

Cloud Native Contrail Networking

Installation and Life Cycle Management Guide for OpenShift Container Platform

Published
2023-09-08

Juniper Networks, Inc.
1133 Innovation Way
Sunnyvale, California 94089
USA
408-745-2000
www.juniper.net

Juniper Networks, the Juniper Networks logo, Juniper, and Junos are registered trademarks of Juniper Networks, Inc. in the United States and other countries. All other trademarks, service marks, registered marks, or registered service marks are the property of their respective owners.

Juniper Networks assumes no responsibility for any inaccuracies in this document. Juniper Networks reserves the right to change, modify, transfer, or otherwise revise this publication without notice.

Cloud Native Contrail Networking Installation and Life Cycle Management Guide for OpenShift Container Platform
Copyright © 2023 Juniper Networks, Inc. All rights reserved.

The information in this document is current as of the date on the title page.

YEAR 2000 NOTICE

Juniper Networks hardware and software products are Year 2000 compliant. Junos OS has no known time-related limitations through the year 2038. However, the NTP application is known to have some difficulty in the year 2036.

END USER LICENSE AGREEMENT

The Juniper Networks product that is the subject of this technical documentation consists of (or is intended for use with) Juniper Networks software. Use of such software is subject to the terms and conditions of the End User License Agreement ("EULA") posted at <https://support.juniper.net/support/eula/>. By downloading, installing or using such software, you agree to the terms and conditions of that EULA.

Table of Contents

1

Introduction

Cloud-Native Contrail Networking Overview | 2

Terminology | 4

CN2 Components | 5

Fabric Requirements | 11

Deployment Models | 12

Single Cluster Deployment | 12

Multi-Cluster Deployment | 13

System Requirements | 14

2

Install

Overview | 16

Before You Install | 18

Install with User-Managed Networking | 26

Install with Cluster-Managed Networking | 38

Manifests and Tools | 51

Manifests and Tools in Release 22.4 | 51

3

Monitor

Overview | 57

Install Contrail Analytics | 57

Install Contrail Analytics in Release 22.4 | 57

Kubectl Contrailstatus | 59

4

Manage

Overview | 65

Back Up and Restore Contrail Etcd | 65

Back Up the Contrail Etcd Database in Release 22.4 | **66**

Restore the Contrail Etcd Database in Release 22.4 | **67**

Upgrade CN2 | 70

5

Appendix

Configure Repository Credentials | 74

Replace a Control Plane Node | 75

Remove an Unhealthy Control Plane Node | **75**

Add a Replacement Control Plane Node | **78**

Add a Worker Node | 85

Delete a Worker Node | 91

Back Up the Etcd Database | 92

1

CHAPTER

Introduction

Cloud-Native Contrail Networking Overview | 2

Terminology | 4

CN2 Components | 5

Fabric Requirements | 11

Deployment Models | 12

System Requirements | 14

Cloud-Native Contrail Networking Overview

SUMMARY

Learn about Cloud-Native Contrail Networking (CN2).

IN THIS SECTION

- [Benefits of Cloud-Native Contrail Networking | 4](#)

NOTE: This section is intended to provide a brief and general overview of the Cloud-Native Contrail Networking solution and might contain a description of features not supported in the Kubernetes distribution that you're using. See the Cloud-Native Contrail Networking Release Notes for information on features in the current release for your distribution.

Unless otherwise indicated, all references to Kubernetes are made generically and are not intended to single out a particular distribution.

Contrail Networking is an SDN solution that automates the creation and management of virtualized networks to connect, isolate, and secure cloud workloads and services seamlessly across private and public clouds.

Cloud-Native Contrail Networking (CN2) brings this rich SDN feature set natively to Kubernetes as a networking platform and container network interface (CNI) plug-in.

Redesigned for cloud-native architectures, CN2 takes advantage of the benefits that Kubernetes offers, from simplified DevOps to turnkey scalability, all built on a highly available platform. These benefits include leveraging standard Kubernetes tools and practices to manage Contrail throughout its life cycle:

- Manage CN2 using standard Kubernetes and third-party tools.
- Scale CN2 by adding or removing nodes.
- Configure CN2 by using custom resource definitions (CRDs).
- Upgrade CN2 software by applying updated manifests.
- Uninstall CN2 by deleting Contrail namespaces and resources (upstream Kubernetes only).

More than a CNI plug-in, CN2 is a networking platform that provides dynamic end-to-end virtual networking and security for cloud-native containerized and virtual machine (VM) workloads, across multi-cluster compute and storage environments, all from a central point of control. It supports hard

multi-tenancy for single or multi-cluster environments shared across many tenants, teams, applications, or engineering phases, scaling to thousands of nodes.

The CN2 implementation consists of a set of Contrail controllers that typically reside on Kubernetes control plane nodes but can reside on worker nodes as in the case for Amazon EKS. The Contrail controllers manage a distributed set of data planes implemented by a CNI plug-in and vRouter on every node. Integrating a full-fledged vRouter alongside the workloads provides CN2 the flexibility to support a wide range of networking requirements, from small single clusters to multi-cluster deployments, including:

- Full overlay networking including load balancing, security and multi-tenancy, elastic and resilient VPNs, and gateway services in single-cluster and multi-cluster deployments
- Highly available and resilient network controller overseeing all aspects of the network configuration and control planes
- Analytics services using telemetry and industry standard monitoring and presentation tools such as Prometheus and Grafana
- Support for both CRI-O and containerd runtimes
- Support for container and VM workloads (using kubevirt)
- Support for DPDK data plane acceleration

The Contrail controller automatically detects workload provisioning events such as a new workload being instantiated, network provisioning events such as a new virtual network being created, routing updates from internal and external sources, and unexpected network events such as link and node failures. The Contrail controller reports and logs these events where appropriate and reconfigures the vRouter data plane as necessary.

Although any single node can contain only one Contrail controller, a typical deployment contains multiple controllers running on multiple nodes. When there are multiple Contrail controllers, the controllers keep in synchronization by using iBGP to exchange routes. If a Contrail controller goes down, the Contrail controllers on the other nodes retain all database information and continue to provide the network control plane uninterrupted.

On the worker nodes where workloads reside, each vRouter establishes communications with two Contrail controllers, such that the vRouter can continue to receive instruction if any one controller goes down.

By natively supporting Kubernetes, the CN2 solution leverages the simplicity, flexibility, scalability, and availability inherent to the Kubernetes architecture, while supporting a rich SDN feature set that can meet the requirements of enterprises and service providers alike. Enterprises and service providers can now manage Contrail using simplified and familiar DevOps tools and processes without needing to learn a new life cycle management (LCM) paradigm.

Benefits of Cloud-Native Contrail Networking

- Support a rich networking feature set for your overlay networks.
- Deploy a highly scalable and highly available SDN solution on both upstream and commercial Kubernetes distributions.
- Manage CN2 using familiar, industry-standard tools and practices.
- Leverage the skill set of your existing DevOps engineers to quickly get CN2 up and running.
- Combine with Juniper Networks fabric devices and fabric management solutions or use your own fabric.

Terminology

Table 1: Terminology

Term	Meaning
Kubernetes control plane	The Kubernetes control plane is the collection of pods that manage containerized workloads on the worker nodes in a cluster.
Kubernetes control plane node	This is the virtual or physical machine that hosts the Kubernetes control plane, formerly known as a master node.
Kubernetes node or worker node	Also called a worker node, a Kubernetes node is a virtual or physical machine that hosts containerized workloads in a cluster. To reduce ambiguity, we refer to this strictly as a worker node in this document.
Contrail compute node	This is equivalent to a worker node. It is the node where the Contrail vRouter is providing the data plane function.

Table 1: Terminology (*Continued*)

Term	Meaning
Network control plane	<p>The network control plane provides the core SDN capability. It uses BGP to interact with peers such as other controllers and gateway routers, and XMPP to interact with the data plane components.</p> <p>CN2 supports a centralized network control plane architecture where the routing daemon runs centrally within the Contrail controller and learns and distributes routes from and to the data plane components.</p> <p>This centralized architecture facilitates virtual network abstraction, orchestration, and automation.</p>
Network configuration plane	The network configuration plane interacts with Kubernetes control plane components to manage all CN2 resources. You configure CN2 resources using custom resource definitions (CRDs).
Network data plane	The network data plane resides on all nodes and interacts with containerized workloads to send and receive network traffic. Its main component is the Contrail vRouter.
Contrail controller	<p>This is the part of CN2 that provides the network configuration and network control plane functionality. The Contrail controller typically resides on a Kubernetes control plane node but can reside on a worker node in some cases.</p> <p>This name is purely conceptual – there is no corresponding Contrail controller object or entity in the UI.</p>
Central cluster	In a multi-cluster deployment, this is the central Kubernetes cluster that houses the Contrail controller.
Workload cluster	In a multi-cluster deployment, this is the distributed cluster that contains the workloads.

CN2 Components

NOTE: This section is intended to provide a brief and general overview of the components that make up the Cloud-Native Contrail Networking (CN2) solution. These components are generally

common to all Kubernetes distributions running CN2. Nevertheless, differences do exist, and these are called out explicitly where necessary.

The CN2 architecture consists of pods that perform the network configuration plane and network control plane functions, and pods that perform the network data plane functions.

- The network configuration plane refers to the functionality that enables CN2 to manage its resources and interact with the rest of the Kubernetes control plane.
- The network control plane represents CN2's full-featured SDN capability. It uses BGP to communicate with other controllers and XMPP to communicate with the distributed data plane components on the worker nodes.
- The network data plane refers to the packet transmit and receive function on every node, especially on worker nodes where the workloads reside.

The pods that perform the configuration and control plane functions reside on Kubernetes control plane nodes in most distributions. The pods that perform the data plane functions reside on both Kubernetes control plane nodes and Kubernetes worker nodes. (In Amazon EKS, all CN2 pods reside on worker nodes.)

Table 2 on page 6 describes the main CN2 components. Depending on configuration, there might be other components as well (not shown) that perform ancillary functions such as certificate management and status monitoring.

Table 2: CN2 Components

Pod Name		Where	Description
Configuration Plane ¹	contrail-k8s-apiserver	Control Plane Node ²	<p>This pod is an aggregated API server that is the entry point for managing all Contrail resources. It is registered with the regular kube-apiserver as an APIService. The regular kube-apiserver forwards all network-related requests to the contrail-k8s-apiserver for handling.</p> <p>There is one contrail-k8s-apiserver pod per Kubernetes control plane node.</p>

Table 2: CN2 Components *(Continued)*

Pod Name		Where	Description
Control Plane ¹	contrail-k8s-controller	Control Plane Node ²	<p>This pod performs the Kubernetes control loop function to reconcile networking resources. It constantly monitors networking resources to make sure the actual state of a resource matches its intended state.</p> <p>There is one contrail-k8s-controller pod per Kubernetes control plane node.</p>
	contrail-k8s-kubemanager	Control Plane Node ²	<p>This pod is the interface between Kubernetes resources and Contrail resources. It watches the kube-apiserver for changes to regular Kubernetes resources such as service and namespace and acts on any changes that affect the networking resources.</p> <p>In a single-cluster deployment, there is one contrail-k8s-kubemanager pod per Kubernetes control plane node.</p> <p>In a multi-cluster deployment, there is additionally one contrail-k8s-kubemanager pod for every distributed workload cluster.</p>
	contrail-control	Control Plane Node ²	<p>This pod passes configuration to the worker nodes and performs route learning and distribution. It watches the kube-apiserver for anything affecting the network control plane and then communicates with its BGP peers and/or vRouter agents (over XMPP) as appropriate.</p> <p>There is one contrail-control pod per Kubernetes control plane node.</p>

Table 2: CN2 Components *(Continued)*

Pod Name		Where	Description
Data Plane	contrail-vrouter-nodes	Worker Node	<p>This pod contains the vRouter agent and the vRouter itself.</p> <p>The vRouter agent acts on behalf of the local vRouter when interacting with the Contrail controller. There is one agent per node. The agent establishes XMPP sessions with two Contrail controllers to perform the following functions:</p> <ul style="list-style-type: none"> • translates configuration from the control plane into objects that the vRouter understands • interfaces with the control plane for the management of routes • collects and exports statistics from the data plane <p>The vRouter provides the packet send and receive function for the co-located pods and workloads. It provides the CNI plug-in functionality.</p>
	contrail-vrouter-masters ³	Control Plane Node	<p>This pod provides the same functionality as the contrail-vrouter-nodes pod, but resides on the control plane nodes.</p>
<p>¹The components that make up the network configuration plane and the network control plane are collectively called the Contrail controller.</p> <p>²Worker node if running on Amazon EKS.</p> <p>³Not present if running on Amazon EKS.</p>			

Figure 1 on page 9 shows these components in the context of a Kubernetes cluster in most distributions and Figure 2 on page 10 shows these components in the context of Amazon EKS.

For clarity and to reduce clutter, the figures do not show the data plane pods on the node with the Contrail controller.

Figure 1: CN2 Components (Most Distributions)

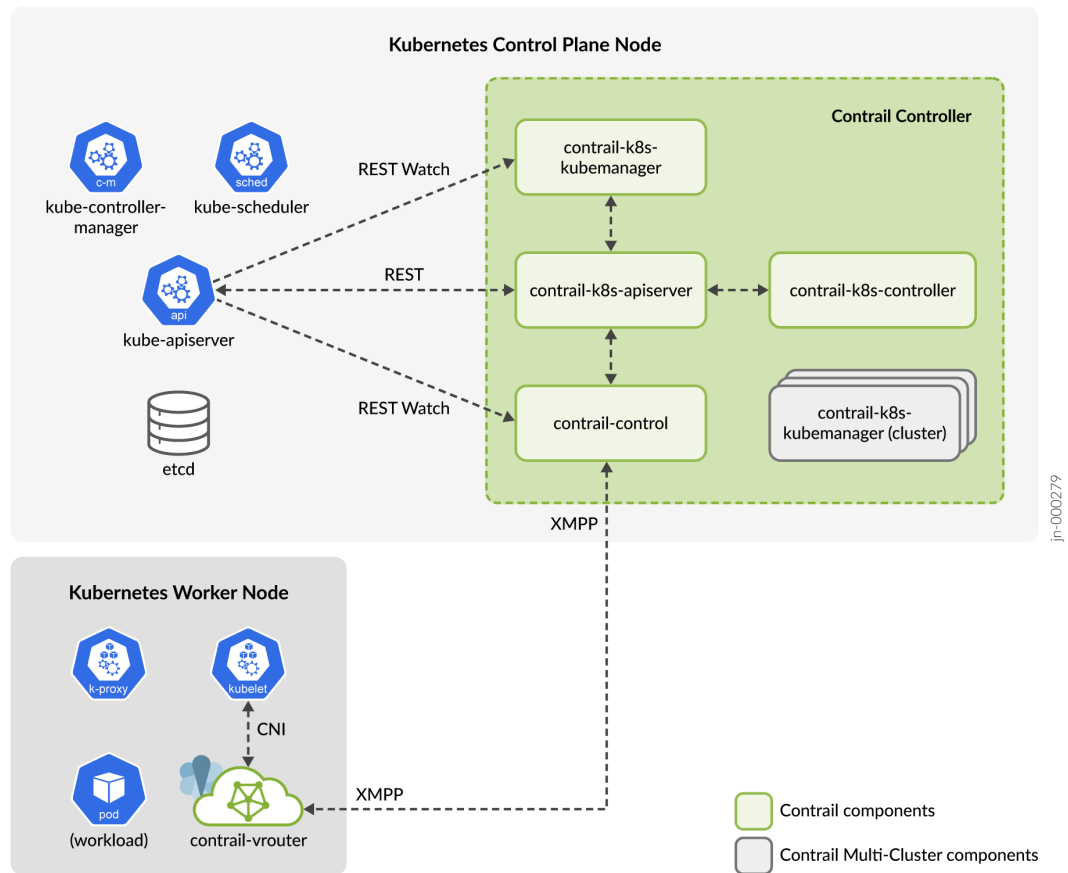
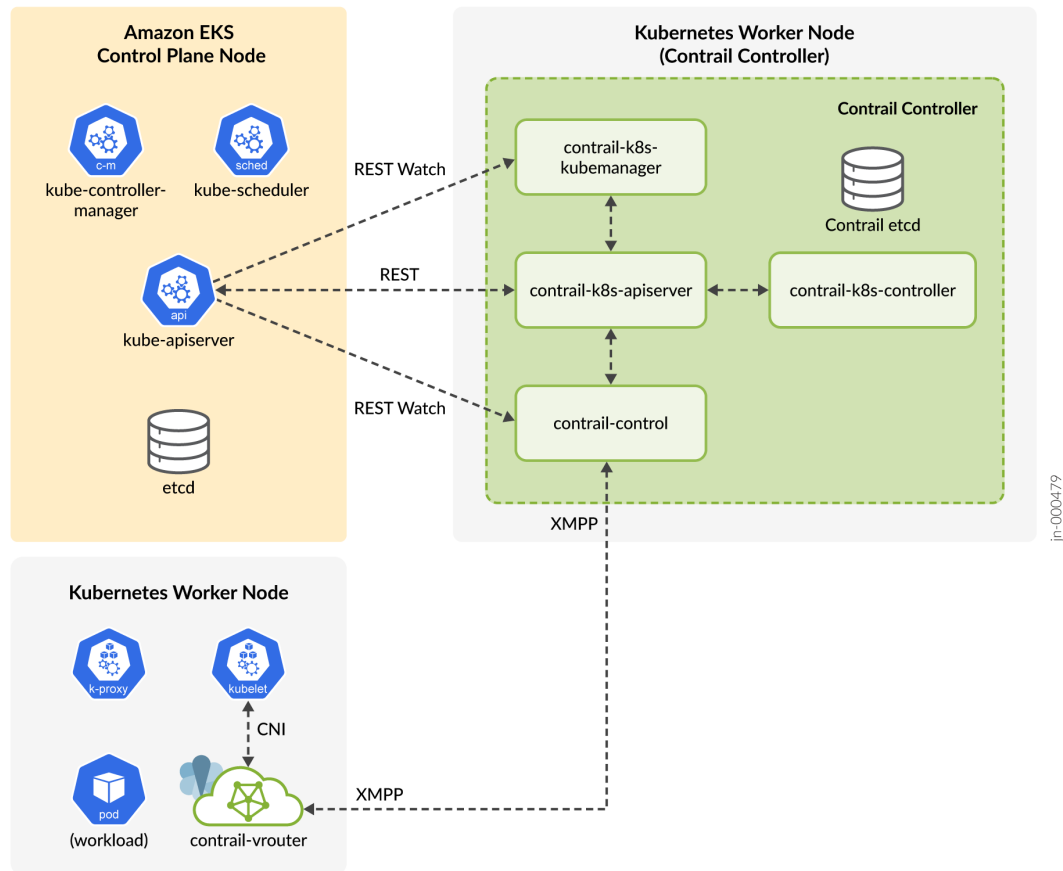


Figure 2: CN2 Components (Amazon EKS)



When running on upstream Kubernetes, CN2 uses the main Kubernetes etcd database. When running on OpenShift or when running on Amazon EKS, CN2 uses its own etcd database.

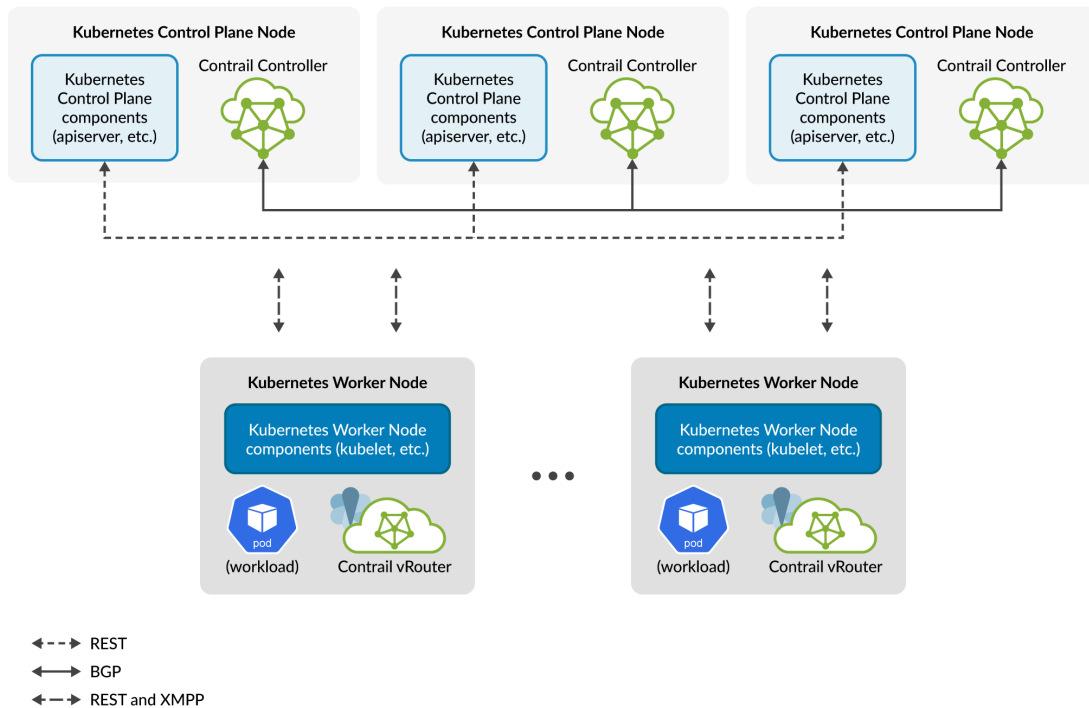
For all distributions, the kube-apiserver is the entry point for Kubernetes REST API calls for the cluster. It directs all networking requests to the contrail-k8s-apiserver, which is the entry point for Contrail API calls. The contrail-k8s-apiserver translates incoming networking requests into REST API calls to the respective CN2 objects. In some cases, these calls may result in the Contrail controller sending XMPP messages to the vRouter agent on one or more worker nodes or sending BGP messages (not shown) to other control plane nodes or external routers. These XMPP and BGP messages are sent outside of regular Kubernetes node-to-node communications.

The contrail-k8s-kubemanager (cluster) components are only present in multi-cluster deployments. For more information on the different types of deployment, see [Deployment Models](#).

[Figure 3 on page 11](#) shows a cluster with multiple Contrail controllers. These controllers reside on control plane nodes (most distributions). The Kubernetes components communicate with each other using REST. The Contrail controllers exchange routes with each other using iBGP, outside of the regular

Kubernetes REST interface. For redundancy, the vRouter agents on worker nodes always establish XMPP communications with two Contrail controllers.

Figure 3: Multiple Contrail Controllers (Most Distributions)



Fabric Requirements

CN2 is agnostic to the fabric technology used. The fabric can consist of Juniper Networks QFX Series switches managed by Juniper Apstra or it can consist of switches from other vendors.

Depending on the type of installation, however, you may need to set up more than one fabric virtual network to connect the cluster nodes. Where necessary, we describe specific fabric requirements in the respective installation sections.

Deployment Models

SUMMARY

Learn about the different ways you can deploy Contrail.

IN THIS SECTION

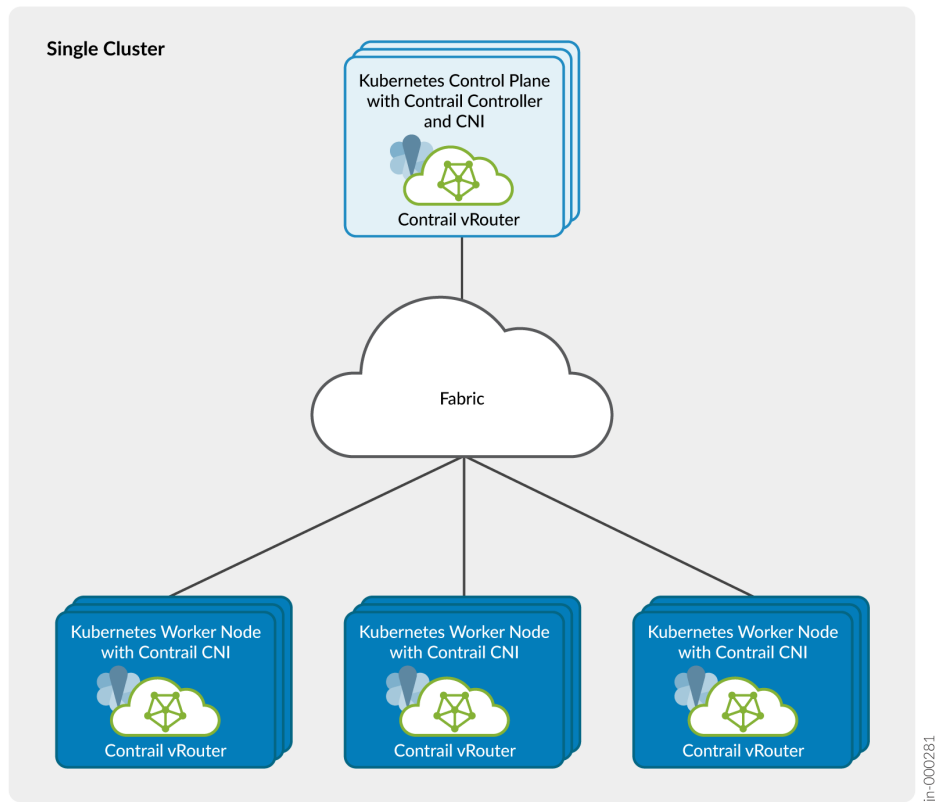
- [Single Cluster Deployment | 12](#)
- [Multi-Cluster Deployment | 13](#)

Single Cluster Deployment

Cloud-Native Contrail Networking (CN2) is available as an integrated networking platform in a single Kubernetes cluster, watching where workloads are instantiated and connecting those workloads to the appropriate overlay networks.

In a single-cluster deployment ([Figure 4 on page 13](#)), the Contrail controller sits in the Kubernetes control plane and provides the network configuration and network control planes for the host cluster. The Contrail data plane components sit in all nodes and provide the packet send and receive function for the workloads.

Figure 4: Single Cluster Deployment



Multi-Cluster Deployment

In a multi-cluster deployment, the Contrail controller resides in its own Kubernetes cluster and provides networking to other clusters.

CN2 does not support multi-cluster deployment on an OpenShift cluster.

System Requirements

Table 3: System Requirements for OpenShift Installation with CN2

Machine	CPU	RAM	Storage ¹	Notes
AI Client	4	16 GB	100 GB	
Control Plane Nodes	8	32 GB	400 GB	Processor must support the AVX2 instruction set if running DPDK.
Worker Nodes ²	4	16 GB	100 GB	Processor must support the AVX2 instruction set if running DPDK.
¹ SSD is recommended.				
² Based on workload requirements.				

2

CHAPTER

Install

[Overview | 16](#)

[Before You Install | 18](#)

[Install with User-Managed Networking | 26](#)

[Install with Cluster-Managed Networking | 38](#)

[Manifests and Tools | 51](#)

Overview

IN THIS SECTION

- [Benefits of OpenShift with CN2 | 18](#)

Red Hat OpenShift is an enterprise Kubernetes container platform that packages Kubernetes with a rich set of DevOps services and tools. Built on Red Hat Enterprise Linux and Kubernetes, OpenShift Container Platform (OCP) provides secure and scalable multi-tenant container orchestration for enterprise-class applications, complete with integrated application runtimes and libraries.

Red Hat OpenShift is available as a fully managed cloud service, or as a self-managed software offering for organizations requiring more customization and control.

When augmented with CN2, Red Hat OpenShift gains a full-featured CNI and networking platform that can meet the complex networking requirements of enterprises and service providers alike.

Red Hat provides an Assisted Installer to perform the heavy lifting for much of the cluster installation process, including automatically running pre-flight validations to ensure a smooth experience. We have integrated the installation of CN2 within the Assisted Installer framework, so you can install CN2 seamlessly as part of Assisted Installer cluster installation.

You can use the Assisted Installer service hosted by Red Hat or you can download and install a local instance of the Assisted Installer in your own infrastructure. The Assisted Installer service, whether installed locally or hosted by Red Hat, is accessible through a UI (via your browser) or through REST API calls.

NOTE: This installation guide covers the use of the hosted Assisted Installer service through REST API calls. Installing a cluster through API calls makes it easier for you to automate the procedure.

The Assisted Installer service supports early binding and late binding of the hosts to the cluster. In early binding, the hosts are bound to the cluster upon cluster registration. In late binding, the hosts are bound to the cluster after cluster registration. In both cases, the hosts are not bound to the cluster directly, but through an intermediary resource that provides the necessary decoupling required to support late binding.

This installation guide follows the early binding procedure, which is summarized as follows:

- Create a cluster resource that defines your cluster. A cluster resource is an object within the Assisted Installer service. As part of creating the cluster resource, you specify that you want CN2 to be the CNI for this cluster.
- Create an `infra_env` resource referencing the cluster resource. The `infra_env` resource is an object within the Assisted Installer service that acts as an intermediary, containing attributes describing the infrastructure environment in which the cluster is to be created. By referencing the cluster resource now (early binding), the hosts will boot up directly into the cluster.
- Generate and download a live ISO image that is customized for the infrastructure environment and cluster. This image is based on Red Hat Core OS (RHCOS) and is configured to automatically run an agent upon booting. The agent uses preconfigured tokens and keys to contact the Assisted Installer service and to communicate with other hosts within the cluster.
- Boot the cluster hosts with the live ISO image. Upon boot-up, the agent on each host contacts the Assisted Installer service via API calls, announcing its presence and disclosing basic hardware specification. The Assisted Installer service authenticates these API calls by verifying the tokens within the message. The Assisted Installer then performs preflight checks and other validations to ensure the hosts are capable of supporting the cluster defined by the cluster resource. Once the Assisted Installer service validates the hosts, the Assisted Installer service waits for you to initiate cluster installation.
- Apply the CN2 manifests to the cluster resource. This augments the cluster resource with CN2 configuration. Installation of CN2 is fully integrated within OpenShift installation.
- Start the installation of the cluster. The Assisted Installer service starts the cluster installation process by first selecting one host to be the bootstrap node. The Assisted Installer service configures this host to bootstrap the other control plane hosts. You will see the other control plane hosts reboot one by one into the cluster. Once all the other control plane hosts reboot into the cluster, the bootstrap node follows suit and reboots into the cluster as well. Once the control plane nodes come back up, they boot up the worker nodes.
- Watch the progress of the installation on the Red Hat Hybrid Cloud Console (<https://console.redhat.com/openshift>). The Red Hat Hybrid Cloud Console is the UI for the Assisted Installer service.

When you use the Assisted Installer service, you can choose to install OpenShift with user-managed networking or with cluster-managed networking:

- User-managed networking refers to a deployment where you explicitly provide an external load balancer for your installation. The load balancer distributes internal and external control plane API calls to the control plane nodes that you set up, as well as distribute application traffic to the worker nodes as required.

With this option, you'll only need a single fabric virtual network. This single virtual network carries installation traffic to and from the Assisted Installer service, Kubernetes control plane traffic between

nodes, Contrail network control plane traffic, and user data plane traffic. Each node has a single interface that connects to the fabric.

- Cluster-managed networking refers to a deployment where the Assisted Installer installs an integrated load balancer and ingress in the cluster for you.

With this option, you'll need two virtual networks in the underlay fabric. The first virtual network carries installation traffic to and from the Assisted Installer service as well as Kubernetes control plane traffic between nodes. The second virtual network carries Contrail network control plane traffic and user data plane traffic. Each node has two interfaces, connecting to each of the aforementioned virtual networks.

The reason for the two virtual networks is that CN2 internally implements VRRP to provide redundancy, which might conflict with some of the functionality that OCP provides. By splitting out the Contrail (vhost0) traffic, both the Kubernetes control plane and the Contrail control plane can achieve redundancy.

CN2 handles all aspects of this VRRP implementation internally. There is no need for external routers to participate in VRRP nor is there a need for external routers to route between the two virtual networks.

NOTE: The Contrail Networking implementation of the ClusterIP service uses ECMP load balancing. Session affinity is therefore enacted per flow, not per client IP address.

Benefits of OpenShift with CN2

- Industry-leading enterprise Kubernetes platform together with industry-leading CNI
- Full featured SDN solution for enterprises and service providers alike
- Feature-rich graphical user interface
- Automated workflows and seamless integration with CI/CD pipelines
- Assisted Installer facilitates installation

Before You Install

1. Set up an account with Red Hat and set up an account with Juniper Networks.

You'll need the Red Hat account to use the hosted Assisted Installer service, and you'll need the Juniper Networks account to download the CN2 manifests, container packages, and tools.

2. Set up the fabric network and connect your nodes to the fabric depending on whether you're installing with user-managed networking ([Figure 5 on page 27](#)) or cluster-managed networking ([Figure 6 on page 39](#)).
3. Configure the Assisted Installer client machine.
 - a. Install a fresh OS on the Assisted Installer client machine, configuring the OS minimally for the following:
 - static IP address and mask (for example, 172.16.0.10/24) and gateway
 - access to one or more DNS servers
 - SSH connectivity including root SSH access
 - NTP
 - curl
 - jq

The Assisted Installer client machine used in our examples is running a RHEL OS.

- b. Install Helm 3.0 or later (optional). Helm is needed if you want to install Contrail Analytics. The following steps are copied from <https://helm.sh/docs/intro/install/> for your convenience:
 - Download the get_helm.sh script:

```
curl -fsSL -o get_helm.sh https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3
```

- Install Helm.

```
chmod 700 get_helm.sh
```

```
./get_helm.sh
```

4. Download the CN2 Networking Manifests and Tools package from Juniper Networks.
 - a. Download the CN2 Networking Manifests and Tools package (see "[Manifests and Tools](#)" on [page 51](#)) onto your local computer.

- b. Copy the downloaded manifests and tools package to the Assisted Installer client machine and extract.

```
user@ai-client~/tmp$ tar -xzf contrail-manifests-openshift-<version>.tgz
```

- c. Identify the manifests you want to use and copy them to a separate directory. For an explanation of the manifests, see ["Manifests and Tools" on page 51](#) for your release.

Make sure you copy over all the manifests that you plan on using, including the manifests from the subdirectories if applicable. In our example, we copy the manifests to a **manifests** directory. Don't copy the subdirectories themselves. The **manifests** directory should be a flat directory.

- d. Populate the manifests with your repository login credentials.

Add your repository login credentials to the **contrail-manifests-openshift/auth-registry** manifests. See ["Configure Repository Credentials" on page 74](#).

- e. Customize the manifests for your environment as necessary.

If you're running your cluster nodes on VMs, edit the following files to reference the actual names of your interfaces. These manifests disable checksum offloads on the named interface on the VM. (Checksum offload is usually only supported on real NICs on bare metal servers.)

- 99-disable-offload-master.yaml - This manifest disables offload on the control plane nodes on the interface used for Kubernetes control plane traffic. This is the interface that attaches to the 172.16.0.0/24 network in our examples.
- 99-disable-offload-worker.yaml - This manifest disables offload on the worker nodes on the interface used for Kubernetes control plane traffic. This is the interface that attaches to the 172.16.0.0/24 network in our examples.
- 99-disable-offload-master-vrrp.yaml - This manifest disables offload on the control plane nodes on the interface used for Contrail control plane and user data plane traffic. Include this only when running a separate interface for Contrail control and data traffic (such as when using cluster-managed networking). This is the interface that attaches to the 10.16.0.0/24 network in our cluster-managed networking example.
- 99-disable-offload-worker-vrrp.yaml - This manifest disables offload on the worker nodes on the interface used for Contrail control plane and user data plane traffic. Include this only when running a separate interface for Contrail control and data traffic (such as when using cluster-managed networking). This is the interface that attaches to the 10.16.0.0/24 network in our cluster-managed networking example.

Look for the line `ExecStart=/sbin/ethtool -K ens3 tx off` or `ExecStart=/sbin/ethtool -K ens4 tx off` in these manifests and change the interface name to match the interface name on your control plane or worker node as appropriate.

- f. Specify the Contrail control and data network if you're using cluster-managed networking.

Edit the following file to reference the subnet and gateway that you're using for Contrail control plane and user data plane traffic:

- `99-network-configmap.yaml` - This manifest specifies the network for Contrail control plane and user data plane traffic. Uncomment the `contrail-network-config` ConfigMap specification in the manifest and specify the appropriate subnet and gateway (for example, `10.16.0.0/24` and `10.16.0.254`).

- g. If you're integrating CN2 with Juniper Apstra, configure your Juniper Apstra login credentials.

Configure your Apstra login credentials in the **`contrail-manifests-openshift/plugins/111-apstra-secret.yaml`** manifest. Make sure the username and password that you specify are base64-encoded. For more information, see <https://www.juniper.net/documentation/us/en/software/cn-cloud-native22.4/cn-cloud-native-feature-guide/index.html>.

5. Install `contrailstatus` on the Assisted Installer client machine. `Contrailstatus` is a `kubectl` plug-in you can use to query CN2 microservices and CN2-specific resources.

The `contrailstatus` executable is packaged within the downloaded tools package. Extract and copy the **`kubectl-contrailstatus`** executable to `/usr/local/bin`.

6. Install a load balancer (if you're running with user-managed networking). This step is not required when running with cluster-managed networking.

In this example, we run `haproxy` on the Assisted Installer client machine. You can choose to run a different load balancer for your installation.

- a. Install the load balancer.

For example:

```
sudo dnf install haproxy
```

- b. Configure the load balancer.

We use a single IP address (`172.16.0.10`) that distributes API and ingress traffic to the nodes in the cluster.

Table 4: Example Load Balancer Entries

Type of Traffic	Front End	Back End
api	172.16.0.10:6443	172.16.0.11:6443
		172.16.0.12:6443
		172.16.0.13:6443

Table 4: Example Load Balancer Entries *(Continued)*

Type of Traffic	Front End	Back End
api-int	172.16.0.10:22623	172.16.0.11:22623 172.16.0.12:22623 172.16.0.13:22623
https	172.16.0.10:443	172.16.0.14:443 172.16.0.15:443
http	172.16.0.10:80	172.16.0.14:80 172.16.0.15:80

Here's an example snippet of haproxy configuration (`/etc/haproxy/haproxy.cfg`) that performs the above mappings:

```
frontend api
  bind 172.16.0.10:6443
  default_backend controlplaneapi

frontend apiinternal
  bind 172.16.0.10:22623
  default_backend controlplaneapiinternal

frontend secure
  bind 172.16.0.10:443
  default_backend secure

frontend insecure
  bind 172.16.0.10:80
  default_backend insecure

backend controlplaneapi
  balance roundrobin
  server cp0 172.16.0.11:6443 check
  server cp1 172.16.0.12:6443 check
  server cp2 172.16.0.13:6443 check
```

```

backend controlplaneapiinternal
    balance roundrobin
    server cp0 172.16.0.11:22623 check
    server cp1 172.16.0.12:22623 check
    server cp2 172.16.0.13:22623 check

backend secure
    balance roundrobin
    server worker0 172.16.0.14:443 check
    server worker1 172.16.0.15:443 check

backend insecure
    balance roundrobin
    server worker0 172.16.0.14:80 check
    server worker1 172.16.0.15:80 check

```

- c. Start the load balancer.

For example:

```
systemctl start haproxy
```

NOTE: If you're running with selinux, you may need to explicitly allow haproxy to listen on ports (`setsebool -P haproxy_connect_any 1`).

7. Install a DNS/DHCP server in your network to serve the Kubernetes nodes.

In this example, we run `dnsmasq` on the Assisted Installer client machine. You can choose to run a different DNS/DHCP server for your installation.

- a. Install the DNS/DHCP server.

`Dnsmasq` is preinstalled on some RHEL OS packages. If it's not preinstalled, you can install it as follows:

```
sudo dnf install dnsmasq
```

- b. Configure the domain name and DHCP entries.

Table 5: Example DHCP Assignments

Fully-Qualified Domain Name	IP Address
ocp1.mycluster.contrail.lan	172.16.0.11
ocp2.mycluster.contrail.lan	172.16.0.12
ocp3.mycluster.contrail.lan	172.16.0.13
ocp4.mycluster.contrail.lan	172.16.0.14
ocp5.mycluster.contrail.lan	172.16.0.15
<p>NOTE: When using the Assisted Installer service, the fully-qualified domain name is constructed as follows: <hostname>.<cluster name>.<domain name></p> <p>In this example, we use ocp<i>n</i> as the hostname, mycluster as the cluster name, and contrail.lan as the domain name.</p>	

- c. Configure your DNS entries.

Table 6: Example DNS Entries

Hostname	IP Address	Note
ocp1.mycluster.contrail.lan	172.16.0.11	Same as DHCP assignment
ocp2.mycluster.contrail.lan	172.16.0.12	Same as DHCP assignment
ocp3.mycluster.contrail.lan	172.16.0.13	Same as DHCP assignment
ocp4.mycluster.contrail.lan	172.16.0.14	Same as DHCP assignment
ocp5.mycluster.contrail.lan	172.16.0.15	Same as DHCP assignment
api.mycluster.contrail.lan	172.16.0.10	Load balancer for external API traffic. Required for user-managed networking only.

Table 6: Example DNS Entries (*Continued*)

Hostname	IP Address	Note
api-int.mycluster.contrail.lan	172.16.0.10	Load balancer for internal API traffic. Required for user-managed networking only.
apps.mycluster.contrail.lan	172.16.0.10	Load balancer for ingress traffic. Required for user-managed networking only.
*.apps.mycluster.contrail.lan	172.16.0.10	Load balancer for ingress traffic. Required for user-managed networking only.

Here's an example snippet of dnsmasq configuration (`/etc/dnsmasq.conf`) that performs the above assignments:

```
dhcp-host=52:54:00:00:11:11,ocp1.mycluster.contrail.lan,172.16.0.11
dhcp-host=52:54:00:00:11:22,ocp2.mycluster.contrail.lan,172.16.0.12
dhcp-host=52:54:00:00:11:33,ocp3.mycluster.contrail.lan,172.16.0.13
dhcp-host=52:54:00:00:11:44,ocp4.mycluster.contrail.lan,172.16.0.14
dhcp-host=52:54:00:00:11:55,ocp5.mycluster.contrail.lan,172.16.0.15
host-record=api.mycluster.contrail.lan,172.16.0.10
address=/.apps.mycluster.contrail.lan/172.16.0.10
address=/api-int.mycluster.contrail.lan/172.16.0.10
```

- d. Start the DNS/DHCP server.

For example:

```
systemctl start dnsmasq
```

8. Download the OpenShift command line interface tool (oc) from Red Hat. This package includes kubectl.
 - a. On the browser of your local computer, go to <https://console.redhat.com/openshift/downloads#tool-oc> and download the OpenShift command line interface tool (oc).

- b. Copy the downloaded package to the Assisted Installer client machine and untar.

```
user@ai-client:~/$ tar -xvzf openshift-client-linux.tar.gz
README.md
oc
kubectl
```

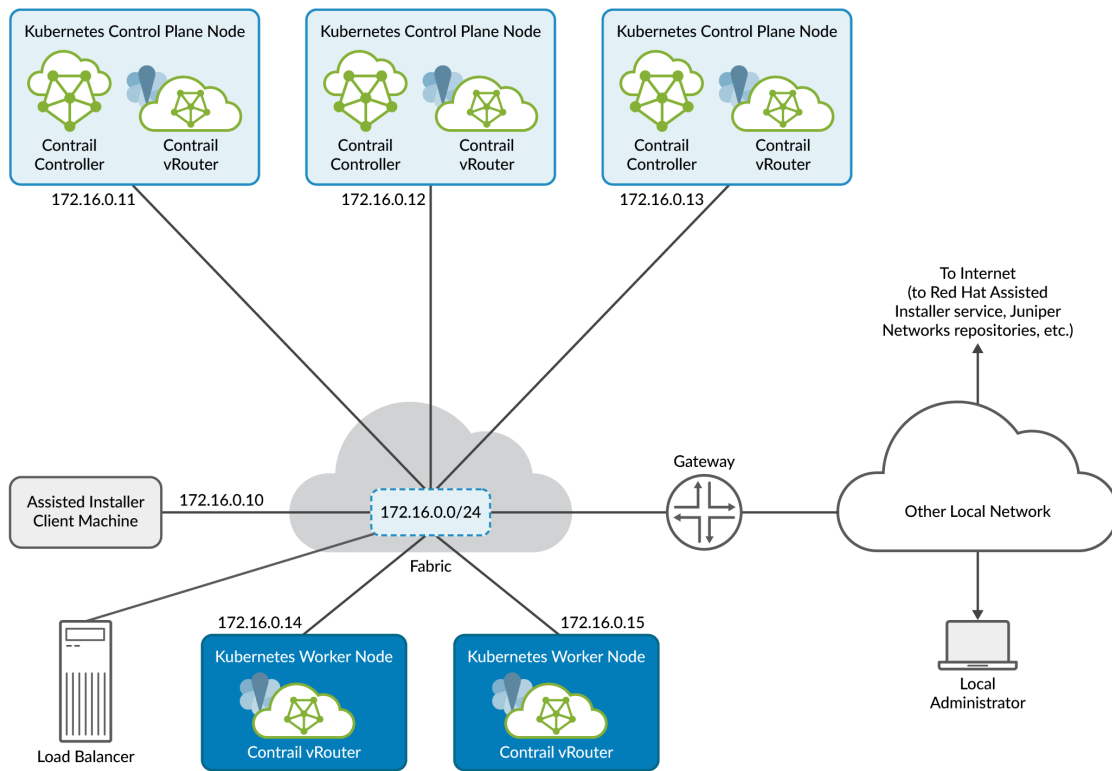
- c. Copy the oc and kubectl executables into a directory in your path (for example, `/usr/local/bin`).

Install with User-Managed Networking

Use this procedure to bring up a cluster with user-managed networking. User-managed networking refers to a deployment where you explicitly provide an external load balancer for your installation.

[Figure 5 on page 27](#) shows a cluster of 3 control plane nodes and 2 worker nodes running on bare metal servers or virtual machines in a user-managed networking setup that includes an external load balancer. All communication between nodes in the cluster and between nodes and external sites take place over the single 172.16.0.0/24 fabric virtual network.

Figure 5: CN2 OpenShift Cluster on Bare Metal Servers (or VMs) with User-Managed Networking



A separate machine acts as the Assisted Installer client. The Assisted Installer client is where you use curl commands to issue API calls to the Assisted Installer service to create the cluster. In the example in this procedure, we use the Assisted Installer client machine to host a DNS/DHCP server for the subnet as well.

The local administrator is shown attached to a separate network reachable through a gateway. This is typical of many installations where the local administrator manages the fabric and cluster from the corporate LAN. In the procedures that follow, we refer to the local administrator station as your local computer.

NOTE: Connecting all nodes together is the data center fabric, which is shown in the example as a single subnet. In real installations, the data center fabric is a network of spine-and-leaf switches that provide the physical connectivity for the cluster.

In an Apstra-managed data center, you would specify this connectivity through the overlay virtual networks that you create across the underlying fabric switches.

This procedure uses the Assisted Installer service hosted by Red Hat, and covers the more common early binding use case where you register the cluster before bringing up the hosts.

1. Log in to the Assisted Installer client machine. The Assisted Installer client machine is where you issue Assisted Installer API calls to the Assisted Installer service.
2. Prepare the deployment by setting the environment variables that you'll use in later steps.
 - a. Create an SSH key that you'll use to access the nodes in your cluster. Save the key to an environment variable.

```
root@ai-client:~# ssh-keygen
```

In this example, we've stored the SSH key in its default location `~/.ssh/id_rsa.pub`

```
export CLUSTER_SSHKEY=$(cat ~/.ssh/id_rsa.pub)
```

- b. Download the image pull secret from your Red Hat account onto your local computer. The pull secret allows your installation to access services and registries that serve container images for OpenShift components.

If you're using the Red Hat hosted Assisted Installer, you can download the pull secret file (**pull-secret**) from the <https://console.redhat.com/openshift/downloads> page. Copy the pull-secret file to the Assisted Installer client machine. In this example, we store the pull-secret in a file called **pull-secret.txt**.

Strip out any whitespace, convert the contents to JSON string format, and store to an environment variable, as follows:

```
export PULL_SECRET=$(sed '/^[[:space:]]*$/d' pull-secret.txt | jq -R .)
```

- c. Copy the offline access token from your Red Hat account. The OpenShift Cluster Manager API Token allows you (on the Assisted Installer client machine) to interact with the Assisted Installer API service hosted by Red Hat.

The token is a string that you can copy and paste to a local environment variable. If you're using the Red Hat hosted Assisted Installer, you can copy the API token from <https://console.redhat.com/openshift/downloads>.

```
export OFFLINE_ACCESS_TOKEN='<paste offline access token here>'
```

- d. Generate (refresh) the token from the OFFLINE_ACCESS_TOKEN. You will use this generated token whenever you issue API commands.

```
export TOKEN=$(curl --silent --data-urlencode "grant_type=refresh_token" --data-
urlencode "client_id=cloud-services" --data-urlencode "refresh_token=$
{OFFLINE_ACCESS_TOKEN}" https://sso.redhat.com/auth/realms/redhat-external/protocol/
openid-connect/token | jq -r .access_token)
```

NOTE: This token expires regularly. When this token expires, you will get an HTTP 4xx response whenever you issue an API command. Refresh the token when it expires, or alternatively, refresh the token regularly prior to expiry. There is no harm in refreshing the token when it hasn't expired.

- e. Set up the remaining environment variables.

[Table 7 on page 29](#) lists all the environment variables that you need to set in this procedure, including the ones described in the previous steps.

Table 7: Environment Variables

Variable	Description	Example
CLUSTER_SSHKEY	The (public) ssh key that you generated. This key will be installed on all cluster nodes.	–
PULL_SECRET	The image pull secret that you downloaded, stripped and converted to JSON string format.	–
OFFLINE_ACCESS_TOKEN	The OpenShift Cluster Manager API Token that you copied.	–
TOKEN	The token that you generated (refreshed) from the OFFLINE_ACCESS_TOKEN.	–

Table 7: Environment Variables (*Continued*)

Variable	Description	Example
CLUSTER_NAME	The name you want to call the cluster. This is the name that will appear in the Red Hat Hybrid Cloud Console UI. NOTE: This name must be in lowercase.	mycluster
CLUSTER_DOMAIN	The base domain that you want to assign to the cluster. Cluster objects will be assigned names in this domain.	contrail.lan
CLUSTER_NET	The overlay cluster network. Pods are assigned IP addresses on this network.	10.128.0.0/14
CLUSTER_SVC_NET	The overlay service network. Services are assigned IP addresses on this network.	172.31.0.0/16
CLUSTER_HOST_PFX	The subnet prefix length for assigning IP addresses from CLUSTER_NET. This defines the subset of CLUSTER_NET IP addresses to use for pod IP address assignment.	23
AI_URL	The URL of the Assisted Installer service. This example uses the Red Hat hosted Assisted Installer.	https://api.openshift.com

3. Register the cluster with the Assisted Installer service. By registering, you're telling the Assisted Installer service about the attributes of the cluster you want to create. In response, the Assisted Installer service creates a cluster resource and returns a cluster identifier that uniquely identifies that cluster resource.

- a. Create the file that describes the cluster you want to create. In this example, we name the file **deployment.json**.

```
cat << EOF > ./deployment.json
{
  "kind": "Cluster",
  "name": "$CLUSTER_NAME",
  "openshift_version": "<openshift_version>",
  "ocp_release_image": "quay.io/openshift-release-dev/ocp-release:<coreOS_version>-x86_64",
  "base_dns_domain": "$CLUSTER_DOMAIN",
  "hyperthreading": "all",
  "cluster_network_cidr": "$CLUSTER_NET",
  "cluster_network_host_prefix": $CLUSTER_HOST_PFX,
  "service_network_cidr": "$CLUSTER_SVC_NET",
  "user_managed_networking": true,
  "ssh_public_key": "$CLUSTER_SSHKEY",
  "pull_secret": $PULL_SECRET
}
EOF
```

NOTE: Ensure you specify a *<coreOS_version>* (for example, 4.8.39) that is listed in https://www.juniper.net/documentation/en_US/release-independent/contrail-cloud-native/topics/reference/cloud-native-contrail-supported-platforms.pdf for the *<openshift_version>* (for example, 4.8) you're using.

- b. Register the cluster and save the CLUSTER_ID to an environment variable. Reference the **deployment.json** file you just created.

```
export CLUSTER_ID=$(curl -s -X POST "$AI_URL/api/assisted-install/v2/clusters" -d @./deployment.json --header "Content-Type: application/json" -H "Authorization: Bearer $TOKEN" | jq -r '.id')
```

- c. Update the cluster attributes to specify that you want the cluster to use CN2 as the networking technology.

```
curl --header "Content-Type: application/json" --request PATCH --data '{"\networking\": {"networkType\":"Contrail\"},"' -H "Authorization: Bearer $TOKEN" $AI_URL/api/assisted-install/v2/clusters/$CLUSTER_ID/install-config
```

- d. Review the changes.

```
curl -s -X GET --header "Content-Type: application/json" -H "Authorization: Bearer $TOKEN" $AI_URL/api/assisted-install/v2/clusters/$CLUSTER_ID/install-config | jq -r
```

Once you've registered your cluster, you can point your browser to the Red Hat Hybrid Cloud Console (<https://console.redhat.com/openshift>) to watch the progress of the installation. You can search for your cluster by cluster name or cluster ID.

4. Generate the discovery boot ISO. You will use this ISO to boot the nodes in your cluster.

The ISO is customized to your infrastructure based on the infrastructure environment that you'll set up.

- a. Create a file that describes the infrastructure environment. In this example, we name it **infra-envs.json**.

```
cat << EOF > ./infra-envs.json
{
  "name": "$CLUSTER_NAME",
  "ssh_authorized_key": "$CLUSTER_SSHKEY",
  "pull_secret": $PULL_SECRET,
  "image_type": "full-iso",
  "cluster_id": "$CLUSTER_ID",
  "openshift_version": "4.8",
  "cpu_architecture": "x86_64"
}
EOF
```

With early binding, the infrastructure environment includes the cluster details.

- d. Download the ISO. In this example, we call it ai-liveiso-*\$CLUSTER_ID*.iso, referencing the cluster identifier in the name.

```
export IMAGE_URL=$(curl -H "Authorization: Bearer $TOKEN" -L "$AI_URL/api/assisted-
install/v2/infra-envs/$INFRA_ENVS_ID/downloads/image-url" | jq -r '.url')
```

```
curl -H "Authorization: Bearer $TOKEN" -L "$IMAGE_URL" -o ./ai-liveiso-$CLUSTER_ID.iso
```

5. Boot the 3 control plane nodes with the discovery boot ISO.
 - a. Choose the boot method most convenient for your infrastructure. Ensure that the machines boot up attached to a network that has access to the Red Hat hosted Assisted Installer service. In the example network shown in [Figure 5 on page 27](#), the nodes have a single interface and that interface is attached to the 172.16.0.0/24 network, which has external connectivity to the Assisted Installer service.
 - b. Check the cluster status:

```
curl -s -X GET --header "Content-Type: application/json" -H "Authorization: Bearer
$TOKEN" -H "get_unregistered_clusters: false" $AI_URL/api/assisted-install/v2/clusters/
$CLUSTER_ID?with_hosts=true | jq -r '.status'
```

The status should indicate **ready** when the nodes come up successfully.

- c. Check the validations status:

```
curl -s -X GET --header "Content-Type: application/json" -H "Authorization: Bearer
$TOKEN" -H "get_unregistered_clusters: false" $AI_URL/api/assisted-install/v2/clusters/
$CLUSTER_ID?with_hosts=true | jq -r '.validations_info' | jq .
```

The validations status shows whether you've defined your cluster properly. There should be no errors in the output.

- d. Check the hosts:

```
curl -s -X GET --header "Content-Type: application/json" -H "Authorization: Bearer
$TOKEN" -H "get_unregistered_clusters: false" $AI_URL/api/assisted-install/v2/clusters/
$CLUSTER_ID?with_hosts=true | jq -r '.hosts'
```

The output shows details on all the nodes you've booted.

You can filter for specific information, such as host IDs:

```
curl -s -X GET --header "Content-Type: application/json" -H "Authorization: Bearer $TOKEN" -H "get_unregistered_clusters: false" $AI_URL/api/assisted-install/v2/clusters/$CLUSTER_ID?with_hosts=true | jq -r '.hosts[].id'
```

and host roles:

```
curl -s -X GET --header "Content-Type: application/json" -H "Authorization: Bearer $TOKEN" -H "get_unregistered_clusters: false" $AI_URL/api/assisted-install/v2/clusters/$CLUSTER_ID?with_hosts=true | jq -r '.hosts[].role'
```

- e. Confirm that the roles have been assigned.

On your browser, go to the Red Hat Hybrid Cloud Console (<https://console.redhat.com/openshift>) and click on your cluster to see details of your cluster. You can search for your cluster by cluster name or by cluster ID.

Proceed to the next step only when the nodes (hosts) come up and the roles have been assigned successfully as control plane nodes. Since we've only booted the 3 control plane nodes, you'll see the roles assigned in the UI as **control plane**, **worker**.

6. Repeat step 5 to boot up the worker nodes.

When you query the host roles, you'll see the worker nodes in **auto-assign** state. This is expected. The Assisted Installer service assigns roles for these nodes later when you install the cluster.

7. If you're running your worker nodes with a DPDK data path, then label each worker node that is running a DPDK data path as follows:

```
curl --location --request PATCH $AI_URL'/api/assisted-install/v2/infra-envs/$INFRA_ENVS_ID/hosts/$HOST_ID --header 'Content-Type: application/json' -H "Authorization: Bearer $TOKEN" --data-raw '{"node_labels":[{"key": "agent-mode", "value": "dpdk"}, {"key": "node-role.kubernetes.io/dpdk", "value": ""}]}'
```

where \$HOST_ID is the host identifier of the worker node that you want to run DPDK.

8. Upload the CN2 manifests (that you downloaded in ["Before You Install" on page 18](#)) to the Assisted Installer service.

For convenience, you can use the following bash script. The script assumes you've placed the manifests in the **manifests** directory. If you use this script, make sure that:

- you place all the manifests you want to use in this directory, including all manifests that you want to use from the subdirectories

- you don't place any other YAML files in this directory

The script loops through all the ***.yaml** files in the **manifests** directory, encodes them in base64, and applies them to the cluster.

Although the downloaded manifests package already contains the base64-encoded versions, we'll run this script to re-create them.

```
#!/bin/bash

MANIFESTS=(manifests/*.yaml)

total=${#MANIFESTS[@]}
i=0
for file in "${MANIFESTS[@]"; do
    i=$(( i + 1 ))
    eval "CONTRAIL_B64_MANIFEST=$(cat $file | base64 -w0)";
    eval "BASEFILE=$(basename $file)";
    eval "echo '{\"file_name\": \"\$BASEFILE\", \"folder\": \"manifests\", \"content\": \"\$CONTRAIL_B64_MANIFEST\"}' > $file.b64";
    printf "\nProcessing file: $file\n"
    curl -s -X POST "$AI_URL/api/assisted-install/v2/clusters/$CLUSTER_ID/manifests" -d @
$file.b64 --header "Content-Type: application/json" -H "Authorization: Bearer $TOKEN"
done
echo
echo
echo "Total Manifests: $total"
echo
echo "Manifest List:"
curl -s -X GET "$AI_URL/api/assisted-install/v2/clusters/$CLUSTER_ID/manifests" --header
"Content-Type: application/json" -H "Authorization: Bearer $TOKEN" | jq -r
```

9. Start the installation of the cluster.

```
curl -X POST "$AI_URL/api/assisted-install/v2/clusters/$CLUSTER_ID/actions/install" -H
"accept: application/json" -H "Content-Type: application/json" -H "Authorization: Bearer
$TOKEN" | jq
```

The Assisted Installer service creates the cluster based on the cluster resource that you defined. First, the Assisted installer assigns one of the control plane nodes as the bootstrap node, which in turn prepares the other nodes. One by one, you'll see the non-bootstrap nodes reboot into the

cluster, with the non-bootstrap control plane nodes rebooting first, then the worker nodes, and finally the bootstrap node.

The installation can take an hour or more. You can watch the progress of the installation on the Red Hat Hybrid Cloud Console (<https://console.redhat.com/openshift>).

If the Red Hat Hybrid Cloud Console shows the installation stalling, log in to each node with username **core** and make sure the host can resolve domain names, especially the ones you configured in your DNS/DHCP server as well as the Red Hat Assisted Installer service and Juniper Networks repository domain names. Most common installation problems can be traced back to network configuration errors such as incorrect DNS configuration. Also, in some environments, the nodes might shut down instead of rebooting. In that case, manually boot the nodes that have shut down.

10. Download the kubeconfig for the cluster.

- a. Download the kubeconfig to a local kubeconfig file.

```
curl -s -X GET "$AI_URL/api/assisted-install/v2/clusters/$CLUSTER_ID/downloads/credentials?file_name=kubeconfig" -H "accept: application/octet-stream" -H "Authorization: Bearer $TOKEN" > kubeconfig
```

- b. Copy the kubeconfig to **~/.kube/config**. The kubeconfig must be at that default **~/.kube/config** location because the **contrailstatus** command that you'll use later expects the kubeconfig to be at its default location.

11. Verify the installation.

- a. Check the status of all the pods. Confirm that all pods are either in Running or Completed states.

```
kubectl get pods -A -o wide
```

- b. For those pods not in Running or Completed states, use the **kubectl describe** command to investigate further.

```
kubectl describe pod <pod name> -n <namespace>
```

A common problem is failing to download an image. If this is the case:

- check that your network can reach the Juniper Networks repository
- check that the node where the failed pod resides is configured to access a DNS server
- check that the node where the failed pod resides can ping the repository by hostname

12. (Optional) Log in to the OpenShift Web Console to monitor your cluster.

The Web Console URL is in the form: **https://console-openshift-console.apps.<cluster-name>.<cluster-domain>**.

- a. Ensure the browser where you want to access the Web Console is on a machine that has access to the Web Console URL. You may need to add an entry for the hostname of that console to the `/etc/hosts` file on that machine. Recall that this mapping is the `*.apps.mycluster.contrail.lan` mapping that you configured in the DNS server in ["Before You Install" on page 18](#).
- b. Download the **kubeadmin** password.

```
curl -s -X GET "$AI_URL/api/assisted-install/v2/clusters/$CLUSTER_ID/downloads/credentials?file_name=kubeadmin-password" -H "accept: application/octet-stream" -H "Authorization: Bearer $TOKEN"
```

- c. Log in to the OpenShift Web Console with username **kubeadmin** and the downloaded password.

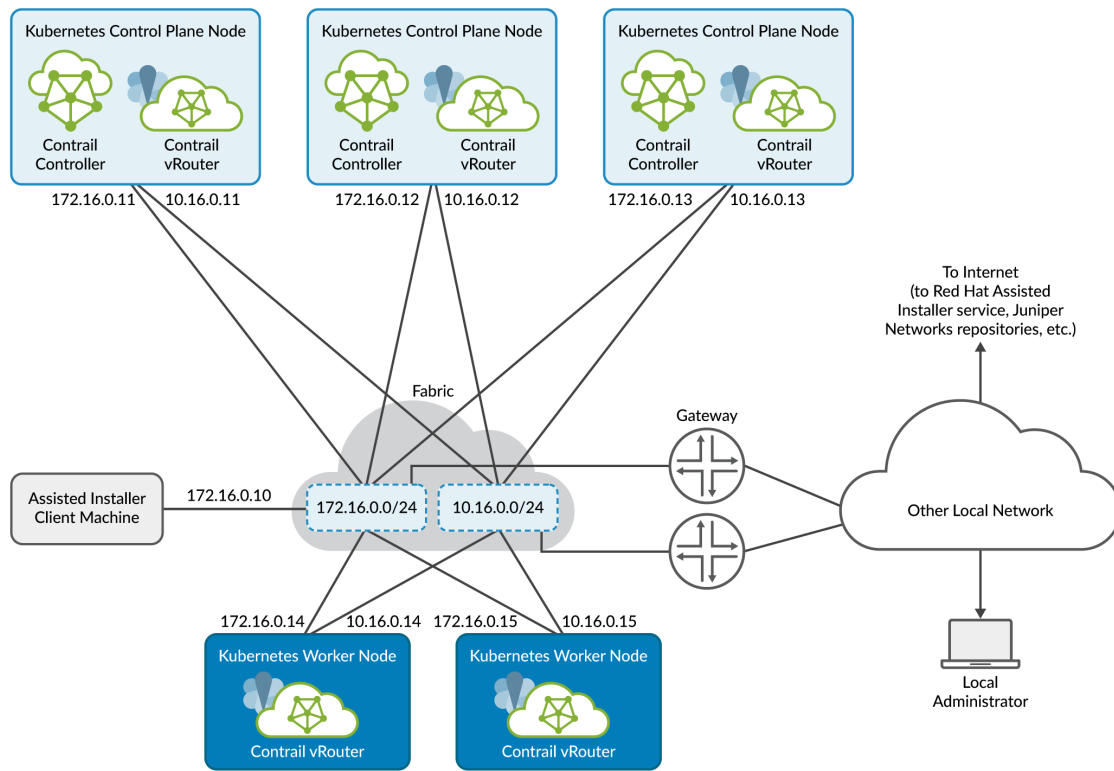
You have successfully installed OpenShift with CN2.

Install with Cluster-Managed Networking

Use this procedure to bring up a cluster with cluster-managed networking. Cluster-managed networking refers to a deployment where the Assisted Installer installs an integrated load balancer and ingress in the cluster for you.

[Figure 6 on page 39](#) shows a cluster of 3 control plane nodes and 2 worker nodes running on bare metal servers or virtual machines on an cluster-managed networking setup where the load balancer (not shown) is integrated within the cluster. Assisted Installer traffic and Kubernetes control plane traffic use the 172.16.0.0/24 fabric virtual network. Contrail control plane traffic and user data plane traffic use the 10.16.0.0/24 fabric virtual network.

Figure 6: CN2 OpenShift Cluster on Bare Metal Servers (or VMs) with Cluster-Managed Networking



A separate machine acts as the Assisted Installer client. The Assisted Installer client is where you use curl commands to issue API calls to the Assisted Installer service to create the cluster. In the example in this procedure, we use the Assisted Installer client machine to host a DNS/DHCP server for the subnet as well.

The local administrator is shown attached to a separate network in both diagrams. This is typical of many installations where the local administrator manages the fabric and cluster from the corporate LAN. In the procedures that follow, we refer to the local administrator station as your local computer.

NOTE: Connecting all nodes together is the data center fabric, which is shown in the example as two subnets. In real installations, the data center fabric is a network of spine-and-leaf switches that provide the physical connectivity for the cluster.

In an Apstra-managed data center, you would specify this connectivity through the overlay virtual networks that you create across the underlying fabric switches.

This procedure uses the Assisted Installer service hosted by Red Hat, and covers the more common early binding use case where you register the cluster before bringing up the hosts.

1. Log in to the Assisted Installer client machine. The Assisted Installer client machine is where you issue Assisted Installer API calls to the Assisted Installer service.
2. Prepare the deployment by setting the environment variables that you'll use in later steps.
 - a. Create an SSH key that you will use to access the nodes in your cluster. Save the key to an environment variable.

```
root@ai-client:~# ssh-keygen
```

In this example, we store the SSH key in its default location `~/.ssh/id_rsa.pub`.

```
export CLUSTER_SSHKEY=$(cat ~/.ssh/id_rsa.pub)
```

- b. Download the image pull secret from your Red Hat account onto your local computer. The pull secret allows your installation to access services and registries that serve container images for OpenShift components.

If you're using the Red Hat hosted Assisted Installer, you can download the pull secret file (**pull-secret**) from the [downloads](#) page. Copy the pull-secret file to the Assisted Installer client machine. In this example, we store the pull-secret in a file called **pull-secret.txt**.

Strip out any whitespace, convert the contents to JSON string format, and store to an environment variable, as follows:

```
export PULL_SECRET=$(sed '/^[[:space:]]*$/d' pull-secret.txt | jq -R .)
```

- c. Copy the offline access token from your Red Hat account. The OpenShift Cluster Manager API Token allows you (on the Assisted Installer client machine) to interact with the Assisted Installer API service hosted by Red Hat.

The token is a string that you can copy and paste to a local environment variable. If you're using the Red Hat hosted Assisted Installer, you can copy the API token from the [downloads](#) page.

```
export OFFLINE_ACCESS_TOKEN='<paste offline access token here>'
```

- d. Generate (refresh) the token from the OFFLINE_ACCESS_TOKEN. You will use this generated token whenever you issue API commands.

```
export TOKEN=$(curl --silent --data-urlencode "grant_type=refresh_token" --data-urlencode "client_id=cloud-services" --data-urlencode "refresh_token=$
```

```
{OFFLINE_ACCESS_TOKEN}" https://sso.redhat.com/auth/realms/redhat-external/protocol/
openid-connect/token | jq -r .access_token)
```

NOTE: This token expires regularly. When this token expires, you will get an HTTP 400 response whenever you issue an API command. Refresh the token when it expires, or alternatively, refresh the token regularly prior to expiry. There is no harm in refreshing the token when it hasn't expired.

- e. Set up the remaining environment variables.

[Table 8 on page 41](#) lists all the environment variables that you need to set in this procedure, including the ones described in the previous steps.

Table 8: Environment Variables

Variable	Description	Example
CLUSTER_SSHKEY	The (public) ssh key that you generated. This key will be installed on all cluster nodes.	–
PULL_SECRET	The image pull secret that you downloaded, stripped and converted to JSON string format.	–
OFFLINE_ACCESS_TOKEN	The OpenShift Cluster Manager API Token that you copied.	–
TOKEN	The token that you generated (refreshed) from the OFFLINE_ACCESS_TOKEN.	–
CLUSTER_NAME	The name you want to call the cluster. This is the name that will appear in the Red Hat Hybrid Cloud Console UI. NOTE: This name must be in lowercase.	mycluster

Table 8: Environment Variables (*Continued*)

Variable	Description	Example
CLUSTER_DOMAIN	The base domain that you want to assign to the cluster. Cluster objects will be assigned names in this domain.	contrail.lan
CLUSTER_NET	The overlay cluster network. Pods are assigned IP addresses on this network.	10.128.0.0/14
CLUSTER_SVC_NET	The overlay service network. Services are assigned IP addresses on this network.	172.31.0.0/16
CLUSTER_HOST_PFX	The subnet prefix length for assigning IP addresses from CLUSTER_NET. This defines the subset of CLUSTER_NET IP addresses to use for pod IP address assignment.	23
CLUSTER_MACHINE_NET	The machine network where the hosts attach. This is the fabric virtual network for Kubernetes control plane traffic.	172.16.0.0/24
CLUSTER_API_VIP	<p>The virtual IP address that you want to assign to the API endpoint. This is the front end IP address that the internal load balancer will use to distribute API traffic to back end nodes.</p> <p>Select an IP address within the CLUSTER_MACHINE_NET that is not in use.</p>	172.16.0.100

Table 8: Environment Variables (*Continued*)

Variable	Description	Example
CLUSTER_INGRESS_VIP	The virtual IP address that you want to assign as the ingress. This is the front end IP address that the internal load balancer will use to distribute application traffic to back end nodes. Select an IP address within the CLUSTER_MACHINE_NET that is not in use.	172.16.0.101
AI_URL	The URL of the Assisted Installer service. This example uses the Red Hat hosted Assisted Installer.	https://api.openshift.com

3. Register the cluster with the Assisted Installer service. By registering, you're telling the Assisted Installer service about the attributes of the cluster you want to create. In response, the Assisted Installer service creates a cluster resource and returns a cluster identifier that uniquely identifies that cluster resource.
 - a. Create the file that describes the cluster you want to create. In this example, we name the file **deployment.json**.

```
cat << EOF > ./deployment.json
{
  "kind": "Cluster",
  "name": "$CLUSTER_NAME",
  "openshift_version": "<openshift_version>",
  "ocp_release_image": "quay.io/openshift-release-dev/ocp-release:<coreOS_version>-x86_64",
  "base_dns_domain": "$CLUSTER_DOMAIN",
  "hyperthreading": "all",
  "machine_networks": [{"cidr": "$CLUSTER_MACHINE_NET"}],
  "ingress_vip": "$CLUSTER_INGRESS_VIP",
  "cluster_network_cidr": "$CLUSTER_NET",
  "cluster_network_host_prefix": $CLUSTER_HOST_PFX,
  "service_network_cidr": "$CLUSTER_SVC_NET",
  "user_managed_networking": false,
```

```
"vip_dhcp_allocation": false,
"ssh_public_key": "$CLUSTER_SSHKEY",
"pull_secret": $PULL_SECRET
}
EOF
```

NOTE: Ensure you specify a *<coreOS_version>* (for example, 4.8.39) that is listed in https://www.juniper.net/documentation/en_US/release-independent/contrail-cloud-native/topics/reference/cloud-native-contrail-supported-platforms.pdf for the *<openshift_version>* (for example, 4.8) you're using.

- b. Register the cluster and save the CLUSTER_ID to an environment variable. Reference the **deployment.json** file you just created.

```
export CLUSTER_ID=$(curl -s -X POST "$AI_URL/api/assisted-install/v2/clusters" -d @./
deployment.json --header "Content-Type: application/json" -H "Authorization: Bearer
$TOKEN" | jq -r '.id')
```

- c. Update the cluster attributes to specify that you want the cluster to use CN2 as the networking technology.

```
curl --header "Content-Type: application/json" --request PATCH --data '{"\networking\":
{\networkType\":"Contrail\"}' -H "Authorization: Bearer $TOKEN" $AI_URL/api/assisted-
install/v2/clusters/$CLUSTER_ID/install-config
```

- d. Review the changes.

```
curl -s -X GET --header "Content-Type: application/json" -H "Authorization: Bearer
$TOKEN" $AI_URL/api/assisted-install/v2/clusters/$CLUSTER_ID/install-config | jq -r
```

Once you've registered your cluster, you can point your browser to the Red Hat Hybrid Cloud Console (<https://console.redhat.com/openshift>) to watch the progress of the installation. You can search for your cluster by cluster name or cluster ID.

4. Generate the discovery boot ISO. You will use this ISO to boot the nodes in your cluster. The ISO is customized to your infrastructure based on the infrastructure environment that you'll set up.


```
JlY3RsIC1uIGt1YmUtc3lzdGVtIHBhdGNoIGNvbWZpZ21hcCBleHRlbnNpb24tYXBpc2VydmlWLF1dGhlbnRpY2F0aW9uIC0tcGF0Y2gtZmlsZSAke3RtcH0KICBpZiBbWyAkPyAtZXEgMCBdXTsgdGhlbgoJcm0gJHt0bXB9CglzdWNjZXNzPTIKICBmaQogIHJtICR7dG1wfQogIHNSZWVwIDYwCmRvbmUK\}}\},\kernelArguments\":{\shouldExist\":[\ipv6.disable=1\}}}]}
```

Apply the ignition file that you just created.

```
curl -s -X PATCH $AI_URL/api/assisted-install/v2/infra-envs/$INFRA_ENVS_ID -d @infra-ignition.json --header "Content-Type: application/json" -H "Authorization: Bearer $TOKEN" | jq -r '.id'
```

- d. Download the ISO. In this example, we call it ai-liveiso-*\$CLUSTER_ID*.iso, referencing the cluster identifier in the name.

```
export IMAGE_URL=$(curl -H "Authorization: Bearer $TOKEN" -L "$AI_URL/api/assisted-install/v2/infra-envs/$INFRA_ENVS_ID/downloads/image-url" | jq -r '.url')

curl -H "Authorization: Bearer $TOKEN" -L "$IMAGE_URL" -o /var/lib/libvirt/images/ai-liveiso-$CLUSTER_ID.iso
```

5. Boot the 3 control plane nodes with the discovery boot ISO.
 - a. Choose the boot method most convenient for your infrastructure. Ensure that the machines boot up attached to a network that has access to the Red Hat hosted Assisted Installer service. In the example network shown in [Figure 6 on page 39](#), the nodes have two interfaces, with one interface attached to the 172.16.0.0/24 network and the other interface attached to the 10.16.0.0/24 network.

NOTE: Configure your DHCP server to assign IP addresses on the 10.16.0.0/24 network (for example, 10.16.0.11 through 10.16.0.13 for the control plane nodes and 10.16.0.14 and 10.16.0.15 for the worker nodes.)

Be sure to configure the 99-network-configmap.yaml manifest according to step [4.e](#) in ["Before You Install" on page 18](#).

- b. Check the cluster status:

```
curl -s -X GET --header "Content-Type: application/json" -H "Authorization: Bearer $TOKEN" -H "get_unregistered_clusters: false" $AI_URL/api/assisted-install/v2/clusters/$CLUSTER_ID?with_hosts=true | jq -r '.status'
```

The status should indicate **ready** when the nodes come up successfully.

- c. Check the validations status:

```
curl -s -X GET --header "Content-Type: application/json" -H "Authorization: Bearer $TOKEN" -H "get_unregistered_clusters: false" $AI_URL/api/assisted-install/v2/clusters/$CLUSTER_ID?with_hosts=true | jq -r '.validations_info' | jq .
```

The validations status shows whether you've defined your cluster properly. There should be no errors in the output.

- d. Check the hosts:

```
curl -s -X GET --header "Content-Type: application/json" -H "Authorization: Bearer $TOKEN" -H "get_unregistered_clusters: false" $AI_URL/api/assisted-install/v2/clusters/$CLUSTER_ID?with_hosts=true | jq -r '.hosts'
```

The output shows details on all the nodes you've booted.

You can filter for specific information, such as host IDs:

```
curl -s -X GET --header "Content-Type: application/json" -H "Authorization: Bearer $TOKEN" -H "get_unregistered_clusters: false" $AI_URL/api/assisted-install/v2/clusters/$CLUSTER_ID?with_hosts=true | jq -r '.hosts[].id'
```

and host roles:

```
curl -s -X GET --header "Content-Type: application/json" -H "Authorization: Bearer $TOKEN" -H "get_unregistered_clusters: false" $AI_URL/api/assisted-install/v2/clusters/$CLUSTER_ID?with_hosts=true | jq -r '.hosts[].role'
```

- e. Confirm that the roles have been assigned.

On your browser, go to the Red Hat Hybrid Cloud Console (<https://console.redhat.com/openshift>) and click on your cluster to see details of your cluster. You can search for your cluster by cluster name or by cluster ID.

Proceed to the next step only when the nodes (hosts) come up and the roles have been assigned successfully as control plane nodes. Since we've only booted the 3 control plane nodes, you'll see the roles assigned in the UI as **control plane, worker**.

6. Set the virtual IP address for API traffic.

```
curl -X PATCH "$AI_URL/api/assisted-install/v2/clusters/$CLUSTER_ID" -H "accept: application/json" -H "Content-Type: application/json" -H "Authorization: Bearer $TOKEN" -d "{ \"api_vip\": \" $CLUSTER_API_VIP\"}"
```

7. Repeat step 5 to boot up the worker nodes.

When you query the host roles, you'll see the worker nodes in **auto-assign** state. This is expected. The Assisted Installer service assigns roles for these nodes later when you install the cluster.

8. If you're running your worker nodes with a DPDK data path, then label each worker node that is running a DPDK data path as follows:

```
curl --location --request PATCH $AI_URL'/api/assisted-install/v2/infra-envs/$INFRA_ENVS_ID/hosts/$HOST_ID --header 'Content-Type: application/json' -H "Authorization: Bearer $TOKEN" --data-raw '{"node_labels":[{"key": "agent-mode", "value": "dpdk"}, {"key": "node-role.kubernetes.io/dpdk", "value": ""}]}'
```

where \$HOST_ID is the host identifier of the worker node that you want to run DPDK.

9. Upload the CN2 manifests (that you downloaded in ["Before You Install" on page 18](#)) to the Assisted Installer service. Be sure to include the additional manifests that are specific to cluster-managed networking.

For convenience, you can use the following bash script. The script assumes you've placed the manifests in the **manifests** directory. If you use this script, make sure that:

- you place all the manifests you want to use in this directory, including all manifests that you want to use from the subdirectories
- you don't place any other YAML files in this directory

The script loops through all the *.yaml files in the **manifests** directory, encodes them in base64, and applies them to the cluster.

```
#!/bin/bash

MANIFESTS=(manifests/*.yaml)

total=${#MANIFESTS[@]}
i=0
for file in "${MANIFESTS[@]"; do
    i=$(( i + 1 ))
    eval "CONTRAIL_B64_MANIFEST=$(cat $file | base64 -w0)";
```

```

eval "BASEFILE=$(basename $file)";
eval "echo '{\"file_name\": \"\$BASEFILE\", \"folder\": \"manifests\", \"content
\": \"\$CONTRAIL_B64_MANIFEST\"}' > \$file.b64";
printf "\nProcessing file: $file\n"
curl -s -X POST "$AI_URL/api/assisted-install/v2/clusters/$CLUSTER_ID/manifests" -d @
$file.b64 --header "Content-Type: application/json" -H "Authorization: Bearer $TOKEN"
done
echo
echo
echo "Total Manifests: $total"
echo
echo "Manifest List:"
curl -s -X GET "$AI_URL/api/assisted-install/v2/clusters/$CLUSTER_ID/manifests" --header
"Content-Type: application/json" -H "Authorization: Bearer $TOKEN" | jq -r

```

10. Start the installation of the cluster.

```

curl -X POST "$AI_URL/api/assisted-install/v2/clusters/$CLUSTER_ID/actions/install" -H
"accept: application/json" -H "Content-Type: application/json" -H "Authorization: Bearer
$TOKEN" | jq

```

The Assisted Installer service creates the cluster based on the cluster resource that you defined. First, the Assisted installer assigns one of the control plane nodes as the bootstrap node, which in turn prepares the other nodes. One by one, you'll see the non-bootstrap nodes reboot into the cluster, with the non-bootstrap control plane nodes rebooting first, then the worker nodes, and finally the bootstrap node.

The installation can take an hour or more. You can watch the progress of the installation on the Red Hat Hybrid Cloud Console (<https://console.redhat.com/openshift>).

If the Red Hat Hybrid Cloud Console shows the installation stalling, log in to each node with username **core** and make sure the host can resolve domain names, especially the ones you configured in your DNS/DHCP server as well as the Red Hat Assisted Installer service and Juniper Networks repository domain names. Most common installation problems can be traced back to network configuration errors such as incorrect DNS configuration. Also, in some environments, the nodes might shut down instead of rebooting. In that case, manually start up the nodes that have shut down.

11. Download the kubeconfig for the cluster.

- a. Download the kubeconfig to a local kubeconfig file.

```
curl -s -X GET "$AI_URL/api/assisted-install/v2/clusters/$CLUSTER_ID/downloads/
credentials?file_name=kubeconfig" -H "accept: application/octet-stream" -H
"Authorization: Bearer $TOKEN" > kubeconfig
```

- b. Copy the kubeconfig to `~/.kube/config`. The kubeconfig must be at that default `~/.kube/config` location because the `contrailstatus` command that you'll use later expects the kubeconfig to be at its default location.

12. Verify the installation.

- a. Check the status of all the pods. Confirm that all pods are either in Running or Completed states.

```
kubectl get -pods -A -o wide
```

- b. For those pods not in Running or Completed states, use the `kubectl describe` command to investigate further.

```
kubectl describe pod <pod name> -n <namespace>
```

A common problem is failing to download an image. If this is the case:

- check that your network can reach the Juniper Networks repository
- check that the node where the failed pod resides is configured to access a DNS server
- check that the node where the failed pod resides can ping the repository by hostname

13. (Optional) Log in to the OpenShift Web Console to monitor your cluster.

The Web Console URL is in the form: `https://console-openshift-console.apps.<cluster-name>.<cluster-domain>`.

- a. Ensure the browser where you want to access the Web Console is on a machine that has access to the Web Console URL. You may need to add an entry for the hostname of that console to the `/etc/hosts` file on that machine. Recall that this mapping is the `*.apps.mycluster.contrail.lan` mapping that you configured in the DNS server in ["Before You Install" on page 18](#).

- b. Download the `kubeadmin` password.

```
curl -s -X GET "$AI_URL/api/assisted-install/v2/clusters/$CLUSTER_ID/downloads/
credentials?file_name=kubeadmin-password" -H "accept: application/octet-stream" -H
"Authorization: Bearer $TOKEN"
```

- c. Log in to the OpenShift Web Console with username **kubeadmin** and the downloaded password.

You have successfully installed OpenShift with CN2.

Manifests and Tools

SUMMARY

We provide tools and sample manifests to make your installation easier. You can download these manifests and tools from the Juniper Networks software download site or from GitHub.

IN THIS SECTION

- [Manifests and Tools in Release 22.4](#) | 51

Manifests and Tools in Release 22.4

IN THIS SECTION

- [Contrail Analytics](#) | 55

The Manifests and Tools for OpenShift package is available for download from the Juniper Networks software download site (<https://support.juniper.net/support/downloads/?p=contrail-networking>) or from GitHub (<https://github.com/Juniper/contrail-networking/tree/main/releases/22.4>).

If you're downloading from the Juniper Networks software download site, you'll need an account to download. If you don't have an account, contact your Juniper Networks sales representative to have one created for you. The package is called **contrail-manifests-openshift-22.4.0.284.tgz**.

The following table lists the manifests in that package.

Table 9: Manifests for Release 22.4

Manifests	User-Managed Networking	Cluster-Managed Networking	Description
contrail-manifests-openshift/100*.yaml through 113*.yaml	Mandatory	Mandatory	Main CN2 manifests.
contrail-manifests-openshift/auth-registry/*.yaml	Mandatory	Mandatory	<p>Image pull secrets for pulling images from the Juniper Networks repository.</p> <p>Edit these manifests to replace the placeholder credentials with your repository login credentials. See step 4.d in "Before You Install" on page 18.</p>
contrail-manifests-openshift/cert-manager-1.8/*.yaml	Mandatory	Mandatory	Manifests needed for managing certificates to access the Contrail etcd database.
contrail-manifests-openshift/99-disable-offload-master.yaml	<p>Mandatory if running on VMs</p> <p>Otherwise, not needed.</p>	<p>Mandatory if running on VMs</p> <p>Otherwise, not needed.</p>	<p>Disables checksum offloads on the named interface.</p> <p>Edit this manifest to reference the interface used for Kubernetes control plane traffic in the VM that is acting as a control plane node. See step 4.e in "Before You Install" on page 18.</p>

Table 9: Manifests for Release 22.4 (Continued)

Manifests	User-Managed Networking	Cluster-Managed Networking	Description
contrail-manifests-openshift/99-disable-offload-worker.yaml	Mandatory if running on VMs Otherwise, not needed.	Mandatory if running on VMs Otherwise, not needed.	Disables checksum offloads on the named interface. Edit this manifest to reference the interface used for Kubernetes control plane traffic in the VM that is acting as a worker node. See step 4.e in "Before You Install" on page 18 .
contrail-manifests-openshift/vrrp/*.yaml			Manifests used for cluster-managed networking. See entries below for usage.
contrail-manifests-openshift/vrrp/99-disable-offload-master-vrrp.yaml	Not needed	Mandatory if running on VMs Otherwise, not needed.	Disables checksum offloads on the named interface. Edit this manifest to reference the interface used for Contrail control and data traffic in the VM that is acting as a control plane node. See step 4.e in "Before You Install" on page 18 .

Table 9: Manifests for Release 22.4 (Continued)

Manifests	User-Managed Networking	Cluster-Managed Networking	Description
contrail-manifests-openshift/vrrp/99-disable-offload-worker-vrrp.yaml	Not needed	Mandatory if running on VMs. Otherwise, not needed.	Disables checksum offloads on the named interface. Edit this manifest to reference the interface used for Contrail control and data traffic in the VM that is acting as a worker node. See step 4.e in "Before You Install" on page 18 .
contrail-manifests-openshift/vrrp/99-network-configmap.yaml	Not needed	Mandatory	Specifies the network used for Contrail control plane and user data plane traffic. Edit this manifest to reference the correct fabric network. See step 4.f in "Before You Install" on page 18 .
contrail-manifests-openshift/plugins/*.yaml			Manifests used to integrate with non-CN2 components. See entries below for usage.
contrail-manifests-openshift/plugins/*apstra*.yaml	Optional	Optional	Manifests used to integrate CN2 with Juniper Apstra. By integrating CN2 with Juniper Apstra, you can extend virtual networks into the fabric. See step 4.g in "Before You Install" on page 18 .

The following table lists miscellaneous tools in that package.

Table 10: Miscellaneous Tools for Release 22.4

Tools	Description
contrail-tools/contrail-readiness/contrail-readiness-controller.yaml	The ContrailReadiness controller that runs postflight checks
contrail-tools/contrail-readiness/contrail-readiness-preflight.yaml	Not applicable for OpenShift installations
contrail-tools/contrail-readiness/contrail-readiness-postflight.yaml	ContrailReadiness postflight custom resource
contrail-tools/contrail-readiness/crds	ContrailReadiness custom resource definitions for postflight checks
contrail-tools/uninstall.tgz	Not applicable for OpenShift installations
contrail-tools/kubectl-contrailstatus.tgz	The kubectl contrailstatus plug-in

Contrail Analytics

The optional Contrail Analytics package is available for download from the Juniper Networks software download <https://support.juniper.net/support/downloads/?p=contrail-networking> site. Select the Contrail Analytics package from the same release page that you select the Contrail Networking manifests. Contrail Analytics is compatible with Contrail Networking within the same release only.

You'll need an account to download. If you don't have an account, contact your Juniper Networks sales representative to have one created for you.

The release 22.4 package is called **contrail-analytics-22.4.0.284.tgz**. See "[Install Contrail Analytics](#)" on [page 57](#).

3

CHAPTER

Monitor

[Overview](#) | 57

[Install Contrail Analytics](#) | 57

[Kubectl Contrailstatus](#) | 59

Overview

You can monitor CN2 the same way you monitor other OpenShift components, using the Red Hat OpenShift console or any other OpenShift tool.

You can also install the optional Contrail Analytics package, which integrates CN2 telemetry exporters within the OpenShift monitoring framework to provide you with insight into the general health, performance, and traffic trends of the network.

Additionally, we provide a kubectl plug-in that you can invoke to check the status of CN2 components. The contrailstatus plug-in allows you to query the CN2 configuration, control, and data plane components as well as BGP and XMPP relationships.

Install Contrail Analytics

SUMMARY

Learn how to install Contrail Analytics.

IN THIS SECTION

- [Install Contrail Analytics in Release 22.4 | 57](#)

Contrail Analytics integrates CN2's telemetry exporters in the OpenShift monitoring framework to provide a unified way for you to monitor and analyze your network and network infrastructure. Information collected includes logs, metrics, status' of various components, and flows.

When you install Contrail Analytics, all analytics components are preconfigured to work with each other.

NOTE: We use Helm charts to install Contrail Analytics. Install Helm 3.0 or later on the host that you're using to install Contrail Analytics.

Install Contrail Analytics in Release 22.4

Use this procedure to install Contail Analytics in release 22.4

1. Install OpenShift Logging.

Contrail Analytics requires OpenShift Logging to be installed. For information on how to install OpenShift Logging, see <https://docs.openshift.com/container-platform/4.8/logging/cluster-logging-deploying.html>.

2. Download the Contrail Analytics package. See "Contrail Analytics" on page 55 in "Manifests and Tools" on page 51.

```
user@ai-client:~/contrail# ls
contrail-analytics-<version>.tgz
```

3. Create the project and namespace for the Contrail Analytics pods.

```
oc new-project contrail-analytics --display-name="Contrail analytics"
```

4. Extract the **openshift-values.yaml** file from the Contrail Analytics package. This file contains values that override some OpenShift defaults.

```
user@ai-client:~/contrail# tar --strip=1 -xzf contrail-analytics-<version>.tgz contrail-analytics/openshift-values.yaml
```

```
user@ai-client:~/contrail# ls
contrail-analytics-<version>.tgz
openshift-values.yaml
```

5. Install the Contrail Analytics package.

```
user@ai-client:~/contrail# helm -n contrail-analytics install analytics contrail-analytics-<version>.tgz -f openshift-values.yaml
```

6. Verify that the analytics components are installed and running.

```
helm -n contrail-analytics list
```

```
kubectl get pods -n contrail-analytics
```

7. To uninstall Contrail Analytics:

```
helm -n contrail-analytics uninstall analytics
```

```
oc delete project contrail-analytics
```

8. To upgrade Contrail Analytics:

```
helm -n contrail-analytics upgrade analytics contrail-analytics-<version>.tgz -f openshift-values.yaml
```

Kubectl Contrailstatus

IN THIS SECTION

- [Syntax | 59](#)
- [Description | 60](#)
- [Options | 60](#)
- [Additional Information | 61](#)
- [Output Fields | 61](#)
- [Sample Output | 62](#)
- [Release Information | 63](#)

Syntax

```
kubectl contrailstatus deployment --plane { config | control | data [--wide] }
kubectl contrailstatus resource { bgprouter [BGP | XMPP] | globalssystemconfig | routinginstance
| virtualnetwork [--wide] }
kubectl contrailstatus cresource all [detail]
kubectl contrailstatus configdump
```

```
kubectrl contrailstatus --all [--wide]
kubectrl contrailstatus version
```

Description

This command displays the status' of various CN2 components. You can display the status' of the Configuration plane components, the Control plane components, the Data plane components, and the BGP routers and other resources.

Options

kubectrl contrailstatus deployment --plane config	Displays the status of the Configuration plane components: <ul style="list-style-type: none"> • contrail-k8s-apiserver • contrail-k8s-controller • contrail-k8s-kubemanager
kubectrl contrailstatus deployment --plane control	Displays the status of the Control plane components: <ul style="list-style-type: none"> • contrail-control
kubectrl contrailstatus deployment --plane data	Displays the status of the Data plane components: <ul style="list-style-type: none"> • contrail-vrouter-masters • contrail-vrouter-nodes
kubectrl contrailstatus resource bgprouter	Displays the status' of the various BGP and XMPP neighbor relationships.
kubectrl contrailstatus resource globalsystemconfig	Displays the status of the GlobalSystemConfig.
kubectrl contrailstatus resource routinginstance	Displays the status' of the various RoutingInstances in CN2.
kubectrl contrailstatus resource virtualnetwork	Displays the status' of the various VirtualNetworks in CN2.
kubectrl contrailstatus cresource all	Displays all information about all resources (useful for displaying all information in a single command for debugging). If the detail option is used, the output is displayed in JSON format.

- kubectl contrailstatus configdump

Lists the resources and their quantities.
- kubectl contrailstatus --all

Displays the status' of the Configuration/Control/Data planes and the BGP and XMPP relationships.
- kubectl contrailstatus version

Displays the versions of the various container images.

Additional Information

The `--wide` qualifier displays more information (if available) on the queried component.

Use the `--help` qualifier to display the help at any point in the command.

This command looks for the kubeconfig file in the default `~/.kube/config` location. You can't use the `kubectl --kubeconfig` option to specify the location of the kubeconfig file.

Output Fields

[Table 11 on page 61](#) lists some of the output fields for the `kubectl contrailstatus` command.

Table 11: kubectl contrailstatus Output Fields

Field Name	Field Description
NAME	The name of the pod or resource.
STATUS	The status of the pod or resource.
NODE	The name of the node on which the pod is running.
IP	The (machine) IP address of the node on which the pod is running.
MESSAGE	Not used.
LOCAL BGPROUTER	The name of the node on which the local BGP router is running.
NEIGHBOR BGPROUTER	The name of the node on which the neighbor BGP router is running.

Table 11: kubectl contrailstatus Output Fields (Continued)

Field Name	Field Description
ENCODING	Whether this connection is XMPP or BGP.
STATE	The state of this connection.
POD	The name of the pod on which the local BGP router is running.

Sample Output

kubectl contrailstatus --all

```
user@host> kubectl contrailstatus --all
```

```
NAME(CONFIG)          STATUS  NODE   IP           MESSAGE
contrail-k8s-apiserver-6d79c8598d-8lfnm    ok      ocp1   172.16.0.11
contrail-k8s-apiserver-6d79c8598d-q7klk    ok      ocp3   172.16.0.13
contrail-k8s-apiserver-6d79c8598d-szdzf    ok      ocp2   172.16.0.12
contrail-k8s-controller-96964f568-csk2k    ok      ocp1   172.16.0.11
contrail-k8s-controller-96964f568-dshn6    ok      ocp3   172.16.0.13
contrail-k8s-controller-96964f568-hfrpl    ok      ocp2   172.16.0.12
contrail-k8s-kubemanager-79b577ff86-6v8qt  ok      ocp3   172.16.0.13
contrail-k8s-kubemanager-79b577ff86-cbh5n  ok      ocp1   172.16.0.11
contrail-k8s-kubemanager-79b577ff86-vmckw  ok      ocp2   172.16.0.12
```

```
NAME(CONTROL)        STATUS  NODE   IP           MESSAGE
contrail-control-0    ok      ocp1   172.16.0.11
contrail-control-1    ok      ocp2   172.16.0.12
contrail-control-2    ok      ocp3   172.16.0.13
```

```
LOCAL  BGPROUTER  NEIGHBOR  BGPROUTER    ENCODING    STATE    POD
```

ocp1	ocp2	BGP	Established ok	contrail-control-0
ocp1	ocp3	BGP	Established ok	contrail-control-0
ocp1	ocp1	XMPP	Established ok	contrail-control-0
ocp1	ocp2	XMPP	Established ok	contrail-control-0
ocp1	ocp3	XMPP	Established ok	contrail-control-0
ocp1	ocp4	XMPP	Established ok	contrail-control-0
ocp1	ocp5	XMPP	Established ok	contrail-control-0
ocp2	ocp3	BGP	Established ok	contrail-control-1
ocp2	ocp1	BGP	Established ok	contrail-control-1
ocp2	ocp2	XMPP	Established ok	contrail-control-1
ocp2	ocp3	XMPP	Established ok	contrail-control-1
ocp2	ocp4	XMPP	Established ok	contrail-control-1
ocp2	ocp5	XMPP	Established ok	contrail-control-1
ocp3	ocp1	BGP	Established ok	contrail-control-2
ocp3	ocp2	BGP	Established ok	contrail-control-2
ocp3	ocp1	XMPP	Established ok	contrail-control-2

NAME(DATA)	STATUS	NODE	IP	MESSAGE
contrail-vrouter-masters-dspzb	ok	ocp3	172.16.0.13	
contrail-vrouter-masters-ks249	ok	ocp2	172.16.0.12	
contrail-vrouter-masters-tn6jz	ok	ocp1	172.16.0.11	
contrail-vrouter-nodes-mjwt2	ok	ocp4	172.16.0.14	
contrail-vrouter-nodes-rp5np	ok	ocp5	172.16.0.15	

Release Information

Table 12: Summary of Changes

Release	Changes
22.1	Initial release.
22.4	Updated version command to include image versions. Added cresource and configdump commands.

4

CHAPTER

Manage

[Overview](#) | 65

[Back Up and Restore Contrail Etcd](#) | 65

[Upgrade CN2](#) | 70

Overview

The way that you manage an OpenShift cluster does not change when CN2 is the CNI plug-in. Once CN2 is installed, CN2 components work seamlessly with OpenShift components to provide the networking infrastructure.

The Contrail controller is constantly watching and reacting to cluster events as they occur. When you add a new node, the Contrail data plane components are automatically deployed. When you delete a node, the Contrail controller automatically deletes networking resources associated with that node. CN2 works seamlessly with `kubectl` and `oc` commands as well as with the OpenShift console.

You cannot uninstall CN2. You cannot remove a CNI plug-in from an OpenShift cluster.

See the Appendix for examples of how to perform some common tasks on an OpenShift cluster. These examples are not specific to CN2 and are provided with no warranty.

The remainder of this chapter contains tasks that are specific to CN2.

Back Up and Restore Contrail Etcd

SUMMARY

Learn how to back up and restore the Contrail etcd database.

IN THIS SECTION

- [Back Up the Contrail Etcd Database in Release 22.4 | 66](#)
- [Restore the Contrail Etcd Database in Release 22.4 | 67](#)

In release 22.1, CN2 stores its data in the main OpenShift etcd database. When you back up and restore the main etcd database in release 22.1, you implicitly back up and restore CN2 data.

Starting in release 22.2, CN2 stores its data in its own etcd database. Use the procedures in this section to back up and restore the Contrail etcd database.

Back Up the Contrail Etcd Database in Release 22.4

Use this example procedure in release 22.4 to back up the Contrail etcd database.

1. Install etcdctl on all control plane nodes.

- a. Log in to one of the control plane nodes.

For example:

```
ssh core@172.16.0.11
```

- b. Download etcd. This example downloads to the **/tmp** directory.

```
ETCD_VER=v3.4.13
curl -L https://storage.googleapis.com/etcd/${ETCD_VER}/etcd-${ETCD_VER}-linux-
amd64.tar.gz -o /tmp/etcd-${ETCD_VER}-linux-amd64.tar.gz
```

- c. Untar and move the etcd executable to a directory in your path (for example **/usr/local/bin**).

```
tar -xzf /tmp/etcd-${ETCD_VER}-linux-amd64.tar.gz -C /tmp
sudo mv /tmp/etcd-${ETCD_VER}-linux-amd64/etcdctl /usr/local/bin
```

- d. Check that you've installed etcd.

```
[core@ocp1]$ etcdctl version
etcdctl version: 3.4.13
API version: 3.4
```

- e. Repeat on all the control plane nodes.

2. Get a list of the contrail-etcd pods.

```
user@ai-client:~# kubectl get pods -A | grep contrail-etcd
```

Take note of the contrail-etcd pod names, the IP addresses, and the nodes they're running on. You'll need this information in the next few steps.

3. Copy the etcd certificate and key files from the pods to the control plane nodes.

We run kubectl on the control plane nodes in this step. We assume you've set up kubeconfig on the control plane nodes in its default location (**~/.kube/config**).

- a. Pick a contrail-etcd pod (for example, contrail-etcd-0) and log in to the control plane node that's hosting that pod.
- b. Copy the certificate and key files from that contrail-etcd pod to the hosting control plane node. In this example, we're copying the certificates and key files from the contrail-etcd-0 pod to local files on this node.

```
kubectl exec --namespace contrail-system contrail-etcd-0 -c contrail-etcd -- cat /etc/
member-tls/ca.crt > ./ca.crt
kubectl exec --namespace contrail-system contrail-etcd-0 -c contrail-etcd -- cat /etc/
member-tls/tls.crt > ./tls.crt
kubectl exec --namespace contrail-system contrail-etcd-0 -c contrail-etcd -- cat /etc/
member-tls/tls.key > ./tls.key
```

This copies the certificate and key files from the contrail-etcd-0 pod to **ca.crt**, **tls.crt**, and **tls.key** in the current directory on this control plane node.

- c. Repeat for each contrail-etcd pod.
4. Back up the etcd database on one of the control plane nodes. You only need to back up the database on one node.
 - a. Log back in to one of the control plane nodes.
 - b. Back up the etcd database.

This example saves the database to **/tmp/etcdbackup.db** on this control plane node.

```
etcdctl snapshot save /tmp/etcdbackup.db --endpoints=<etcd-pod-ip>:<etcd-port>
--cacert=ca.crt --cert=tls.crt --key=tls.key
```

where **<etcd-pod-ip>** is the IP address of the pod on this node and the **<etcd-port>** is the port that etcd is listening on (by default, 12379).

5. Copy the database to a safe location.

Restore the Contrail Etcd Database in Release 22.4

Use this example procedure in release 22.4 to restore the Contrail etcd database from a snapshot.

1. Copy the snapshot you want to restore to all the control plane nodes.

The steps below assume you've copied the snapshot to **/tmp/etcdbackup.db** on all the control plane nodes.

2. Restore the snapshot.

- a. Log in to one of the control plane nodes. In this example, we're logging in to the control plane node that is hosting contrail-etcd-0.
- b. Restore the etcd database to the contrail-etcd-0 pod on this control plane node.

This creates a **contrail-etcd-0.etcd** directory on the node.

```
ETCDCTL_API=3 etcdctl snapshot restore /tmp/etcdBackup.db \
--name=contrail-etcd-0 \
--initial-cluster=contrail-etcd-0=https://<contrail-etcd-0-ip>:12380,\
contrail-etcd-1=https://<contrail-etcd-1-ip>:12380,\
contrail-etcd-2=https://<contrail-etcd-2-ip>:12380 \
--initial-advertise-peer-urls=https://<contrail-etcd-0-ip>:12380 \
--cacert=ca.crt --cert=tls.crt --key=tls.key
```

where `--name=contrail-etcd-0` specifies that this command is restoring the database to contrail-etcd-0, `--initial-cluster=...` lists all the contrail-etcd members in the cluster, and `--initial-advertise-peer-urls=...` refers to the IP address and port number that the contrail-etcd-0 pod is listening on.

- c. Repeat for the other contrail-etcd pods on their respective control plane nodes, substituting the `--name` and `--initial-advertise-peer-urls` values with the respective contrail-etcd pod name and IP address.

3. Stop the contrail-etcd pods.

This sets the replicas to 0, which effectively stops the pods.

```
user@ai-client:~# kubectl patch etcds.datastore.juniper.net contrail-etcd -n contrail-system
--type=merge -p '{"spec": {"common": {"replicas": 0}}}'
```

4. Replace contrail-etcd data with the data from the snapshot.

- a. SSH into one of the control plane nodes.
- b. Replace the data. Recall that the snapshot is stored in the **contrail-etcd-*<xxx>*.etcd** directory.

```
sudo rm -rf /var/lib/contrail-etcd/snapshots
sudo mv /var/lib/contrail-etcd/etcd/member /var/lib/contrail-etcd/etcd/member.bak
sudo mv contrail-etcd-<xxx>.etcd/member /var/lib/contrail-etcd/etcd/
```

where *contrail-etcd-xxx* is the name of the contrail-etcd pod on the control plane node that you logged in to.

c. Repeat for the other control plane nodes.

5. Start the contrail-etcd pods.

This sets the replicas to 3, which effectively starts the pods.

```
user@ai-client:~# kubectl patch etcds.datastore.juniper.net contrail-etcd -n contrail-system
--type=merge -p '{"spec": {"common": {"replicas": 3}}}'
```

6. Restart the contrail-system apiserver and controller.

Delete all the contrail-k8s-apiserver and contrail-k8s-controller pods.

```
kubectl delete pod <contrail-k8s-apiserver-xxx> -n contrail-system
```

```
kubectl delete pod <contrail-k8s-controller-xxx> -n contrail-system
```

These pods will automatically restart.

7. Restart the vrouters.

Delete all the contrail-vrouter-masters and contrail-vrouter-nodes pods.

```
kubectl delete pod <contrail-vrouter-masters-xxx> -n contrail
```

```
kubectl delete pod <contrail-vrouter-nodes-xxx> -n contrail
```

These pods will automatically restart.

8. Check that all pods are in running state.

```
user@ai-client:~# kubectl get pods -n contrail-system
```

```
user@ai-client:~# kubectl get pods -n contrail
```

Upgrade CN2

Use this procedure to upgrade CN2.

The Contrail controller consists of Deployments and StatefulSets, which are configured for rolling updates. During the upgrade, the pods in each Deployment and StatefulSet are upgraded one at a time. The remaining pods in that Deployment or StatefulSet remain operational. This enables Contrail controller upgrades to be hitless.

The Contrail data plane consists of a DaemonSet with a single vRouter pod. During the upgrade procedure, this single pod is taken down and upgraded. Because of this, Contrail data plane upgrades are not hitless. If desired, migrate traffic off of the node being upgraded prior to performing the upgrade.

You upgrade CN2 software by porting the contents of your existing manifests to the new manifests, and then applying the new manifests. All CN2 manifests must reference the same software version.

NOTE: Before you upgrade, check to make sure that each node has at least one allocatable pod available. The upgrade procedure temporarily allocates an additional pod, which means that your node cannot be running at maximum pod capacity when you perform the upgrade. You can check pod capacity on a node by using the `kubectl describe node` command.

1. Download the manifests for the new release.
2. Locate the (old) manifests that you used to create the existing CN2 installation. These could be the manifests you used in step 4.c in ["Before You Install" on page 18](#) or these could be manifests that you customized for your setup.
3. Port over any changes from the old manifests to the new manifests.

The new manifests can contain constructs that are specific to the new release. Identify all changes that you've made to the old manifests and copy them over to the new manifests. This includes repository credentials, interface names, network configuration, and other customizations.

NOTE: If you have a large number of nodes, use node selectors to group your upgrades to a more manageable number.

4. Upgrade CN2 by applying all the manifests one at a time in ascending order from **99-*.yaml** onwards. For convenience, you can use the following bash script. The script assumes you've placed the manifests in the **manifests** directory. If you use this script, make sure that:
 - you place all the manifests you want to use in this directory, including all manifests that you want to use from the subdirectories

- you don't place any other YAML files in this directory

The script loops through all the *.yaml files in the **manifests** directory and applies them to the cluster in the proper order.

```
#!/bin/bash

# First rename 99-*.yaml files to 099-*.yaml so they get executed first
MANIFESTS=(manifests/*.yaml)
for file in "${MANIFESTS[@]"; do
    num=$(echo $file | cut -d "/" -f 2 | cut -d "-" -f 1)
    if [ "$num" = "99" ]; then
        rfile=$(echo $file | sed 's/99/099/')
        printf "renaming file: $file to $rfile\n"
        eval "mv $file $rfile"
    fi
done
# Now apply the manifests in order
MANIFESTS=(manifests/*.yaml)
total=${#MANIFESTS[@]}
for file in "${MANIFESTS[@]"; do
    printf "applying file: $file\n"
    eval "kubectl apply -f $file"
done
echo
echo
echo "Total manifests applied: $total"
echo
```

The pods in each Deployment and Stateful set will upgrade one at a time. The vRouter DaemonSet will go down and come back up.

5. Use standard kubectl commands to check on the upgrade.

Check the status of the nodes.

```
kubectl get nodes
```

Check the status of the pods.

```
kubectl get pods -A -o wide
```

If some pods remain down, debug the installation as you normally do. Use the `kubectl describe` command to see why a pod is not coming up. A common error is a network or firewall issue preventing the node from reaching the Juniper Networks repository.

5

CHAPTER

Appendix

[Configure Repository Credentials | 74](#)

[Replace a Control Plane Node | 75](#)

[Add a Worker Node | 85](#)

[Delete a Worker Node | 91](#)

[Back Up the Etcd Database | 92](#)

Configure Repository Credentials

Use this procedure to configure your repository login credentials in your manifests.

1. Install docker if you don't already have docker installed.
2. Log in to the Juniper Networks repository where you pull the container images.

```
docker login enterprise-hub.juniper.net
```

Enter your login credentials when prompted.

Once you've logged in, your credentials are automatically stored in `~/.docker/config.json`. (If you installed docker using snap, then the credentials are stored in the `~/snap/docker` directory hierarchy.)

3. Encode your credentials in base64 and store the resulting string.

```
ENCODED_CREDS=$(base64 -w 0 config.json)
```

4. Replace the credentials placeholder in the manifests with the encoded string.

The manifests have a `<base64-encoded-credential>` credentials placeholder. Simply replace the placeholder with the encoded string in all manifests.

```
sed -i s/'<base64-encoded-credential>'/${ENCODED_CREDS}/ *.yaml
```

RELATED DOCUMENTATION

https://www.juniper.net/documentation/en_US/day-one-books/topics/concept/secrets.html

Replace a Control Plane Node

SUMMARY

Learn how to identify and replace an unhealthy node in an OpenShift cluster.

IN THIS SECTION

- [Remove an Unhealthy Control Plane Node | 75](#)
- [Add a Replacement Control Plane Node | 78](#)

Replacing a control plane node requires you to first identify and remove the unhealthy node. After you remove the unhealthy node, you can then add the new replacement node.

We provide these example procedures purely for informational purposes. See Red Hat OpenShift documentation (<https://docs.openshift.com/>) for the official procedure.

Remove an Unhealthy Control Plane Node

Use this procedure to identify and remove an unhealthy control plane node.

1. Check the status of the control plane nodes to identify the unhealthy member.

```
user@ai-client:~# oc get nodes -l node-role.kubernetes.io/master
```

NAME	STATUS	ROLES	AGE	VERSION
ocp1.mycluster.contrail.lan	Ready	master	16d	v1.21.6+bb8d50a
ocp2.mycluster.contrail.lan	Ready	master	16d	v1.21.6+bb8d50a
ocp3.mycluster.contrail.lan	NotReady	master	16d	v1.21.6+bb8d50a

In this example, ocp3 is the unhealthy node.

2. Back up the etcd database on one of the healthy nodes by following the procedure in "[Back Up the Etcd Database](#)" on page 92.
3. List the etcd members.

```
user@ai-client:~# oc get pods -n openshift-etcd -o wide | grep -v etcd-quorum-guard | grep etcd
```

NAME	READY	RESTARTS	AGE	IP
etcd-ocp1	4/4	Running	0	18d 172.16.0.11
ocp1	<none>	<none>		

etcd-ocp2			4/4	Running	0	18d	172.16.0.12
ocp2	<none>	<none>					
etcd-ocp3			4/4	Running	0	18d	172.16.0.13
ocp3	<none>	<none>					

4. Open a remote shell to an etcd pod on a healthy node (for example, ocp1).

```
user@ai-client:~# oc rsh -n openshift-etcd etcd-ocp1
Defaulted container "etcdctl" out of: etcdctl, etcd, etcd-metrics, etcd-health-monitor,
setup (init), etcd-ensure-env-vars (init), etcd-resources-copy (init)
```

5. List the etcd members.

```
sh-4.4# etcdctl member list -w table
+-----+-----+-----+-----+-----+
+-----+
|      ID      | STATUS | NAME |      PEER ADDRS      |      CLIENT ADDRS      |
| IS LEARNER |
+-----+-----+-----+-----+-----+
+-----+
| 19faf45778a1ddd3 | started | ocp3 | https://172.16.0.13:2380 | https://172.16.0.13:2379 |
|      false |
| ad4840148f3f241c | started | ocp1 | https://172.16.0.11:2380 | https://172.16.0.11:2379 |
|      false |
| b1f7a55fb3caa3b6 | started | ocp2 | https://172.16.0.12:2380 | https://172.16.0.12:2379 |
|      false |
+-----+-----+-----+-----+-----+
+-----+
```

6. Remove the etcd member on the unhealthy node.

```
sh-4.4# etcdctl member remove 19faf45778a1ddd3
Member 19faf45778a1ddd3 removed from cluster 60f3fdb1b921fd7
```

7. View the member list again.

```
sh-4.4# etcdctl member list -w table
+-----+-----+-----+-----+-----+
+-----+
|      ID      | STATUS | NAME |      PEER ADDRS      |      CLIENT ADDRS      |
| IS LEARNER |
+-----+-----+-----+-----+-----+
```

```

+-----+-----+-----+-----+-----+
+-----+
| ad4840148f3f241c | started | ocp1 | https://172.16.0.11:2380 | https://172.16.0.11:2379
|      false      |
| b1f7a55fb3caa3b6 | started | ocp2 | https://172.16.0.12:2380 | https://172.16.0.12:2379
|      false      |
+-----+-----+-----+-----+-----+
+-----+

```

8. Type **exit** to exit the remote shell.
9. Back on the AI client node, remove the old secrets for the unhealthy etcd member.
 - a. List the secrets for the unhealthy (removed) member.

```

user@ai-client:~# oc get secrets -n openshift-etcd | grep ocp3
etcd-peer-ocp3          kubernetes.io/tls          2      18d
etcd-serving-metrics-ocp3 kubernetes.io/tls          2      18d
etcd-serving-ocp3       kubernetes.io/tls          2      18d

```

- b. Delete the peer secrets.

```

user@ai-client:~# oc delete secret -n openshift-etcd etcd-peer-ocp3
secret "etcd-peer-ocp3" deleted

```

- c. Delete the metrics secrets.

```

user@ai-client:~# oc delete secret -n openshift-etcd etcd-serving-metrics-ocp3
secret "etcd-serving-metrics-ocp3" deleted

```

- d. Delete the serving secrets.

```

user@ai-client:~# oc delete secret -n openshift-etcd etcd-serving-ocp3
secret "etcd-serving-ocp3" deleted

```

10. Finally, delete the unhealthy node.

- a. Cordon the unhealthy node.

```
user@ai-client:~# oc adm cordon ocp3
node/ocp3 cordoned
```

- b. Drain the unhealthy node.

```
user@ai-client:~# oc adm drain ocp3 --ignore-daemonsets=true --delete-emptydir-data --
force
node/ocp3 already cordoned
<trimmed>
```

- c. Delete the unhealthy node.

```
user@ai-client:~# oc delete node ocp3
node "ocp3" deleted
```

- d. List the nodes.

```
user@ai-client:~# oc get nodes
```

NAME	STATUS	ROLES	AGE	VERSION
ocp1	Ready	master	19d	v1.21.6+bb8d50a
ocp2	Ready	master	19d	v1.21.6+bb8d50a
ocp4	Ready	worker	18d	v1.21.6+bb8d50a
ocp5	Ready	worker	18d	v1.21.6+bb8d50a

You have now identified and removed the unhealthy node.

Add a Replacement Control Plane Node

Use this procedure to add a replacement control plane node to an existing OpenShift cluster. An OpenShift cluster has exactly 3 control plane nodes. You cannot use this procedure to add a node to a cluster that already has 3 control plane nodes.

This procedure shows an example of late binding. In late binding, you generate an ISO and boot the node with that ISO. After the node boots, you bind the node to the existing cluster.

This causes one or more CertificateSigningRequests (CSRs) to be sent from the new node to the existing cluster. A CSR is simply a request to obtain the client certificates for the (existing) cluster. You'll need to explicitly approve these requests. Once approved, the existing cluster provides the client certificates to the new node, and the new node is allowed to join the existing cluster.

1. Log in to the machine (VM or BMS) that you're using as the Assisted Installer client. The Assisted Installer client machine is where you issue Assisted Installer API calls to the Assisted Installer server hosted by Red Hat.
2. Prepare the deployment by setting the environment variables that you'll use in later steps.

- a. Set up the same SSH key that you use for the existing cluster.

In this example, we retrieve that SSH key from its default location `~/.ssh/id_rsa.pub` and store into a variable.

```
export CLUSTER_SSHKEY=$(cat ~/.ssh/id_rsa.pub)
```

- b. If you no longer have the image pull secret, then download the image pull secret from your Red Hat account onto your local computer. The pull secret allows your installation to access services and registries that serve container images for OpenShift components.

If you're using the Red Hat hosted Assisted Installer, you can download the pull secret file (**pull-secret**) from the <https://console.redhat.com/openshift/downloads> page. Copy the pull-secret file to the Assisted Installer client machine. In this example, we store the pull-secret in a file called **pull-secret.txt**.

Strip out any whitespace, convert the contents to JSON string format, and store to an environment variable, as follows:

```
export PULL_SECRET=$(sed '/^[:space:]]*$/d' pull-secret.txt | jq -R .)
```

- c. If you no longer have your offline access token, then copy the offline access token from your Red Hat account. The OpenShift Cluster Manager API Token allows you (on the Assisted Installer client machine) to interact with the Assisted Installer API service hosted by Red Hat.

The token is a string that you can copy and paste to a local environment variable. If you're using the Red Hat hosted Assisted Installer, you can copy the API token from <https://console.redhat.com/openshift/downloads>.

```
export OFFLINE_ACCESS_TOKEN='<paste offline access token here>'
```

- d. Generate (refresh) the token from the OFFLINE_ACCESS_TOKEN. You will use this generated token whenever you issue API commands.

```
export TOKEN=$(curl --silent --data-urlencode "grant_type=refresh_token" --data-urlencode "client_id=cloud-services" --data-urlencode "refresh_token=${OFFLINE_ACCESS_TOKEN}" https://sso.redhat.com/auth/realms/redhat-external/protocol/openid-connect/token | jq -r .access_token)
```

NOTE: This token expires regularly. When this token expires, you will get an HTTP 4xx response whenever you issue an API command. Refresh the token when it expires, or alternatively, refresh the token regularly prior to expiry. There is no harm in refreshing the token when it hasn't expired.

- e. Get the OpenShift cluster ID of the existing cluster.

For example:

```
oc get clusterversion -o jsonpath='{.items[].spec.clusterID}{"\n"}'
1777102a-1fe1-407a-9441-9d0bad4f5968
```

Save it to a variable:

```
export $OS_CLUSTER_ID="1777102a-1fe1-407a-9441-9d0bad4f5968"
```

- f. Set up the remaining environment variables.

[Table 13 on page 80](#) lists all the environment variables that you need to set in this procedure, including the ones described in the previous steps.

Table 13: Environment Variables

Variable	Description	Example
CLUSTER_SSHKEY	The (public) SSH key you use for the existing cluster. You must use this same key for the new node you're adding.	–

Table 13: Environment Variables (*Continued*)

Variable	Description	Example
PULL_SECRET	The image pull secret that you downloaded, stripped and converted to JSON string format.	–
OFFLINE_ACCESS_TOKEN	The OpenShift Cluster Manager API Token that you copied.	–
TOKEN	The token that you generated (refreshed) from the OFFLINE_ACCESS_TOKEN.	–
CLUSTER_NAME	The name of the existing cluster.	mycluster
CLUSTER_DOMAIN	The base domain of the existing cluster.	contrail.lan
OS_CLUSTER_ID	The OpenShift cluster ID of the existing cluster.	1777102a-1fe1-407a-9441-9d0bad4f5968
AI_URL	The URL of the Assisted Installer service. This example uses the Red Hat hosted Assisted Installer.	https://api.openshift.com

3. Generate the discovery boot ISO. You will use this ISO to boot the node that you're adding to the cluster.

The ISO is customized to your infrastructure based on the infrastructure environment that you'll set up.

- a. Create a file that describes the infrastructure environment. In this example, we name it **infra-envs-addhost.json**.

```
cat << EOF > ./infra-envs-addhost.json
{
  "name": "<InfraEnv Name>",
  "ssh_authorized_key": "$CLUSTER_SSHKEY",
  "pull_secret": $PULL_SECRET,
  "openshift_version": "4.8",
```

```

    "user_managed_networking": <same as for existing cluster>,
    "vip_dhcp_allocation": <same as for existing cluster>,
    "base_dns_domain": "$CLUSTER_DOMAIN",
  }
EOF

```

where:

- *InfraEnv Name* is the name you want call the InfraEnv.
 - `user_managed_networking` and `vip_dhcp_allocation` are set to the same values as for the existing cluster.
- b. Register the InfraEnv. In response, the Assisted Installer service assigns an InfraEnv ID and builds the discovery boot ISO based on the specified infrastructure environment.

```

curl -X POST "$AI_URL/api/assisted-install/v2/infra-envs" -H "accept: application/json" -
H "Content-Type: application/json" -H "Authorization: Bearer $TOKEN" -d @infra-envs-
addhosts.json

```

When you register the InfraEnv, the Assisted Installer service returns an InfraEnv ID. Look carefully for the InfraEnv ID embedded in the response. For example:

```

"id": "5c858ed9-26cf-446d-817c-4c4261541657"

```

Store the InfraEnv ID into a variable. For example:

```

export INFRA_ENV_ID="5c858ed9-26cf-446d-817c-4c4261541657"

```

- c. Get the image download URL.

```

curl -s $AI_URL/api/assisted-install/v2/infra-envs/$INFRA_ENV_ID/downloads/image-url -H
"accept: application/json" -H "Content-Type: application/json" -H "Authorization: Bearer
$TOKEN" | jq '.url'

```

The Assisted Installer service returns the image URL.

- d. Download the ISO and save it to a file. In this example, we save it to **ai-liveiso-addhosts.iso**.

```

curl -L "<image URL>" -H "Authorization: Bearer $TOKEN" -o ./ai-liveiso-addhosts.iso

```

4. Boot the new node with the discovery boot ISO. Choose the boot method most convenient for your infrastructure. Ensure that the new node boots up attached to a network that has access to the Red Hat hosted Assisted Installer.

Check the status of the host:

```
curl -s -X GET --header "Content-Type: application/json" -H "Authorization: Bearer $TOKEN"
$AI_URL/api/assisted-install/v2/infra-envs/$INFRA_ENV_ID/hosts | jq '.'
```

Store the host ID into a variable.

```
export HOST_ID=$(curl -s -X GET --header "Content-Type: application/json" -H
"Authorization: Bearer $TOKEN" $AI_URL/api/assisted-install/v2/infra-envs/$INFRA_ENV_ID/
hosts | jq -r '.[].id')
```

5. Configure the new node as a control plane node.

```
curl -X PATCH --header "Content-Type: application/json" "$AI_URL/api/assisted-install/v2/
infra-envs/$INFRA_ENV_ID/hosts/$HOST_ID" -H "Authorization: Bearer $TOKEN" -d
"{ \"machine_config_pool_name\": \"master\"}" | jq -r
```

Check to see that the following is embedded in the response:

```
"machine_config_pool_name": "master",
```

6. Import the existing cluster.

```
curl -X POST "$AI_URL/api/assisted-install/v2/clusters/import" -H "accept: application/
json" -H "Content-Type: application/json" -H "Authorization: Bearer $TOKEN" -d "{ \"name
\": \"$CLUSTER_NAME\", \"openshift_cluster_id\": \"$OS_CLUSTER_ID\", \"api_vip_dnsname
\": \"api.$CLUSTER_NAME.$CLUSTER_DOMAIN\"}"
```

When you import the cluster, the Assisted Installer service returns a cluster ID for the AddHostsCluster. Look carefully for the cluster ID embedded in the response. For example:

```
"id": "c5bbb159-78bc-41c9-99b7-d8a4727a3890"
```

7. Bind the new host to the cluster, referencing the cluster ID of the AddHostsCluster.

```
curl -X POST "$AI_URL/api/assisted-install/v2/infra-envs/$INFRA_ENV_ID/hosts/$HOST_ID/
actions/bind" -H "accept: application/json" -H "Content-Type: application/json" -H
"Authorization: Bearer $TOKEN" -d '{"cluster_id":"c5bbb159-78bc-41c9-99b7-d8a4727a3890"}'
```

Check the status of the host regularly:

```
curl -s -X GET --header "Content-Type: application/json" -H "Authorization: Bearer $TOKEN"
$AI_URL/api/assisted-install/v2/infra-envs/$INFRA_ENV_ID/hosts | jq '.'
```

Proceed to the next step when you see the following output:

```
"status": "known",
"status_info": "Host is ready to be installed",
```

8. Install the new node.

```
curl -X POST -H "accept: application/json" -H "Content-Type: application/json" -H
"Authorization: Bearer $TOKEN" "$AI_URL/api/assisted-install/v2/infra-envs/$INFRA_ENV_ID/
hosts/$HOST_ID/actions/install"
```

Check on the status of the node:

```
curl -s -X GET --header "Content-Type: application/json" -H "Authorization: Bearer $TOKEN"
$AI_URL/api/assisted-install/v2/infra-envs/$INFRA_ENV_ID/hosts | jq '.'
```

Look for the following status, which indicates that the node has rebooted:

```
"status_info": "Host has rebooted and no further updates will be posted. Please check
console for progress and to possibly approve pending CSRs",
```

9. Once the new node has rebooted, it will try to join the existing cluster. This causes one or more CertificateSigningRequests (CSRs) to be sent from the new node to the existing cluster. You will need to approve the CSRs.
 - a. Check for the CSRs.

For example:

```
root@ai-client:~/contrail# oc get csr -A
```

NAME	AGE	SIGNERNAME	CONDITION
REQUESTOR			
csr-gblnm	20s	kubernetes.io/kube-apiserver-client-kubelet	
system:serviceaccount:openshift-machine-config-operator:node-bootstrapper			Pending

You may need to repeat this command periodically until you see pending CSRs.

- b. Approve the CSRs.

For example:

```
root@ai-client:~/contrail# oc adm certificate approve csr-gblnm
certificatesigningrequest.certificates.k8s.io/csr-gblnm approved
```

10. Verify that the new node is up and running in the existing cluster.

```
oc get nodes
```

Add a Worker Node

Use this procedure to add a worker node to an OpenShift cluster.

We provide this example procedure purely for informational purposes. See Red Hat OpenShift documentation (<https://docs.openshift.com/>) for the official procedure.

This procedure shows an example of early binding. In early binding, you generate an ISO that is preconfigured for the existing cluster. When the node boots with that ISO, the node automatically reaches out to the existing cluster.

This causes one or more CertificateSigningRequests (CSRs) to be sent from the new node to the existing cluster. A CSR is simply a request to obtain the client certificates for the (existing) cluster. You'll need to explicitly approve these requests. Once approved, the existing cluster provides the client certificates to the new node, and the new node is allowed to join the existing cluster.

1. Log in to the machine (VM or BMS) that you're using as the Assisted Installer client. The Assisted Installer client machine is where you issue Assisted Installer API calls to the Assisted Installer service hosted by Red Hat.
2. Prepare the deployment by setting the environment variables that you'll use in later steps.

- a. Set up the same SSH key that you use for the existing cluster.

In this example, we retrieve that SSH key from its default location `~/.ssh/id_rsa.pub` and store into a variable.

```
export CLUSTER_SSHKEY=$(cat ~/.ssh/id_rsa.pub)
```

- b. If you no longer have the image pull secret, then download the image pull secret from your Red Hat account onto your local computer. The pull secret allows your installation to access services and registries that serve container images for OpenShift components.

If you're using the Red Hat hosted Assisted Installer, you can download the pull secret file (**pull-secret**) from the <https://console.redhat.com/openshift/downloads> page. Copy the pull-secret file to the Assisted Installer client machine. In this example, we store the pull-secret in a file called **pull-secret.txt**.

Strip out any whitespace, convert the contents to JSON string format, and store to an environment variable, as follows:

```
export PULL_SECRET=$(sed '/^[:space:]]*$/d' pull-secret.txt | jq -R .)
```

- c. If you no longer have your offline access token, then copy the offline access token from your Red Hat account. The OpenShift Cluster Manager API Token allows you (on the Assisted Installer client machine) to interact with the Assisted Installer API service hosted by Red Hat.

The token is a string that you can copy and paste to a local environment variable. If you're using the Red Hat hosted Assisted Installer, you can copy the API token from <https://console.redhat.com/openshift/downloads>.

```
export OFFLINE_ACCESS_TOKEN='<paste offline access token here>'
```

- d. Generate (refresh) the token from the OFFLINE_ACCESS_TOKEN. You will use this generated token whenever you issue API commands.

```
export TOKEN=$(curl --silent --data-urlencode "grant_type=refresh_token" --data-urlencode "client_id=cloud-services" --data-urlencode "refresh_token=${OFFLINE_ACCESS_TOKEN}")
```

```
https://sso.redhat.com/auth/realms/redhat-external/protocol/openid-connect/token | jq -r .access_token)
```

NOTE: This token expires regularly. When this token expires, you will get an HTTP 4xx response whenever you issue an API command. Refresh the token when it expires, or alternatively, refresh the token regularly prior to expiry. There is no harm in refreshing the token when it hasn't expired.

- e. Get the OpenShift cluster ID of the existing cluster.

For example:

```
oc get clusterversion -o jsonpath='{.items[].spec.clusterID}{"\n"}'
1777102a-1fe1-407a-9441-9d0bad4f5968
```

Save it to a variable:

```
export $OS_CLUSTER_ID="1777102a-1fe1-407a-9441-9d0bad4f5968"
```

- f. Set up the remaining environment variables.

[Table 14 on page 87](#) lists all the environment variables that you need to set in this procedure, including the ones described in the previous steps.

Table 14: Environment Variables

Variable	Description	Example
CLUSTER_SSHKEY	The (public) SSH key you use for the existing cluster. You must use this same key for the new node you're adding.	–
PULL_SECRET	The image pull secret that you downloaded, stripped and converted to JSON string format.	–
OFFLINE_ACCESS_TOKEN	The OpenShift Cluster Manager API Token that you copied.	–

Table 14: Environment Variables (*Continued*)

Variable	Description	Example
TOKEN	The token that you generated (refreshed) from the OFFLINE_ACCESS_TOKEN.	–
CLUSTER_NAME	The name of the existing cluster.	mycluster
CLUSTER_DOMAIN	The base domain of the existing cluster.	contrail.lan
OS_CLUSTER_ID	The OpenShift cluster ID of the existing cluster.	1777102a-1fe1-407a-9441-9d0bad4f5968
AI_URL	The URL of the Assisted Installer service. This example uses the Red Hat hosted Assisted Installer.	https://api.openshift.com

3. Import the existing cluster.

```
curl -X POST "$AI_URL/api/assisted-install/v2/clusters/import" -H "accept: application/json"
-H "Content-Type: application/json" -H "Authorization: Bearer $TOKEN" -d "{\"name
\": \"$CLUSTER_NAME\", \"openshift_cluster_id\": \"$OS_CLUSTER_ID\", \"api_vip_dnsname
\": \"api.$CLUSTER_NAME.$CLUSTER_DOMAIN\"}"
```

When you import the cluster, the Assisted Installer service returns a cluster ID for the AddHostsCluster. Look carefully for the cluster ID embedded in the response. For example:

```
"id": "19b809b5-69c4-42d8-9e5e-56aae4aba386"
```

4. Generate the discovery boot ISO. You will use this ISO to boot up the node that you're adding to the cluster.

The ISO is customized to your infrastructure based on the infrastructure environment that you'll set up.

- a. Create a file that describes the infrastructure environment. In this example, we name it **infra-envs-addhost.json**.

```
cat << EOF > ./infra-envs-addhost.json
{
  "name": "<InfraEnv Name>",
  "ssh_authorized_key": "$CLUSTER_SSHKEY",
  "pull_secret": $PULL_SECRET,
  "cluster_id": "<AddHostsCluster ID>",
  "openshift_version": "4.8",
  "user_managed_networking": <same as for existing cluster>,
  "vip_dhcp_allocation": <same as for existing cluster>,
  "base_dns_domain": "$CLUSTER_DOMAIN",
}
EOF
```

where:

- *InfraEnv Name* is the name you want call the InfraEnv.
 - *AddHostsCluster ID* is the cluster ID of the AddHostsCluster (obtained in the previous step).
 - *user_managed_networking* and *vip_dhcp_allocation* are set to the same values as for the existing cluster.
- b. Register the InfraEnv. In response, the Assisted Installer service assigns an InfraEnv ID and builds the discovery boot ISO based on the specified infrastructure environment.

```
curl -X POST "$AI_URL/api/assisted-install/v2/infra-envs" -H "accept: application/json" -H
"Content-Type: application/json" -H "Authorization: Bearer $TOKEN" -d @infra-envs-
addhosts.json
```

When you register the InfraEnv, the Assisted Installer service returns an InfraEnv ID. Look carefully for the InfraEnv ID embedded in the response. For example:

```
"id": "78d20699-f25b-462c-bc1d-4738590a9344"
```

Store the InfraEnv ID into a variable. For example:

```
export INFRA_ENV_ID="78d20699-f25b-462c-bc1d-4738590a9344"
```

- c. Get the image download URL.

```
curl -s $AI_URL/api/assisted-install/v2/infra-envs/$INFRA_ENV_ID/downloads/image-url -H
"accept: application/json" -H "Content-Type: application/json" -H "Authorization: Bearer
$TOKEN" | jq '.url'
```

The Assisted Installer service returns the image URL.

- d. Download the ISO and save it to a file. In this example, we save it to **ai-liveiso-addhosts.iso**.

```
curl -L "<image URL>" -H "Authorization: Bearer $TOKEN" -o ./ai-liveiso-addhosts.iso
```

5. Boot the new worker node with the discovery boot ISO. Choose the boot method most convenient for your infrastructure. Ensure that the new node boots up attached to a network that has access to the Red Hat hosted Assisted Installer.
6. Install the new node to the existing cluster.
 - a. Inspect the new node to make sure its role is set to worker.

```
curl -s -X GET --header "Content-Type: application/json" -H "Authorization: Bearer $TOKEN"
$AI_URL/api/assisted-install/v2/infra-envs/$INFRA_ENV_ID/hosts | jq '.'
```

Proceed only when the host role is set to worker.

- b. Get the host ID of the new node.

```
export HOST_ID=$(curl -s -X GET --header "Content-Type: application/json" -H
"Authorization: Bearer $TOKEN" $AI_URL/api/assisted-install/v2/infra-envs/$INFRA_ENV_ID/
hosts | jq -r '.[].id')
```

- c. Install the new node.

```
curl -X POST "$AI_URL/api/assisted-install/v2/infra-envs/$INFRA_ENV_ID/hosts/$HOST_ID/
actions/install" -H "accept: application/json" -H "Authorization: Bearer $TOKEN" | jq -r
```

- d. Check on the progress of the installation.

```
curl -s -X GET --header "Content-Type: application/json" -H "Authorization: Bearer $TOKEN"
$AI_URL/api/assisted-install/v2/infra-envs/$INFRA_ENV_ID/hosts | jq '.'
```

The new node will eventually reboot.

7. Once the new node has rebooted, it will try to join the existing cluster. This causes one or more CertificateSigningRequests (CSRs) to be sent from the new node to the existing cluster. You will need to approve the CSR requests.

- a. Check for pending CSRs.

For example:

```
root@ai-client:~/contrail# oc get csr -A
NAME          AGE    SIGNERNAME
REQUESTOR
csr-gblnm     20s    kubernetes.io/kube-apiserver-client-kubelet
system:serviceaccount:openshift-machine-config-operator:node-bootstrapper Pending
```

You may need to repeat this command periodically until you see a pending CSR.

- b. Approve the CSRs.

For example:

```
root@ai-client:~/contrail# oc adm certificate approve csr-gblnm
certificatesigningrequest.certificates.k8s.io/csr-gblnm approved
```

8. Verify that the new node is up and running in the existing cluster.

```
oc get nodes
```

Delete a Worker Node

Use this procedure to delete a worker node.

We provide this example procedure purely for informational purposes. See Red Hat OpenShift documentation (<https://docs.openshift.com/>) for the official procedure.

1. Cordon the node you want to delete. In this example, we're deleting ocp6.

```
root@ai-client:~/contrail# oc get nodes
<trimmed>
ocp6.mycluster.contrail.lan    Ready    worker    74m    v1.21.8+ed4d8fd
```

```
oc adm cordon ocp6.mycluster.contrail.lan
```

2. Drain the node.

```
oc adm drain ocp6.mycluster.contrail.lan --force --delete-emptydir-data --ignore-daemonsets
```

3. Ensure there are no current jobs/cronjobs scheduled for this node.
4. Power off the node.
5. Delete the node.

```
oc delete node ocp6.mycluster.contrail.lan
```

6. Verify the node is no longer present.

```
oc get nodes
```

Back Up the Etcd Database

Use this example procedure to back up the etcd database.

We provide this example procedure purely for informational purposes. See Red Hat OpenShift documentation (<https://docs.openshift.com/>) for the official procedure.

The use of CN2 as a CNI plug-in does not affect how you back up and restore the etcd database. Use the tools that you're most familiar with to manage the database, such as etcdctl.

1. Get a list of the running nodes.

```
user@ai-client:~# oc get nodes
```

NAME	STATUS	ROLES	AGE	VERSION
ocp1	Ready	master	18d	v1.21.6+bb8d50a
ocp2	Ready	master	18d	v1.21.6+bb8d50a
ocp3	Ready	master	18d	v1.21.6+bb8d50a
ocp4	Ready	worker	18d	v1.21.6+bb8d50a
ocp5	Ready	worker	18d	v1.21.6+bb8d50a

2. Log in to one of the control plane nodes as root.

You cannot simply do this through SSH because root login is disabled by default. You have to launch a debug pod and chroot into the host filesystem.

- Launch a debug pod on one of the control plane nodes. When you do this, you're automatically placed into a root shell of the debug pod. This example launches a debug pod on ocp1.

```
user@ai-client:~# oc debug node/ocp1
Starting pod/ocp1-debug ...
To use host binaries, run `chroot /host`

Pod IP: 172.16.0.11
If you don't see a command prompt, try pressing enter.
sh-4.4#
sh-4.4# whoami
root
```

The debug pod mounts the host (node) filesystem at **/host**, as you can see here:

```
sh-4.4# df | grep "/host$"
/dev/vda4      167236588 17160016 150076572  11% /host
```

- To change to the host filesystem as root, use the chroot command.

By doing this, you are effectively logged in to the host node as root.

```
sh-4.4# chroot /host
```

You can verify that you're in the host filesystem by searching for the device name that was mounted as **/host** previously.

```
sh-4.4# df | grep /dev/vda4
/dev/vda4      167236588 17250336 149986252  11% /
```

3. Back up the etcd database.

The following backs up the database to the **/home/core/assets/backup** directory. This directory is created as part of the `cluster-backup.sh` script. This script is provided as part of the etcd Cluster Operator and is a wrapper around the **etcdctl snapshot save** command. You don't have to install etcdctl. The script installs etcdctl automatically.

```
sh-4.4# /usr/local/bin/cluster-backup.sh /home/core/assets/backup
found latest kube-apiserver: /etc/kubernetes/static-pod-resources/kube-apiserver-pod-14
found latest kube-controller-manager: /etc/kubernetes/static-pod-resources/kube-controller-
manager-pod-6
found latest kube-scheduler: /etc/kubernetes/static-pod-resources/kube-scheduler-pod-6
found latest etcd: /etc/kubernetes/static-pod-resources/etcd-pod-4
<trimmed>
snapshot db and kube resources are successfully saved to /home/core/assets/backup
```

NOTE: It is normal to see CNI errors in the above output.

Two files are created by the script:

- **snapshot_<timestamp>.db** - this is the etcd snapshot
- **static_kuberresources_<timestamp>.tar.gz** - this contains the resources for the static pods

4. Type **exit** to exit the shell and terminate the debug pod.