER:         <set to the key 'ilo_server' in secret 'illumio-
kubelink-config'>         Optional: false
      ILO_CLUSTER_UUID:   <set to the key 'ilo_cluster_uuid' in secret
'illumio-kubelink-config'>   Optional: false
      ILO_CLUSTER_TOKEN:  <set to the key 'ilo_cluster_token' in secret
'illumio-kubelink-config'>  Optional: false
      CLUSTER_TYPE:       Kubernetes
      IGNORE_CERT:        <set to the key 'ignore_cert' in secret 'illumio-
kubelink-config'>  Optional: true
      DEBUG_LEVEL:        <set to the key 'log_level' in secret 'illumio-
kubelink-config'>    Optional: true
    Mounts:
      /etc/pki/tls/ilo_certs/ from root-ca (rw)
      /var/run/secrets/kubernetes.io/serviceaccount from illumio-kubelink-
token-7mvgk (ro)
Conditions:
```

```
  Type              Status
  Initialized       True
  Ready             False
  ContainersReady   False
  PodScheduled      True
Volumes:
  root-ca:
    Type:      ConfigMap (a volume populated by a ConfigMap)
    Name:      root-ca-config
    Optional:  false
  illumio-kubelink-token-7mvgk:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  illumio-kubelink-token-7mvgk
    Optional:    false
QoS Class:       BestEffort
Node-Selectors:  <none>
Tolerations:     node-role.kubernetes.io/master:NoSchedule
                 node.kubernetes.io/not-ready:NoExecute for 300s
                 node.kubernetes.io/unreachable:NoExecute for 300s
Events:
  Type     Reason          Age                  From               Message
  ----     ------          ----                 ----               -------
  Normal   Scheduled       <unknown>            default-scheduler
Successfully assigned kube-system/illumio-kubelink-87fd8d9f6-nmh25 to node2
  Normal   SandboxChanged  45s                  kubelet, node2     Pod
sandbox changed, it will be killed and re-created.
  Normal   BackOff         14s (x4 over 45s)  kubelet,
node2     Back-off pulling image "registry.poc.segmentationpov.com/illumio-
kubelink:2.0.x.xxxxxx"
  Warning  Failed          14s (x4 over 45s)  kubelet, node2     Error:
ImagePullBackOff
  Normal   Pulling         1s (x3 over 46s)   kubelet, node2     Pulling
image "registry.poc.segmentationpov.com/illumio-kubelink:2.0.x.xxxxxx"
  Warning  Failed          1s (x3 over 46s)   kubelet,
node2     Failed to pull image "registry.poc.segmentationpov.com/illumio-
kubelink:2.0.x.xxxxxx": rpc error: code = Unknown desc = Error response
from daemon: unauthorized: authentication required
  Warning  Failed          1s (x3 over 46s)   kubelet, node2     Error:
ErrImagePull
```

The messages at the end of the output above are self-explanatory that there is a problem with the authentication against the container registry. Verify the credentials you entered in the secret for your private container registry and reapply it after fixing the issue.

Kubelink Pod in CrashLoopBackOff State

In some cases, your Kubelink Pod is in `CrashLoobBackOff` state after the deployment:

```
$ kubectl -n kube-system get Pods
NAME                                    READY   STATUS             RESTARTS
AGE
coredns-58687784f9-h4pp2                1/1     Running            8
174d
coredns-58687784f9-znn9j                1/1     Running            9
174d
```

```
dns-autoscaler-79599df498-m55mg     1/1     Running           9
174d
illumio-kubelink-8648c6fb68-mdh8p   0/1     CrashLoopBackOff   1
16s
```

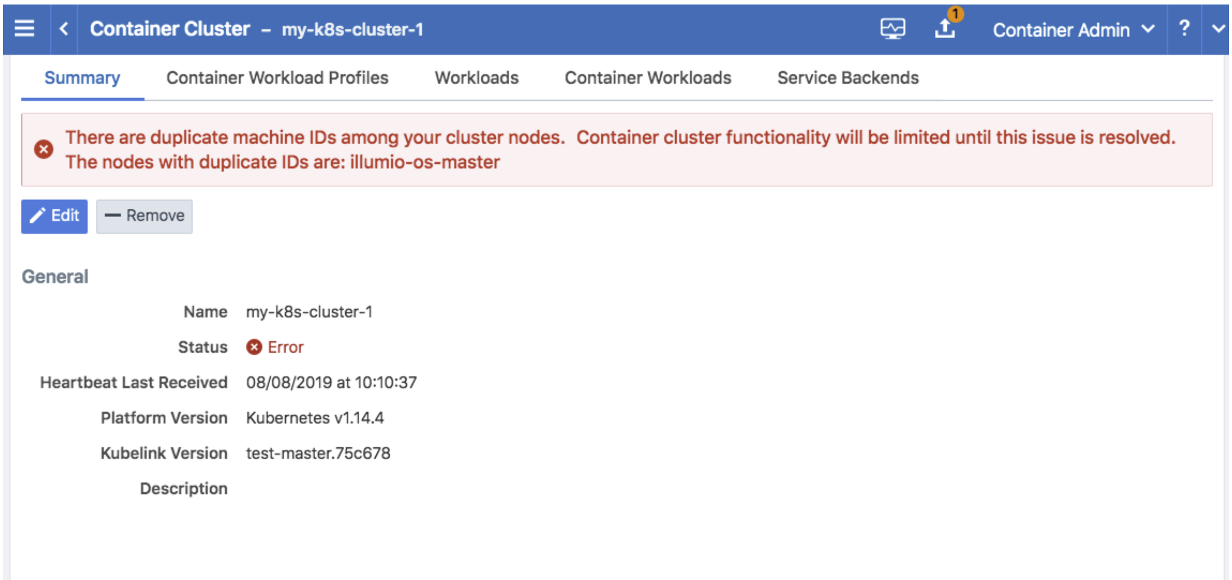
In this case, check the logs of your Pods using the following command:

```
$ kubectl -n kube-system logs illumio-kubelink-8648c6fb68-mdh8p
I, [2020-04-03T01:46:33.587761 #19]  INFO -- : Starting Kubelink for PCE
https://mypce.example.com:8443
I, [2020-04-03T01:46:33.587915 #19]  INFO -- : Found 1 custom certs
I, [2020-04-03T01:46:33.594212 #19]  INFO -- : Installed custom certs to
/etc/pki/tls/certs/ca-bundle.crt
I, [2020-04-03T01:46:33.619976 #19]  INFO -- : Connecting to PCE
https://mypce.example.com:8443
E, [2020-04-03T01:46:33.651410 #19] ERROR -- : Received a non retriable
error
401
/illumio/kubelink.rb:163:in `update_pce_resource': HTTP status code 401
uri:
https://mypce.example.com:8443/api/v2/orgs/10/container_clusters/
42083a4d-dd92-49e6-b495-6f84a940073c/put_from_cluster, request_id:
21bdfc05-7b02-442d-a778-e6f2da2a462b response:  request_body:
{"kubelink_version":"2.0.x.xxxxxx","errors":[],"manager_type":"Kubernetes
v1.16.0"} (Illumio::PCEHttpException)
    from /illumio/kubelink.rb:113:in `initialize'
    from /illumio/main.rb:39:in `new'
    from /illumio/main.rb:39:in `block in main'
    from /external/lib/ruby/gems/2.4.0/gems/em-synchrony-1.0.6/
    lib/em-synchrony.rb:39:in `block (2 levels) in synchrony'
```

In the example above, the request is rejected by the PCE because of a wrong identifier. Open your secret file for Kubelink, verify your cluster UUID and token, and make sure you copy-pasted the same string provided by the PCE during cluster creation.

Container Cluster in Error

In some cases, the container cluster page displays an error indicating that duplicate machine IDs were detected and functionality will be limited. See the screenshot below.



To resolve this error, follow the steps in the section below. After following those steps, restart the C-VEN Pod on each of the affected Kubernetes cluster node.

Verify Machine IDs on All Nodes

To verify machine-ids and resolve any duplicate IDs across nodes:

1. Check the machineID of all your cluster nodes with the following command:

   ```
   kubectl get node -o yaml | grep machineID
   ```

   ```
   $ kubectl get node -o yaml | grep machineID
         machineID: ec2eefcfc1bdfa9d38218812405a27d9
         machineID: ec2bcf3d167630bc587132ee83c9a7ad
         machineID: ec2bf11109b243671147b53abe1fcfc0
   ```
2. As an alternative, you can also to check content of the `/etc/machine-id` file on all cluster nodes. The output should be a single newline-terminated, hexadecimal, 32-character, and lowercase ID.
3. If the machine-id string is unique for each node, then the environment is OK. If the machine-id is duplicated across any of the nodes, then you must generate a machine-id for each node which has the same machine-id.
4. Running the following command displays the output of the machine-id:

   ```
   cat /etc/machine-id
   ```

   ```
   root@k8s-c2-node1:~# cat /etc/machine-id
   2581d13362cd4220b20020ff728efff8
   ```

Generate a New Machine ID

If the machineID is duplicated on some or all of the Kubernetes nodes, use the following steps to generate a new machine-id.

- For CentOS or Red Hat:

```
rm -rf /etc/machine-id; systemd-machine-id-setup;
systemctl restart kubelet
```

- For Ubuntu:

```
rm -rf /etc/machine-id; rm /var/lib/dbus/machine-id; systemd-machine-id-setup;
systemctl restart kubelet
```

> **NOTE**
> Check the machine-id again after doing the above steps to verify that each Kubernetes cluster node has a unique machine-id.

Pods and Services Not Detected

In some cases, the Container Workloads page under **Infrastructure > Container Clusters > MyClusterName** is empty although the Workloads page has all the cluster nodes in it. This issue typically occurs when the wrong container runtime is monitored by Illumio. To resolve this issue:

1. Validate which container runtime is used in your Kubernetes or OpenShift cluster.
2. Open your configuration file for the C-VEN DaemonSet.
3. Modify the `unixsocks` mount configuration to point to the right socket path on your hosts.

> **NOTE**
> This issue typically occurs when containerd or cri-o is the primary container runtime on Kubernetes or OpenShift nodes and there is an existing docker container runtime on the nodes that is not "active" (the socket still present on the nodes and process still running, mostly some leftover from the staging phase of the servers).

Pods Stuck in Terminating State

In a Kubernetes cluster running containerd 1.2.6-10 as the container runtime, on deleting a Pod while the C-VEN is deployed may result in the Pod being stuck in a terminating state. If you see this error, redeploy the C-VEN and modify the socket path as follows:

Change the `volumeMount` and `hostPath` from `/var/run` to `/var/run/containerd` in the illumio-ven.yaml file

Enable Firewall Coexistence

> **NOTE**
> If Kubelink was deployed on the Kubernetes cluster and is "In Sync" with the PCE **prior to the VEN installation**, the manual configuration of firewall coexistence **is not required**.

The Illumio C-VEN configures iptables on each host. By default, Illumio Core coexistence mode is set to **Exclusive** meaning the C-VEN will take full control of iptables and flush any rules or chains which are not created by Illumio. In containerized environments, this may affect communications to/from container components (Docker, Kubernetes, and Illumio Kubelink). Therefore, Illumio Core must allow firewall coexistence in order to achieve non-disruptive installation and deployment.

In order to allow firewall coexistence, you must set a scope of Illumio labels in the firewall coexistence configuration. Once you provision a firewall coexistence scope, the PCE will enable firewall coexistence configuration on C-VENs whose labels fall within the scope.

> **NOTE**
> Labels assigned to Kubernetes cluster nodes must fall within the firewall coexistence scope. This is not a requirement for the labels assigned to container workloads.

To manually configure firewall coexistence:

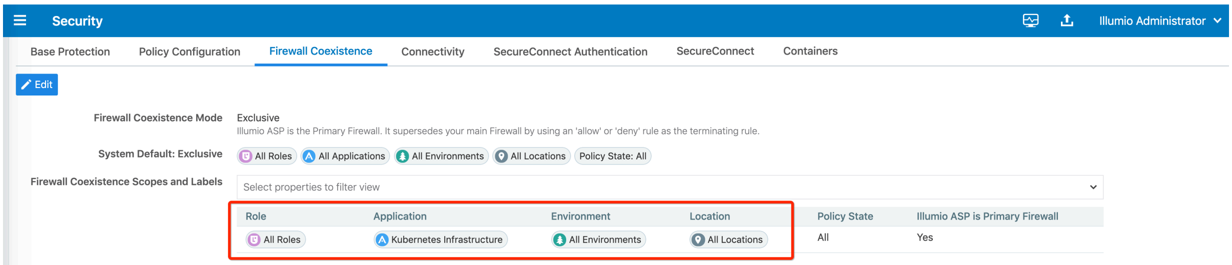
1. Log in to the PCE UI and navigate to **Settings** > **Security**.
2. On the Security page, navigate to the **Manage Firewall Coexistence** tab.
3. Select **Edit**.
4. In the edit wizard, click **Add**. The **Add Firewall Coexistence Labels and Policy State** wizard will pop-up.
5. Select a scope of Illumio labels. The scope must include the labels you intend to use for your Kubernetes cluster nodes.
   a. Select **All** for Policy State.
   b. Illumio Core is Primary Firewall - Select your preference.
      i. **Yes** = (Recommended) Illumio iptable chains will be at the top of iptables at all times. Non-Illumio iptable chains can coexist, but will follow after Illumio chains.
      ii. **No** = (Not Recommended) Non-Illumio iptable chains may coexist and can be placed before Illumio chains.
   c. Click **OK**.
6. Click **Save**.
7. Provision the changes.

> **IMPORTANT**
> Be sure to provision the saved changes or else firewall coexistence will not take effect.

Below is an example of a Firewall Coexistence scope for an Kubernetes cluster which has the following labels:

- Role: Master OR Worker
- Application: Kubernetes Infrastructure
- Environment: Development
- Location: Data Center 1

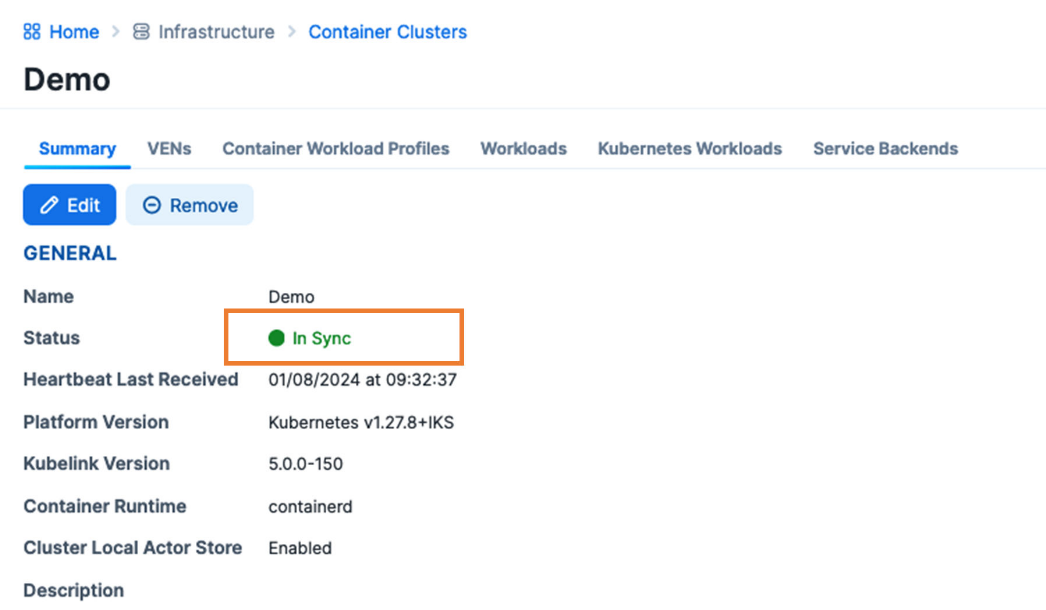
The firewall coexistence scope in the example uses the 'All Roles', 'All Environments', 'All Locations' objects to cover future Kubernetes clusters.

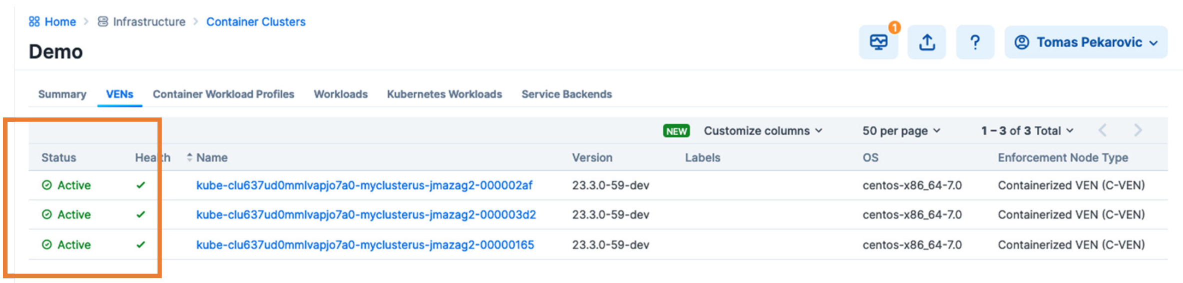


Troubleshooting CLAS Mode Architecture

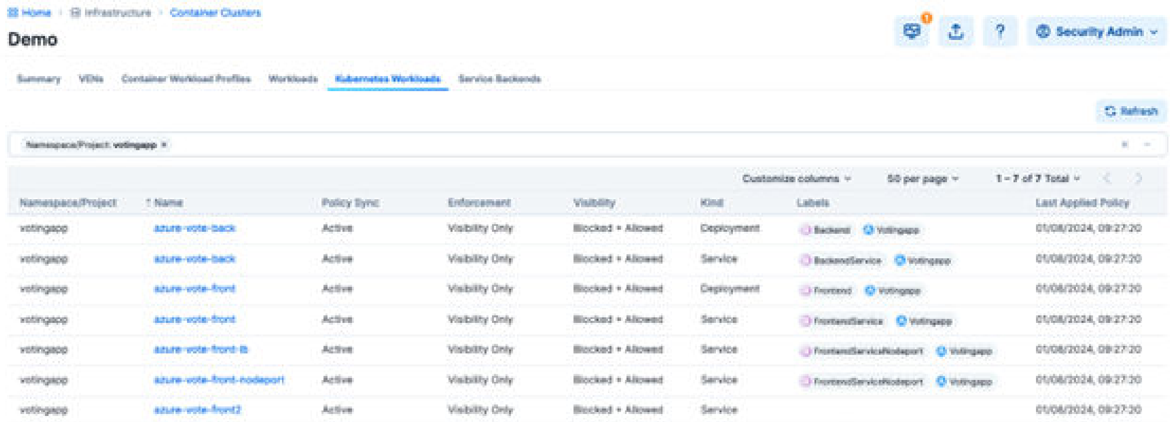
If your upgrade or installation of CLAS-enabled clusters exhibits unusual behavior, follow these steps to troubleshoot the issue:

1. Check that the Kubelinks and C-VENs are operating. Verify at the Container Clusters page, in the Summary and VENs tabs.

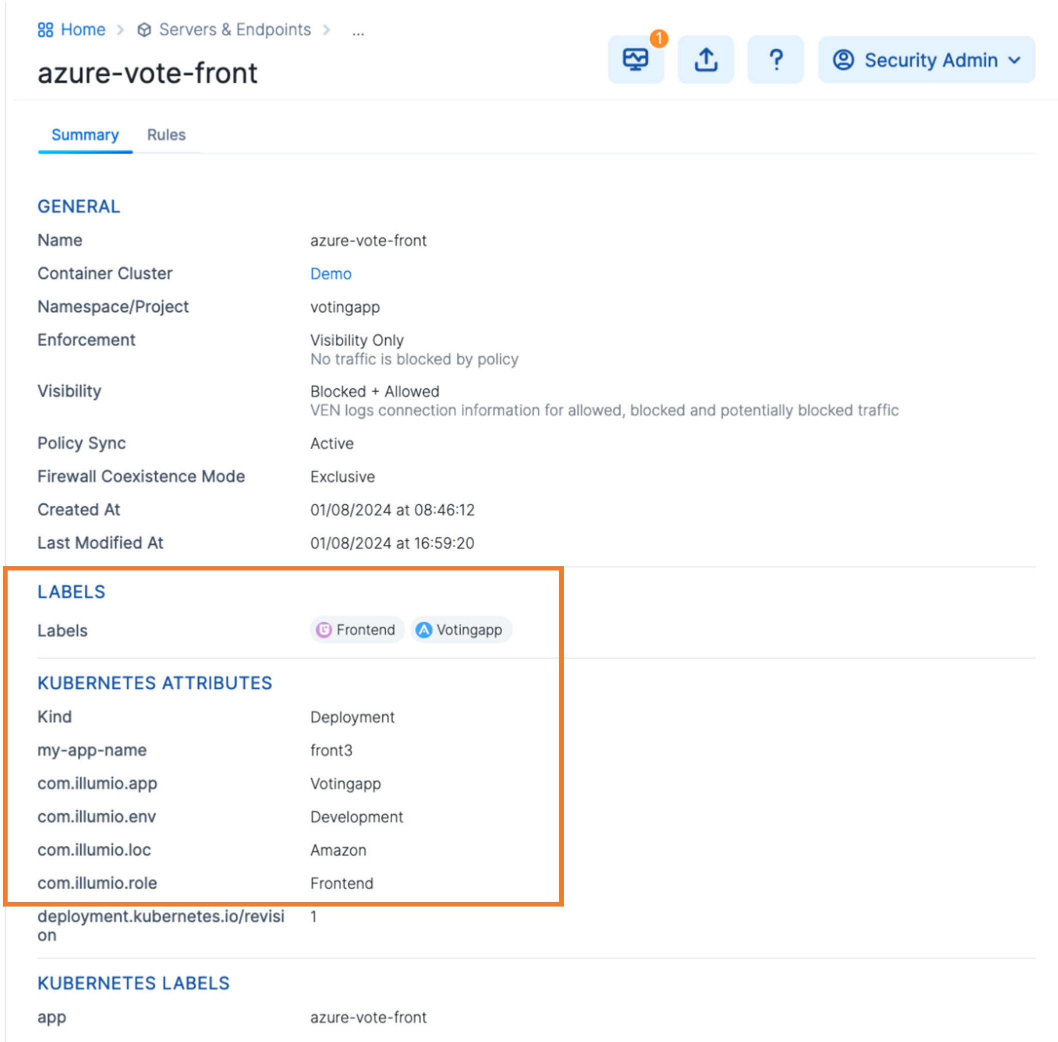




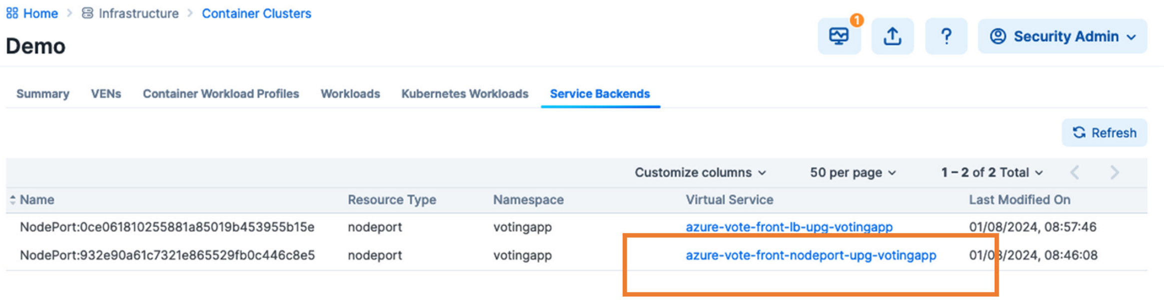
2. Check that Kubernetes Namespace and Kubernetes Workload objects are created, and are present in the PCE. Verify from the Kubernetes Workloads tab.

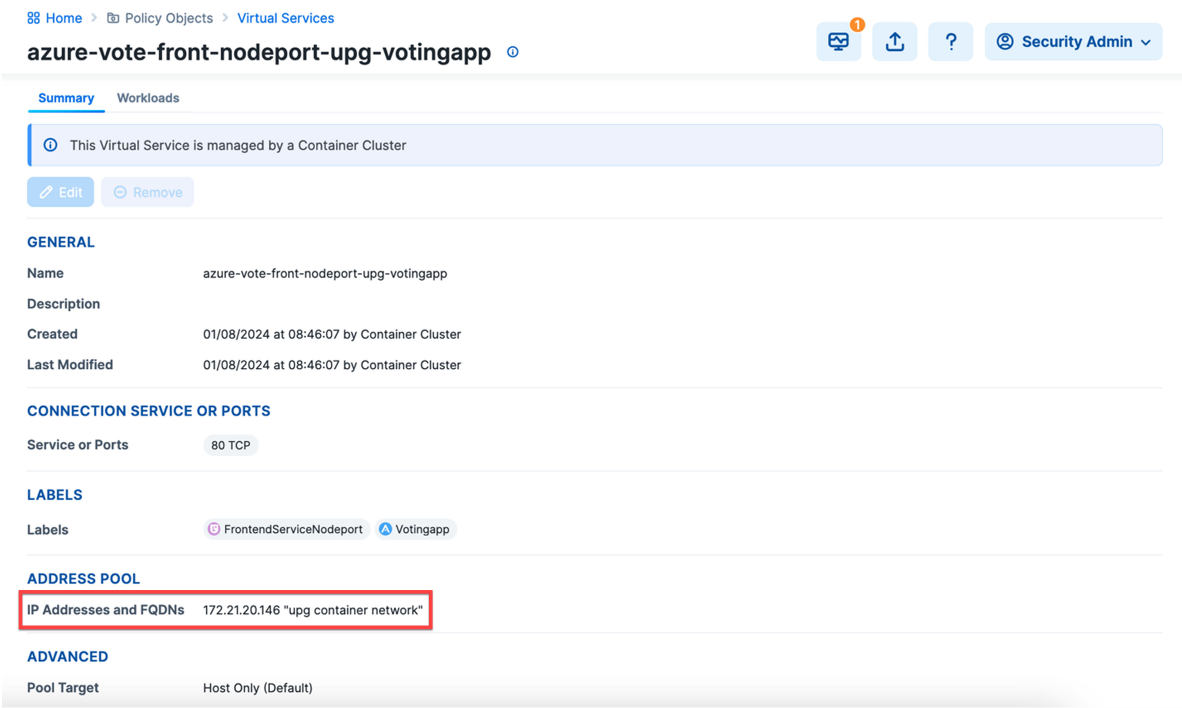


3. Check that labeling is done correctly by examining each Kubernetes Workload:

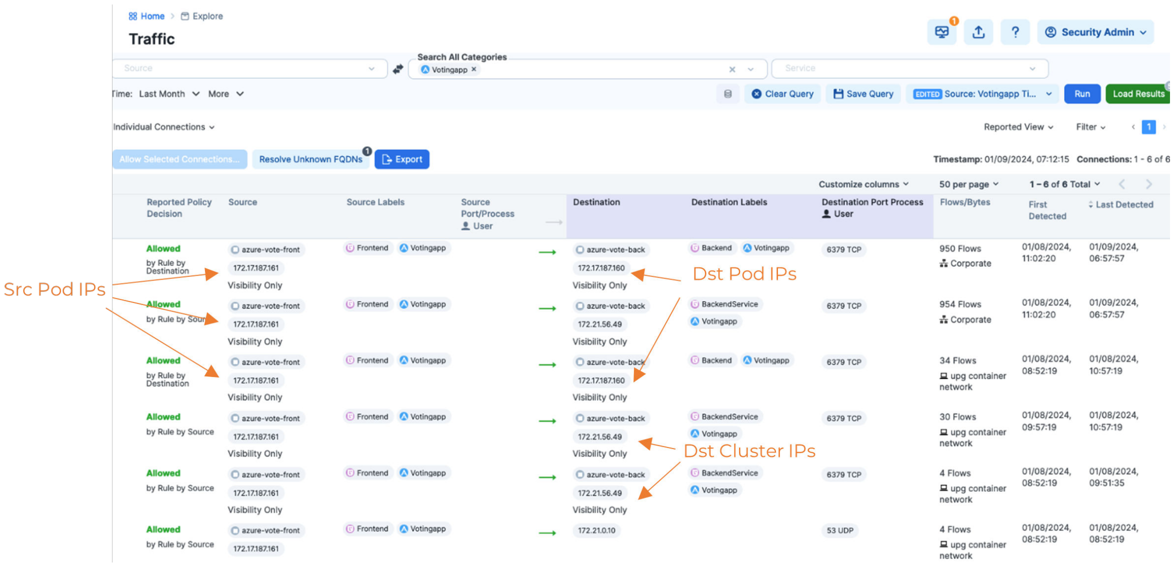


4. Check that NodePort Services have correct IPs. Find NodePort IPs on the Service Backends tab, under the Virtual Services table column.





5. Check that traffic is flowing properly. Find the pod-to-pod, and pod-to-application flows with IP information from the Traffic view. Select "Individual Connections" to see the name of the Kubernetes Workloads and the IPs of the Pods sending and receiving the traffic.



Known Limitations

The known limitations in this release are:

- Kube-proxy mode set to IPVS is currently not supported.
- If a C-VEN on a server hosting containers is paired directly into the Enforced policy state, other nodes may lose connectivity with the master node until policy is synchronized across all the nodes.

- Pods which run on the host network stack (inherit the host IP address) are not reported to the PCE. Any rules written for the host will also be inherited by any *hostNetworked Pods* on the host.
- If you are using an external load balancer, the policy configuration will be dependent on the type of the load balancer used.
- Kubernetes uses NAT tables, which depend on traffic being tracked and stateful. Therefore, it is not recommended to use stateless rules.
- If a Kubernetes service has both port 1234/TCP and port 2345/UDP configured, a rule configured with the Pod as Consumer and the virtual service as Provider will open up both ports 1234/TCP and 2345/TCP, and 1234/UDP and 2345/UDP on the Pod's firewall (outbound rule).

  In case of a Kubernetes service configured with a `port` and `targetPort` statement in the manifest file as shown in the example below:

  ```
  apiVersion: v1
  kind: Service
  metadata:
    name: web-frontend-svc
    namespace: app1
    labels:
      app: app1
      tier: web-frontend
    annotations:
      com.illumio.role: Web
  spec:
    type: ClusterIP
    ports:
    - port: 8080
      targetPort: 80
      protocol: TCP
    - port: 8081
      targetPort: 81
      protocol: UDP
    selector:
      app: app1
      tier: web-frontend
  ```

  This configuration is supported with Illumio Core. In this case, only the port number associated to the `port` statement will show this issue, the port number associated to the `targetPort` statement will not show this issue and will use the `protocol` specified in the Service yaml file.

Kubelink Monitoring and Troubleshooting

If you deployed Illumio Core for Kubernetes 3.0.0 or later, Kubelink is deployed as part of the overall Helm Chart deployment, as described in Deployment with Helm Chart (Core for Kubernetes 3.0.0 and Higher) [10].

Kubelink Process

Kubelink uses a single Ruby process which runs as: **`ruby /illumio/init.rb`**.

Kubelink Startup Log Messages

After deploying Kubelink (whether by Helm Chart or manually), verify your deployment with the **`kubectl get pods -n illumio-system`** command. The **`kubelinkpod`** should be shown with the Running status. In addition, you can review the log file entries after the deployment with the **kubectl logs** command pointing to the Kubelink pod name.

```
kubectl logs <kubelink_pod_name> -n illumio-system
```

A typical successful Kubelink deployment produces log entries similar to these:

```
I, [2022-05-23T14:36:53.847248 #10]  INFO -- : Starting Kubelink for PCE
https://192.168.88.127:10443
I, [2022-05-23T14:36:53.847502 #10]  INFO -- : Metrics reporting enabled;
reporting window 30
I, [2022-05-23T14:36:53.847520 #10]  INFO -- : PCE fqdn https://
192.168.88.127:10443
[WARNING; em-http-request] TLS hostname validation is disabled (use 'tls:
{verify_peer: true}'), see CVE-2020-13482 and https://github.com/igrigorik/
em-http-request/issues/339 for details
I, [2022-05-23T14:36:53.893048 #10]  INFO -- : Successfully connected to PCE
I, [2022-05-23T14:36:53.893170 #10]  INFO -- : begin sync on resource
namespaces
[WARNING; em-http-request] TLS hostname validation is disabled (use 'tls:
{verify_peer: true}'), see CVE-2020-13482 and https://github.com/igrigorik/
em-http-request/issues/339 for details
I, [2022-05-23T14:36:53.904369 #10]  INFO -- : Synchronized 6 namespaces.
I, [2022-05-23T14:36:53.904424 #10]  INFO -- : sync on resource namespaces
successful, setting up resource version to 184232
I, [2022-05-23T14:36:53.904522 #10]  INFO -- : Start watch on namespaces
with version 184232
I, [2022-05-23T14:36:53.905678 #10]  INFO -- : begin sync on resource nodes
[WARNING; em-http-request] TLS hostname validation is disabled (use 'tls:
{verify_peer: true}'), see CVE-2020-13482 and https://github.com/igrigorik/
em-http-request/issues/339 for details
[WARNING; em-http-request] TLS hostname validation is disabled (use 'tls:
{verify_peer: true}'), see CVE-2020-13482 and https://github.com/igrigorik/
em-http-request/issues/339 for details
I, [2022-05-23T14:36:53.918093 #10]  INFO -- : Synchronized 1 nodes.
I, [2022-05-23T14:36:53.918143 #10]  INFO -- : sync on resource nodes
successful, setting up resource version to 184232
I, [2022-05-23T14:36:53.918175 #10]  INFO -- : Start watch on nodes with
version 184232
I, [2022-05-23T14:36:53.919265 #10]  INFO -- : begin sync on resource pods
I, [2022-05-23T14:36:53.935536 #10]  INFO -- : sync on resource pods
successful, setting up resource version to 184232
I, [2022-05-23T14:36:53.935601 #10]  INFO -- : Start watch on pods with
version 184232
I, [2022-05-23T14:36:53.936938 #10]  INFO -- : begin sync on resource
services
[WARNING; em-http-request] TLS hostname validation is disabled (use 'tls:
{verify_peer: true}'), see CVE-2020-13482 and https://github.com/igrigorik/
em-http-request/issues/339 for details
[WARNING; em-http-request] TLS hostname validation is disabled (use 'tls:
{verify_peer: true}'), see CVE-2020-13482 and https://github.com/igrigorik/
em-http-request/issues/339 for details
[WARNING; em-http-request] TLS hostname validation is disabled (use 'tls:
{verify_peer: true}'), see CVE-2020-13482 and https://github.com/igrigorik/
em-http-request/issues/339 for details
```

```
I, [2022-05-23T14:36:54.029965 #10]  INFO -- : Synchronized 3 services,
full=true, force=false
I, [2022-05-23T14:36:54.030013 #10]  INFO -- : sync on resource services
successful, setting up resource version to 184232
I, [2022-05-23T14:36:54.030046 #10]  INFO -- : Start watch on services with
version 184232
I, [2022-05-23T14:36:54.031042 #10]  INFO -- : begin sync on resource
replica_sets
I, [2022-05-23T14:36:54.100090 #10]  INFO -- : Nothing to sync
I, [2022-05-23T14:36:54.100237 #10]  INFO -- : sync on resource
replica_sets successful, setting up resource version to 184232
I, [2022-05-23T14:36:54.100281 #10]  INFO -- : Start watch on replica_sets
with version 184232
I, [2022-05-23T14:36:54.101226 #10]  INFO -- : begin sync on resource
stateful_sets
I, [2022-05-23T14:36:54.170175 #10]  INFO -- : Nothing to sync
I, [2022-05-23T14:36:54.170220 #10]  INFO -- : sync on resource
stateful_sets successful, setting up resource version to 184232
I, [2022-05-23T14:36:54.170267 #10]  INFO -- : Start watch on stateful_sets
with version 184232
I, [2022-05-23T14:36:54.171159 #10]  INFO -- : begin sync on resource
daemon_sets
I, [2022-05-23T14:36:54.245866 #10]  INFO -- : Nothing to sync
I, [2022-05-23T14:36:54.246025 #10]  INFO -- : sync on resource daemon_sets
successful, setting up resource version to 184232
I, [2022-05-23T14:36:54.246210 #10]  INFO -- : Start watch on daemon_sets
with version 184232
I, [2022-05-23T14:36:54.247946 #10]  INFO -- : begin sync on resource
replication_controllers
I, [2022-05-23T14:36:54.324925 #10]  INFO -- : Nothing to sync
I, [2022-05-23T14:36:54.324977 #10]  INFO -- : sync on resource
replication_controllers successful, setting up resource version to 184232
I, [2022-05-23T14:36:54.325032 #10]  INFO -- : Start watch on
replication_controllers with version 184232
[WARNING; em-http-request] TLS hostname validation is disabled (use 'tls:
{verify_peer: true}'), see CVE-2020-13482 and https://github.com/igrigorik/
em-http-request/issues/339 for details
[WARNING; em-http-request] TLS hostname validation is disabled (use 'tls:
{verify_peer: true}'), see CVE-2020-13482 and https://github.com/igrigorik/
em-http-request/issues/339 for details
[WARNING; em-http-request] TLS hostname validation is disabled (use 'tls:
{verify_peer: true}'), see CVE-2020-13482 and https://github.com/igrigorik/
em-http-request/issues/339 for details
[WARNING; em-http-request] TLS hostname validation is disabled (use 'tls:
{verify_peer: true}'), see CVE-2020-13482 and https://github.com/igrigorik/
em-http-request/issues/339 for details
[WARNING; em-http-request] TLS hostname validation is disabled (use 'tls:
{verify_peer: true}'), see CVE-2020-13482 and https://github.com/igrigorik/
em-http-request/issues/339 for details
I, [2022-05-23T14:36:54.505403 #10]  INFO -- : replica_sets MODIFIED
I, [2022-05-23T14:37:24.312086 #10]  INFO -- : Heart beating to PCE
I, [2022-05-23T14:37:24.312191 #10]  INFO -- : Attaching metrics report
to heartbeat: {:pod_changes=>[{:namespace=>"illumio-system", "added"=>0,
"modified"=>0, "deleted"=>1}], :service_changes=>[], :duration_seconds=>30}
[WARNING; em-http-request] TLS hostname validation is disabled (use 'tls:
```

```
{verify_peer: true}'), see CVE-2020-13482 and https://github.com/igrigorik/
em-http-request/issues/339 for details
I, [2022-05-23T14:37:54.343467 #10]  INFO -- : Heart beating to PCE
I, [2022-05-23T14:37:54.343874 #10]  INFO -- : Attaching metrics report to
heartbeat: {:pod_changes=>[], :service_changes=>[], :duration_seconds=>30}
[WARNING; em-http-request] TLS hostname validation is disabled (use 'tls:
{verify_peer: true}'), see CVE-2020-13482 and https://github.com/igrigorik/
em-http-request/issues/339 for details
I, [2022-05-23T14:38:24.373847 #10]  INFO -- : Heart beating to PCE
I, [2022-05-23T14:38:24.373924 #10]  INFO -- : Attaching metrics report to
heartbeat: {:pod_changes=>[], :service_changes=>[], :duration_seconds=>30}
[WARNING; em-http-request] TLS hostname validation is disabled (use 'tls:
{verify_peer: true}'), see CVE-2020-13482 and https://github.com/igrigorik/
em-http-request/issues/339 for details
I, [2022-05-23T14:38:54.380933 #10]  INFO -- : Heart beating to PCE
I, [2022-05-23T14:38:54.381009 #10]  INFO -- : Attaching metrics report to
heartbeat: {:pod_changes=>[], :service_changes=>[], :duration_seconds=>30}
[WARNING; em-http-request] TLS hostname validation is disabled (use 'tls:
{verify_peer: true}'), see CVE-2020-13482 and https://github.com/igrigorik/
em-http-request/issues/339 for details
I, [2022-05-23T14:39:24.401636 #10]  INFO -- : Heart beating to PCE
I, [2022-05-23T14:39:24.401748 #10]  INFO -- : Attaching metrics report to
heartbeat: {:pod_changes=>[], :service_changes=>[], :duration_seconds=>30}
[WARNING; em-http-request] TLS hostname validation is disabled (use 'tls:
{verify_peer: true}'), see CVE-2020-13482 and https://github.com/igrigorik/
em-http-request/issues/339 for details
I, [2022-05-23T14:39:54.422494 #10]  INFO -- : Heart beating to PCE
I, [2022-05-23T14:39:54.422595 #10]  INFO -- : Attaching metrics report to
heartbeat: {:pod_changes=>[], :service_changes=>[], :duration_seconds=>30}
[WARNING; em-http-request] TLS hostname validation is disabled (use 'tls:
{verify_peer: true}'), see CVE-2020-13482 and https://github.com/igrigorik/
em-http-request/issues/339 for details
I, [2022-05-23T14:40:24.453077 #10]  INFO -- : Heart beating to PCE
I, [2022-05-23T14:40:24.453217 #10]  INFO -- : Attaching metrics report to
heartbeat: {:pod_changes=>[], :service_changes=>[], :duration_seconds=>30}
[WARNING; em-http-request] TLS hostname validation is disabled (use 'tls:
{verify_peer: true}'), see CVE-2020-13482 and https://github.com/igrigorik/
em-http-request/issues/339 for details
I, [2022-05-23T14:40:54.466210 #10]  INFO -- : Heart beating to PCE
I, [2022-05-23T14:40:54.466455 #10]  INFO -- : Attaching metrics report to
heartbeat: {:pod_changes=>[], :service_changes=>[], :duration_seconds=>30}
[WARNING; em-http-request] TLS hostname validation is disabled (use 'tls:
{verify_peer: true}'), see CVE-2020-13482 and https://github.com/igrigorik/
em-http-request/issues/339 for details
I, [2022-05-23T14:41:24.296410 #10]  INFO -- : Verify watches
for ["namespaces", "nodes", "pods", "services", "replica_sets",
"stateful_sets", "daemon_sets", "replication_controllers"]
I, [2022-05-23T14:41:24.296468 #10]  INFO -- : Watch client namespaces
Connection Idle: 270.3355407714844s
I, [2022-05-23T14:41:24.296485 #10]  INFO -- : Watch client nodes
Connection Idle: 179.93679809570312s
I, [2022-05-23T14:41:24.296499 #10]  INFO -- : Watch client pods Connection
Idle: 240.5237274169922s
I, [2022-05-23T14:41:24.296513 #10]  INFO -- : Watch client services
Connection Idle: 270.0260314941406s
```

```
I, [2022-05-23T14:41:24.296526 #10]  INFO -- : Watch client replica_sets
Connection Idle: 269.85888671875s
I, [2022-05-23T14:41:24.296542 #10]  INFO -- : Watch client stateful_sets
Connection Idle: 270.0269775390625s
I, [2022-05-23T14:41:24.296573 #10]  INFO -- : Watch client daemon_sets
Connection Idle: 270.02490234375s
I, [2022-05-23T14:41:24.296731 #10]  INFO -- : Watch client
replication_controllers Connection Idle: 270.02490234375s
[WARNING; em-http-request] TLS hostname validation is disabled (use 'tls:
{verify_peer: true}'), see CVE-2020-13482 and https://github.com/igrigorik/
em-http-request/issues/339 for details
I, [2022-05-23T14:41:24.300532 #10]  INFO -- : Synchronized 3 services,
full=true, force=true
I, [2022-05-23T14:41:24.452846 #10]  INFO -- : Heart beating to PCE
[WARNING; em-http-request] TLS hostname validation is disabled (use 'tls:
{verify_peer: true}'), see CVE-2020-13482 and https://github.com/igrigorik/
em-http-request/issues/339 for details
W, [2022-05-23T14:41:54.186807 #10]  WARN -- : watch client for
stateful_sets error callback invoked. Resetting watch ...
W, [2022-05-23T14:41:54.186863 #10]  WARN -- : Watch on stateful_sets
ended. Resetting it after 3 seconds
I, [2022-05-23T14:41:54.441880 #10]  INFO -- : Heart beating to PCE
I, [2022-05-23T14:41:54.441991 #10]  INFO -- : Attaching metrics report to
heartbeat: {:pod_changes=>[], :service_changes=>[], :duration_seconds=>60}
[WARNING; em-http-request] TLS hostname validation is disabled (use 'tls:
{verify_peer: true}'), see CVE-2020-13482 and https://github.com/igrigorik/
em-http-request/issues/339 for details
I, [2022-05-23T14:41:57.193339 #10]  INFO -- : begin sync on resource
stateful_sets
I, [2022-05-23T14:41:57.267375 #10]  INFO -- : Nothing to sync
I, [2022-05-23T14:41:57.267411 #10]  INFO -- : sync on resource
stateful_sets successful, setting up resource version to 184451
I, [2022-05-23T14:41:57.267424 #10]  INFO -- : Start watch on stateful_sets
with version 184451
[WARNING; em-http-request] TLS hostname validation is disabled (use 'tls:
{verify_peer: true}'), see CVE-2020-13482 and https://github.com/igrigorik/
em-http-request/issues/339 for details
I, [2022-05-23T14:42:24.483142 #10]  INFO -- : Heart beating to PCE
I, [2022-05-23T14:42:24.483224 #10]  INFO -- : Attaching metrics report to
heartbeat: {:pod_changes=>[], :service_changes=>[], :duration_seconds=>30}
[WARNING; em-http-request] TLS hostname validation is disabled (use
'tls: {verify_peer: true}'), see CVE-2020-13482 and https://github.com/
igrigorik/em-http-request/issues/339 for details
```

Verify Kubelink Deployment

To verify your Kubelink deployment.

- To check the Kubelink Pod status for Kubernetes:

```
kubectl get pods -n illumio-system
```

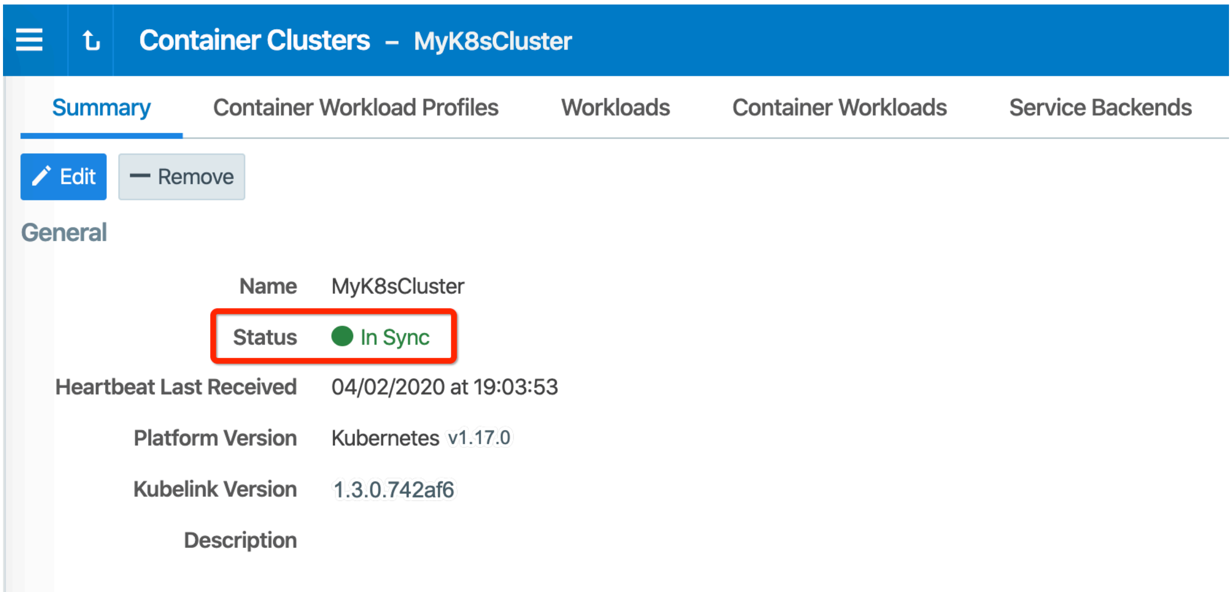
- To check the Kubelink Pod status for OpenShift:

```
oc get pods -n illumio-system
```

The **illumio-kubelink-xxxxxxxxxx-xxxxx** Pod should be in the "Running" state. If the either **get pods -n illumio-system** command shows the kubelink pod is not successfully running, check the log file for any ERROR messages.

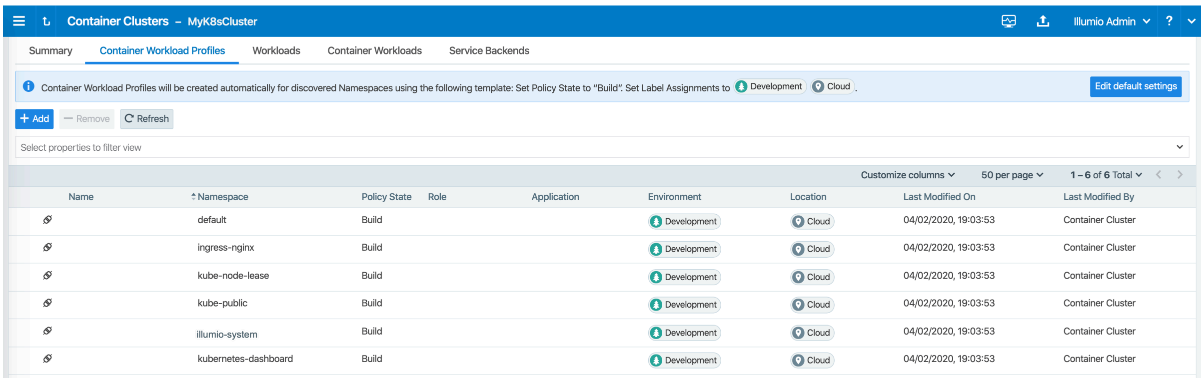
After Kubelink is successfully deployed, you can check the cluster information in the Illumio PCE UI. From the main menu, navigate to **Infrastructure > Container Clusters**.

Below is an example of a healthy container cluster state reported by Kubelink, where Status is "In Sync".

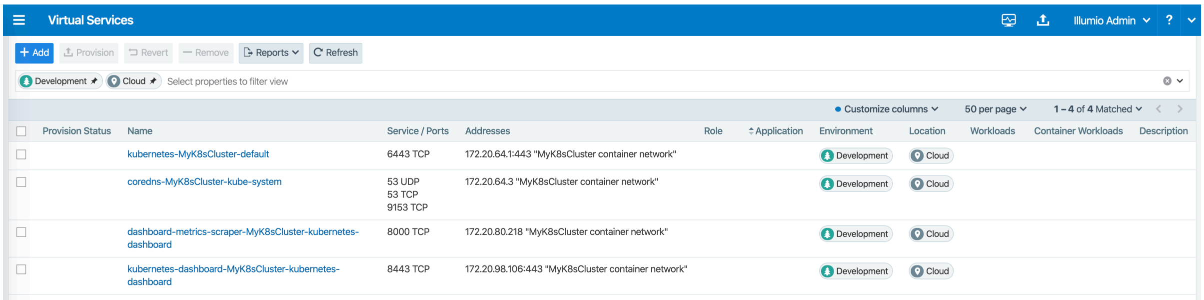


You can also verify in the PCE UI that Kubelink was successfully deployed by checking the following:

- Under the **Container Workload Profiles** tab, namespaces created in your Kubernetes or OpenShift cluster should be listed. An example is shown below.



- Under **Policy Objects > Virtual Services**, services created in your Kubernetes or OpenShift cluster should be listed. An example is shown below.



PCE-Kubelink Connection and Heartbeat

The Kubelink heartbeat to the PCE is logged in its log file. Use the **`kubectl logs`** command, and search for the string **`Heart beating to PCE`** to confirm. To confirm PCE-Kubelink connectivity, check the PCE UI, which will show the Kubelink pod as being offline if the heartbeat is missing 2-3 times (about 10 minutes).

Additional Kubelink Monitoring

Other Kubelink actions that can be confirmed in the Kubelink log file include:

API request succeeds

When Kubelink successfully sets up a watch with the Kubernetes API, the related log entry is:

```
sync on resource <RESOURCE> successful, setting up resource version to
<RESOURCE VERSION>
```

Information sent to PCE

When Kubelink successfully sends information to the PCE, the related log entry is:

```
Synchronized 2 <RESOURCE>, full=..., force=...
```

Setting Log Verbosity

The log verbosity level is set by default to include INFO, WARNING, and ERROR messages in the log. If your log appears to be extremely small (showing only ERRORs, for example), or is extremely large (which could indicate being set at the DEBUG level), you can check the `log_level` setting in the `illumio-kubelink-secret.yml` file. Values for this setting are:

log_level Setting	Description
0	Debug
1	Info (default)
2	Warn
3	Error

Values are cumulative, in that a setting includes all other settings greater than it. For example, the default setting of '1' includes in the log file all INFO, WARNING, and ERROR messages. Whereas a setting of '3' would only include ERROR messages.

Aggregating Logs from Kubelink and C-VEN Pods

There are many log aggregation solutions; this topic describes one example of using Fluent Bit to aggregate our logs. Fluent Bit is a lightweight version of Fluentd with many outputs. See https://docs.fluentbit.io/manual/pipeline/outputs for official details about supported Fluent Bit output plugins.

Loki is used as storage in this example. Change the output section of your Fluent Bit yaml file to suit your needs.

Loki and Grafana

As an example installation for testing, Loki and Grafana are installed in the illumio-system namespace. Loki is installed in monolithic mode to use file system storage For more details, see https://grafana.com/docs/loki/latest/setup/install/helm/install-monolithic/.

```
helm repo add grafana https://grafana.github.io/helm-charts
helm repo update
helm upgrade --install loki grafana/loki --values loki-values.yaml -n
illumio-system
```

Example contents of `loki-values.yaml`:

```
loki:
  commonConfig:
    replication_factor: 1
  storage:
    type: 'filesystem'
  auth_enabled: false

singleBinary:
  replicas: 1

# lokiCanary:
#   enabled: false
# gateway:
#   enabled: false
# grafanaAgent:
#   installOperator: true
```

```
helm upgrade --install --wait -n illumio-system --set admin.username=admin
--set admin.password=UseYourPassword --set persistence.enabled=false -f
grafana-values.yaml grafana
oci://registry-1.docker.io/bitnamicharts/grafana
kubectl -n illumio-system expose deployment grafana --type=NodePort --
name=grafana-service
kubectl -n illumio-system get svc grafana-service -o go-
template='{{range.spec.ports}}
{{if .nodePort}}{{.nodePort}}{{"\n"}}{{end}}{{end}}'
```

Example contents of `grafana-values.yaml`:

```
dashboardsProvider:
  enabled: true
```

Fluent Bit

The following procedure shows one way of downloading and installing Fluent Bit:

```
helm repo add fluent https://fluent.github.io/helm-charts
helm repo update
helm upgrade --install fluent-bit fluent/fluent-bit --version 0.40.0 --
values fluentbit-values.yaml
-n illumio-system
kubectl --namespace illumio-system patch daemonsets.apps fluent-bit --patch-
file
fluentbit-patch-nodename.yaml
```

Example contents of `fluentbit-values.yaml`:

```
labels
  app: IllumioFluentBit

image:
  pullPolicy: IfNotPresent

extraVolumes:
  - name: illumio-ven-data
    hostPath:
    path: /opt/illumio_ven_data
    type: Directory

extraVolumeMounts:
  - name: illumio-ven-data
    mountPath: /opt/illumio_ven_data

config:
  service: |
    [SERVICE]
        daemon Off
        flush {{ .Values.flush }}
        log_level debug
        parsers_file parsers.conf
        parsers_file custom_parsers.conf
        http_server On
        http_listen 0.0.0.0
        http_port {{ .Values.metricsPort }}
        health_check On

  inputs: |
    [INPUT]
        Name tail
        Path /var/log/containers/illumio-kubelink*.log
        Tag kubelink.*
        Multiline.parser docker,cri
        Read_From_Head true
        Buffer_Chunk_Size 3MB
```

```
        Buffer_Max_Size 10MB
        Mem_Buf_Limit 10MB
        Skip_Long_Lines Off
    [INPUT]
        Name tail
        Path /opt/illumio_ven_data/log/*.log
        Tag cven.*
        Read_From_Head true
        Buffer_Chunk_Size 3MB
        Buffer_Max_Size 10MB
        Mem_Buf_Limit 10MB
        Skip_Long_Lines Off

  filters: |
    [FILTER]
        Name kubernetes
        Match kubelink.*
        Merge_Log On
        Kube_Tag_Prefix  kubelink.var.log.containers.
        Merge_Log_Key log_processed
    [FILTER]
        Name parser
        Parser cvenparser
        Match cven.*
        Key_name log
        Preserve_key false
        Reserve_data true
    [FILTER]
        Name record_modifier
        Match cven.*
        Record nodename ${K8S_NODE_NAME}

  upstream: {}

  outputs: |
    [OUTPUT]
        #for debugging only should be turned off in PROD
        #PLEASE TURN OFF IN PROD
        Name stdout
        Match *

    [OUTPUT]
        Name                       loki
        Match                      *
        Host                       loki.illumio-system.svc.cluster.local
        Port                       3100
        Labels                     job=fluentbit

  customParsers: |
    [PARSER]
      Name        cvenparser
      Format      regex
      Regex       ^(?<time>[^ ]+) (?<message>.+)$
      Time_Key       time
      Time_Format %Y-%m-%dT%H:%M:%S.%L
```

```
  extraFiles {}

logLevel: info
```

Example contents of `fluentbit-patch-nodeport.yaml`:

```
spec:
  template:
      spec:
      containers:
      - name: fluent-bit
        env:
          - name: K8S_NODE_NAME
            valueFrom:
              fieldRef:
                fieldPath: spec.nodeName
```

Reference: OpenShift Deployment

Prepare OpenShift for Illumio Core

If the prerequisite steps are not performed prior to VEN and Kubelink installation, containerized environments and Kubelink may be disrupted.

Unique Machine ID

Some of the functionalities and services provided by the Illumio VEN and Kubelink depend on the Linux machine-id of each OpenShift cluster node. Each machine-id must be unique in order to take advantage of the functionalities. By default, the Linux OS generates a random machine-id to give each Linux host uniqueness. However, there are cases when machine-id's can be duplicated across machines. This is common across deployments that clone machines from a golden image, for example, spinning up virtual machines from VMware templates or creating Amazon EC2 instances from an AMI.

To verify machine-ids and resolve any duplicate machine-ids across nodes:

1. `ssh` into every node of the OpenShift cluster (master, infra, and worker) as the root user.
2. Check the contents of the `/etc/machine-id` file. The output is a string of letters and numbers.
3. If the machine-id string is unique for each node, then the environment is ok. If the machine-id is duplicated across any of the nodes, you must generate a machine-id for each node which has the same machine-id.

You can run the following command to view the output of machine-id:

```
cat /etc/machine-id
```

If the machine-id is duplicated, then run the command listed below to generate a new machine-id. You will also need to restart the `atomic-OpenShift-node` service on each node. If the machine-id is not duplicated, go to the next section.

```
rm -rf /etc/machine-id; touch /etc/machine-id; systemd-machine-id-setup;
service atomic-OpenShift-node restart
```

NOTE
Check the machine-id again to verify that each machine has a unique machine-id.

Create Labels

For details on creating labels, see "Labels and Label Groups" in the Security Policy Guide.

The labels listed below are used in examples throughout this document. You are not required to use the same labels.

Name	Label type
Openshift Infrastructure	Application
Development	Environment
HQ	Location
Kubelink	Role
Master	Role
Infra	Role
Compute	Role

Create Pairing Profiles

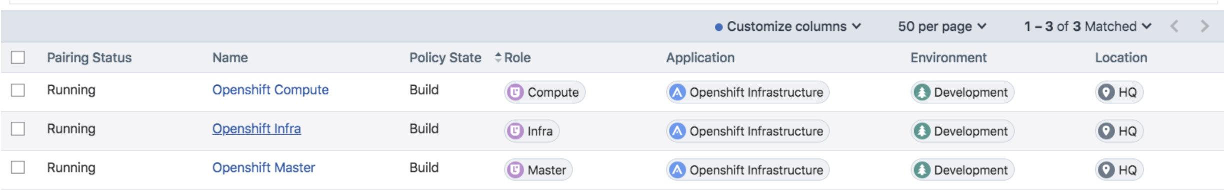
After creating labels for your OpenShift cluster nodes, you can use those labels to create pairing profiles. You do not need to create pairing profiles for container workloads.

For ease of configuration and management, consider applying the same Application, Environment, and Location labels across all nodes of the same OpenShift cluster. The screenshot below show examples of three pairing profiles for one OpenShift Enterprise cluster. The pairing profiles are used for pairing either master, compute, or infrastructure nodes of an OpenShift cluster.

TIP

It is recommended that all pairing profiles for OpenShift nodes **not** use Enforced policy state.

Move into Enforced state after you have completed all other configuration steps in this guide (setup Kubelink, discover services, and write rules).



Customize columns | 50 per page | 1 – 3 of 3 Matched

Pairing Status	Name	Policy State	Role	Application	Environment	Location
Running	Openshift Compute	Build	Compute	Openshift Infrastructure	Development	HQ
Running	Openshift Infra	Build	Infra	Openshift Infrastructure	Development	HQ
Running	Openshift Master	Build	Master	Openshift Infrastructure	Development	HQ

Deploy Kubelink

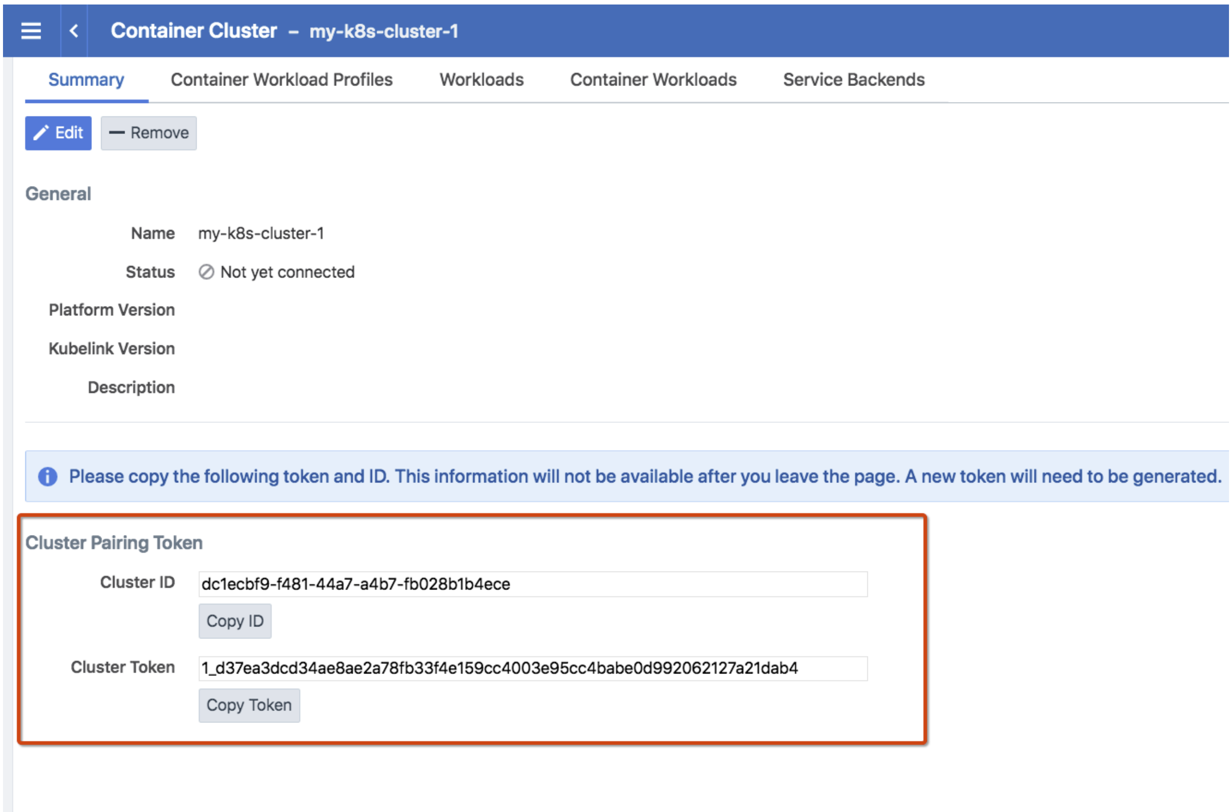
Download the required resources such as Kubelink docker image, secret file, and deployment file from the Illumio Support portal (login required).

Prerequisites

- Kubelink deployment file provided by Illumio. For OpenShift deployments, the file name is `illumio-kubelink-openshift.yml`.
- Kubelink secret file provided by Illumio. This file name is `illumio-kubelink-secret.yml`.
- Illumio's Kubelink docker image uploaded to your private docker registry.

Create Container Cluster

1. Log into the PCE as a user with Global Organization Owner privileges.
2. From the PCE web console menu, choose **Infrastructure** > **Container Clusters**.
3. Click **Add**.
   a. Enter a Name.
   b. **Save** the Container Cluster.
4. You will see a summary page of the new Container Cluster. Copy the values of the Cluster ID and Cluster Token found under the Cluster Pairing Token section.
5. Once you have the values, you can exit the Container Cluster page.



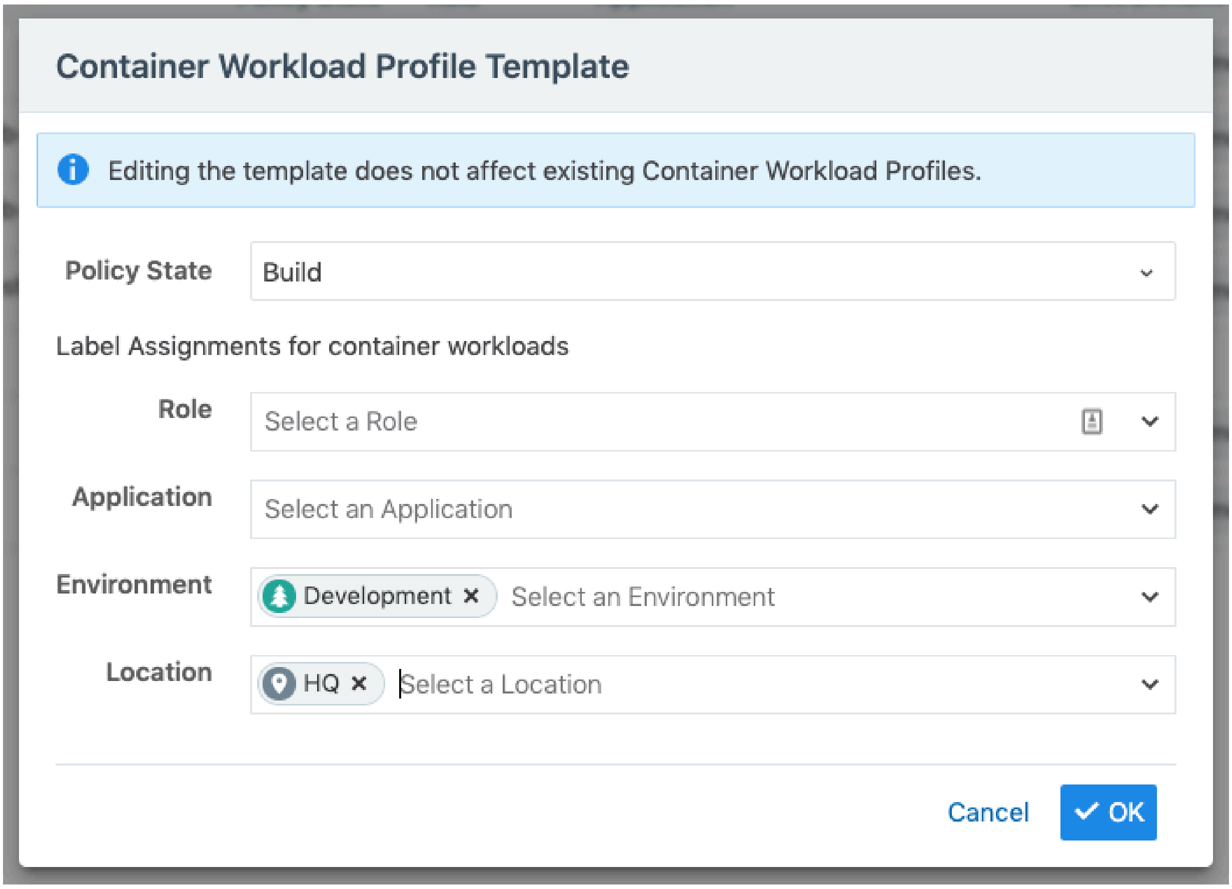
Configure Container Workload Profile

When configuring a new Container Cluster, it is recommended to set the default settings shared by all the Container Workload Profiles. Illumio provides a Container Workload Profile template that can be used for that purpose. By defining the default Policy State and minimum set of labels common to all namespaces in the cluster, you will save time later on when new namespaces are discovered by Kubelink. Each new profile created will inherit what was defined in the template.

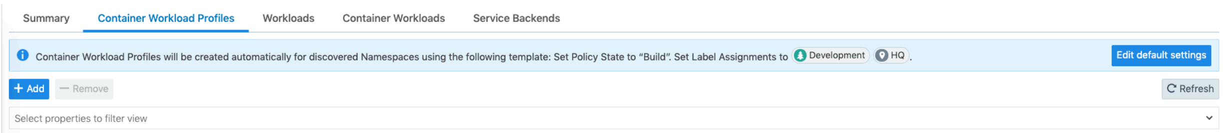
SSL Verification

Illumio does not provide a simple way to redefine all at once the labels associated to each profile all at once in this release, so it is strongly recommended to use this template to define the default values for all profiles part of the same cluster.

To define the default parameters for all profiles using a template, under Container Workload Profiles, click on Edit default settings and fill in the different fields. An example is shown below:



Once you validate, you should see something like the following:



Configure Kubelink Secret

This step assumes that you have created a Container Cluster object in the PCE. You will need the Cluster ID and Cluster Token values for the Kubelink secret.

1. `ssh` to the master node.
2. Open the kubelink secret YAML file and modify the `stringData`.
   a. `ilo_server` = the PCE URL and port. Example: `https://mypce.example.com:8443`
   b. `ilo_cluster_uuid` = Cluster ID value from previous step. Example: `dc1ecbf9-f481-44a7-a4b7-fb028b1b4ece`
   c. `ilo_cluster_token` = Cluster Token from previous step. Example: `1_d37ea3dcd34ae8ae2a78fb33f4e159cc4003e95cc4babe0d992062127a21dab4`
   d. `ignore_cert` = SSL verification. The value is boolean and is recommended to be set to false so that Kubelink requires PCE certificate verification. Example: '`false`'
   e. `log_level` = Log level where '`0`' for debug, '`1`' for info, '`2`' for warn, or '`3`' for error. Example: '`1`'

   SSL Verification

Illumio does not recommend turning off SSL verification (`ignore_cert`: '`true`'); however, this is an option for deployments in which the PCE uses a self-signed certificate.

Contents of a modified illumio-kubelink-secret.yml file are shown below.

```
#
# Copyright 2013-2020 Illumio, Inc. All Rights Reserved.
#

apiVersion: v2
kind: Secret
metadata:
  name: illumio-kubelink-config
  namespace: kube-system
type: Opaque
stringData:
  ilo_server: https://mypce.example.com:8443 # Example: https://
mypce.example.com:8443
  ilo_cluster_uuid: dc1ecbf9-f481-44a7-a4b7-fb028b1b4ece # Example:
cc4997c1-408b-4f1d-a72b-91495c24c6a0
  ilo_cluster_token:
1_d37ea3dcd34ae8ae2a78fb33f4e159cc4003e95cc4babe0d992062127a21dab4 #
Example: 170b8aa3dd6d8aa3c284e9ea016e8653f7b51cb4b0431d8cbdba11508763f3a3
  ignore_cert: 'false' # Set to 'true' to ignore the PCE certificate
  log_level: '1' # Default log level is info
```

NOTE

If you are going to use a private PKI to sign the PCE certificate, see Implement Kubelink with a Private PKI [123] before deploying Kubelink.

3. Save the changes.
4. Create the Kubelink secret using the file.

```
oc create -f illumio-kubelink-secret.yml
```

Deploy Kubelink

Modify the Kubelink configuration file to point to the correct docker image. The example in this document has `kubelink:<version#>` uploaded to `registry.example.com:443/illumio`, which means the image link in this example is `registry.example.com:443/illumio/kubelink:<version#>`

1. Edit the Kubelink configuration YAML file. For OpenShift clusters, the file name will be `illumio-kubelink-openshift.yml`.
   - Inside the YAML you will find the `spec: > template: > spec: > containers:` section. Paste the image link in the `image:` attribute. The string should be wrapped in single quotes as shown in the example below.
2. Save the changes.

   Below is a snippet from an example of the Kubelink configuration for OpenShift to illustrate the image location.

```
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: illumio-kubelink
```

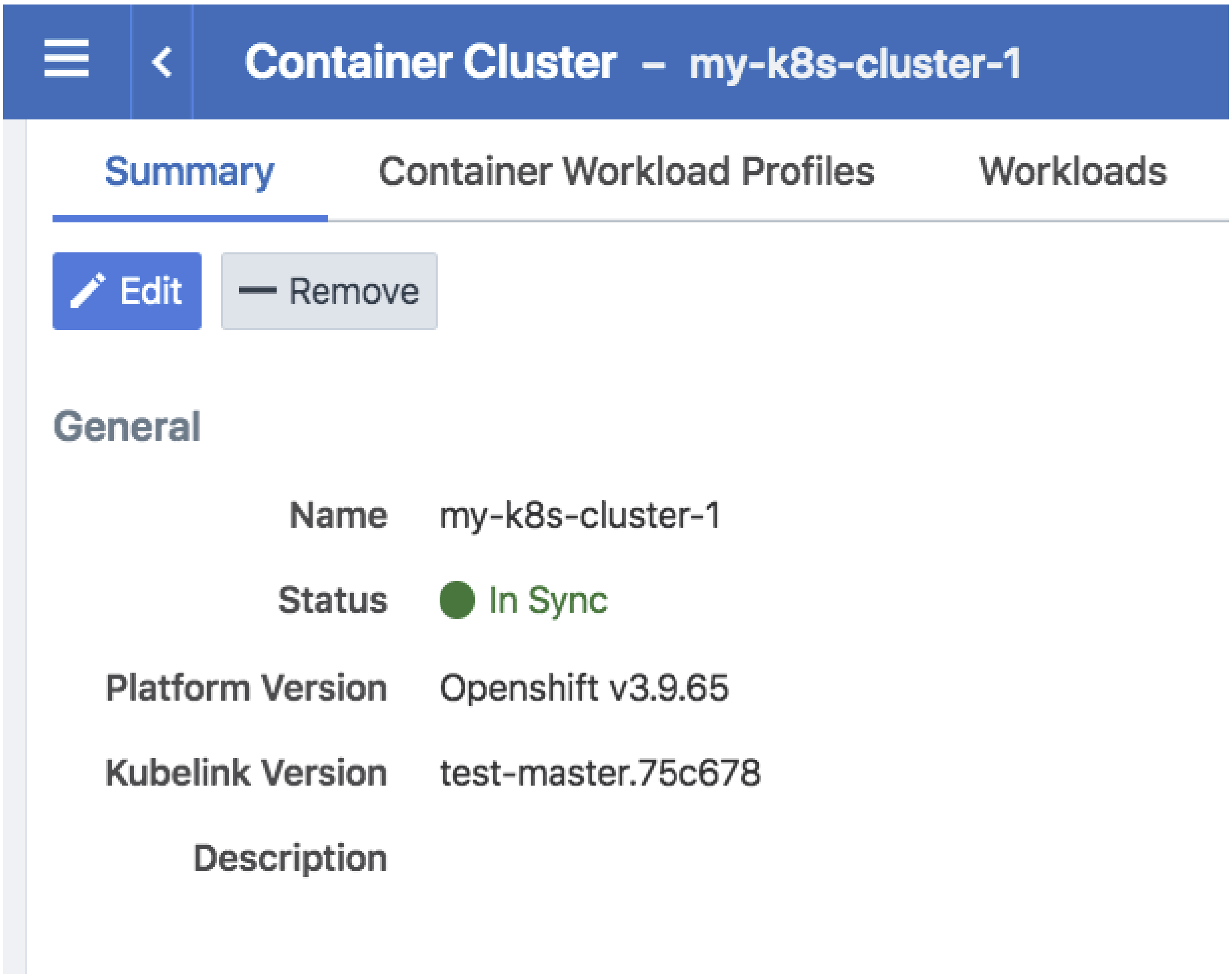
```
  namespace: kube-system
spec:
  replicas: 1
  selector:
    matchLabels:
      app: illumio-kubelink
  template:
    metadata:
      labels:
        app: illumio-kubelink
    spec:
#      nodeSelector:
#        node-role.kubernetes.io/master: ""
      serviceAccountName: illumio-kubelink
      tolerations:
      - key: node-role.kubernetes.io/master
        effect: NoSchedule
      containers:
      - name: illumio-kubelink
        image: 'registry.example.com:443/illumio/illumio-
kubelink:<version#>'
        imagePullPolicy: Always
        env:
          - name: ILO_SERVER
            valueFrom:
              secretKeyRef:
                name: illumio-kubelink-config
                key: ilo_server
```

3. (Optional) If you're using a private PKI to sign the PCE certificate, make sure you add the references to the root CA certificate that signed the PCE certificate. For more details, see Implement Kubelink with a Private PKI [123].
4. To deploy Kubelink, run the following command:

```
oc apply -f illumio-kubelink-openshift.yml
```

After Kubelink is successfully installed, you can check the cluster information by using the Illumio PCE web console. From the main menu, navigate to **Infrastructure > Container Clusters**.

Below is an example of a healthy container cluster state reported by Kubelink.



Implement Kubelink with a Private PKI

This section describes how to implement Kubelink with a PCE using a certificate signed by a private PKI. It describes how to configure Kubelink to accept the certificate from the PCE signed by a private root or intermediate Certificate Authority (CA) and ensure that Kubelink can communicate in a secure way with the PCE.

> **NOTE**
> The steps described below are not applicable for a PCE using a self-signed certificate.

Prerequisites

- Access to the root CA to download the root CA certificate.
- Access to your Kubernetes cluster and can run `kubectl` commands.
- Correct privileges in your Kubernetes cluster to create resources like a configmaps, secrets, and pods.
- Access to the PCE UI as a Global Organization Owner.

Download the Root CA Certificate

Before you begin, ensure that you have access to the root CA certificate. The root CA certificate is a file that can be exported from the root CA without compromising the security of the company. It is usually made available to external entities to ensure a proper SSL handshake between a server and its clients.

You can download the root CA cert in the CRT format on your local machine. Below is an example of a root CA certificate:

```
$ cat root.democa.illumio-demo.com.crt
-----BEGIN CERTIFICATE-----
MIIGSzCCBDOgAwIBAgIUAPw0NfPAivJW4YmKZ499eHZH3S8wDQYJKoZIhvcNAQEL
---output suppressed---
wPG0lug46K1EPQqMA7YshmrwOd6ESy6RGNFFZdhk9Q==
-----END CERTIFICATE-----
```

You can also get the content of your root CA certificate in a readable output format by running the following command:

```
$ openssl x509 -text -noout -in ./root.democa.illumio-demo.com.crt
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            fc:34:35:f3:c0:8a:f2:56:e1:89:8a:67:8f:7d:78:76:47:dd:2f
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=US, ST=California, L=Sunnyvale, O=Illumio, OU=Technical
Marketing, CN=Illumio Demo Root CA 1/emailAddress=tme-team@illumio.com
        Validity
            Not Before: Jan 20 00:05:36 2020 GMT
            Not After : Jan 17 00:05:36 2030 GMT
        Subject: C=US, ST=California, L=Sunnyvale, O=Illumio, OU=Technical
Marketing, CN=Illumio Demo Root CA 1/emailAddress=tme-team@illumio.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (4096 bit)
                Modulus:
                    00:c0:e5:48:7d:97:f8:5b:8c:ef:ac:16:a8:8c:aa:
                    68:b8:48:af:28:cd:17:8f:02:c8:82:e9:69:62:e2:
                    89:2b:be:bd:34:fc:e3:4d:3f:86:5e:d7:e6:89:34:
                    71:60:e6:54:61:ac:0f:26:1c:99:6f:80:89:3f:36:
                    b3:ad:78:d1:6c:3f:d7:23:1e:ea:51:14:48:74:c3:
                    e8:6e:a2:79:b1:60:4c:65:14:2a:f1:a0:97:6c:97:
                    50:43:67:07:b7:51:5d:2c:12:49:81:dc:01:c9:d1:
                    57:48:32:2e:87:a8:d2:c0:b9:f8:43:b2:58:10:af:
                    54:59:09:05:cb:3e:f0:d7:ef:70:cc:fc:53:48:ee:
                    a4:a4:61:f1:d7:5b:7c:a9:a8:92:dc:77:74:f4:4a:
                    c0:4a:90:71:0f:6d:9e:e7:4f:11:ab:a5:3d:cd:4b:
                    8b:79:fe:82:1b:16:27:94:8e:35:37:db:dd:b8:fe:
                    fa:6d:d9:be:57:f3:ca:f3:56:aa:be:c8:57:a1:a8:
                    c9:83:dd:5a:96:5a:6b:32:2d:5e:ae:da:fc:85:76:
                    bb:77:d5:c2:53:f3:5b:61:74:e7:f3:3e:4e:ad:10:
                    7d:4f:ff:90:69:7c:1c:41:2f:67:e4:13:5b:e6:3a:
                    a3:2f:93:61:3b:07:56:59:5a:d9:bc:34:4d:b3:54:
```

```
                    b5:c6:e5:0a:88:e9:62:7b:4b:85:d2:9e:4c:ee:0b:
                    0d:f4:72:b1:1b:44:04:93:cf:cc:bb:18:31:3a:d4:
                    83:4a:ff:15:42:2d:91:ca:d0:cb:36:d9:8d:62:c0:
                    41:59:1a:93:c7:27:79:08:94:b2:a2:50:3c:57:27:
                    33:af:f0:b6:92:44:49:c5:09:15:a7:43:2a:0f:a9:
                    02:61:b3:66:4f:c3:de:d3:63:1e:08:b1:23:ea:69:
                    90:db:e8:e9:1e:21:84:e0:56:e1:8e:a1:fa:3f:7a:
                    08:0f:54:0a:82:41:08:6b:6e:bb:cf:d6:5b:80:c6:
                    ea:0c:80:92:96:ab:95:5d:38:6d:4d:da:38:6b:42:
                    ef:7c:88:58:83:88:6d:da:28:62:62:1f:e5:a7:0d:
                    04:9f:0d:d9:52:39:46:ba:56:7c:1d:77:38:26:7c:
                    86:69:58:4d:b0:47:3a:e2:be:ee:1a:fc:4c:de:67:
                    f3:d5:fe:e6:27:a2:ef:26:86:19:5b:05:85:9c:4c:
                    02:24:76:58:42:1a:f8:e0:e0:ed:78:f2:8f:c8:5a:
                    20:a9:2d:0b:d4:01:fa:57:d4:6f:1c:0a:31:30:8c:
                    32:7f:b0:01:1e:fe:94:96:03:ee:01:d7:f4:4a:83:
                    f5:06:fa:60:43:15:05:9a:ca:88:59:5c:f5:13:09:
                    82:69:7f
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                3D:3D:3D:61:E6:88:09:FE:34:0F:1D:5E:5E:52:72:71:C7:DE:15:92
            X509v3 Authority Key Identifier:

keyid:3D:3D:3D:61:E6:88:09:FE:34:0F:1D:5E:5E:52:72:71:C7:DE:15:92

            X509v3 Basic Constraints: critical
                CA:TRUE
            X509v3 Key Usage: critical
                Digital Signature, Certificate Sign, CRL Sign
    Signature Algorithm: sha256WithRSAEncryption
         28:24:86:91:a6:4a:88:e4:8d:6b:fc:67:2a:68:08:67:35:e5:
         a6:77:ff:07:4b:89:53:99:2e:6d:95:df:12:81:28:6a:8e:6f:
         5a:98:95:5b:4a:21:ae:f0:20:a4:4e:06:b2:4e:5a:67:c1:6a:
         06:f1:0f:c1:f7:7e:f2:e0:b3:9d:d8:54:26:6a:b2:1c:19:b8:
         b5:5c:c7:03:6b:f7:70:9e:72:85:c9:29:55:f9:f4:a4:f2:b4:
         3b:3d:ce:25:96:67:32:1e:8d:e2:00:22:55:4b:05:4f:ee:0e:
         67:ac:db:1b:61:da:5f:9c:10:1c:0c:05:66:c0:5b:5f:b9:95:
         59:a9:58:5b:e7:69:ac:b0:bd:b3:c2:a3:35:58:01:a4:ff:c0:
         8d:ac:1c:19:21:41:50:fb:8e:e0:f5:a9:ad:ec:de:cb:53:04:
         a9:d8:ac:76:8a:09:0d:7c:c6:1a:bc:06:74:bb:10:1c:aa:07:
         f6:cb:b2:1b:0c:0c:65:03:45:2b:51:d5:6e:a0:4d:91:ce:c5:
         ed:8d:a9:e7:f6:37:7d:ab:1b:a4:a2:a3:3b:76:17:5b:d9:3a:
         9c:c1:df:cc:cd:a0:b0:a9:5c:74:61:d7:a0:1d:04:67:68:ee:
         a6:7b:1e:41:a4:02:fc:65:9e:e3:c1:c2:57:b2:2e:b0:ff:a9:
         86:82:35:4d:29:b2:fe:74:2e:b8:37:5d:2b:e8:69:f2:80:29:
         19:f1:1e:7a:5d:e3:d2:51:50:46:30:54:7e:b8:ad:59:61:24:
         45:a8:5a:fe:19:ff:09:31:d0:50:8b:e2:15:c0:a2:f1:20:95:
         63:55:18:a7:a2:ad:16:25:c7:a3:d1:f2:e5:be:6d:c0:50:4b:
         15:ac:e0:10:5e:f3:7b:90:9c:75:1a:6b:e3:fb:39:88:e4:e6:
         9f:4c:85:60:67:e8:7d:2e:85:3d:87:ed:06:1d:13:0b:76:d7:
         97:a5:b8:05:76:67:d6:41:06:c5:c0:7a:bd:f4:c6:5b:b2:fd:
         23:6f:1f:57:2e:df:95:3f:26:a5:13:4d:6d:96:12:56:98:db:
         2e:7d:fd:56:f5:71:b7:19:2b:c9:de:2d:b9:c8:17:cc:20:de:
         7c:19:7a:aa:12:97:1c:80:b7:d3:67:d3:b7:a7:96:f0:c9:4d:
```

```
        f5:8b:0e:10:3b:b9:4e:09:90:5a:3b:51:c9:48:a2:ca:9f:db:
        72:44:87:59:db:49:fa:75:44:b5:f6:7f:c5:26:e1:01:ae:7b:
        6f:4a:75:d1:b5:b3:68:c0:31:48:f8:5c:06:c0:f1:b4:96:e8:
        38:e8:ad:44:3d:0a:8c:03:b6:2c:86:6a:f0:39:de:84:4b:2e:
        91:18:d1:45:65:d8:64:f5
```

Create a configmap in Kubernetes Cluster

After downloading the certificate locally on your machine, create a configmap in the Kubernetes cluster that will copy the root CA certificate on your local machine into the Kubernetes cluster.

To create configmap, run the following command:

```
$ kubectl -n kube-system create configmap root-ca-config \
    --from-file=./certs/root.democa.illumio-demo.com.crt
```

The `--from-file` option points to the path where the root CA certificate is stored on your local machine.

To verify that configmap was created correctly, run the following command:

```
$ kubectl -n kube-system create configmap root-ca-config \
> --from-file=./certs/root.democa.illumio-demo.com.crt
configmap/root-ca-config created
$
$ kubectl -n kube-system get configmap
NAME                                 DATA   AGE
calico-config                        8      142d
cluster-info                         4      142d
coredns                              1      142d
coredns-autoscaler                   1      142d
crn-info-ibmc                        6      142d
extension-apiserver-authentication   6      142d
iaas-subnet-config                   1      142d
ibm-cloud-cluster-ingress-info       2      142d
ibm-cloud-provider-data              1      142d
ibm-cloud-provider-ingress-cm        6      142d
ibm-master-proxy-config              1      142d
ibm-network-interfaces               1      142d
kube-dns                             0      142d
kubernetes-dashboard-settings        1      44d
metrics-server-config                1      142d
node-local-dns                       1      142d
root-ca-config                       1      12s
subnet-config                        1      142d
$
$ kubectl -n kube-system describe configmap root-ca-config
Name:         root-ca-config
Namespace:    kube-system
Labels:       <none>
Annotations:  <none>

Data
```

```
====
root.democa.illumio-demo.com.crt:
----
-----BEGIN CERTIFICATE-----
MIIGSzCCBDOgAwIBAgIUAPw0NfPAivJW4YmKZ499eHZH3S8wDQYJKoZIhvcNAQEL
---output suppressed---
wPG0lug46K1EPQqMA7YshmrwOd6ESy6RGNFFZdhk9Q==
-----END CERTIFICATE-----

Events:  <none>
$
```

`root-ca-config` is the name used to designate configmap. You can modify it according to your naming convention.

Modify Kubelink Manifest File to Use Certificate

After creating the configmap in your Kubernetes cluster, modify the YAML file that describes Kubelink.

The current manifest file provided by Illumio does not include this modification, by default. Open the .yml file and add the following code blocks:

- volumeMounts (under `spec.template.spec.containers`)
- volumes (under `spec.template.spec`)

```
       volumeMounts:
          - name: root-ca
            mountPath: /etc/pki/tls/ilo_certs/
            readOnly: false
      volumes:
      - name: root-ca
        configMap:
          name: root-ca-config
```

NOTE

In a YAML file, the indentation matters. Make sure that the indentation in the file is as specified.

`root-ca` is the name used to designate the new volume mounted in the container. You can modify it according to your naming convention.

After successfully modifying the manifest file, deploy Kubelink. For more details, see Deploy Kubelink [118].

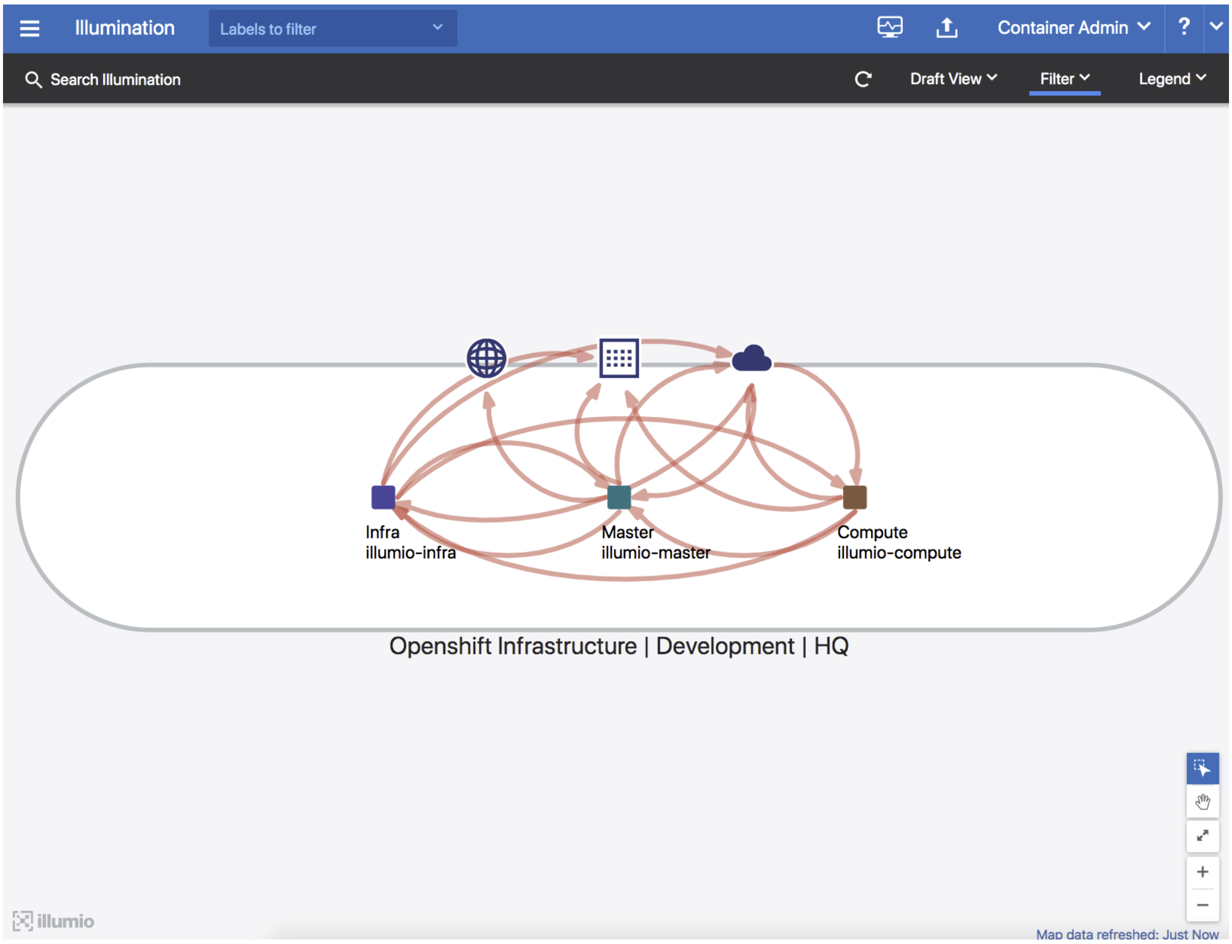
Install and Pair VENs for Containers

Using the pairing profiles mentioned earlier in this guide to install the VEN on each node of your OpenShift cluster. For more information about installing VENs, see the VEN Installation and Upgrade Guide.

Ensure that either of the two requirements below have been met prior to installing the VEN:

- Kubelink is deployed on the OpenShift cluster and in sync with the PCE
- Firewall coexistence is enabled

Below is a screenshot of Illumination with a master, compute, and infra node after deploying and pairing the Illumio VEN.



Manage OpenShift Namespaces

After activating the VENs on the OpenShift cluster nodes and Kubelink is in sync with the PCE, you can start managing the OpenShift projects (or namespaces). By default, all namespaces are unmanaged, which means Illumio Core does not apply any inbound or outbound controls to the pods within those namespaces. Any pods or services within unmanaged namespaces do not show up in the PCE inventory and Illumination.

After an Illumio Core PCE administrator changes an OpenShift namespace from unmanaged to managed, the pods and services will show up in Illumination and inherit the labels of each

OpenShift namespace. The pods are represented in Illumio Core as Container Workloads. If there are frontend services, then Illumio Core represents each one as a Virtual Service.

The following section describes how to change a namespace from unmanaged to managed.

Container Workload Profiles

Log into the PCE Web Console.

1. From the PCE web console menu, choose **Infrastructure** > **Container Clusters**.
2. Select the **Container Cluster** you want to manage.
3. Select the **Container Workload Profiles** tab.
4. You will see a list of all namespaces in the cluster. Select the namespace you want to manage.
5. Click **Edit**:
    a. Name is optional.
    b. Select a Container Workload Policy State (anything other than unmanaged).
    c. Assign Labels (optional).
    d. Click **Save**.

When assigning labels, you can assign no labels, some labels, or all labels to the namespace. If there is a label which is not assigned, then you can insert annotations into the deployment configuration (or application configuration) to assign labels. If there is a conflict between a label assigned via the Container Workload Profile and the annotations in the deployment configuration, then the label from the Container Workload Profile will override the deployment configuration. Regardless of how you assign labels, it is not required for pods or services to have all labels in order for the PCE to manage them. Below are instructions on how to assign labels via the deployment configuration.

Using Annotations

For Deployment Configurations (Pods)

1. Open the OpenShift Web Console.
2. Navigate to the desired deployment/daemon set and click **Edit YAML**.
    a. Inside the configuration YAML navigate to `spec: > template: > metadata: > annotations:`. If `annotations:` does not exist, then create an `annotations:` section underneath `metadata:`.
    b. The following Illumio label key fields which can go under the the `annotations:` section.
        - `com.illumio.role:`
        - `com.illumio.app:`
        - `com.illumio.env:`
        - `com.illumio.loc:`
    c. Fill in the appropriate labels.
    d. Save the file and exit.

For Service Configurations (Services)

1. Open the OpenShift Web Console.
2. Navigate to the desired service and click **Edit YAML**.
    a. Inside the configuration YAML navigate to `metadata: > annotations:`. If `annotations:` does not exist, then create an `annotations:` section underneath `metadata:`.

b. The following Illumio label key fields which can go under the the **`annotations:`** section.
- `com.illumio.role:`
- `com.illumio.app:`
- `com.illumio.env:`
- `com.illumio.loc:`

c. Fill in the appropriate labels.

d. Save the file and exit.

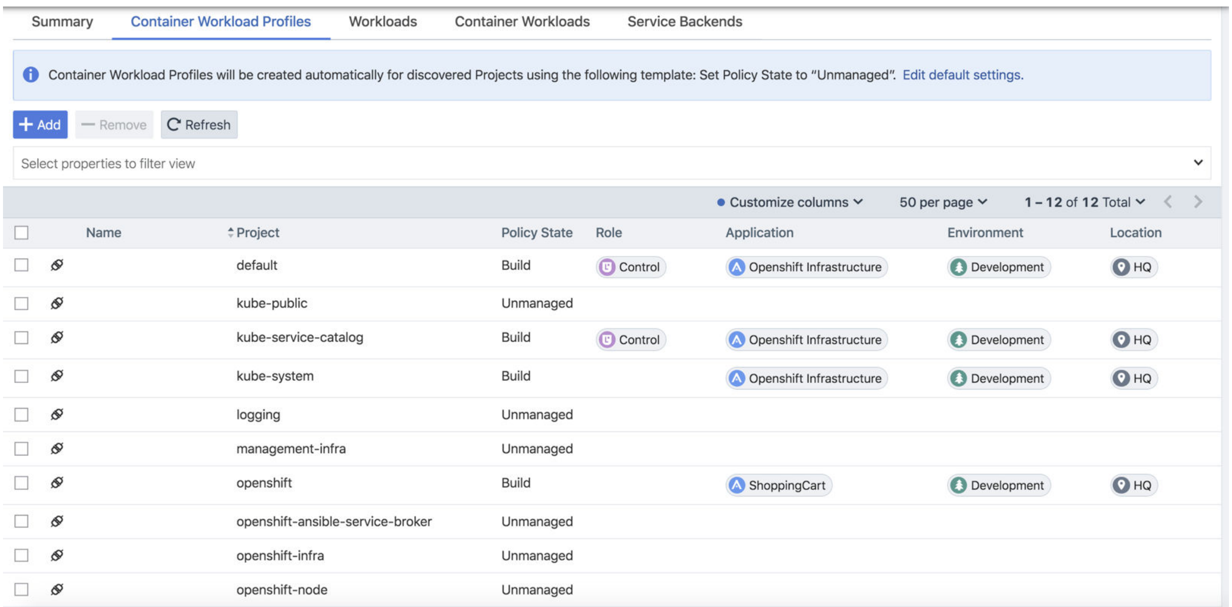
When using the annotations method, you may need to restart the pods or service after saving the changes to the YAML for the labels to get assigned.

Below are examples of pods and namespaces which use label assignments via either Container Workload Profiles or a mix of Container Workload Profiles plus annotation insertion.

This example changes unmanaged namespaces of Openshift infrastructure services (such as apiserver, registry-console, etc.) into managed namespaces.

Things to notice about the example shown below:

- There are Openshift infrastructure services, or control plane pods, that exist within namespaces like **`default`**, **`kube-service-catalog`**, etc. They will inherit all four R-A-E-L labels, including a Role label called "Control", from what has been configured in the Container Workload Profile(s). The Application, Environment, and Location labels are the same as the Openshift cluster nodes. This will minimize the complexity of writing policy which is mentioned later in this guide.
- The Kubelink pod exists in the kube-system. This pod will get the same application, environment, and location labels as the Openshift cluster nodes. The role label is left blank and will be specified later using the annotations. These labels are assigned to the Kubelink pod through the Container Workload Profile associated to the kube-system namespace.
- There is a namespace called **`openshift`** which contains two different deployments or a two-tier shopping cart application (Web and Database). To achieve tier-to-tier segmentation across the application they would need different Role labels; therefore, a Role label will be inserted into the annotations of each deployment configuration.



Snippet of **illumio-kubelink** deployment configuration file shown here. Role label of "Kubelink" inserted under **spec: > template: > metadata: > annotations:** section.

illumio-kubelink-openshift.yml

```
apiVersion: apps/v2
kind: Deployment
metadata:
  name: illumio-kubelink
  namespace: kube-system
spec:
  replicas: 1
  selector:
    matchLabels:
      app: illumio-kubelink
  template:
    metadata:
      labels:
        app: illumio-kubelink
      annotations:
        com.illumio.role: Kubelink
```

Snippet of the Shopping-Cart Web deployment configuration file shown here. Role label of "Web" inserted under **spec: > template: > metadata: > annotations:** section.

shopping-cart-web.yml

```
spec:
  replicas: 3
  revisionHistoryLimit: 10
  selector:
    name: shopping-cart-web
  strategy:
    activeDeadlineSeconds: 21600
    resources: {}
    rollingParams:
      intervalSeconds: 1
      maxSurge: 25%
      maxUnavailable: 25%
      timeoutSeconds: 600
      updatePeriodSeconds: 1
    type: Rolling
  template:
    metadata:
      annotations:
        com.illumio.role: Web
        openshift.io/generated-by: OpenShiftNewApp
      creationTimestamp: null
      labels:
```

Snippet of the Shopping-Cart Database deployment configuration file shown here. Role label of "Database" inserted under **spec: > template: > metadata: > annotations:** section.

shopping-cart-db.yml

```
spec:
  replicas: 2
  revisionHistoryLimit: 10
  selector:
    name: postgresql
  strategy:
    activeDeadlineSeconds: 21600
    recreateParams:
      timeoutSeconds: 600
    resources: {}
    type: Recreate
  template:
    metadata:
      annotations:
        com.illumio.role: Database
        openshift.io/generated-by: OpenShiftNewApp
      creationTimestamp: null
      labels:
```

Below is the final outcome of the label assignment from the example.

Policy State	Policy Sync	Namespace	Name	Role	Application	Environment	Location
Build	Active	default	registry-console-1-r85jf	Control	Openshift Infrastructure	Development	HQ
Build	Active	kube-service-catalog	apiserver-bqf5g	Control	Openshift Infrastructure	Development	HQ
Build	Active	kube-service-catalog	controller-manager-x4vb2	Control	Openshift Infrastructure	Development	HQ
Build	Active	default	docker-registry-6-2jfcn	Control	Openshift Infrastructure	Development	HQ
Build	Active	openshift-template-service-broker	apiserver-x8bx7	Control	Openshift Infrastructure	Development	HQ
Build	Active	kube-system	illumio-kubelink-554688b759-k2mxt	Kubelink	Openshift Infrastructure	Development	HQ
Build	Active	openshift	postgresql-1-2v99p	Database	ShoppingCart	Development	HQ
Build	Active	openshift	shopping-cart-web-1-shgbc	Web	ShoppingCart	Development	HQ
Build	Active	openshift	shopping-cart-web-1-ljq78	Web	ShoppingCart	Development	HQ
Build	Active	openshift	shopping-cart-web-1-ghv2t	Web	ShoppingCart	Development	HQ
Build	Active	openshift	postgresql-1-qbl6z	Database	ShoppingCart	Development	HQ

Daemonsets and Replicasets

The steps above apply only to services in OpenShift which are bound to `deployment` or `deploymentconfig`. This is due to the Kubelink's dependency on pod hash templates which daemonset and replicaset configurations do not have. If you discover pods derived from daemonset or replicaset configurations and also discover services bound to those pods, then Kubelink will **not** automatically bind the virtual service and service backends for the PCE. The absence of this binding will create limitations with Illumio policies written against the virtual service. To get around this limitation for daemonsets and replicasets follow the steps below.

1. Log into the CLI of any OpenShift node and generate a random uuid using the `uuidgen` command.
2. Copy the output of the `uuidgen` command.
3. In the OpenShift web console, navigate to the configuration of the daemonset or replicaset and edit the YAML file.

4. Find the `spec: > template: > metadata: > labels:` field in the YAML and create field called `pod-template-hash:` under the `labels:` section.
5. Paste the new uuid to the value of the `pod-template-hash:` field.
6. Save the changes.

Repeat steps 1 through 6 for each daemonset or replicaset configuration.

See screenshots below for DaemonSet or ReplicaSet reference.


```
[root@master ~]# cat nginx-ds.yaml
apiVersion: extensions/v1
kind: DaemonSet
metadata:
  name: nginx-webserver
spec:
  template:
    metadata:
      annotations:
        com.illumio.pairing_key: 4229fb4a718c62861e11139749580112068b35394639954eac02a1395c87888e307d72676fc0
      labels:
        app: nginx-webserver
        pod-template-hash: be85a690-613a-4b24-a7f7-5765befbe11d
    spec:
      containers:
        - name: webserver
          image: rstarmer/nginx-curl
          imagePullPolicy: IfNotPresent
          ports:
          - containerPort: 80
```

OpenShift Deployment Reference

After you set up your clusters, make sure you perform the steps in the order provided in this section.

1. Prepare OpenShift for Illumio Core.
2. Deploy Kubelink.
3. Implement Kubelink with a private KPI.
4. Install and Pair VENs for Containers.
5. Manage OpenShift Namespaces.

Illumio Core for Kubernetes Release Notes 5.2

November 6, 2024

About Illumio Core for Kubernetes 5.2

These release notes describe the resolved issues, known issues, and related information for the 5.2.*x* releases of Illumio Core for Kubernetes, formerly known as Illumio Containerized VEN, or C-VEN. Illumio Core for Kubernetes also includes the related required component, Kubelink. Because of this heritage, many references to this product as "C-VEN" are still used throughout the documentation.

Document Last Revised: November 2024

Product Version

Compatible PCE Versions: 23.5.0+A1 and later releases

Current Illumio Core for Kubernetes Version: 5.2.1, which includes:

- C-VEN version: 23.4.0
- Kubelink version: 5.2.1
- Helm Chart version: 5.2.1

What's New in Release 5.2.1

- **Helm Chart option to Disable NodePort Forwarding**
  A new option was added to Helm Chart for C-VEN that disables NodePort forwarding on host workloads. After setting `enforceNodePortTraffic: never` in the Helm values file, C-VEN behaves like before in its 22.5 version-- that is, the forward chain on Node is open, and custom iptables rules must be used to enforce traffic in this chain.

Updates for Core for Kubernetes 5.2.1

Kubelink

Resolved Issues

- **Kubelink can't start on OpenShift because of fsGroup 1001** (E-120425)
  When using Helm Chart 5.2.0 on OpenShift, Kubelink would not start because of fsGroup 1001.

C-VEN

Resolved Issues

In an early version of these Release Notes issues E-119682 and E-119110 were incorrectly listed as being resolved.

- **NodePort access is working when it should be blocked** (E-120655)
  NodePort traffic was being always allowed, with or without a rule allowing the traffic from an external resource to the NodePort service. This issue was fixed by adding missing legacy iptables command line utilities to the UBI9-based C-VEN.
- **Move C-VEN base image to a smaller image** (E-118492)
  C-VEN now uses a UBI9-micro image as its base image, using the current latest version 9.4-15.

What's New in Release 5.2.0

- **"Wait for Policy" Feature**
  With a new Wait For Policy feature, CLAS-enabled Kubelink can be configured to automatically and transparently delay the start of an application container in a pod until a policy is properly applied to the pod. This feature replaces the local policy convergence controller, the Illumio readiness gate. A readiness gate required adding the `readinessGates.conditionType` into the spec YAML file of the Kubernetes Workload. Instead, Wait For Policy uses an automatically injected init container, with no change of the user application needed. When enabled, Wait For Policy synchronizes the benefit of Kubernetes automatic container creation with the protection of proper policy convergence into the new container.
  For more information, see "Wait For Policy" Feature [139].
- **CLAS Flat Network Support**
  Starting in version 5.2.0, the Kubelink Operator supports flat network CNIs in CLAS mode, a feature that was previously only available in non-CLAS mode. This update includes compatibility with flat network types such as Azure CNI Pod Subnet and Amazon VPC CNI. To enable a flat network CNI, set the `networkType` parameter to `flat` in the Helm Chart's `illumio-values.yaml` file during installation.
  Also note that in CLAS-enabled flat networks, if a pod communicates with a virtual machine outside the cluster using private IP addresses, you must enable the annotation `meta.illumio.podIPObservability`. This is a scenario in which the virtual machine is in a private network and has an IP address from the same range as cluster nodes and pods. In this case, the PCE needs to know the private IP address of the pod to be able to open a connection on the virtual machine. The main benefit of CLAS is that the PCE no longer directly manages individual pods, so the implementation expects a specific annotation on such pods. Traffic between such private IPs will be blocked without this annotation, and will appear in the UI as blocked.
  In this case, when the application communicates through private IPs, add the following annotation so that Kubelink can then report the private IPs of Kubernetes Workloads to the PCE:

  ```
  metadata:
      annotations:
          meta.illumio.podIPObservability: "true"
  ```
- **Kubelink Support Bundle**
  To assist the Illumio Support team with more details for troubleshooting, Kubelink now provides a support bundle that collects up to 2 GB of logs, metrics, and other data inside its pod. Future versions will add the option to upload these support bundles to the PCE.

Currently, you must copy this support bundle by running the script `/support_bundle.sh` inside the Kubelink pod. The script generates debug data, creates a gzipped tar archive using stdout as output, and encodes this data using Base64.

Use the following command to generate and transfer the Kubelink support bundle from its pod: (Note that the backslash (\) character is included to indicate the continuation of a long command line that will be truncated by the right margin of this document in PDF form.)

```
kubectl --namespace illumio-system exec deploy/illumio-kubelink \
-- /support_bundle.sh | base64 --decode > /tmp/kubelink_support.tgz
```

Send the resulting compressed archive file to Illumio Support when requested.

- **Base OS Upgraded to UBI9**

  The base OS has been upgraded to Red Hat Universal Base Image 9 (micro UBI9 for Kubelink, mini UBI9 for C-VEN).

  > **IMPORTANT**
  >
  > **Important Notice:** With the base image upgrade for both Kubelink and C-VEN, you must adjust resource allocations according to the guidance described below in the "Resource Allocation Guidelines [137]" section. You must ensure that resources are updated prior to the upgrade to achieve optimal performance, and to avoid any potential degradation in product performance.

- **Enhanced Pod Stability for Kubelink and C-VEN**

  To address the challenge of pod eviction during Kubernetes cluster issues or space shortages, Kubelink was previously the first pod to be evicted, which led to failures in policy enforcement. Recognizing the critical need for stability, Helm Chart version 5.2.0 introduces default priority classes for both Kubelink and C-VEN. Kubelink is now assigned the priority class of `system-cluster-critical`, while C-VENs receive `system-node-critical`. This implementation significantly enhances the resilience of your deployments, ensuring that key components remain operational even under resource constraints.

- **Changes to Supported Orchestration Platforms and Components in 5.2.0**

  The 5.2.0 release contains several changes to supported platforms and components. For full details, see Kubernetes Operator OS Support and Dependencies on the Illumio Support portal (log in required).

Resource Allocation Guidelines

New resource allocation guidelines have been developed to help configure deployments to achieve optimal performance and cost-efficiency.

These guidelines are grouped into the following general deployment sizes:

- **Small-scale:** Customers with limited Kubernetes deployments and moderate workloads.
- **Medium-scale:** Customers with moderate-sized Kubernetes environments and growing workloads.
- **Large-scale:** Customers with extensive Kubernetes deployments and high-performance requirements.

The following variables determine the deployment sizes listed above:

- Number of nodes per cluster
- Total number of workloads per cluster
- Total policy size per cluster

Set the `resources` values in the appropriate pod spec (Kubelink or C-VEN) `yaml` file under the `storage` section, as shown in the following example:

```
storage:
  sizeGi: 1
  resources:
    limits:
      memory: 600Mi
    requests:
      memory: 500Mi
      cpu: 500m
```

If you have two parameters that match one category, and a third parameter that matches another, it's important to select the category based on the highest value among them.

For instance, if the number of nodes per cluster is 8, and the total number of Kubernetes workloads is 500, but the average size of the policy is 1 Gi, the resource allocation should align with the large-scale resource allocation. This ensures that your resources are appropriately scaled to meet the demands of your workloads, optimizing performance and stability.

In practice, monitor these resources, and if usage is at 80% of these limits, then consider increasing.

NOTE that amounts are expressed in mebibytes (Mi) and gibibytes (Gi) and not in megabytes (MB) or gigabytes (GB).

Small-scale resource allocation

Customer Category	Nodes per Cluster	Total K8s Workloads	Total Policy Size	
Small-scale	1 - 10	0 - 1000	0 - 1.5 Mi	
Resources		**C-VEN**	**Kubelink**	**Storage**
Requests	CPU	0.5	0.5	0.5
Requests	memory	600 Mi	500 Mi	500 Mi
Limits	CPU	1	1	1
Limits	memory	700 Mi	600 Mi	600 Mi
Volumes	size limits	n/a	n/a	1 Gi

Medium-scale resource allocation

Customer Category	Nodes per Cluster	Total K8s Workloads	Total Policy Size	
Medium-scale	10 - 20	1000 - 5000	1.5 Mi - 500 Mi	
Resources		**C-VEN**	**Kubelink**	**Storage**
Requests	CPU	2	2	1
Requests	memory	3 Gi	5 Gi	5 Gi
Limits	CPU	3	2	2
Limits	memory	5 Gi	7 Gi	7 Gi
Volumes	size limits	n/a	n/a	5 Gi

Large-scale resource allocation

Customer Category	Nodes per Cluster	Total K8s Workloads	Total Policy Size	
Large-scale	20+	5000 - 8000	500 Mi - 1.5 Gi	
Resources		**C-VEN**	**Kubelink**	**Storage**
Requests	CPU	2	3	1
Requests	memory	6 Gi	10 Gi	10 Gi
Limits	CPU	3	4	2
Limits	memory	8 Gi	12 Gi	12 Gi
Volumes	size limits	n/a	n/a	10 Gi

"Wait For Policy" Feature

With a new *Wait For Policy* feature, CLAS-enabled Kubelink can be configured to automatically and transparently delay the start of an application container in a pod until a policy is properly applied to that container. This synchronizes the benefit of automatic container creation with the protection of proper policy convergence into the new container.

This Wait For Policy feature replaces the existing local policy convergence controller, also known as a readiness gate. A readiness gate required manually adding the `readinessGate` condition into the spec of the Kubernetes Workload. Instead, Wait For Policy uses an automatically injected init container, which requires no change to the user application.

Behavior

When Wait For Policy is enabled, Kubelink creates a new `MutatingWebhookConfiguration`. This webhook injects an Illumio init container into every new pod. Now a new pod lifecycle consists of the following sequence of actions:

1. Kubernetes creates a pod.
2. The pod creation request is intercepted by a mutating webhook.
3. Kubernetes requests MutatingAdmissionWebhook Controller running in Kubelink.
4. Controller returns with a new pod patched with an Illumio init container.
5. Init container starts in the pod, and periodically checks the policy status of the pod using the Kubelink status server.
6. At the same time, Kubelink is preparing a policy for the new pod, and is sending the policy to the pod's C-VEN.
7. The C-VEN applies policy to the pod, and sends an acknowledgment to Kubelink.
8. Kubelink reports that the policy is now applied to the init container.
9. The Init container exits, and allows the original container to start.
10. If a policy is not applied within the configured time (see Configuration [140] section for Helm Chart `waitForPolicy.timeout` parameter), the init container exits anyway, and allows the original container to start.

The Illumio init container must be accessible from all namespaces that use Wait for Policy. An easy way to ensure this accessibility is to make init available from a public repository. However, a private repository can be used if you manage the secret deployment properly, such as by deploying init from the same repository as all other containers, or by using a secret management tool.

Configuration

The Wait For Policy feature is disabled by default. To enable it, change the `waitForPolicy: enabled` value to `true` in the Helm Chart `illumio-values.yaml` file. The following is the default Helm Chart configuration for Wait For Policy:

```
## Wait for Policy - Illumio delays the start of Pods until policy is
applied
waitForPolicy:
  ## @param waitForPolicy.enabled Enable Wait for Policy feature
  enabled: false
  ## @param waitForPolicy.ignoredNamespaces List of namespaces where
Illumio
  ## doesn't delay start of Pods. kube-system and
  ## illumio-system name are ignored by Kubelink for this feature by
default,
  ## even if not specified in this list.
  ignoredNamespaces:
    - kube-system
    - illumio-system
  ## @param waitForPolicy.timeout How long will pods wait for policy, in
seconds
  timeout: 130
```

Pods starting in namespaces listed in `ignoredNamespaces` start immediately, without an Illumio init container injected into them. The namespaces `kube-system` and `illumio-system` are always ignored by the MutatingAdmissionWebhook Controller running in Kubelink, even if those are not specified in the configuration. The default value of `ignoredNamespaces` contains `kube-system` and `illumio-system` for reference, and can be extended with custom namespaces.

The `timeout` value is a total allowed run time of the init container. After this time elapses, the init container exits even if policy is not applied, and allows the original container to start.

Updates for Core for Kubernetes 5.2.0

Kubelink

Resolved Issues

- **Helm: pull secret to quay gets created even if no credentials are set** (E-119659)
  Helm chart now creates Illumio pull secret only if credentials are specified and also externally passed secret names are included.
- **Kubelink: error concurrent map read and map write** (E-119626)
  Kubelink was restarted because previous container exited with the message "`fatal error concurrent map read and map write`."
- **Kubelink: Update base image to address vulnerabilities** (E-119429)
  The Unified Base Image was upgraded to address CVE-2023-45288.
- **Kubelink needs to have higher priority assigned to avoid going to evicted state** (E-113920)
  If the Kubernetes cluster encounters problems or runs out of space, Kubelink was the first pod to be put into the evicted state, which caused policy enforcement to fail. To prevent permanent eviction, in Helm chart version 5.2.0 the Kubelink Deployment and C-VEN DaemonSets are assigned priority classes by default -- `system-cluster-critical` for Kubelink and `system-node-critical` for C-VENs.

C-VEN

Resolved Issues

- **CVEN: Update base image to address vulnerabilities** (E-119428)
  The 23.4 C-VEN Unified Base Image was upgraded to the latest UBI9 to address vulnerabilities described in CVE-2014-3566, CVE-2014-3566, CVE-2014-3566, CVE-2022-3358, and CVE-2023-27533.
- **Cannot deploy C-VEN to GKE when using default OS** (E-116506)
  For GKE clusters, when using the default cluster OS (Container-Optimized OS from Google), the node filesystems are read-only. This prevented C-VEN from mounting `/opt/illumio_ven_data` and writing into it for persistent storage.

  To resolve this issue, a new variable `` `cven.hostBasePath `` was added to the 5.2.0 Helm Chart to specify where the C-VEN DaemonSet mounts its data directory. The default value is `/opt`. Use this variable to specify where the C-VEN DaemonSet mounts its data directory. If using a Container-Optimized OS, you can set the directory to `/var`.
- **[CVEN]: Failed to load policy** (E-115231)
  The log message "`Error: Failed to load policy`" was appearing during scenarios that were obvious or expected. The log level for this message has been changed from Error to Info.

- **Re-adding node does not re-pair it** (E-98120)
  When deleting and then re-adding the same node, the node would not reappear, and its policy disappeared.

Illumio Core for Kubernetes Release Notes 5.1

Published: September 4, 2024

Core for Kubernetes 5.1.10

Compatible PCE Versions: 23.5.10 and most later releases

Current Illumio Core for Kubernetes Version: 5.1.10, which includes:

- C-VEN version: 23.3.1
- Kubelink version: 5.1.10
- Helm Chart version: 5.1.10

Before deploying any Illumio Core for Kubernetes 5.1.x version, confirm your PCE version supports it. For example, currently Illumio Core for Kubernetes versions 5.1.0 and 5.1.2 are supported **only** with PCE versions 23.5.10 (for On Premises customers) or 24.1.x (for SaaS customers), but NOT on PCE versions 23.5.1 or 23.6.0, or any lower versions. For complete compatibility details, see the Kubernetes Operator OS Support and Dependencies page on the Illumio Support Portal.

Illumio Core release numbering uses the following format: “a.b.c-d+e”.

- “a.b”: Standard or LTS release number, for example, “2.2”
- “.c”: Maintenance release number, for example, “.1”
- “-d”: Optional descriptor for pre-release versions, for example, “preview2”

Limitations

- **NodePort**
  The following limitations exist regarding NodePort policy enforcement and flows:
  - Only NodePort Services with `externalTrafficPolicy` set to "`cluster`" are supported. (This is the default and most frequently used value for this setting.)
  - When writing rules to allow traffic to flow from external (to the cluster) entities and NodePort Service, the source side of the rule must contain all nodes in the cluster.
    For example, given the following setup:
    - Worker nodes in the cluster are labeled as Role: Worker Node
    - Clients accessing the Service running in the Kubernetes cluster are labeled Role: Client
    - The NodePort Service is labeled Role: Ingress
    Normally, the rule would be written as Role: Client -> Role: Ingress. However, for thisrelease the rule must also include all nodes in the cluster to work correctly: Role: Client + Role: Worker Node -> Role: Ingress.
- **Flat Network support in CLAS mode**
  Using EKS or AKS in a flat network topology, such as EKS with AWS VPC CNI or AKS with Azure CNI, is not supported in CLAS-enabled clusters.

Updates for Core for Kubernetes 5.1.10

Kubelink

Resolved Issues

- **Last updated policy timestamp for C-VENs reflects Kubernetes Workload policy changes** (E-118372)
  The last updated policy timestamp on C-VENs now updates after a C-VEN successfully updates the policy for its pods.
- **Unexpected Potentially Blocked traffic in Explorer (CLAS mode)** (E-116105)
  In CLAS environments, some allowed traffic flows were wrongly reported as Potentially Blocked because of missing IP sets in the firewall test database.

Updates for Core for Kubernetes 5.1.7

Kubelink

Resolved Issues

- **Kubelink: policy service blocked when agent disconnects while receiving policy message** (E-117099)
  In some situations, policies stopped being sent due to a policy channel lock after C-VEN disconnected while receiving a policy update.
- **Kubelink: policy service blocked if one agent is not reading policy message** (E-116967)
  In some situations, policies stopped being sent after a C-VEN became unresponsive.
- **Kubelink can't save sets because of message size limit** (E-116825)
  Policy updates were being interrupted when large policy sets were being sent. The message size has been increased to permit larger policy transmissions .
- **Kubelink: workload events processing is slowed down by policy updates** (E-116706)
  The processing of workload events from Kubernetes sometimes became slow when handling thousands of Kubernetes Workloads, or the policy PCE requests were taking too long, or if there was no previous policy version in storage.
- **Kubelink sends wrong workload href in policy ACK request** (E-116640)
  In some CLAS-enabled clusters that host large numbers of workloads, the Kubernetes Workloads page showed an old policy apply date. Kubelink incorrectly sent a policy ACK for some Kubernetes Workloads with the host workload URI. The PCE responded with a 406 error, and a "no policy" ACK was stored.

Updates for Core for Kubernetes 5.1.3

Kubelink

Resolved Issues

- **Kubelink can't save policy to storage** (E-116539)
  Kubelink could not store cluster policy due to storage size limitations. To permit increased storage sizes, the Helm chart now includes new `resources` values under the `storage` component, as well as under `cven` and `kubelink` (note that amounts are in MiB not MB, and GiB not GB):

```
kubelink:
  resources:
    limits:
      memory: 500Mi
    requests:
      memory: 200Mi
      cpu: 200m

cven:
  resources:
    limits:
      memory: 300Mi
    requests:
      memory: 100Mi
      cpu: 250m

storage:
  resources:
    limits:
      memory: 500Mi
    requests:
      memory: 200Mi
      cpu: 100m
```

- **Pod to pod flows and pod labels are missing from Explorer search results** (E-116271, E-116272)
  In CLAS-enabled clusters, Explorer was not showing pod labels, only workload labels. In addition, Explorer did not return some traffic flows, even when trying with label-based search, or port-based search, or even searching using workload labels + pod labels. Also, pod traffic was being mapped to workloads.

Updates for Core for Kubernetes 5.1.2

Kubelink

Resolved Issues

- **Helm Chart: etcd storage size limit** (E-115417)

Kubelink in CLAS mode uses etcd as a local cache for policy and runtime data. The Helm Chart now accepts a new variable called `storage.sizeGi` to set the size (in GiB not GB) of ephemeral storage. The default value is 1.

- **Kubelink - Unable to process policy with custom iptables rules** (E-115250)
  Kubelink in CLAS mode failed to process policy received from the PCE when custom iptables rules were present, producing the error message "json: cannot unmarshal object into Go struct field."
- **Kubelink to PCE connectivity issues - connection reset by peer** (E-115049)
  CLAS-enabled Kubelink was entering degraded mode too soon because of PCE connectivity problems. Now Kubelink also retries requests after network and OS errors, which avoids premature degraded mode entry.
- **C-VEN reporting potentially blocked traffic between worker nodes** (E-114691)
  CLAS processing of outbound rules to a ClusterIP Service replaced the "All Services" destination in the rule with actual ports from the Kubernetes Service. If a destination label included a Kubernetes Service, this caused a missing iptables rule between nodes.
- **Max policy message size between Kubelink and C-VEN is too small** (E-113714)
  The default gRPC message size was set to too small of a value, which caused C-VENs to reject policy messages that were larger than this value. The default gRPC message size is now larger, to avoid this problem.

Updates for Core for Kubernetes 5.1.0

What's New in the 5.1.0 Release

The following are new and changed items in the 5.1.0 release from the previous releases of C-VEN and Kubelink:

- **New CLAS architecture option**
  Kubelink now can be deployed with a Cluster Local Actor Store (CLAS) module, which manages flows from C-VENs to PCE, and policies from PCE to C-VENs. The CLAS-enabled Kubelink tracks individual pods, and when they are created or destroyed, instead of this being communicated directly to the PCE. To migrate from an existing (non-CLAS) environment to a CLAS-enabled one, set the `clusterMode` parameter to `migrateLegacyToClas` in your deployment YAML file (typically named `illumio-values.yaml`). See the `README.md` file accompanying the Helm Chart for full details on this and other Helm Chart parameters.
- **Workloads more closely match Kubernetes architecture**
  In CLAS-enabled environments, workloads are now conceptually tied to their containers, instead of being referred to in context of their pods, which more closely matches Kubernetes practice. To reflect this change, such workloads in CLAS environments are called *Kubernetes Workloads*, regardless of what containers have been spun up or destroyed to run the applications. In non-CLAS environments, the existing term *Container Workloads* is still used as in prior releases, corresponding to Pods. In mixed environments (with both non-CLAS and CLAS-enabled clusters), the PCE UI shows both Container Workloads and Kubernetes Workloads, as appropriate.
- **Degraded mode for CLAS-enabled Kubelink**
  If a CLAS-enabled Kubelink detects that its connection with the PCE becomes unavailable (for example, due to connectivity problems or an upgrade), Kubelink by default enters a *degraded mode*. In this degraded mode, new Pods of existing Kubernetes Workloads get the latest policy version cached in CLAS storage. When Kubelink detects a new Kubernetes Workload with exactly the same label sets and in the same namespace as an existing

Kubernetes Workload, Kubelink delivers the existing, cached policy to Pods to this new Workload. If Kubelink cannot find a cached policy (that is, when labels of a new Workload do not match those of any existing Workload in the same namespace), Kubelink delivers a “fail open” or “fail closed” policy based on the Helm Chart parameter `degradedModePolicyFail`. The degraded mode can also be turned on or off by the Helm Chart parameter `disableDegradedMode`.

- **Illumio annotations in CLAS mode specified on the workload and not on Pod's template**
  Illumio annotations when in CLAS mode are now specified on the Kubernetes Workload and not on the pod's template.
- **Docker support dropped**
  The Docker CRI is no longer supported as of the 5.0.0 release of Illumio Core for Kubernetes.

C-VEN

Resolved Issue

- **Permanently delete Kubernetes Workloads after certain period when they are unpaired** (E-112362)
  Kubernetes Workloads (from a CLAS environment) are pruned from the PCE one day (by default) after they are unpaired. The length of time that elapses (in seconds) before this pruning occurs is configurable with the `vacuum_entities_wait_before_vacuum_seconds` parameter, which is set in the PCE `agent.yml` file. The default value for this parameter is 86400 (24 hours).

Known Issues

- **When C-VEN starts first, a 404 from PCE when getting CLAS token** (E-109259)
  When C-VEN is started first, it tries to contact the PCE in order to obtain CLAS token, but receives a 404 error. This is expected behavior for this scenario, which is only momentary. Kubelink eventually starts normally, and C-VEN obtains the CLAS tokens as expected.
- **Helm install fails with Helm version 3.12.2 but works with 3.10** (E-108128)
  When installing with Helm version 3.12.2, the installation fails with a YAML parse error.
  Workaround: Use Helm version 3.10, or version 3.12.3 or later.
- **Re-adding node does not re-pair it** (E-98120)
  After deleting a node and re-adding the same node, the node does not reappear, and previously established policy disappears from the node.
  Workaround: Uninstall and re-install Illumio Core for Kubernetes from scratch with the node present.

Kubelink

Resolved Issues

- **CLAS: NodePort - pod rules are not removed after disabling rule** (E-111689)
  After disabling a NodePort rule that opens it to outside VMs, iptable entries for pods with a virtual service's targetPort were not being removed as expected. Now the pod no longer remains opened. Host iptables are removed, so traffic does not go through, and the pod ports are properly closed.
- **CLAS - The etcd pod crashes when node reboots** (E-106236)
  The etcd pod would crash if one of the nodes in the cluster was rebooted.

Known Issues

- **CLAS-mode Kubelink pod gets restarted once when deploying Illumio Core for Kubernetes** (E-109284)

  The Kubelink pod is restarted after deploying Illumio Core for Kubernetes in CLAS mode.

  There is no workaround. Kubelink runs properly after this single restart.
- **CLAS: Container Workload Profile label change is not applied to Kubernetes Workloads, only to Virtual Services** (E-109168)

  When removing labels in a Container Workload Profile, existing Kubernetes Workloads that are managed by that profile do not have their labels changed automatically to labels based on annotations. These existing Kubernetes Workloads must be updated with the `kubectl apply` command for the labels change to take effect. New Kubernetes Workloads created after the profile label change will have the new labels.

  This works as designed.

Security Information for Core for Kubernetes 5.1

For information about security issues, security advisories, and other security guidance pertaining to this release, see Illumio’s Knowledge Base in Illumio's Support portal.

Illumio Core for Kubernetes Release Notes 5.0.0

About Illumio Core for Kubernetes 5.0

These release notes describe the resolved issues, known issues, and related information for the 5.0.0 release of Illumio Core for Kubernetes, formerly known as Illumio Containerized VEN, or C-VEN. Illumio Core for Kubernetes also includes the related required component, Kubelink. Because of this heritage, many references to this product as "C-VEN" are still used throughout the documentation.

Document Last Revised: January 2024

Product Version

Compatible PCE Versions: 23.5.0+A1 and later releases

Current Illumio Core for Kubernetes Version: 5.2.1, which includes:

- C-VEN version: 23.4.0
- Kubelink version: 5.2.1
- Helm Chart version: 5.0.0

Illumio Core release numbering uses the following format: “a.b.c-d+e”.

- “a.b”: Standard or LTS release number, for example, “2.2”
- “.c”: Maintenance release number, for example, “.1”
- “-d”: Optional descriptor for pre-release versions, for example, “preview2”

What’s New in C-VEN and Kubelink

The following are new and changed items in this release from the previous releases of C-VEN and Kubelink:

- **New CLAS architecture option**
  Kubelink now can be deployed with a Cluster Local Actor Store (CLAS) module, which manages flows from C-VENs to PCE, and policies from PCE to C-VENs. The CLAS-enabled Kubelink tracks individual pods, and when they are created or destroyed, instead of this being communicated directly to the PCE. To migrate from an existing (non-CLAS) environment to a CLAS-enabled one, set the `clusterMode` parameter to `migrateLegacyToClas` in your deployment YAML file (typically named `illumio-values.yaml`). See the `README.md` file accompanying the Helm Chart for full details on this and other Helm Chart parameters.
- **Workloads more closely match Kubernetes architecture**
  In CLAS-enabled environments, workloads are now conceptually tied to their containers, instead of being referred to in context of their pods, which more closely matches Kubernetes practice. To reflect this change, such workloads in CLAS environments are called *Kubernetes Workloads*, regardless of what containers have been spun up or destroyed to

run the applications. In non-CLAS environments, the existing term *Container Workloads* is still used as in prior releases, corresponding to Pods. In mixed environments (with both non-CLAS and CLAS-enabled clusters), the PCE UI shows both Container Workloads and Kubernetes Workloads, as appropriate.

- **Illumio annotations in CLAS mode specified on the workload and not on Pod's template**

  Illumio annotations when in CLAS mode are now specified on the Kubernetes Workload and not on the pod's template.
- **Docker support dropped**

  The Docker CRI is no longer supported as of this 5.0.0 release of Illumio Core for Kubernetes.

NodePort Limitations

- **NodePort**

  Here are some limitations around NodePort policy enforcement and flows:
  - Only NodePort Services with `externalTrafficPolicy` set to "`cluster`" are supported. (This is the default and most frequently used value for this setting.)
  - When writing rules to allow traffic to flow from external (to the cluster) entities and NodePort Service, the source side of the rule must contain all nodes in the cluster.

    For example, given the following setup:

    - Worker nodes in the cluster are labeled as Role: Worker Node

    - Clients accessing the Service running in the Kubernetes cluster are labeled Role: Client

    - The NodePort Service is labeled Role: Ingress
  - Normally, the rule would be written as Role: Client -> Role: Ingress. However, for this beta1 release the rule must also include all nodes in the cluster to work correctly: Role: Client + Role: Worker Node -> Role: Ingress.

Updates for Core for Kubernetes 5.0.0-LA

C-VEN

Resolved Issues

- **Scaling a Deployment with changed labels was not being updated on PCE** (E-107274)

  After deploying a workload with a non-existing label, create labels on the PCE and wait a few minutes before updating and applying the YAML to change the number of replicas. The deployment was not properly updated on the PCE. This issue is resolved.

Known Issues

- **When C-VEN starts first, a 404 from PCE when getting CLAS token** (E-109259)

  When C-VEN is started first, it tries to contact the PCE in order to obtain CLAS token, but receives a 404 error. This is expected behavior for this scenario, which is only momentary. Kubelink eventually starts normally, and C-VEN obtains the CLAS tokens as expected.
- **Helm install fails with Helm version 3.12.2 but works with 3.10** (E-108128)

  When installing with Helm version 3.12.2, the installation fails with a YAML parse error.

  Workaround: Use Helm version 3.10, or version 3.12.3 or later.
- **Re-adding node does not re-pair it** (E-98120)

After deleting a node and re-adding the same node, the node does not reappear, and previously established policy disappears from the node.

Workaround: Uninstall and re-install Illumio Core for Kubernetes from scratch with the node present.

Kubelink

Resolved Issues

- **CLAS on IKS with Calico, the flow of ClusterIP is not displayed correctly** (E-109238)

  In a CLAS environment on IKS with Calico, when running traffic to a clusterIP service from a pod, flows were being displayed incorrectly. Sometimes flows were incorrectly shown as Allowed. Other times, flows that should not be present were being shown as Blocked. This issue is resolved.
- **Kubernetes cluster falsely detected as an OpenShift cluster** (E-107910)

  After deployment, Kubelink falsely detected a Kubernetes cluster as an OpenShift cluster based on misinterpretations of installed VolumeReplicationClass and VolumeReplications APIs on the cluster. This issue is resolved.
- **Problem when label from PCE was deleted after Kubelink starts** (E-107779)

  When creating a new workload on PCE, Kubelink uses cached or preloaded labels to label a workload. However, if the label was deleted before the workload was actually created, the PCE responded with a 406 status error. This issue is resolved.
- **Kubelink did not properly apply label mappings with PCE using two-sided management ports** (E-105391)

  Label mappings were not properly applied when using the LabelMap CRD if the PCE used two-sided management ports. This issue is resolved.

Known Issues

- **CLAS: NodePort - pod rules are not removed after disabling rule** (E-111689)

  After disabling a NodePort rule that opens it to outside VMs, iptables entries for pods with a virtual service's targetPort are not removed as expected. The pod is still opened. Host iptables are removed, so traffic does not go through, but the pod ports stay opened towards original IPs.

  There is no workaround available.
- **Non-CLAS mode: Failed to clean up the pods** (E-109687)

  After deleting a non-CLAS container cluster, the cluster gets deleted but Container Workloads are not deleted, and remain present.
- **CLAS-mode Kubelink pod gets restarted once when deploying Illumio Core for Kubernetes** (E-109284)

  The Kubelink pod is restarted after deploying Illumio Core for Kubernetes in CLAS mode.

  There is no workaround. Kubelink runs properly after this single restart.
- **CLAS: Container Workload Profile label change is not applied to Kubernetes Workloads, only to Virtual Services** (E-109168)

  In CLAS environments, after changing a label in a Container Workload Profile, the Kubernetes Workloads that are managed by that Profile do not have their labels changed as expected. No changes to these Kubernetes Workloads occur even when the Profile is changed to "No Label Allowed;" the original labels remain in the Kubernetes Workloads. However, Virtual Services managed by that profile do successfully have their labels changed properly.

  No workaround is available.
- **CLAS - The etcd pod crashes when node reboots** (E-106236)

The etcd pod crashes if one of the nodes in the cluster is rebooted.

There is no workaround available.

Security Information for Core for Kubernetes 5.0.0-LA

For information about security issues, security advisories, and other security guidance pertaining to this release, see Illumio's Knowledge Base in Illumio's Support portal.

Illumio Core for Kubernetes Release Notes 4.3.0

What's New in Kubernetes 4.3.0

These release notes describe the resolved issues and related information for the 4.3.0 release of Illumio Core for Kubernetes, formerly known as Illumio Containerized VEN, or C-VEN.

Illumio Core for Kubernetes also includes the related required component, Kubelink. Because of this heritage, many references to this product as "C-VEN" are still used throughout the documentation.

Here are the new and changed items in this release:

- **New Kubelink 3.3.1**
  This Kubernetes 4.3.0 release includes an upgraded Kubelink component, version 3.3.1 .
- **New C-VEN 22.5.14**
  This Kubernetes 4.3.0 release includes an upgraded C-VEN component, version 22.5.14.

> **NOTE**
> C-VEN 22.5.14 requires PCE version 22.5.0 or later, and supports PCE 23.3.0 or later.

Security Information

For information about security issues, security advisories, and other security guidance pertaining to this release, see Illumio's Knowledge Base in Illumio's Support portal.

Base Image Upgraded

The C-VEN base OS image is upgraded to minimal UBI for Red Hat Linux 7.9-979.1679306063, which is available at https://catalog.redhat.com/software/containers/ubi7/ubi-minimal/5c3594f7dd19c775cddfa777.

Customers are advised to upgrade to Core for Kubernetes 4.1.0 or higher for these security fixes.

Product Version

Compatible PCE Versions: 22.5.0 and later releases

Current Illumio Core for Kubernetes Version: 4.3.0, which includes:

- C-VEN version: 22.5.14
- Kubelink version: 3.3.1
- Helm Chart version: 4.3.0

Illumio Core release numbering uses the following format: “a.b.c-d+e”.

- “a.b”: Standard or LTS release number, for example, “2.2”
- “.c”: Maintenance release number, for example, “.1”
- “-d”: Optional descriptor for pre-release versions, for example, “preview2”

Updates for Core for Kubernetes 4.3.0

C-VEN

Resolved Issues

- **C-VEN support report does not contain container workload firewalls** (EYE-106932)
  VEN support reports for C-VENs were missing the active firewall information for all container workloads. This issue is resolved. Support reports now include full firewalls from each network namespace, as gathered by `iptables-save` and `ipset list` output.
- **Conntrack tear-down for containers with policy updates** (E-44832)
  Although policy was changed to block a container workload from talking to another, traffic was still passing between the workloads, due to a conntrack connection remaining incorrectly active. This issue is resolved. Conntrack connections on sessions affected by a policy change are now properly torn down.

Known Issue

- **C-VENs not automatically cleaned up after AKS upgrade** (E-103895)
  After upgrading an AKS cluster, sometimes a few duplicate C-VENs might not be automatically removed as part of the normal upgrade process, and remain in the PCE as "non-active." Note there is no compromise to the security or other functionality of the product.
  Workaround: Manually prune the extra unmigrated C-VENs from the PCE by clicking the **Unpair** button for each of them.

Kubelink

Resolved Issue

- **Kubelink does not pair with PCE when a separate management port is used** (E-107001)
  Kubelink would crash after start when the PCE had `front_end_management_https_port` set to 9443 instead of 8443, because of a missing label_map URL. This issue is resolved.

Known Issue

- **Kubelink does not properly apply label mappings with PCE using two-sided management ports** (E-105391)
  Label mappings are not properly applied when using the LabelMap CRD if the PCE uses two-sided management ports.

Workaround: Use the label map feature only with a PCE that uses only one management port.

Legal Notice

Resources

- Legal information
- Trademarks statements
- Patent statements
- License statements

Contact Information

- Contact Illumio
- Contact Illumio Legal
- Contact Illumio Documentation