



Illumio Core[®]

Version 21.1.0

C-VEN

Version 21.1.0

Illumio Core for Kubernetes and OpenShift

May 2021

25000-100-21.1.0

Legal Notices

Copyright © 2021 Illumio 920 De Guigne Drive, Sunnyvale, CA 94085. All rights reserved.

The content in this documentation is provided for informational purposes only and is provided "as is," without warranty of any kind, expressed or implied of Illumio. The content in this documentation is subject to change without notice.

Product Versions

PCE Version: 21.1.0 (Standard Release)

VEN Version: 21.1.0 (Standard Release)

C-VEN Version: 21.1.0

NEN Version: 2.1.0

Kubelink Version: 2.0.0

FlowLink Version: 1.1.2

Standard versus LTS Releases

21.1.0-PCE and 21.1.0-VEN are standard releases. Illumio will designate a version of 21.x.x as a Long Term Support (LTS) release. Do not upgrade the PCE or VEN to 21.1.0 if your environment requires an LTS release.

For information on Illumio software support for Standard and LTS releases, see [Versions and Releases](#) on the Illumio Support portal.

Resources

Legal information, see <https://www.illumio.com/legal-information>

Trademarks statements, see <https://www.illumio.com/trademarks>

Patent statements, see <https://www.illumio.com/patents>

License statements, see <https://www.illumio.com/eula>

Open source software utilized by the Illumio Core and their licenses, see [Open Source Licensing Disclosures](#)

Contact Information

To contact Illumio, go to <https://www.illumio.com/contact-us>

To contact the Illumio legal team, email us at legal@illumio.com

To contact the Illumio documentation team, email us at doc-feedback@illumio.com

Contents

Chapter 1 Overview of Containers in Illumio Core	6
About This Guide	6
How to Use This Guide	6
Recommended Skills	7
Concepts	7
Containerized VEN	7
Container Cluster	7
Container Workloads	8
Container Workload Profiles	8
Kubelink	9
Virtual Services	9
Workloads	10
Architecture	10
Chapter 2 Deployment	14
Host and Cluster Requirements	14
General Requirements	15
Supported Configurations for On-premises and IaaS	15
Privileges	16
Prepare Your Environment	17
Unique Machine ID	18
Create Labels	18
Push Kubelink and C-VEN Images to Your Container Registry	19
Create Illumio Namespace	20
Authenticate Kubernetes Cluster with Container Registry	21
Create a ConfigMap to Store Your Root CA Certificate	22
Configure Calico in Append Mode	26
Create a Container Cluster in the PCE	27
Create a Container Cluster	27
Configure a Container Workload Profile Template	28
Deploy Kubelink in Your Cluster	29
Prerequisites	29
Configure Kubelink Secret	29
Deploy Kubelink	31
Deploy C-VENs in Your Cluster	35

Prerequisites	35
Create a Pairing Profile for Your Cluster Nodes	36
Configure C-VEN Secret	36
Deploy C-VEs	38
Re-Label Your Cluster Nodes	43
Chapter 3 Configure Labels for Namespaces, Pods, and Services	45
Use Container Workload Profiles	45
Configure New Container Workload Profiles	46
Dynamic Creation of a Profile	47
Manual Pre-creation of a Profile	48
Labels Restrictions for Kubernetes Namespaces	50
Using Annotations	56
Deployments	57
Services	57
DaemonSets and Replicasets	61
Chapter 4 Configure Security Policies for Containerized Environment	65
IP and FQDN Lists	65
FQDN Services for Kubernetes	65
IP Lists for Kubernetes	66
FQDN Services for OpenShift	66
IP Lists for OpenShift	67
Rules for Kubernetes or OpenShift Cluster	68
Kubernetes	68
OpenShift	70
Rules for Containerized Applications	73
Access Services from within the Cluster	73
Access Services from Outside the Cluster	75
Outbound Connections	78
Liveness Probes	79
Rules for Persistent Storage	80
Kubernetes	81
OpenShift	81
Firewall Coexistence on Pods	82
Chapter 5 Upgrade and Uninstallation	85
Upgrade Illumio Components	85

Upgrade Kubelink	85
Upgrade C-VEN	86
Uninstall Illumio from Your Cluster	86
Unpair C-VEs	86
Delete Illumio Resources	87
Chapter 6 Reference: General	89
<hr/>	
Troubleshooting	89
Failed Authentication with the Container Registry	89
Kubelink Pod in CrashLoopBackOff State	92
Container Cluster in Error	93
Pods and Services Not Detected	95
Pods Stuck in Terminating State	95
Enable Firewall Coexistence	95
Known Limitations	97

Overview of Containers in Illumio Core

This chapter contains the following topics:

About This Guide	6
Concepts	7
Architecture	10

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

About This Guide

How to Use This Guide

This guide explains how to deploy the Illumio Core with Kubernetes or OpenShift on your distributed, on-premises systems.

The guide provides the details to complete the following tasks:

- Preparing your environment
- Creating a container cluster in the PCE
- Deploying Kubelink and C-VEs in your cluster
- Configuring labels for namespaces, pods, and services
- Configuring security policies for containerized environments
- Upgrading and Uninstalling the C-VE in your containerized environments

Recommended Skills

This guide assumes that you have a thorough understanding of the following technologies and concepts:

- 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. For more information, see [Get Started with Docker](#).
- Red Hat OpenShift Container Platform. For more information, see [OpenShift Documentation](#).
- Kubernetes concepts, such as clusters, Pod, and services. For more information, see [Kubernetes Documentation](#).

Concepts

This section describes some key concepts of the solution.

Containerized VEN

Containerized VEN (C-VEN) is an Illumio-provided software component, which 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. 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 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](#).

The C-VEs 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-VEs are deployed on all nodes in the cluster. In cloud-managed deployments, C-VEs are deployed only on the Worker nodes and not on the Master nodes (Master nodes are not managed by Cloud customers).

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 Workloads

Container workloads are containers or the smallest resource that can be assimilated to a container in an orchestration system. In the context of Kubernetes and OpenShift, a Pod is a container workload. Similar to workloads, 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, 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 state (Build, Test or Enforced) in container workload profiles.

Container Workload Profiles

A container workload profile maps to a Kubernetes namespace and defines:

- Policy state (Build, Test, or Enforced) for the Pods and services that belong to the namespace.
- Labels (Role, Application, Environment, and Location) assigned to the Pods and services.

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.

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

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.

Kubelink

Kubelink is a software component provided by Illumio to make the integration between the PCE and Kubernetes easier. 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-VEs 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-VEs with the particular container cluster by matching a unique identifier of the underlying OS called `machine-id` reported by each C-VE with the one reported by the Kubernetes cluster.

Kubelink is not in the critical path for normal Pod creation. When Pods start, stop, scale up or down, or nodes reboot, these events are all discovered by the C-VE and the C-VE provides enough information to the PCE to be able to immediately receive the security policy. There is no dependency on Kubelink to keep that information in sync.

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.

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.

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:

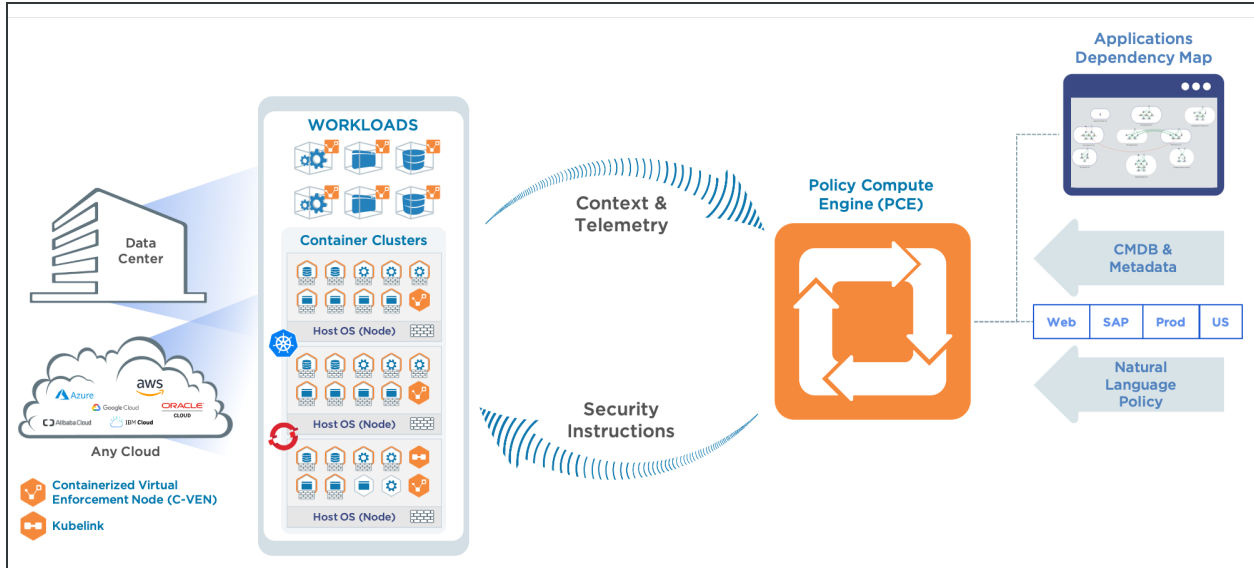
- One or more Master Node(s) - In the control plane of the cluster, these nodes control and manage the cluster.
- One or more Worker Node(s) - In the data plane of the cluster, these nodes run the application (containers).

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.

Architecture

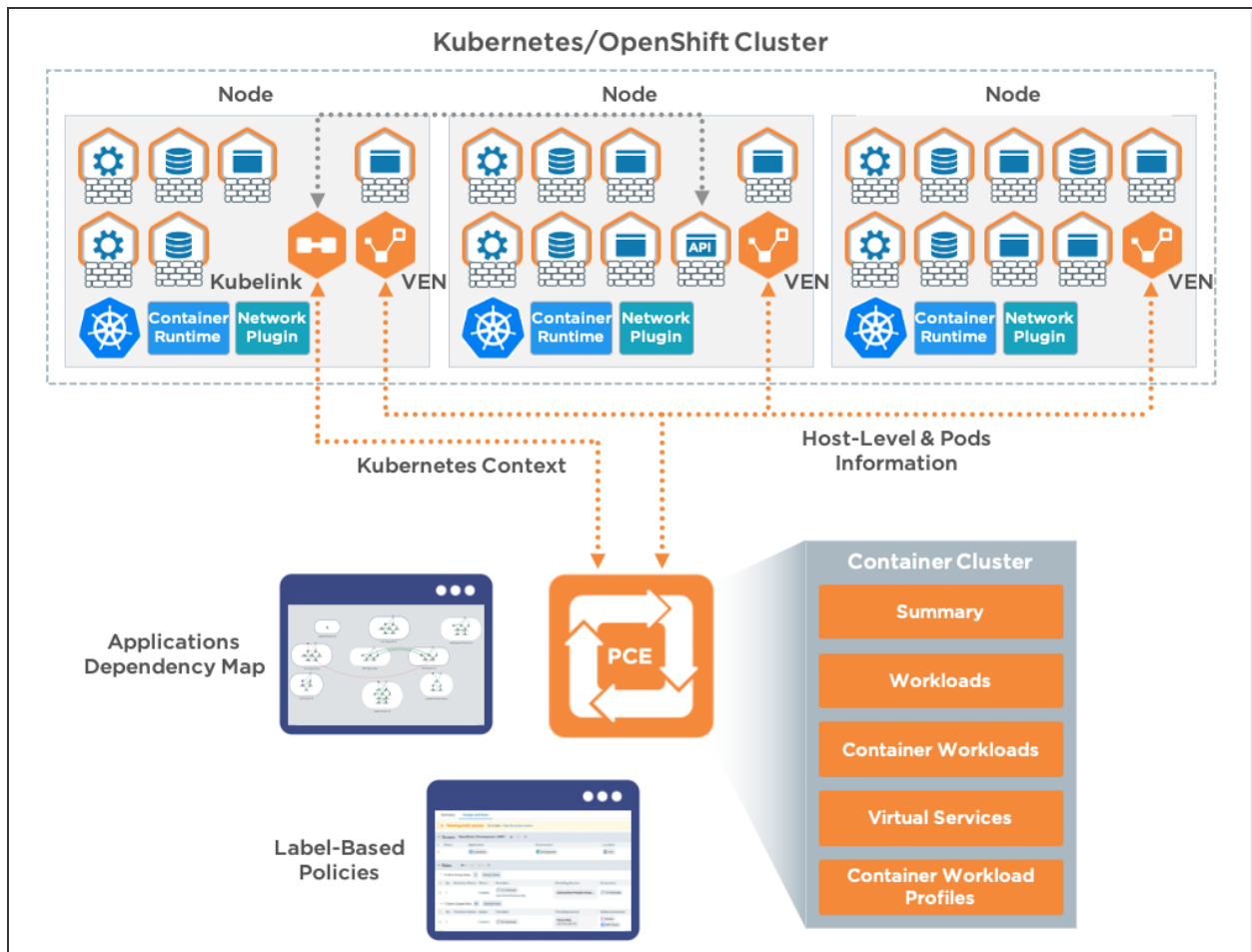
With the increased adoption of containers, the threat of unauthorized lateral movement from vulnerabilities and exploits increases considerably in the east-west attack surface. In addition, consumers and providers may be other containers, bare-metal servers, or virtual machines running on-premises or 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.

Illumio Core for containers orchestrated with Kubernetes or OpenShift, uses the following architecture:



Kubernetes-based orchestration platforms such as, native Kubernetes, OpenShift, AWS EKS, Microsoft Azure AKS, and others integrate with Illumio 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.
- Containerized VEN (C-VEN) - An Illumio software component that provides visibility and enforcement on the nodes and the Pods.

Once these components are deployed in the cluster, they both report the following information to the Policy Compute Engine (PCE):

- Summary - Information about the Kubernetes cluster and Illumio components deployed.
- Workloads - Information about Kubernetes nodes.
- Container Workloads - Information about Kubernetes Pods.
- Virtual Services - Information about Kubernetes services.

- Container Workload Profiles - Information about Kubernetes namespaces and policies.

Illumio Core visibility and enforcement occur at the Pod level in Kubernetes and OpenShift, with policies programmed into the iptables in the namespace provided by the Pod. This means only the Pods can be segmented but containers inside a Pod cannot be segmented. The Pod is represented as a single container workload in the PCE, with the C-VEN providing details about the containers that are a part of the Pod.

Deployment

This chapter contains the following topics:

Host and Cluster Requirements	14
Prepare Your Environment	17
Create a Container Cluster in the PCE	27
Deploy Kubelink in Your Cluster	29
Deploy C-VEs in Your Cluster	35
Re-Label Your Cluster Nodes	43

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



NOTE:

The installation process is the same for Kubernetes and OpenShift, except a few steps or manifest files 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.

General Requirements

- Illumio Core for containers:
 - PCE version 21.1.0
 - C-VEN version 21.1.0
 - Docker image (illumio-ven-21.1.0-xxxx.tgz)
 - Secret file (illumio-ven-secret.yml)
 - Configuration file (illumio-ven-kubernetes.yml or illumio-ven-openshift.yml)
 - Kubelink version 2.0.0
 - Docker image (kubelink-image.tar.gz)
 - Secret file (illumio-kubelink-secret.yml)
 - Configuration file (illumio-kubelink-kubernetes.yml or illumio-kubelink-openshift.yml)
- Private container registry to pull container images from

Supported Configurations for On-premises and IaaS

In this release, Illumio supports the following on-premises and IaaS configurations:

Orchestration Platforms	Versions	Node OS	Container Runtimes	Network Plugins	Kube-proxy Mode
Kubernetes	<ul style="list-style-type: none"> • 1.20 • 1.19 • 1.18 	<ul style="list-style-type: none"> • CentOS 7.9 • Ubuntu 18.04 	<ul style="list-style-type: none"> • Docker 19.03.x • Containerd 1.4.x • CRI-O 1.16.x 	<ul style="list-style-type: none"> • Calico (IP-in-IP) 3.16.x • Flannel (Overlay) 0.13.x 	iptables
IBM Cloud Kubernetes Service (IKS)	1.18	Ubuntu 18.04	Containerd 1.3.x	Calico (IP-in-IP) 3.13.x	iptables
OpenShift	<ul style="list-style-type: none"> • 4.6* (Pre-view) • 4.5* (Pre-view) 	<ul style="list-style-type: none"> • RHCOS 4.6 • RHCOS 4.5 	<ul style="list-style-type: none"> • CRI-O 1.18 • CRI-O 1.17 	OpenShift SDN (OVS)	iptables
OpenShift	3.11**	• RHEL 7.7	Docker 1.13.x	OpenShift SDN (OVS)	iptables

Orchestration Platforms	Versions	Node OS	Container Runtimes	Network Plugins	Kube-proxy Mode
		<ul style="list-style-type: none"> CentOS 7.7 			

*OpenShift 4.x platform is supported in Preview and with C-VEN 19.3.x and Kubelink 1.x.

**OpenShift 3.11 platform is supported only using the standard VEN 19.3.x on RHEL or CentOS with Kubelink 1.x and it does not require the C-VEN package to be deployed in the OpenShift cluster.

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 information, see [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-VEs 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 information, see [Kubernetes Documentation](#).

Prepare Your Environment

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 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](#).

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
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-21.1.0-7637.k8s.x86_64.tgz` package includes:

- 1 docker image
 - `illumio-ven-21.1.0-7637.tgz`
- 2 configuration files:
 - `illumio-ven-secret.yml`
 - `illumio-ven-kubernetes.yml`
 - `illumio-ven-openshift.yml`

Kubelink `illumio-kubelink-2.0.0.tar.gz` package includes:

- 1 docker image
 - `kubelink-image.tar.gz`
- 3 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-21.1.0-7637.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:2.0.0.665e02  
docker tag <illumio-ven-image-id> <docker-registry>/illumio-ven:21.1.0-7637
```

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:2.0.0.665e02  
docker push <docker-registry>/illumio-ven:21.1.0-7637
```

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

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](#), [Create a ConfigMap to Store Your Root CA Certificate](#), and [Configure Calico in Append Mode](#) 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 `kubect1` 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 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-----
MIIGSzCCBD0gAwIBAgIUAPw0NFPAivJW4YmKZ499eHZH3S8wDQYJKoZIhvcNAQEL
---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 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-----
MIIGSzCCBD0gAwIBAgIUAPw0NfPAivJW4YmKZ499eHZH3S8wDQYJKoZIhvcNAQEL
---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.

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

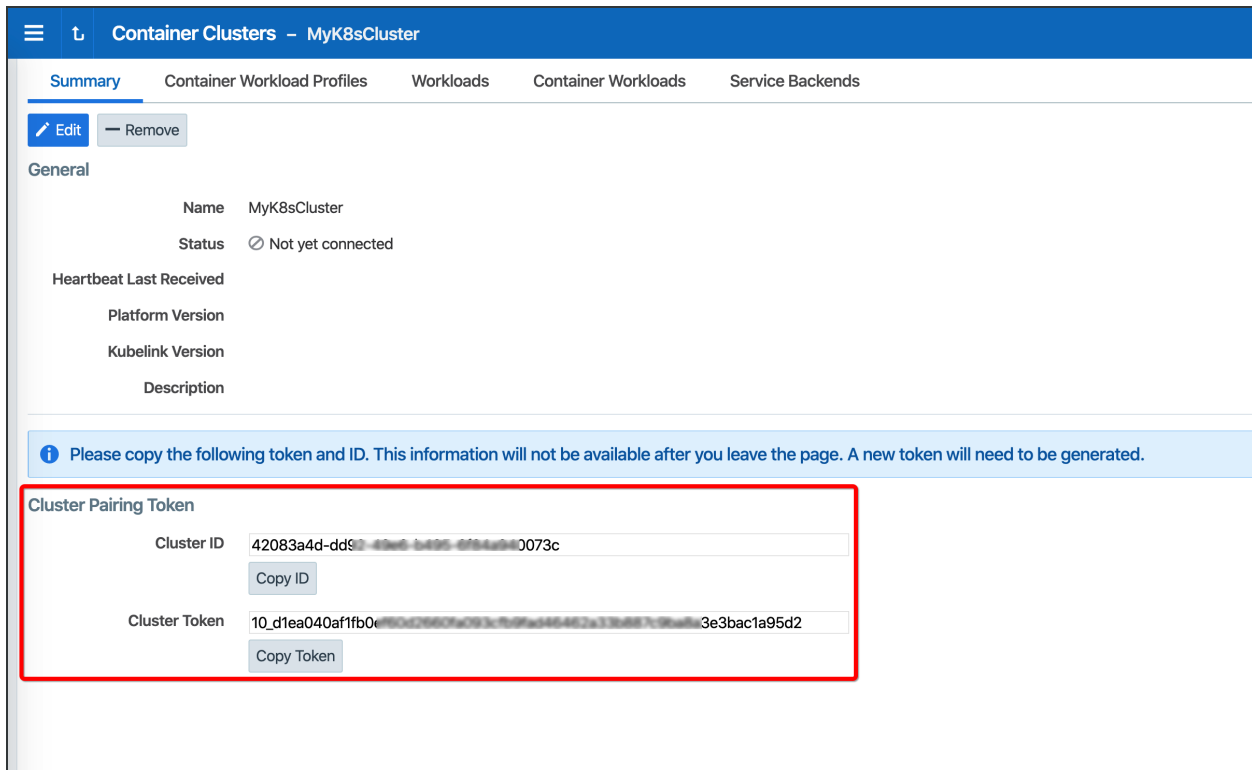
To provide visibility and enforcement to your containerized environment, you first need to create a container cluster in the PCE. Each container cluster maps to an existing Kubernetes or OpenShift cluster.

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.



The screenshot shows the 'Container Clusters - MyK8sCluster' summary page. It includes tabs for 'Summary', 'Container Workload Profiles', 'Workloads', 'Container Workloads', and 'Service Backends'. Under the 'General' section, the name is 'MyK8sCluster' and the status is 'Not yet connected'. A warning message indicates that the pairing token and ID will expire. The 'Cluster Pairing Token' section contains two rows: 'Cluster ID' with the value '42083a4d-dd90-4366-b495-df8a40073c' and a 'Copy ID' button, and 'Cluster Token' with the value '10_d1ea040af1fb0c-990a29907ac93c-f99fa446462c3134887199a33e3bac1a95d2' and a 'Copy Token' button.

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.



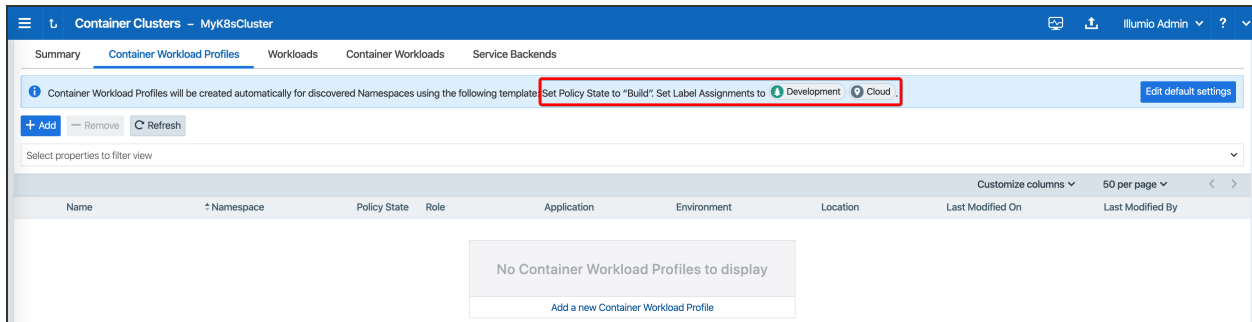
IMPORTANT:

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 [Labels Restrictions for Kubernetes Namespaces](#).

After you click OK, the following information is displayed:



Deploy Kubelink in Your Cluster

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

Prerequisites

- Kubelink deployment file provided by Illumio.
- Kubelink secret file provided by Illumio.
- Illumio's Kubelink docker image uploaded to your private docker registry.

Configure Kubelink Secret

This section 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. Open the Kubelink secret YAML file and modify the following keys that are listed under `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: `15643adc-ac09-40f2-be63-fd9a261f41cc`
 - c. `ilo_cluster_token` = Cluster Token from previous step. Example: `1_e94c116a4485ab1bb8560728afd6a332182b849c841297f63e73a87bf255cc96`

- 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'`

**IMPORTANT:**

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. For PCE deployments using a certificate signed with a private PKI, there is no need to set the `ignore_cert` key to `'false'`. For more details, see [Create a ConfigMap to Store Your Root CA Certificate](#).

The contents of a modified `illumio-kubelink-secret.yml` file are shown below.

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

apiVersion: v1
kind: Secret
metadata:
  name: illumio-config
  namespace: illumio-system
type: Opaque
stringData:
  ilo_server: https://mypce.example.com:8443 # Example:
https://mypce.example.com:8443
  ilo_cluster_uuid: 42083a4d-dd92-49e6-b495-6f84a940073c # Example: cc4997c1-
408b-4f1d-a72b-91495c24c6a0
  ilo_cluster_token: 10_
d1ea040af1fb0ef60d2660fa093cfb9fad46462a33b887c9ba8a3e3bac1a95d # 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 [Create a ConfigMap to Store Your Root CA Certificate](#) before deploying Kubelink.

2. Save the changes.
3. Create the Kubelink secret in your Kubernetes or OpenShift cluster.
 - Deploy Kubelink secret in Kubernetes:

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

- Deploy Kubelink secret in OpenShift:

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

4. Verify the Kubelink secret creation in your Kubernetes cluster.
 - Verify Kubelink secret in Kubernetes:

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

- Verify Kubelink secret in OpenShift:

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

Deploy Kubelink

Modify the Kubelink configuration file to point to the correct Docker image. The example in this document has `illumio-kubelink:2.0.0.665e02` uploaded to `registry.example.com`, so the image link in this example is: `registry.example.com/illumio-kubelink:2.0.0.665e02`

1. Edit the Kubelink configuration YAML file. The file name is `illumio-kubelink-kubernetes.yml` for a Kubernetes cluster or `illumio-kubelink-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 Kubelink configuration for Kubernetes to illustrate the image location.

```
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: illumio-kubelink
  namespace: illumio-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/illumio-kubelink:2.0.0.665e02
          imagePullPolicy: Always
          env:
            - name: ILO_SERVER
              valueFrom:
                secretKeyRef:
                  name: illumio-kubelink-config
                  key: ilo_server
```

3. (Optional) Reference your root CA certificate.

If you are 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. By default,

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

- 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](#) 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. Deploy Kubelink.

- To deploy Kubelink for Kubernetes:

```
kubectl apply -f illumio-kubelink-kubernetes.yaml
```

- To deploy Kubelink for OpenShift:

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

6. Verify your deployment.

- To check the Kubelink Pod status for Kubernetes:

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

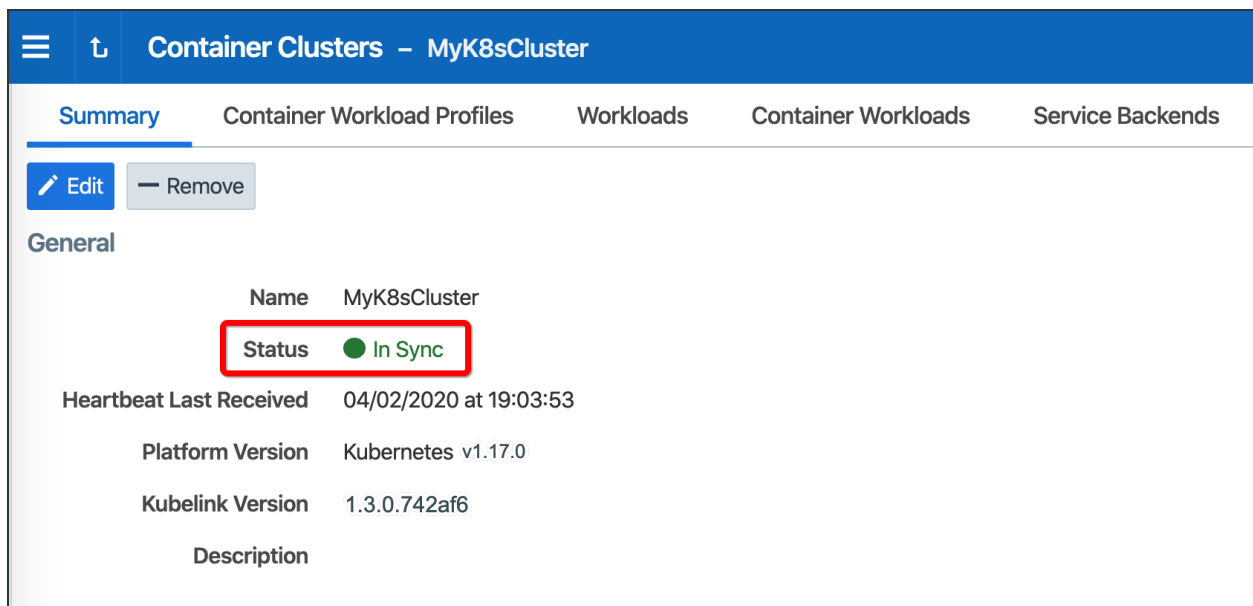
- To check the Kubelink Pod status for OpenShift:

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

The `illumio-kubelink-xxxxxxxx-xxxxx` Pod should be in the "Running" state.

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".



The screenshot shows the 'Container Clusters - MyK8sCluster' page in the Illumio PCE UI. The 'Summary' tab is selected, and the 'Status' is 'In Sync', which is highlighted with a red box. Other details include the Name 'MyK8sCluster', Heartbeat Last Received '04/02/2020 at 19:03:53', Platform Version 'Kubernetes v1.17.0', and Kubelink Version '1.3.0.742af6'.

General	
Name	MyK8sCluster
Status	● In Sync
Heartbeat Last Received	04/02/2020 at 19:03:53
Platform Version	Kubernetes v1.17.0
Kubelink Version	1.3.0.742af6
Description	

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.

Name	Namespace	Policy State	Role	Application	Environment	Location	Last Modified On	Last Modified By
	default	Build			Development	Cloud	04/02/2020, 19:03:53	Container Cluster
	ingress-nginx	Build			Development	Cloud	04/02/2020, 19:03:53	Container Cluster
	kube-node-lease	Build			Development	Cloud	04/02/2020, 19:03:53	Container Cluster
	kube-public	Build			Development	Cloud	04/02/2020, 19:03:53	Container Cluster
	illumio-system	Build			Development	Cloud	04/02/2020, 19:03:53	Container Cluster
	kubernetes-dashboard	Build			Development	Cloud	04/02/2020, 19:03:53	Container Cluster

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

Provision Status	Name	Service / Ports	Addresses	Role	Application	Environment	Location	Workloads	Container Workloads	Description
<input type="checkbox"/>	kubernetes-MyK8sCluster-default	6443 TCP	172.20.64.1:443 "MyK8sCluster container network"			Development	Cloud			
<input type="checkbox"/>	coredns-MyK8sCluster-kube-system	53 UDP 53 TCP 9153 TCP	172.20.64.3 "MyK8sCluster container network"			Development	Cloud			
<input type="checkbox"/>	dashboard-metrics-scraper-MyK8sCluster-kubernetes-dashboard	8000 TCP	172.20.80.218 "MyK8sCluster container network"			Development	Cloud			
<input type="checkbox"/>	kubernetes-dashboard-MyK8sCluster-kubernetes-dashboard	8443 TCP	172.20.98.106:443 "MyK8sCluster container network"			Development	Cloud			

Deploy C-VEs in Your Cluster



IMPORTANT:

Before deploying the C-VE, 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-VE 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`

Create a Pairing Profile for Your Cluster Nodes

Before deploying the C-VE 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.

Pairing Status	Name	Policy State	Role	Application	Environment	Location	Last Modified On	Last Modified By	Last Used On	Description
Running	Kubernetes Nodes	Build	Node	Kubernetes Cluster	Development	Cloud	03/22/2020, 22:39:13		02/04/2020, 03:42:52	



TIP:

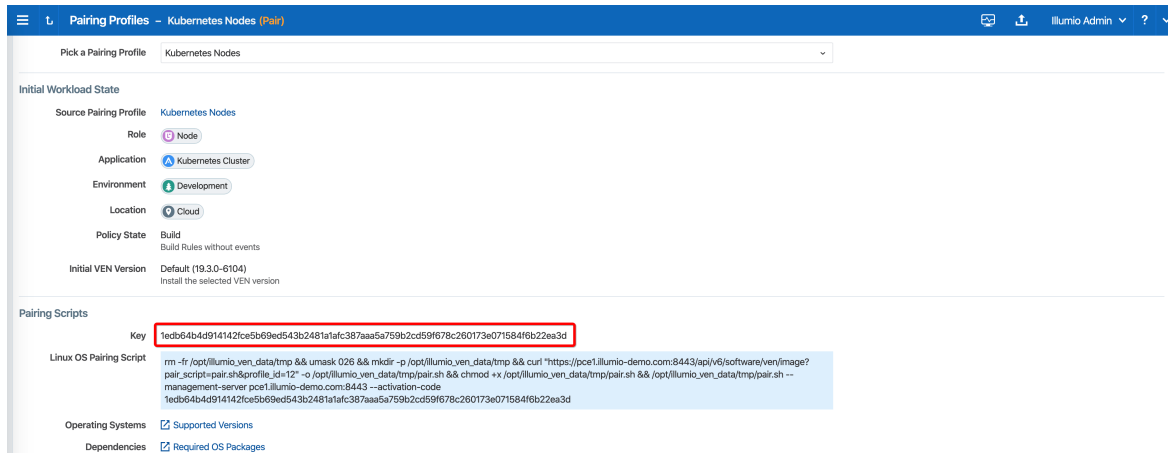
Illumio recommends all pairing profiles for Kubernetes nodes to *not* use Enforced policy state. Use Build or Test mode for initial configuration.

You should only move them into enforced state after you have completed all other configuration steps in this guide, such as setup Kubelink, discover services, and write rules.

Configure C-VE Secret

This section assumes that you have already created a Pairing Profile in the PCE. You will need the activation code for the C-VE 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 2013-2020 Illumio, Inc. All Rights Reserved.
#
# VEN 21.1.0-7637

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:
```

```
1edb64b4d914142fce5b69ed543b2481a1afc387aaa5a759b2cd59f678c260173e071584f6b22ea  
3d # 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-VE 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-VE secret creation in your cluster.
 - To verify the creation of the secret for Kubernetes:

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

- To verify the creation of the secret for OpenShift:

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

Deploy C-VEs

Modify the C-VE configuration file to point to the correct Docker image. The example in this document has `illumio-ven:21.1.0-7637` uploaded to `registry.example.com:443`, so the image link in this example is: `registry.example.com:443/illumio-ven:21.1.0-7637`

1. Edit the C-VE 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-VE configuration for Kubernetes or OpenShift to illustrate the image location.

```
#
# Copyright 2013-2020 Illumio, Inc. All Rights Reserved.
#
# VEN 21.1.0-7637

---
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:21.1.0-7637
  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](#) 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: > 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 release onwards, 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-VE 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-VE 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-VE 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-VE.
 - 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
```

- For OpenShift:

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

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

After C-VEs 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-VEs 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.

Policy State	Policy Sync	Name	Role	Application	Environment	Location	Last Applied Policy
Build	Active	master	Node	Kubernetes Cluster	Development	Cloud	04/02/2020, 23:18:46
Build	Active	worker1	Node	Kubernetes Cluster	Development	Cloud	04/02/2020, 23:18:47
Build	Active	worker2	Node	Kubernetes Cluster	Development	Cloud	04/02/2020, 23:18:46

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

Policy State	Policy Sync	Namespace/Project	Name	Role	Application	Environment	Location
Build	Active	illumio-system	illumio-kubelink-8548c6fb68-6rwxk			Development	Cloud
Build	Active	kube-system	coredns-58687784f9-h4pp2			Development	Cloud
Build	Active	kube-system	dns-autoscaler-79599df498-m55mg			Development	Cloud
Build	Active	kube-system	coredns-58687784f9-zmrfj			Development	Cloud
Build	Active	kubernetes-dashboard	kubernetes-dashboard-7b5bf5d559-zmrvq			Development	Cloud
Build	Active	kubernetes-dashboard	dashboard-metrics-scraper-566cddb686-vmwv2			Development	Cloud

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

Re-Label Your Cluster Nodes

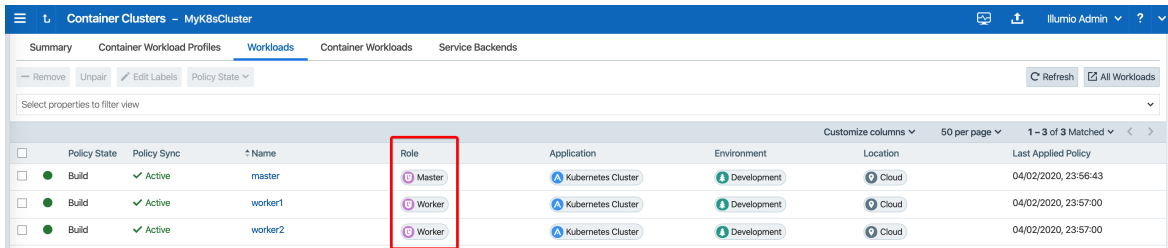


NOTE:
Re-labeling the cluster nodes is optional.

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).



	Policy State	Policy Sync	Name	Role	Application	Environment	Location	Last Applied Policy
<input type="checkbox"/>	Build	Active	master	Master	Kubernetes Cluster	Development	Cloud	04/02/2020, 23:56:43
<input type="checkbox"/>	Build	Active	worker1	Worker	Kubernetes Cluster	Development	Cloud	04/02/2020, 23:57:00
<input type="checkbox"/>	Build	Active	worker2	Worker	Kubernetes Cluster	Development	Cloud	04/02/2020, 23:57:00

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

This chapter contains the following topics:

Use Container Workload Profiles	45
Using Annotations	56

Once Kubelink is deployed on to 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. The Pods are represented in Illumio Core as Container Workloads. If Kubernetes services exist in the respective namespace, Illumio Core represents each service as an Illumio Core Virtual Service object.

This section describes how to change a namespace from unmanaged to managed and how to use custom annotations to add more context to your applications.

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 *Container Workload Policy State* (any state, except unmanaged).
 - c. Assign *Labels* (optional).
 - d. 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 "Build" 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).

Name	Namespace	Policy State	Role	Application	Environment	Location	Last Modified On	Last Modified By
app1	app1	Build		App1	Development	Cloud	04/03/2020, 24:31:25	@illumio.com
Default	default	Build		Default	Development	Cloud	04/02/2020, 23:33:51	@illumio.com
	ingress-nginx	Build			Development	Cloud	04/02/2020, 19:03:53	Container Cluster
	kube-node-lease	Build			Development	Cloud	04/02/2020, 19:03:53	Container Cluster
	kube-public	Build			Development	Cloud	04/02/2020, 19:03:53	Container Cluster
Kube-System	kube-system	Build		Kube-System	Development	Cloud	04/02/2020, 23:34:11	@illumio.com
Dashboard	kubernetes-dashboard	Build		Dashboard	Development	Cloud	04/02/2020, 23:34:50	@illumio.com
	namespace1	Build			Development	Cloud	04/03/2020, 08:42:08	Container Cluster

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.
See [Labels Restrictions for Kubernetes Namespaces](#).
4. After you are done, click **Save**.

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

Name	Namespace	Policy State	Role	Application	Environment	Location	Last Modified On	Last Modified By
app1	app1	Build		App1	Development	Cloud	04/03/2020, 24:31:25	@illumio.com
Default	default	Build		Default	Development	Cloud	04/02/2020, 23:33:51	@illumio.com
	ingress-nginx	Build			Development	Cloud	04/02/2020, 19:03:53	Container Cluster
	kube-node-lease	Build			Development	Cloud	04/02/2020, 19:03:53	Container Cluster
	kube-public	Build			Development	Cloud	04/02/2020, 19:03:53	Container Cluster
Kube-System	kube-system	Build		Kube-System	Development	Cloud	04/02/2020, 23:34:11	@illumio.com
Dashboard	kubernetes-dashboard	Build		Dashboard	Development	Cloud	04/02/2020, 23:34:50	@illumio.com
WebApp1	namespace1	Build		WebApp1	Development	Cloud	04/03/2020, 08:51:37	@illumio.com

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 *Container Workload Policy State*.
4. Assign *Labels* to the profile.
See [Labels Restrictions for Kubernetes Namespaces](#).
5. Click **Save**.

Please copy the following pairing key. This information will not be available after you leave the page. A new profile will need to be created.

Pairing Key

Key abc8aaffdb2101e13a9da02bf492badb8d09d5ce338af116d076aef77558afcd

Copy Key

6. 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.

Name	Namespace	Policy State	Role	Application	Environment	Location	Last Modified On	Last Modified By
app1	app1	Build		App1	Development	Cloud	04/03/2020, 24:31:25	@illumio.com
Default	default	Build		Default	Development	Cloud	04/02/2020, 23:33:51	@illumio.com
	ingress-nginx	Build			Development	Cloud	04/02/2020, 19:03:53	Container Cluster
	kube-node-lease	Build			Development	Cloud	04/02/2020, 19:03:53	Container Cluster
	kube-public	Build			Development	Cloud	04/02/2020, 19:03:53	Container Cluster
Kube-System	kube-system	Build		Kube-System	Development	Cloud	04/02/2020, 23:34:11	@illumio.com
	kubernetes-dashboard	Build		Dashboard	Development	Cloud	04/02/2020, 23:34:50	@illumio.com
WebApp1	namespace1	Build		WebApp1	Development	Cloud	04/03/2020, 08:51:37	@illumio.com
WebApp2		Build		WebApp2	Development	Cloud	04/03/2020, 09:21:17	@illumio.com

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

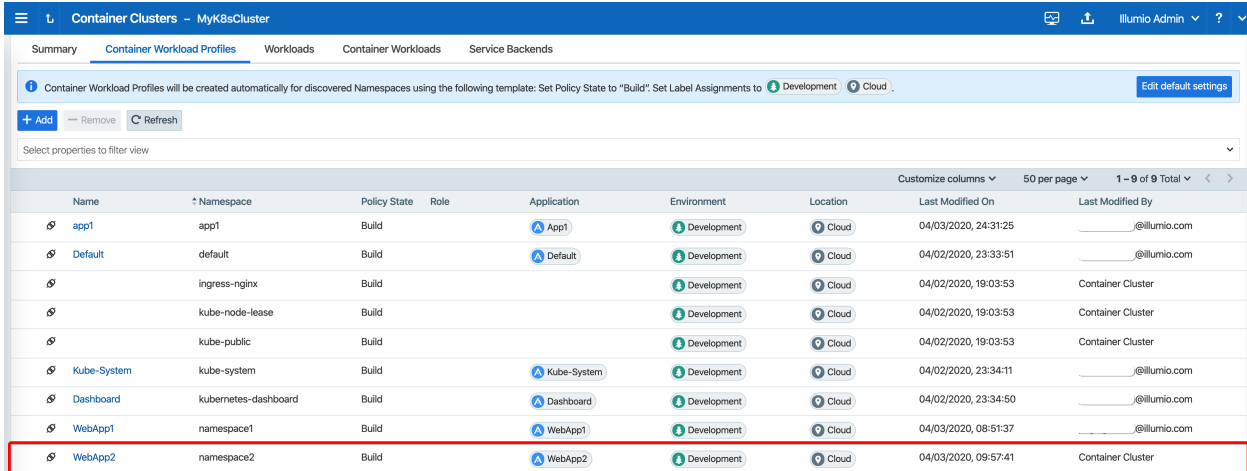
To edit the namespace configuration file to include the pairing key in order to authenticate this namespace with the PCE:

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 show below.

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

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



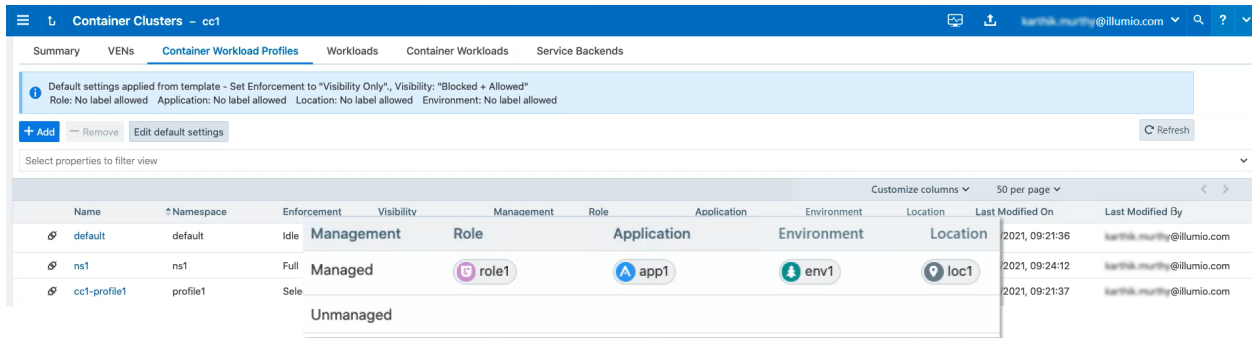
Name	Namespace	Policy State	Role	Application	Environment	Location	Last Modified On	Last Modified By
app1	app1	Build		App1	Development	Cloud	04/03/2020, 24:31:25	@illumio.com
Default	default	Build		Default	Development	Cloud	04/02/2020, 23:33:51	@illumio.com
	ingress-nginx	Build			Development	Cloud	04/02/2020, 19:03:53	Container Cluster
	kube-node-lease	Build			Development	Cloud	04/02/2020, 19:03:53	Container Cluster
	kube-public	Build			Development	Cloud	04/02/2020, 19:03:53	Container Cluster
Kube-System	kube-system	Build		Kube-System	Development	Cloud	04/02/2020, 23:34:11	@illumio.com
Dashboard	kubernetes-dashboard	Build		Dashboard	Development	Cloud	04/02/2020, 23:34:50	@illumio.com
WebApp1	namespace1	Build		WebApp1	Development	Cloud	04/03/2020, 08:51:37	@illumio.com
WebApp2	namespace2	Build		WebApp2	Development	Cloud	04/03/2020, 09:57:41	Container Cluster

Labels Restrictions for Kubernetes Namespaces

At a high level, creating policy for containerized applications functions in the same basic way as for other types of applications running on bare-metal servers and virtual machines protected by the Illumio Core. Container workloads are assigned four-dimensional labels to identify their roles, applications, environments, and locations (RAEL). These labels can then be used to apply security policies to specific parts of the containerized application environment. The PCE converts these label-based policies into rules that can be applied to the container workloads.

In the previous release, the PCE supported two options for assigning labels to container workloads:

- When creating or editing a container workload profile in the PCE web console or by using the Illumio Core REST API, an Illumio administrator assigned labels (RAEL) for the resources in that Kubernetes namespace.
- The Illumio administrator did not assign labels in the container workload profile. The DevOps/SRE team could use custom annotations in the service and deployment manifest files (YAML) to apply labels to the pods and services running in a namespace. On receiving this information from Kubelink, the PCE applied these labels to the container workloads, as long as the labels matched existing labels in the PCE.



Name	Namespace	Enforcement	Visibility	Management	Role	Application	Environment	Location	Last Modified On	Last Modified By
default	default	Idle	Management	Role	Application	Environment	Location		2021, 09:21:36	karthik.murthy@illumio.com
ns1	ns1	Full	Managed	role1	app1	env1	loc1		2021, 09:24:12	karthik.murthy@illumio.com
cc1-profile1	profile1	Sele	Unmanaged						2021, 09:21:37	karthik.murthy@illumio.com

These two ways of assigning labels for container workloads are sufficient for most container segmentation uses cases; however, this approach lacks the flexibility with label assignment for namespaces requested by Illumio customers. In this release, Illumio is providing an alternative between those two options by still allowing developer-s/DevOps teams to assign their own labels for Kubernetes pods and services, but at the same time restricting the list of labels that they can assign. Illumio administrators now have a way to control which labels can be assigned by the developers managing their Kubernetes environments.

Options for Assigning Labels with a Container Workload Profile

You assign labels with container workload profiles in a number of ways:

- By creating a new container workload profile; see Manual Pre-creation of a Profile in Illumio Core for Kubernetes and OpenShift.
- By editing a container workload profile that was dynamically created in the PCE when Kubelink imported a new Kubernetes namespace; see Dynamic Creation of a Profile in Illumio Core for Kubernetes and OpenShift.
- By specifying label assignments in the default settings for the container workload profile template; see Configure a Container Workload Profile Template in Illumio Core for Kubernetes and OpenShift.

In Core 21.1.0, you now have four options when setting labels with a container workload profile:

Labels

Any container annotation label is accepted by default. You can choose to restrict container annotations to one or more labels if needed, or assign your own label [?](#)

Role Use Container Annotations Assign Label None

Application Use Container Annotations Assign Label None

Environment Use Container Annotations Assign Label None

Location Use Container Annotations Assign Label None

- Do not allow a label for a specific label type (the “None” option).
- Allow developers to assign any label from Kubernetes for a specified label type (the “Use Container Annotations” option); so long as the labels match ones in the PCE.

In this release, the PCE web console page has changed. In the previous release, when the PCE administrator left the labels unassigned in the GUI or through the REST API, labels specified in annotations were used. In this release, the “Use Container Annotations” option is selected by default for all labels in a container workload profile (provided the default settings for the cluster are not configured).

- Specify a list of labels that are allowed for that label type. This capability is new in this release.
- Fix a label to a specific label for that label type (the “Assign Label” option).

The following example shows how the label assignments appear for namespaces:

Role	Application	Environment	Location
No label allowed	<input type="text" value="Any"/>	<input type="text" value="2 allowed"/>	<input type="text" value="loc1"/>
<input type="text" value="Any"/>	<input type="text" value="kube-system"/>	<input type="text" value="env1"/>	<input type="text" value="loc1"/>
<input type="text" value="Any"/>	<input type="text" value="Any"/>	<input type="text" value="Any"/>	<input type="text" value="Any"/>
<input type="text" value="Any"/>	<input type="text" value="Any"/>	<input type="text" value="Any"/>	<input type="text" value="Any"/>
<input type="text" value="Any"/>	<input type="text" value="kube-system"/>	<input type="text" value="env1"/>	<input type="text" value="loc1"/>

Example: Assigning Labels with a Container Workload Profile

The following example shows how you can use each of the 4 options:

Labels

Any container annotation label is accepted by default. You can choose to restrict container annotations to one or more labels if needed, or assign your own label [?](#)

Role Use Container Annotations Assign Label None

No label allowed

Application Use Container Annotations Assign Label None





Environment Use Container Annotations Assign Label None

Location Use Container Annotations Assign Label None

- Specifying a Role label in Kubernetes is not allowed; essentially, the Role label annotation (`com.illumio.role:`) is ignored when passed at runtime and reported by Kubelink to the PCE. The PCE ensures that a label is not assigned for the Role label key for the resources in this Kubernetes namespace.
- Developers can specify any label for Applications, so long as the label matches a preexisting label in the PCE.
- For Environment, a list of two labels (`env1` and `env2`) is available. Developers can set either of these labels in Kubernetes. If a developer sets another value for the Environment label as a Kubernetes annotation, the PCE considers it invalid and, as a result, a label is not assigned to that label key. Because the wrong label is assigned, the policy will not allow expected traffic from other services or applications with the Environment label `env1`.

Environment Use Container Annotations Assign Label None




Environment Label:

-  env1
-  env2
-  env3
-  unmanaged

4 Matching Results

- The Location label is fixed as the loc1 label. If a developer assigns another Location label (for example, loc3, which is a label in the PCE) or the developer leaves the Location label empty, the PCE overrides what the developer has specified in the annotation and the PCE assigns loc1 for the Location label.

The label assignments for that namespace appear as follows in the Container Clusters list in the PCE web console:

Role	Application	Environment	Location
No label allowed	 Any	 2 allowed	 loc1

For this example, you can see the label assignments mirrored in the Kubernetes annotation for the namespace:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-app-1
  namespace: demo
spec:
  replicas: 1
  selector:
    matchLabels:
      app: app1
  template:
    metadata:
      annotations:
        com.illumio.role: role1
        com.illumio.app: app1
        com.illumio.env: env1
        com.illumio.loc: loc100
      labels:
        app: app1
    spec:
      containers:
        - name: app1
          image: kodekloud/webapp-conntest
          ports:
            - containerPort: 8080
```

Where a developer set role1, app1, env1, and loc100 for the labels in the annotations. Kubelink passes this data to the PCE at runtime. The PCE ignores the Role label because it's not allowed. It accepts the Application and Environment labels. It ignores the loc100 label and uses loc1 instead.

In the Container Workloads tab, you can see how the label assignments are applied for the pod in this example:

Namespace/Project: **demo** × Select properties to filter view

Customize columns ▾ 50 per page ▾

Policy Sync	Namespace/Project	Name	Enforcement	Visibility	Role	Application	Environment	Location
Syncing	demo	web-app-1-55465549c4-5n77v	Visibility Only	Blocked + Allowed				



NOTE:

If a developer sets another value for the Environment label as a Kubernetes annotation, the PCE considers it invalid and, as a result, a label is not assigned to that label key. Because the wrong label is assigned, the policy will not allow expected traffic from other services or applications.

For example, in the Kubernetes annotations, a developer leaves the Environment label empty or specifies env100, the PCE uses the following labels for the namespace and you won't have policy for applications or services with the Environment label env1.

Namespace/Project: **demo** × Select properties to filter view

Customize columns ▾ 50 per page ▾

Policy Sync	Namespace/Project	Name	Enforcement	Visibility	Role	Application	Environment	Location
Syncing	demo	web-app-1-55465549c4-5n77v	Visibility Only	Blocked + Allowed				

Affect of Upgrading the PCE to Core 21.1.0

After upgrading your PCE to Core 21.1.0, the labels assignments for your Kubernetes namespaces are not impacted operationally. However, you will see changes in the PCE web console and in the REST API.

- The values set in the PCE in the previous Core release are unchanged and the “Assign Label” option is selected in the PCE web console and through the REST API.
- The values left open so that container annotations were used for label assignments are updated to the “Use Container Annotations” option and the label assignments won't be restricted by any settings in the PCE web console or through the REST API.

Using Annotations

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 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 will override 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 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.

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 in to 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

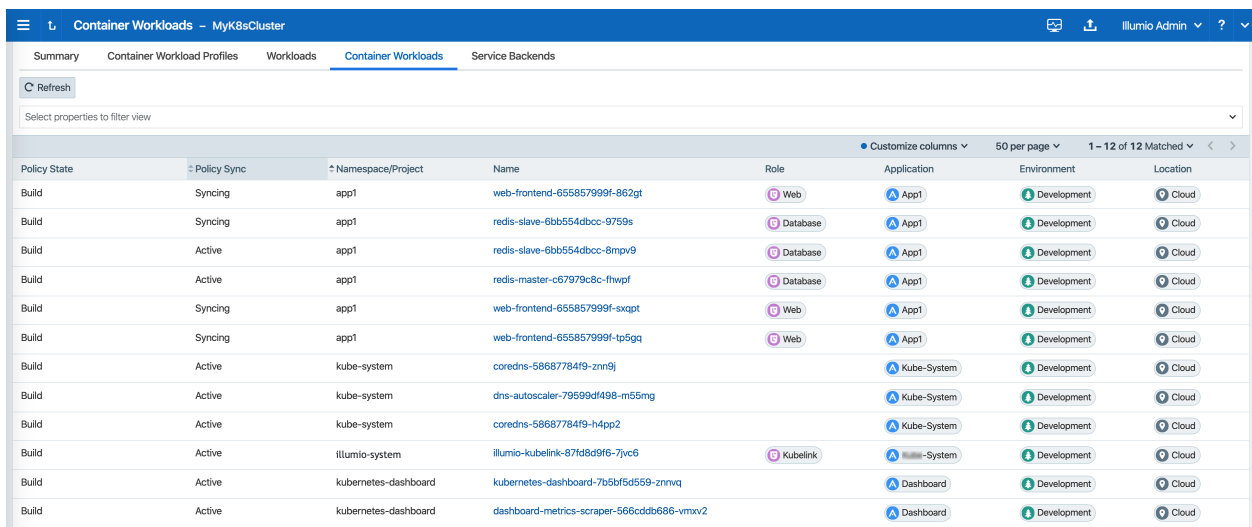
```
spec:
  replicas: 3
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      app: webapp1
      tier: frontend
  strategy:
    activeDeadlineSeconds: 21600
    resources: {}
    rollingParams:
      intervalSeconds: 1
      maxSurge: 25%
      maxUnavailable: 25%
      timeoutSeconds: 600
      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-ftwvpf	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-zrn9j		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	System	Development	Cloud
Build	Active	kubernetes-dashboard	kubernetes-dashboard-7b5bf5d559-znrnq		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 due to the Kubelink's dependency on the Pod hash templates to map resources together 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 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 add 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>

Configure Security Policies for Containerized Environment

This chapter contains the following topics:

IP and FQDN Lists	65
Rules for Kubernetes or OpenShift Cluster	68
Rules for Containerized Applications	73
Rules for Persistent Storage	80
Firewall Coexistence on Pods	82

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

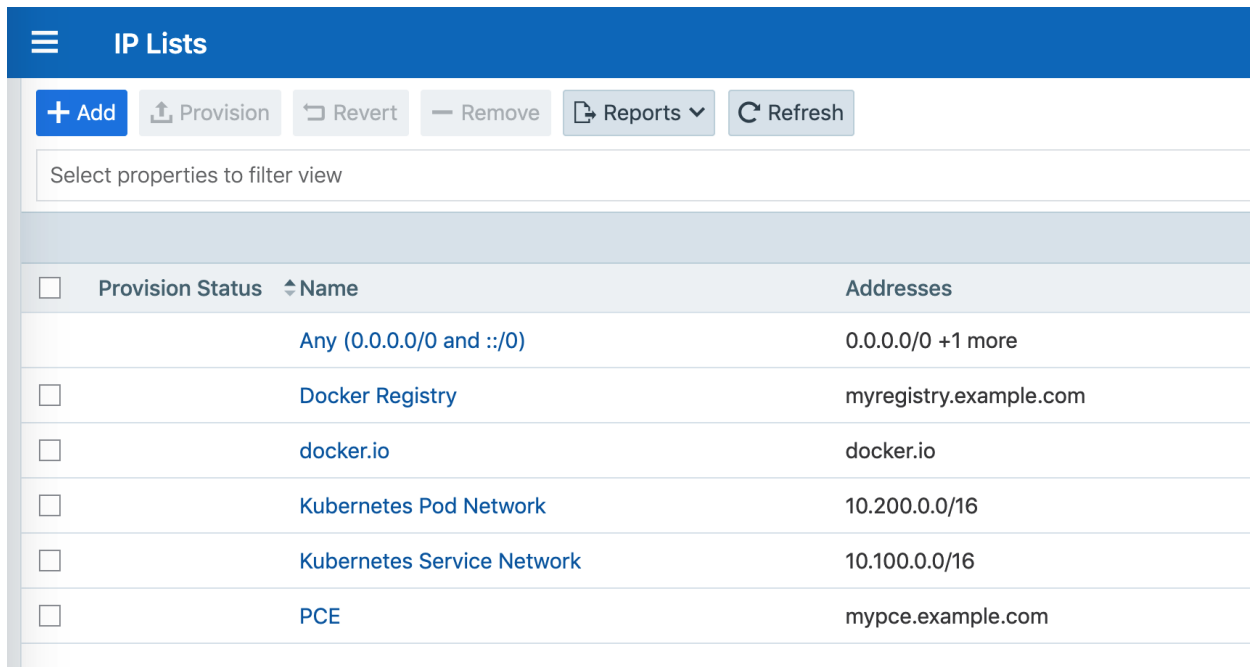
The PCE FQDN is required for Kubelink for example, mypce.example.com.

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.



IP Lists			
<input type="button" value="+ Add"/> <input type="button" value="Provision"/> <input type="button" value="Revert"/> <input type="button" value="Remove"/> <input type="button" value="Reports"/> <input type="button" value="Refresh"/>			
Select properties to filter view			
<input type="checkbox"/>	Provision Status	Name	Addresses
<input type="checkbox"/>		Any (0.0.0.0/0 and ::/0)	0.0.0.0/0 +1 more
<input type="checkbox"/>		Docker Registry	myregistry.example.com
<input type="checkbox"/>		docker.io	docker.io
<input type="checkbox"/>		Kubernetes Pod Network	10.200.0.0/16
<input type="checkbox"/>		Kubernetes Service Network	10.100.0.0/16
<input type="checkbox"/>		PCE	mypce.example.com

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

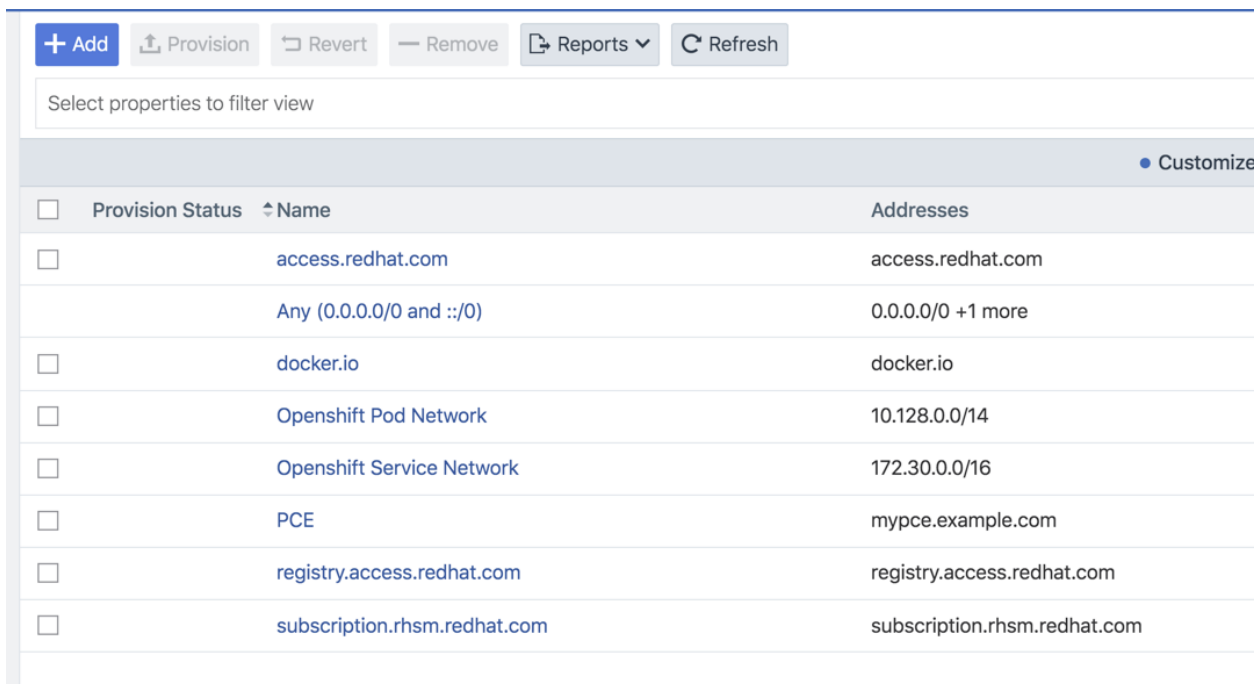
The PCE FQDN is required for Kubelink, for example, mypce.example.com.

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.



The screenshot shows a web interface with a toolbar containing buttons for '+ Add', 'Provision', 'Revert', 'Remove', 'Reports', and 'Refresh'. Below the toolbar is a search bar with the text 'Select properties to filter view'. The main content is a table with a 'Customize' button in the top right corner. The table has three columns: 'Provision Status', 'Name', and 'Addresses'. Each row has a checkbox in the 'Provision Status' column.

<input type="checkbox"/>	Provision Status	Name	Addresses
<input type="checkbox"/>		access.redhat.com	access.redhat.com
<input type="checkbox"/>		Any (0.0.0.0/0 and ::/0)	0.0.0.0/0 +1 more
<input type="checkbox"/>		docker.io	docker.io
<input type="checkbox"/>		Openshift Pod Network	10.128.0.0/14
<input type="checkbox"/>		Openshift Service Network	172.30.0.0/16
<input type="checkbox"/>		PCE	mypce.example.com
<input type="checkbox"/>		registry.access.redhat.com	registry.access.redhat.com
<input type="checkbox"/>		subscription.rhsm.redhat.com	subscription.rhsm.redhat.com



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.

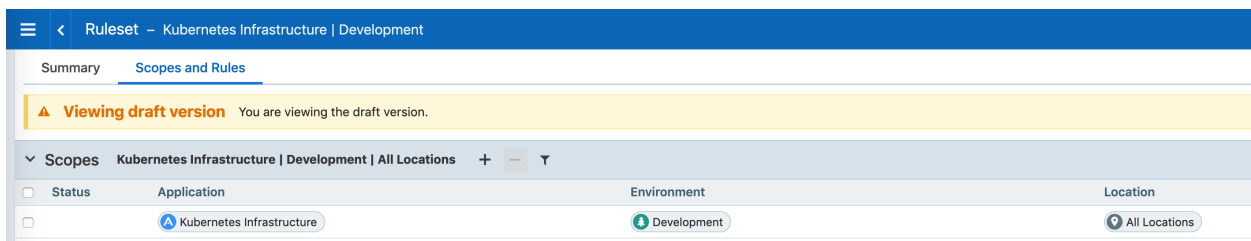
Rules for Kubernetes or OpenShift Cluster

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 oper-

Providers	Services	Consumers	Notes
			ations 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

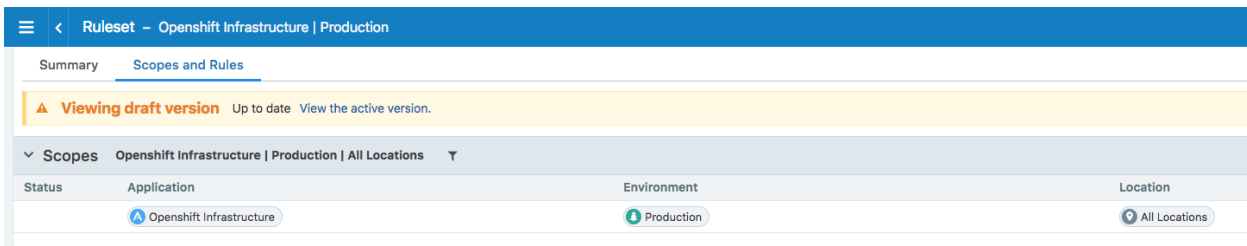
Providers	Services	Consumers	Notes
			(Kubernetes).

Extra-scope Rules

Providers	Services	Consumers	Notes
All Work-loads	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 rule-set.

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.

Providers	Services	Consumers	Notes
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

Providers	Services	Consumers	Notes
			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:

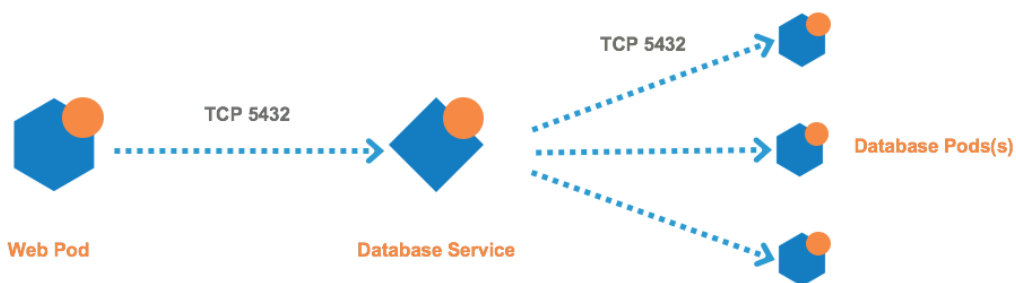
The IP lists referenced in the rulesets are commonly used public registries (for example, docker.io) for container environments. If you have confirmed that your Kubernetes or OpenShift environment does not depend on the public registries mentioned above, then it is recommended that you remove the IP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

Rules 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

Provider	Providing Service	Consumer	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 <i>Derived from Provider Virtual Service</i> .

OpenShift

The rules you need to write are:

Example Ruleset

Scope

Application	Environment	Location
Risk Assessment	Production	HQ

Intra-Scope Rule

Provider	Providing Service	Consumer	Notes
Database (Virtual Service Role label for database service) + Use Virtual Ser-	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

Provider	Providing Service	Consumer	Notes
vices Only			with <i>Derived from Provider Virtual Service.</i>

Access Services from Outside the Cluster

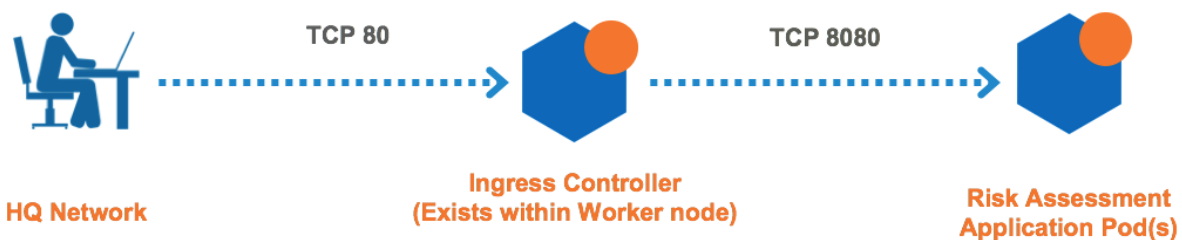
Kubernetes

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 NodePort or 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](#) 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

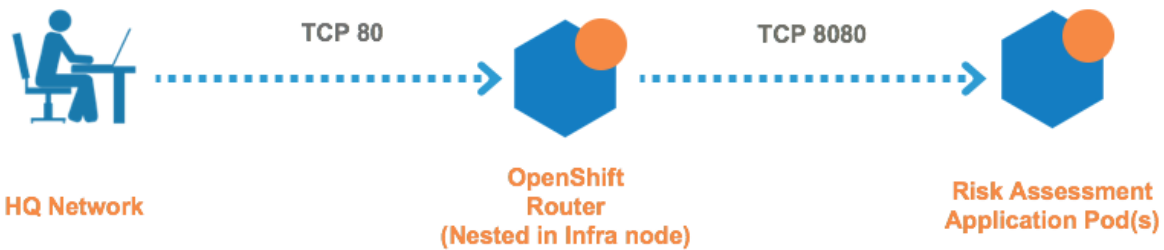
Provider	Providing Service	Consumer	Notes
Worker Node(s)	TCP 80	External Network	This rule is should exist from the Rules for Kubernetes or OpenShift Cluster section. The provider should be the Kubernetes node(s) which contain the ingress controller. The consumer can be an IP List such as 0.0.0.0/0 (any), 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

Provider	Providing Service	Consumer	Notes
			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

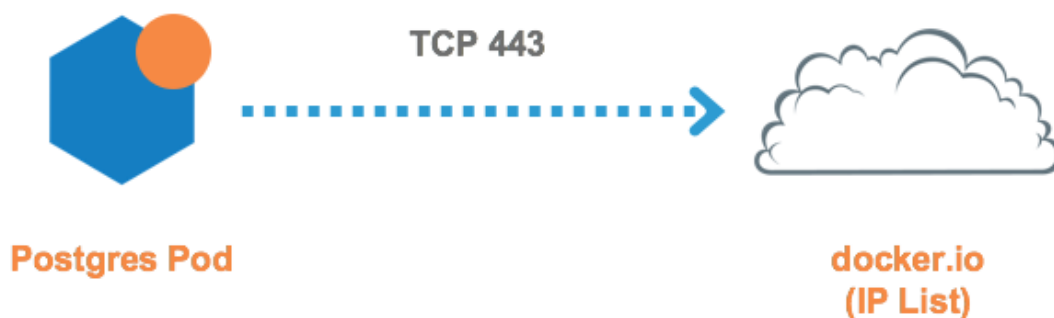
Application	Environment	Location	Notes
IST OpenShift Infrastructure	IST Production	IST HQ	Ruleset is derived from Illumio Segmentation Template. The scope should match the OpenShift cluster.

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

Provider	Providing Service	Consumer	Notes
docker.io (IP List)	All Ser- vices	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 <i>Derived from Provider Virtual Service</i> .

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).

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 Service	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.

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**.

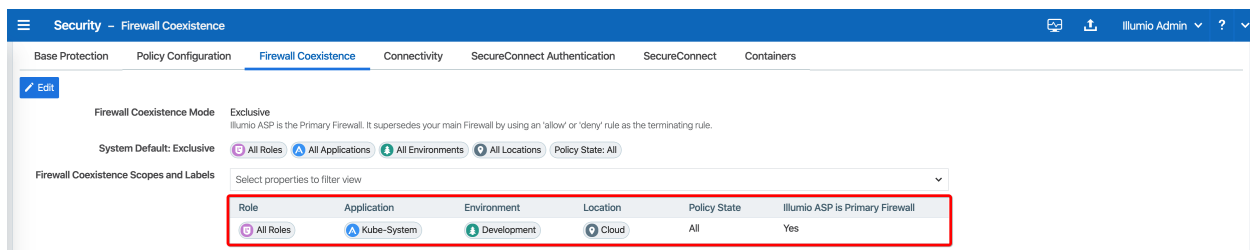
- c. Click **OK**.
6. Click **Save**.
 7. Provision the changes.

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

This chapter contains the following topics:

Upgrade Illumio Components	85
Uninstall Illumio from Your Cluster	86

Follow the steps and sequence described in this section while upgrading or uninstalling Illumio Core components.

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-VEs 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-VEs and then delete the Illumio resources from your cluster.

Unpair C-VEs



IMPORTANT:

Contact Illumio Professional Services to unpair the C-VEs in your Kubernetes or OpenShift clusters.

Deleting C-VEs 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-VEs is by deleting the entire cluster in the PCE UI.



IMPORTANT:

Unpairing an individual C-VE 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 `kubect1 delete node <node_name>` command, the PCE automatically unpairs the C-VE and deletes the workload and Container workloads associated with the C-VE 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-VE Resources

1. Contact Illumio Professional Services to unpair the C-VEs 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-VE installation by using the following command:

```
kubect1 delete -f illumio-ven-kubernetes.yml
kubect1 delete -f illumio-ven-secret.yml
```

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

Delete Kubelink Resources

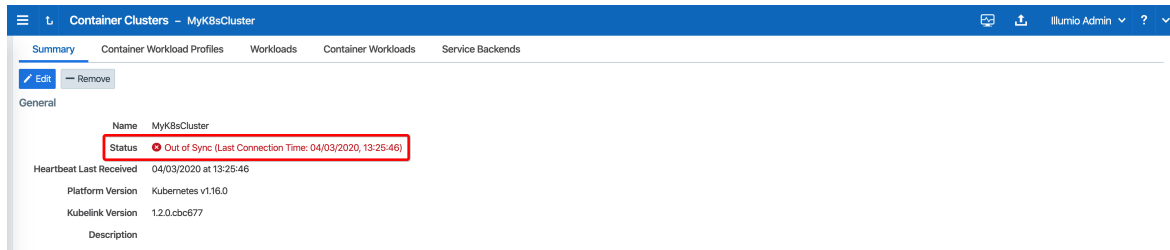
1. Delete the resources created during the Kubelink installation.
2. Delete Kubelink resources from Kubernetes:

```
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 > YourCluster-Name** 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
```


Reference: General

This chapter contains the following topics:

Troubleshooting	89
Known Limitations	97

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

Troubleshooting

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

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:1.2.0.cbc677
    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:1.2.0.cbc677"
  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:1.2.0.cbc677"
  Warning Failed          1s (x3 over 46s) kubelet, node2 Failed to pull image
"registry.poc.segmentationpov.com/illumio-kubelink:1.2.0.cbc677": 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 `CrashLoopBackOff` 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 retrievable 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":"1.2.0.cbc677","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.

The screenshot shows the 'Container Cluster - my-k8s-cluster-1' page. The 'Summary' tab is active, displaying an error message: 'There are duplicate machine IDs among your cluster nodes. Container cluster functionality will be limited until this issue is resolved. The nodes with duplicate IDs are: illumio-os-master'. Below the message are 'Edit' and 'Remove' buttons. The 'General' section shows the following details:

Name	my-k8s-cluster-1
Status	✘ Error
Heartbeat Last Received	08/08/2019 at 10:10:37
Platform Version	Kubernetes v1.14.4
Kubelink Version	test-master.75c678
Description	

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
```

Each machineID should be unique. See the example below:

```
$ kubectl get node -o yaml | grep machineID
  machineID: ec2eefcfc1bd9a9d38218812405a27d9
  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
```

Example of machine-id output:

```
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-VEs 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.

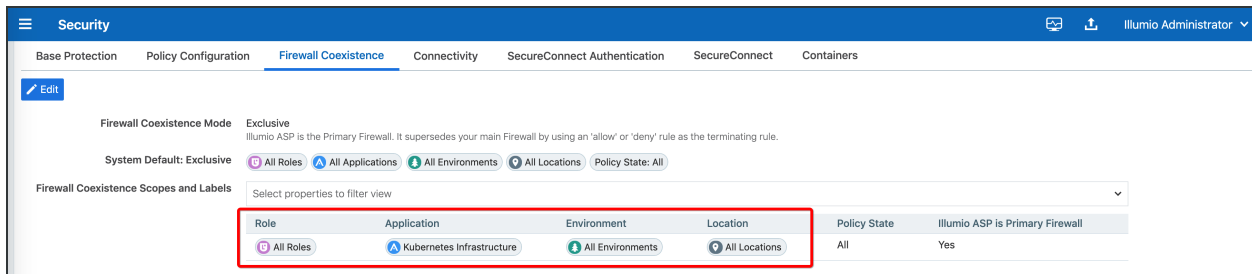
**IMPORTANT:**

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 a 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.



Known Limitations

The known limitations in this release are:

- Kube-proxy mode set to IPVS is currently not supported.
- Exposing an application using NodePort is currently not supported by Illumio Core (no visibility and no enforcement). Consider leveraging hostNetwork'd ingress controllers to expose HTTP applications to better control inbound connections or Network Load Balancers that can preserve the client IP addresses.
- 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.