JUNIPER NETWORKS | Engineering Simplicity

Contrail® Networking

Contrail Networking for Container Networking Environments User Guide

Published
2023-07-13

RELEASE
21.4

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

Contrail® Networking Contrail Networking for Container Networking Environments User Guide
21.4

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

About this guide

This guide covers Contrail Networking in container networking environments that are using Contrail Networking Release 21-based releases.

Starting in Release 22.1, Contrail Networking evolved into Cloud-Native Contrail Networking. Cloud-Native Contrail Networking offers significant enhancements to optimize networking performance in container networking environments. Container networking environments are cloud environments that use Kubernetes for orchestration and Contrail Networking for networking. See the Cloud-Native Contrail Networking Techlibrary homepage.

Use this guide to install and perform foundational tasks when using Contrail Networking in container networking environments.

This guide covers the following scenarios:

- Contrail Networking with Kubernetes Overview
- Contrail Networking with Red Hat Openshift
- Contrail Networking with the Elastic Kubernetes Service (EKS) in Amazon Web Services (AWS)
- Contrail Networking with Google Anthos
- Using KubeVirt
- Using Contrail Networking with Kubernetes

Contrail Networking product documentation is organized into multiple guides as shown in Table 1, according to the task you want to perform or the deployment scenario.

Table 1: Contrail Networking Guides

Guide Name	Description
Contrail Networking Installation and Upgrade Guide	Provides step-by-step instructions to install and bring up Contrail and its various components.
Contrail Networking Fabric Lifecycle Management Guide	Provides information about Contrail underlay management and data center automation.
Contrail Networking and Security User Guide	Provides information about creating and orchestrating highly secure virtual networks.

Table 1: Contrail Networking Guides *(Continued)*

Guide Name	Description
Contrail Networking Service Provider Focused Features Guide	Provides information about the features that are used by service providers.
Contrail Networking Monitoring and Troubleshooting Guide	Provides information about Contrail Insights and Contrail analytics.

CHAPTER 1

Overview: Contrail Networking with Kubernetes

Contrail Integration with Kubernetes

IN THIS SECTION

NOTE: This topic covers Contrail Networking in Kubernetes-orchestrated environments that are using Contrail Networking Release 21-based releases.

Starting in Release 22.1, Contrail Networking evolved into Cloud-Native Contrail Networking. Cloud-Native Contrail Networking offers significant enhancements to optimize networking performance in Kubernetes-orchestrated environments. We recommend using Cloud-Native Contrail for networking in most Kubernetes-orchestrated environments.

For general information about Cloud-Native Contrail, see the Cloud-Native Contrail Networking Techlibrary homepage.

Contrail Networking supports the Container Network Interface (CNI) for integrating Contrail with the Kubernetes automation platform.

What is Kubernetes?

Kubernetes, also called K8s, is an open source platform for automating deployment, scaling, and operations of application containers across clusters of hosts, providing container-centric infrastructure. It provides a portable platform across public and private clouds. Kubernetes supports deployment, scaling, and auto-healing of applications.

Kubernetes supports a pluggable framework called Container Network Interface (CNI) for most of the basic network connectivity, including container pod addressing, network isolation, policy-based security,

a gateway, SNAT, load-balancer, and service chaining capability for Kubernetes orchestration. Contrail Networking is supported as a CNI in Kubernetes environments starting in Contrail Release 4.0.

Kubernetes provides a flat networking model in which all container pods can talk to each other. Network policy is added to provide security between the pods. Contrail integrated with Kubernetes adds additional networking functionality, including multi-tenancy, network isolation, micro-segmentation with network policies, load-balancing, and more.

Table 2 on page 3 lists the mapping between Kubernetes concepts and Tungsten Fabric resources.

Table 2: Kubernetes to Tungsten Fabric Mapping

Kubernetes	Tungsten Fabric Resources
Namespace	Shared or single project
Pod	Virtual-machine, Interface, Instance-ip
Service	ECMP-based native Loadbalancer
Ingress	HAProxy-based L7 Loadbalancer for URL routing
Network policy	Security group based on namespace and pod selectors

What is a Kubernetes Pod?

A Kubernetes pod is a group of one or more containers (such as Docker containers), the shared storage for those containers, and options on how to run the containers. Pods are always co-located and co-scheduled, and run in a shared context. The shared context of a pod is a set of Linux namespaces, cgroups, and other facets of isolation. Within the context of a pod, individual applications might have further sub-isolations applied.

You can find more information about Kubernetes at: http://kubernetes.io/docs/whatisk8s/.

Configuration Modes for Contrail Integrated with Kubernetes

Contrail can be configured in several different modes in Kubernetes. This section describes the various configuration modes.

Default Mode

In Kubernetes, all pods can communicate with all other pods without using network address translation (NAT). This is the default mode of Contrail Kubernetes cluster. In the default mode, Contrail creates a virtual-network that is shared by all namespaces, from which service and pod IP addresses are allocated.

All pods in all namespaces that are spawned in the Kubernetes cluster are able to communicate with one another. The IP addresses for all of the pods are allocated from a pod subnet that is configured in the Contrail Kubernetes manager.

> **NOTE**: System pods that are spawned in the kube-system namespace are not run in the Kubernetes cluster; they run in the underlay, and networking for these pods is not handled by Contrail.

Namespace Isolation Mode

In addition to the default networking model mandated by Kubernetes, Contrail supports additional custom networking models that make available the many rich features of Contrail to the users of the Kubernetes cluster. One such feature is network isolation for Kubernetes namespaces.

For namespace isolation mode, the cluster administrator can configure a namespace annotation to turn on isolation. As a result, services in that namespace are not accessible from other namespaces, unless security groups or network policies are explicitly defined to allow access.

A Kubernetes namespace can be configured as isolated by annotating the Kubernetes namespace metadata:

```
opencontrail.org/isolation : true
```

Namespace isolation provides network isolation to pods, because the pods in isolated namespaces are not reachable to pods in other namespaces in the cluster.

Namespace isolation also provides service isolation to pods. If any Kubernetes service is implemented by pods in an isolated namespace, those pods are reachable only to pods in the same namespace through the Kubernetes service-ip.

To make services remain reachable to other namespaces, service isolation can be disabled by the following additional annotation on the namespace:

```
opencontrail.org/isolation.service : false
```

Disabling service isolation makes the services reachable to pods in other namespaces, however pods in isolated namespaces still remain unreachable to pods in other namespaces.

A namespace annotated as “isolated” for both pod and service isolation has the following network behavior:

- All pods created in an isolated namespace have network reachability with each other.
- Pods in other namespaces in the Kubernetes cluster *cannot* reach pods in the isolated namespace.
- Pods created in isolated namespaces *cannot* reach pods in non-isolated namespaces.
- Pods in isolated namespaces *can* reach non-isolated services in any namespace in the Kubernetes cluster.
- Pods from other namespaces *cannot* reach services in isolated namespaces.

A namespace annotated as “isolated”, with service-isolation disabled and only pod isolation enabled, has the following network behavior:

- All pods created in an isolated namespace have network reachability with each other.
- Pods in other namespaces in the Kubernetes cluster *cannot* reach pods in the isolated namespace.
- Pods created in isolated namespaces *cannot* reach pods in non-isolated namespaces.
- Pods in isolated namespaces *can* reach non-isolated services in any namespace in the Kubernetes cluster.
- Pods from other namespaces *can* reach services in isolated namespaces.

Custom Isolation Mode

Administrators and application developers can add annotations to specify the virtual network in which a pod or all pods in a namespace are to be provisioned. The annotation to specify this custom virtual network is:

```
"opencontrail.org/network: <fq_network_name>"
```

where *fq-network-name* is the name of the virtual network.

Example:

```
annotations: {
    "opencontrail.org/network" : '{"domain":"default-domain", "project": "k8s-default",
"name":"k8s-blue-net-pod-network"}'
  }
```

If this annotation is configured on a pod spec then the pod is launched in that network. If the annotation is configured in the namespace spec then all the pods in the namespace are launched in the provided network.

> **NOTE**: The virtual network must be created using Contrail VNC APIs or Contrail-UI prior to configuring it in the pod or namespace spec.

For additional information on custom isolation, see Isolation (Namespace and Custom) in Github. A Github account may be required.

Nested Mode

Contrail supports the provisioning of Kubernetes cluster inside an OpenStack cluster. While this nesting of clusters by itself is not unique, Contrail provides a *collapsed* control and data plane in which a single Contrail control plane and a single network stack manage and service both the OpenStack and Kubernetes clusters. With unified control and data planes, interworking and configuring these clusters is seamless, and the lack of replication and duplicity makes this a very efficient option.

In nested mode, a Kubernetes cluster is provisioned in the virtual machine of an OpenStack cluster. The CNI-plugin and the Contrail-kubernetes manager of the Kubernetes cluster interface directly with Contrail components that manage the OpenStack cluster.

In a nested-mode deployment, all Kubernetes features, functions, and specifications are supported as is. Nested deployment stretches the boundaries and limits of Kubernetes by allowing it to operate on the same plane as underlying OpenStack cluster.

For more information, see "Provisioning of Kubernetes Clusters" on page 206.

Kubernetes Services

A Kubernetes service is an abstraction that defines a logical set of pods and the policy used to access the pods. The set of pods implementing a service are selected based on the **LabelSelector** field in the service definition. In Contrail, a Kubernetes service is implemented as an ECMP-native load-balancer.

The Contrail Kubernetes integration supports the following **ServiceType**s:

- **`clusterIP`**: This is the default mode. Choosing this **ServiceType** makes the service reachable through the cluster network.

- **`LoadBalancer`**: Designating a **ServiceType** as **`LoadBalancer`** enables the service to be accessed externally. The **`LoadBalancer`** **_Service_** is assigned both CluserIP and ExternalIP addresses. This **ServiceType** assumes that the user has configured the public network with a floating-ip pool.

Contrail Kubernetes Service-integration supports TCP and UDP for protocols. Also, Service can expose more than one port where port and targetPort are different. For example:

```
kind: Service
apiVersion: v1
metadata:
  name: my-service
spec:
    selector:
      app: MyApp
    ports:
      - name: http
        protocol: TCP
        port: 80
        targetPort: 9376
      - name: https
        protocol: TCP
        port: 443
        targetPort: 9377
```

Kubernetes users can specify spec.clusterIP and spec.externalIPs for both **LoadBalancer** and **clusterIP ServiceType**s.

If **ServiceType** is **LoadBalancer** and no spec.externalIP is specified by the user, then contrail-kube-manager allocates a floating-ip from the public pool and associates it to the ExternalIP address.

Ingress

Kubernetes services can be exposed externally or exposed outside of the cluster in many ways. See https://kubernetes.io/docs/concepts/services-networking/ingress/#alternatives for a list of all methods of exposing Kubernetes services externally. Ingress is one such method. Ingress provides Layer 7 load-balancing whereas the other methods provide Layer 4 load-balancing. Contrail supports http-based single-service ingress, simple-fanout ingress, and name-based virtual hosting ingress.

Contrail Kubernetes Solution

Contrail Kubernetes solution includes the following elements.

Contrail Kubernetes Manager

The Contrail Kubernetes implementation requires listening to the Kubernetes API messages and creating corresponding resources in the Contrail API database.

A new module, contrail-kube-manager, runs in a Docker container to listen to the messages from the Kubernetes API server.

ECMP Load-Balancers for Kubernetes Services

Each service in Kubernetes is represented by a load-balancer object. The service IP allocated by Kubernetes is used as the VIP for the load-balancer. Listeners are created for the port on which the service is listening. Each pod is added as a member of the listener pool. The contrail-kube-manager listens for any changes based on service labels or pod labels, and updates the member pool list with any added, updated, or deleted pods.

Load-balancing for services is Layer 4 native, non-proxy load-balancing based on ECMP. The instance-ip (service-ip) is linked to the ports of each of the pods in the service. This creates an ECMP next-hop in Contrail and traffic is load-balanced directly from the source pod.

HAProxy Loadbalancer for Kubernetes Ingress

Kubernetes Ingress is implemented through the HAProxy load-balancer feature in Contrail. Whenever ingress is configured in Kubernetes, contrail-kube-manager creates the load-balancer object in contrail-controller. The Contrail service monitor listens for the load-balancer objects and launches the HAProxy with appropriate configuration, based on the ingress specification rules in active-standby mode.

See *Using Load Balancers in Contrail* for more information on load balancers.

Security Groups for Kubernetes Network Policy

Kubernetes network policy is a specification of how groups of pods are allowed to communicate with each other and other network endpoints. **NetworkPolicy** resources use labels to select pods and define allow list rules which allow traffic to the selected pods in addition to what is allowed by the isolation policy for a given namespace.

For more information about Kubernetes network policies, see https://kubernetes.io/docs/concepts/services-networking/networkpolicies/.

The contrail-kube-manager listens to the Kubernetes network policy events for create, update, and delete, and translates the Kubernetes network policy to Contrail security group objects applied to virtual machine interfaces (VMIs). The VMIs are dynamically updated as pods and labels are added and deleted.

Kubernetes Support for Security Policy

Network policies created in a Kubernetes environment are implemented by using Contrail Security Policy framework. Labels from the Kubernetes environment are exposed as tags in Contrail. Starting in Contrail Release 5.0, you can define tags for a Kubernetes environment. Contrail security policy uses these tags to implement specified Kubernetes policies. You can define tags in the UI or upload configurations in JSON format. The newly-defined tags can be used to create and enforce policies in Contrail Security.

Domain Name Server (DNS)

Kubernetes implements DNS using SkyDNS, a small DNS application that responds to DNS requests for service name resolution from pods. SkyDNS runs as a pod in Kubernetes.

Supported Kubernetes Annotations

Currently, Contrail Networking supports the following Kubernetes annotations:

```
'opencontrail.org/network': '{"domain":"default-domain", "project": "k8s-contrail",
"name":"deu"}'
'opencontrail.org/isolation': 'true'
'opencontrail.org/fip-pool': '{"domain": "default-domain", "project": "k8s-default", "network":
"k8s-default-svc-public", "name": "default"}'
```

For further details, refer to https://kubernetes.io/docs/concepts/overview/working-with-objects/annotations/.

RELATED DOCUMENTATION

Verifying Configuration for CNI for Kubernetes | 224

2
CHAPTER

Contrail Networking with Red Hat Openshift

How to Install Contrail Networking and Red Hat OpenShift 4.6

IN THIS SECTION

NOTE: This topic covers Contrail Networking in Red Hat Openshift environments that are using Contrail Networking Release 21-based releases.

Starting in Release 22.1, Contrail Networking evolved into Cloud-Native Contrail Networking. Cloud-Native Contrail offers significant enhancements to optimize networking performance in Kubernetes-orchestrated environments. Cloud-Native Contrail supports Red Hat Openshift and we strongly recommend using Cloud-Native Contrail for networking in environments using Red Hat Openshift.

For general information about Cloud-Native Contrail, see the Cloud-Native Contrail Networking Techlibrary homepage.

Starting in Contrail Networking Release 2011.L1, you can install Contrail Networking with Red Hat Openshift 4.6 in multiple environments.

This document shows one method of installing Red Hat Openshift 4.6 with Contrail Networking in two separate contexts—on a VM running in a KVM module and within Amazon Web Services (AWS).

There are many implementation and configuration options available for installing and configuring Red Hat OpenShift 4.6 and the scope of all options is beyond this document. For additional information on Red Hat Openshift 4.6 implementation options, see the OpenShift Container Platform 4.6 Documentation from Red Hat.

This document includes the following sections:

How to Install Contrail Networking and Red Hat OpenShift 4.6 using a VM Running in a KVM Module

IN THIS SECTION

This section illustrates how to install Contrail Networking with Red Hat OpenShift 4.6 orchestration, where Contrail Networking and Red Hat Openshift are running on virtual machines (VMs) in a Kernel-based Virtual Machine (KVM) module.

This procedure can also be performed to configure an environment where Contrail Networking and Red Hat OpenShift 4.6 are running in an environment with bare metal servers. You can, for instance, use this procedure to establish an environment where the master nodes host the VMs that run the control plane on KVM while the worker nodes operate on physical bare metal servers.

When to Use This Procedure

This procedure is used to install Contrail Networking and Red Hat OpenShift 4.6 orchestration on a virtual machine (VM) running in a Kernel-based Virtual Machine (KVM) module. Support for Contrail Networking installations onto VMs in Red Hat OpenShift 4.6 environments is introduced in Contrail Networking Release 2011.L1. See Contrail Networking Supported Platforms.

You can also use this procedure to install Contrail Networking and Red Hat OpenShift 4.6 orchestration on a bare metal server.

You cannot incrementally upgrade from an environment using an earlier version of Red Hat OpenShift and Contrail Networking to an environment using Red Hat OpenShift 4.6. You must use this procedure to install Contrail Networking with Red Hat Openshift 4.6.

This procedure should work with all versions of Openshift 4.6.

Prerequisites

This document makes the following assumptions about your environment:

- the KVM environment is operational.

- the server meets the platform requirements for the Contrail Networking installation. See Contrail Networking Supported Platforms.
- Minimum server requirements:
  - Master nodes: 8 CPU, 40GB RAM, 250GB SSD storage

    NOTE: The term *master node* refers to the nodes that build the control plane in this document.

  - Worker nodes: 4 CPU, 16GB RAM, 120GB SSD storage

    NOTE: The term *worker node* refers to nodes running compute services using the data plane in this document.

  - Helper node: 4 CPU, 8GB RAM, 30GB SSD storage
- In single node deployments, do not use spinning disk arrays with low Input/Output Operations Per Second (IOPS) when using Contrail Networking with Red Hat Openshift. Higher IOPS disk arrays are required because the control plane always operates as a high availability setup in single node deployments.

  IOPS requirements vary by environment due to multiple factors beyond Contrail Networking and Red Hat Openshift. We, therefore, provide this guideline but do not provide direct guidance around IOPS requirements.

Install Contrail Networking and Red Hat Openshift 4.6

IN THIS SECTION

Perform these steps to install Contrail Networking and Red Hat OpenShift 4.6 using a VM running in a KVM module:

Create a Virtual Network or a Bridge Network for the Installation

To create a virtual network or a bridge network for the installation:

1. Log onto the server that will host the VM that will run Contrail Networking.

   Download the *virt-net.xml* virtual network configuration file from the Red Hat repository.

   ```
   # wget https://raw.githubusercontent.com/RedHatOfficial/ocp4-helpernode/master/docs/examples/
   virt-net.xml
   ```

2. Create a virtual network using the *virt-net.xml* file.

   You may need to modify your virtual network for your environment.

   Example:

   ```
   # virsh net-define --file virt-net.xml
   ```

3. Set the OpenShift 4 virtual network to autostart on bootup:

   ```
   # virsh net-autostart openshift4
   # virsh net-start openshift4
   ```

   NOTE: If the worker nodes are running on physical bare metal servers in your environment, this virtual network will be a bridge network with IP address allocations within the same subnet. This addressing scheme is similar to the scheme for the KVM server.

Create a Helper Node with a Virtual Machine Running CentOS 7 or 8

This procedure requires a helper node with a virtual machine that is running either CentOS 7 or 8.

To create this helper node:

1. Download the Kickstart file for the helper node from the Red Hat repository:

CentOS 8

```
# wget https://raw.githubusercontent.com/RedHatOfficial/ocp4-helpernode/master/docs/examples/
helper-ks8.cfg -O helper-ks.cfg
```

CentOS 7

```
# wget https://raw.githubusercontent.com/RedHatOfficial/ocp4-helpernode/master/docs/examples/
helper-ks.cfg -O helper-ks.cfg
```

2. If you haven't already configured a root password and the NTP server on the helper node, enter the following commands:

   Example Root Password

   ```
   rootpw --plaintext password
   ```

   Example NTP Configuration

   ```
   timezone America/Los_Angeles --isUtc --
   ntpservers=0.centos.pool.ntp.org,1.centos.pool.ntp.org,2.centos.pool.ntp.org,3.centos.pool.ntp
   .org
   ```

3. Edit the *helper-ks.cfg* file for your environment and use it to install the helper node.

   The following examples show how to install the helper node without having to take further actions:

   CentOS 8

   ```
   # virt-install --name="ocp4-aHelper" --vcpus=2 --ram=4096 \
   --disk path=/var/lib/libvirt/images/ocp4-aHelper.qcow2,bus=virtio,size=50 \
   --os-variant centos8 --network network=openshift4,model=virtio \
   --boot hd,menu=on --location /var/lib/libvirt/iso/CentOS-8.2.2004-x86_64-dvd1.iso \
   --initrd-inject helper-ks.cfg --extra-args "inst.ks=file:/helper-ks.cfg" --noautoconsole
   ```

   CentOS 7

   ```
   # virt-install --name="ocp4-aHelper" --vcpus=2 --ram=4096 \
   --disk path=/var/lib/libvirt/images/ocp4-aHelper.qcow2,bus=virtio,size=30 \
   --os-variant centos7.0 --network network=openshift4,model=virtio \
   ```

```
--boot hd,menu=on --location /var/lib/libvirt/iso/CentOS-7-x86_64-Minimal-2003.iso \
--initrd-inject helper-ks.cfg --extra-args "inst.ks=file:/helper-ks.cfg" --noautoconsole
```

The helper node is installed with the following settings, which are pulled from the *virt-net.xml* file:

- **HELPER_IP**: 192.168.7.77
- **NetMask**: 255.255.255.0
- **Default Gateway**: 192.168.7.1
- **DNS Server**: 8.8.8.8

4. Monitor the helper node installation progress in the viewer:

```
# virt-viewer --domain-name ocp4-aHelper
```

When the installation process is complete, the helper node shuts off.

5. Start the helper node:

```
# virsh start ocp4-aHelper
```

Prepare the Helper Node

To prepare the helper node after the helper node installation:

1. Login to the helper node:

```
# ssh -l root HELPER_IP
```

NOTE: The default *HELPER_IP*, which was pulled from the *virt-net.xml* file, is 192.168.7.77.

2. Install Enterprise Linux and update CentOS.

```
# yum -y install https://dl.fedoraproject.org/pub/epel/epel-release-latest-$(rpm -E
%rhel).noarch.rpm
# yum -y update
# reboot
```

3. Install Ansible and Git and clone the *helpernode* repository onto the helper node.

```
# yum -y install ansible git
# git clone https://github.com/RedHatOfficial/ocp4-helpernode
# cd ocp4-helpernode
```

4. Copy the vars.yaml file into the top-level directory:

```
# cp docs/examples/vars.yaml .
```

Review the vars.yml file. Consider changing any value that requires changing in your environment.

The following values should be reviewed especially carefully:

- The domain name, which is defined using the *domain:* parameter in the *dns:* hierarchy. If you are using local DNS servers, modify the forwarder parameters—*forwarder1:* and *forwarder2:* are used in this example—to connect to these DNS servers.
- Hostnames for master and worker nodes. Hostnames are defined using the *name:* parameter in either the *primaries:* or *workers:* hierarchies.
- IP and DHCP settings. If you are using a custom bridge network, modify the IP and DHCP settings accordingly.
- VM and BMS settings.

  If you are using a VM, set the *disk:* parameter as *disk: vda*.

  If you are using a BMS, set the *disk:* parameter as *disk: sda*.

A sample *vars.yml* file:

```
disk: vda
helper:
  name: "helper"
  ipaddr: "192.168.7.77"
dns:
  domain: "example.com"
  clusterid: "ocp4"
  forwarder1: "8.8.8.8"
  forwarder2: "8.8.4.4"
dhcp:
  router: "192.168.7.1"
  bcast: "192.168.7.255"
```

```
  netmask: "255.255.255.0"
  poolstart: "192.168.7.10"
  poolend: "192.168.7.30"
  ipid: "192.168.7.0"
  netmaskid: "255.255.255.0"
bootstrap:
  name: "bootstrap"
  ipaddr: "192.168.7.20"
  macaddr: "52:54:00:60:72:67"
masters:
  - name: "master0"
    ipaddr: "192.168.7.21"
    macaddr: "52:54:00:e7:9d:67"
  - name: "master1"
    ipaddr: "192.168.7.22"
    macaddr: "52:54:00:80:16:23"
  - name: "master2"
    ipaddr: "192.168.7.23"
    macaddr: "52:54:00:d5:1c:39"
workers:
  - name: "worker0"
    ipaddr: "192.168.7.11"
    macaddr: "52:54:00:f4:26:a1"
  - name: "worker1"
    ipaddr: "192.168.7.12"
    macaddr: "52:54:00:82:90:00"
```

NOTE: If you are using physical servers to host worker nodes, change the provisioning interface for the worker nodes to the mac address.

5. Review the *vars/main.yml* file to ensure the file reflects the correct version of Red Hat OpenShift. If you need to change the Red Hat Openshift version in the file, change it.

   In the following sample *main.yml* file, Red Hat Openshift 4.6 is installed:

```
ssh_gen_key: true
install_filetranspiler: false
staticips: false
force_ocp_download: false
remove_old_config_files: false
ocp_bios: "https://mirror.openshift.com/pub/openshift-v4/dependencies/rhcos/4.6/4.6.8/
```

```
rhcos-4.6.8-x86_64-live-rootfs.x86_64.img"
ocp_initramfs: "https://mirror.openshift.com/pub/openshift-v4/dependencies/rhcos/4.6/4.6.8/
rhcos-4.6.8-x86_64-live-initramfs.x86_64.img"
ocp_install_kernel: "https://mirror.openshift.com/pub/openshift-v4/dependencies/rhcos/
4.6/4.6.8/rhcos-4.6.8-x86_64-live-kernel-x86_64"
ocp_client: "https://mirror.openshift.com/pub/openshift-v4/clients/ocp/4.6.12/openshift-
client-linux-4.6.12.tar.gz"
ocp_installer: "https://mirror.openshift.com/pub/openshift-v4/clients/ocp/4.6.12/openshift-
install-linux-4.6.12.tar.gz"
helm_source: "https://get.helm.sh/helm-v3.5.0-linux-amd64.tar.gz"
chars: (\\_|\\$|\\|\\/|\\=|\\)|\\(|\\&|\\^|\\%|\\$|\\#|\\@|\\!|\\*)
ppc64le: false
uefi: false
chronyconfig:
  enabled: false
setup_registry:
  deploy: false
  autosync_registry: false
  registry_image: docker.io/library/registry:2
  local_repo: "ocp4/openshift4"
  product_repo: "openshift-release-dev"
  release_name: "ocp-release"
  release_tag: "4.6.1-x86_64"
```

6. Run the playbook to setup the helper node:

```
# ansible-playbook -e @vars.yaml tasks/main.yml
```

7. After the playbook is run, gather information about your environment and confirm that all services are active and running:

```
# /usr/local/bin/helpernodecheck services
Status of services:
===================
Status of dhcpd svc         ->    Active: active (running) since Mon 2020-09-28 05:40:10 EDT;
33min ago
Status of named svc         ->    Active: active (running) since Mon 2020-09-28 05:40:08 EDT;
33min ago
Status of haproxy svc     ->    Active: active (running) since Mon 2020-09-28 05:40:08 EDT;
33min ago
Status of httpd svc         ->    Active: active (running) since Mon 2020-09-28 05:40:10 EDT;
33min ago
```

```
Status of tftp svc         ->    Active: active (running) since Mon 2020-09-28 06:13:34 EDT;
1s ago
Unit local-registry.service could not be found.
Status of local-registry svc         ->
```

Create the Ignition Configurations

To create Ignition configurations:

1. On your hypervisor and helper nodes, check that your NTP server is properly configured in the */etc/chrony.conf* file:

   ```
   chronyc tracking
   ```

   The installation fails with a *X509: certificate has expired or is not yet valid* message when NTP is not properly configured.
2. Create a location to store your pull secret objects:

   ```
   # mkdir -p ~/.openshift
   ```
3. From Get Started with Openshift website, download your pull secret and save it in the *~/.openshift/pull-secret* directory.

   ```
   # ls -1 ~/.openshift/pull-secret
   /root/.openshift/pull-secret
   ```
4. (Contrail containers in password protected registries only) If the Contrail containers in your environment are in password protected registries, also add the authentication information for the registries in the *root/.openshift/pull-secret* directory.

   ```
   # cat ~/.openshift/pull-secret
   {
     "auths": {
       "hub.juniper.net": {
         "email": "example@example.com",
         "auth": "<base64 encoded concatenated line username:password>"
       },
       "cloud.openshift.com": {
         "auth": "…",
        …},
   ```

```
…
    }
```

5. An SSH key is created for you in the *~/.ssh/helper_rsa* directory after completing the previous step. You can use this key or create a unique key for authentication.

```
# ls -1 ~/.ssh/helper_rsa
/root/.ssh/helper_rsa
```

6. Create an installation directory.

```
# mkdir ~/ocp4
# cd ~/ocp4
```

7. Create an install-config.yaml file.

   An example file:

```
# cat <<EOF > install-config.yaml
apiVersion: v1
baseDomain: example.com
compute:
- hyperthreading: Enabled
  name: worker
  replicas: 0
controlPlane:
  hyperthreading: Enabled
  name: master
  replicas: 3
metadata:
  name: ocp4
networking:
  clusterNetworks:
  - cidr: 10.254.0.0/16
    hostPrefix: 24
  networkType: Contrail
  serviceNetwork:
  - 172.30.0.0/16
platform:
  none: {}
pullSecret: '$(< ~/.openshift/pull-secret)'
```

```
sshKey: '$(< ~/.ssh/helper_rsa.pub)'
EOF
```

8. Create the installation manifests:

```
# openshift-install create manifests
```

9. Set the **mastersSchedulable:** variable to **false** in the *manifests/cluster-scheduler-02-config.yml* file.

```
# sed -i 's/mastersSchedulable: true/mastersSchedulable: false/g' manifests/cluster-
scheduler-02-config.yml
```

A sample **cluster-scheduler-02-config.yml** file after this configuration change:

```
# cat manifests/cluster-scheduler-02-config.yml
apiVersion: config.openshift.io/v1
kind: Scheduler
metadata:
  creationTimestamp: null
  name: cluster
spec:
  mastersSchedulable: false
  policy:
    name: ""
status: {}
```

This configuration change is needed to prevent pods from being scheduled on control plane machines.

10. Download the tf-openshift installer (**tf-openshift-***release-tag***.tgz**) and the tf-operator (**tf-operator-***release-tag***.tgz**) for your release from the Contrail Networking Software Download Site.

    See the README Access to Contrail Registry 20XX to obtain the release tags for the installer for your version of Contrail Networking.

11. Install the YAML files to apply the Contrail configuration:

    Configure the YAML file for your environment, paying particular attention to the registry, container tag, cluster name, and domain fields.

The container tag for any R2011 and R2011.L release can be retrieved from README Access to Contrail Registry 20XX.

```
yum -y install git jq python3
python3 -m pip install jinja2
export INSTALL_DIR=$PWD
./tf-openshift/scripts/apply_install_manifests.sh $INSTALL_DIR
export CONTRAIL_CONTAINER_TAG="2011.L1.297"
export CONTAINER_REGISTRY="hub.juniper.net/contrail"
export DEPLOYER="openshift"
export KUBERNETES_CLUSTER_NAME="ocp4"
export KUBERNETES_CLUSTER_DOMAIN="example.com"
export CONTRAIL_REPLICAS=3
./tf-operator/contrib/render_manifests.sh
for i in $(ls ./tf-operator/deploy/crds/) ; do
  cp ./tf-operator/deploy/crds/$i $INSTALL_DIR/manifests/01_$i
done
for i in namespace service-account role cluster-role role-binding cluster-role-binding ; do
  cp ./tf-operator/deploy/kustomize/base/operator/$i.yaml $INSTALL_DIR/manifests/02-tf-
operator-$i.yaml
done
oc kustomize ./tf-operator/deploy/kustomize/operator/templates/ | sed -n 'H; /---/h; $
{g;p;}' > $INSTALL_DIR/manifests/02-tf-operator.yaml
oc kustomize ./tf-operator/deploy/kustomize/contrail/templates/ > $INSTALL_DIR/manifests/03-
tf.yaml
```

12. NTP synchronization on all master and worker nodes is required for proper functioning.

    If your environment has to use a specific NTP server, set the environment using the steps in the Openshift 4.x Chrony Configuration document.

13. Generate the Ignition configurations:

```
# openshift-install create ignition-configs
```

14. Copy the Ignition files in the Ignition directory for the webserver:

```
# cp ~/ocp4/*.ign /var/www/html/ignition/
# restorecon -vR /var/www/html/
# restorecon -vR /var/lib/tftpboot/
# chmod o+r /var/www/html/ignition/*.ign
```

Launch the Virtual Machines

To launch the virtual machines:

1. From the hypervisor, use PXE booting to launch the virtual machine or machines. If you are using a bare metal server, use PXE booting to boot the servers.
2. Launch the bootstrap virtual machine:

```
# virt-install --pxe --network bridge=openshift4 --mac=52:54:00:60:72:67 --name ocp4-
bootstrap --ram=16384 --vcpus=4 --os-variant rhel8.0 --disk path=/var/lib/libvirt/images/ocp4-
bootstrap.qcow2,size=120 --vnc
```

   The following actions occur as a result of this step:

   - a bootstrap node virtual machine is created.
   - the bootstrap node VM is connected to the PXE server. The PXE server is our helper node.
   - an IP address is assigned from DHCP.
   - A Red Hat Enterprise Linux CoreOS (RHCOS) image is downloaded from the HTTP server.

   The ignition file is embedded at the end of the installation process.
3. Use SSH to run the helper RSA:

```
# ssh -i ~/.ssh/helper_rsa core@192.168.7.20
```

4. Review the logs:

```
journalctl -f
```

5. On the bootstrap node, a temporary etcd and bootkube is created.

   You can monitor these services when they are running by entering the **sudo crictl ps** command.

```
[core@bootstrap ~]$ sudo crictl ps
CONTAINER      IMAGE         CREATED             STATE    NAME                              POD
ID
33762f4a23d7d  976cc3323...  54 seconds ago      Running  manager
29a...
ad6f2453d7a16  86694d2cd...  About a minute ago  Running  kube-apiserver-insecure-readyz
4cd...
3bbdf4176882f  quay.io/...   About a minute ago  Running  kube-scheduler
```

```
b3e...
57ad52023300e  quay.io/...   About a minute ago  Running  kube-controller-manager
596...
a1dbe7b8950da  quay.io/...   About a minute ago  Running  kube-apiserver
4cd...
5aa7a59a06feb  quay.io/...   About a minute ago  Running  cluster-version-operator
3ab...
ca45790f4a5f6  099c2a...     About a minute ago  Running  etcd-metrics
081...
e72fb8aaa1606  quay.io/...   About a minute ago  Running  etcd-member
081...
ca56bbf2708f7  1ac19399...   About a minute ago  Running  machine-config-server
c11...
```

NOTE: Output modified for readability.

6. From the hypervisor, launch the VMs on the master nodes:

```
# virt-install --pxe --network bridge=openshift4 --mac=52:54:00:e7:9d:67 --name ocp4-master0
--ram=40960 --vcpus=8 --os-variant rhel8.0 --disk path=/var/lib/libvirt/images/ocp4-
master0.qcow2,size=250 --vnc
# virt-install --pxe --network bridge=openshift4 --mac=52:54:00:80:16:23 --name ocp4-master1
--ram=40960 --vcpus=8 --os-variant rhel8.0 --disk path=/var/lib/libvirt/images/ocp4-
master1.qcow2,size=250 --vnc
# virt-install --pxe --network bridge=openshift4 --mac=52:54:00:d5:1c:39 --name ocp4-master2
--ram=40960 --vcpus=8 --os-variant rhel8.0 --disk path=/var/lib/libvirt/images/ocp4-
master2.qcow2,size=250 --vnc
```

You can login to the master nodes from the helper node after the master nodes have been provisioned:

```
# ssh -i ~/.ssh/helper_rsa core@192.168.7.21
# ssh -i ~/.ssh/helper_rsa core@192.168.7.22
# ssh -i ~/.ssh/helper_rsa core@192.168.7.23
```

Enter the **sudo crictl ps** at any point to monitor pod creation as the VMs are launching.

Monitor the Installation Process and Delete the Bootstrap Virtual Machine

To monitor the installation process:

1. From the helper node, navigate to the *~/ocp4* directory.
2. Track the install process log:

```
# openshift-install wait-for bootstrap-complete --log-level debug
```

Look for the *DEBUG Bootstrap status: complete* and the *INFO It is now safe to remove the bootstrap resources* messages to confirm that the installation is complete.

```
INFO Waiting up to 30m0s for the Kubernetes API at https://api.ocp4.example.com:6443...
INFO API v1.13.4+838b4fa up
INFO Waiting up to 30m0s for bootstrapping to complete...
DEBUG Bootstrap status: complete
INFO It is now safe to remove the bootstrap resources
```

Do not proceed to the next step until you see these messages.

3. From the hypervisor, delete the bootstrap VM and launch the worker nodes.

> **NOTE**: If you are using physical bare metal servers as worker nodes, skip this step.
> Boot the bare metal servers using PXE instead.

```
# virt-install --pxe --network bridge=openshift4 --mac=52:54:00:f4:26:a1 --name ocp4-worker0
--ram=16384 --vcpus=4 --os-variant rhel8.0 --disk path=/var/lib/libvirt/images/ocp4-
worker0.qcow2,size=120 --vnc

# virt-install --pxe --network bridge=openshift4 --mac=52:54:00:82:90:00 --name ocp4-worker1
--ram=16384 --vcpus=4 --os-variant rhel8.0 --disk path=/var/lib/libvirt/images/ocp4-
worker1.qcow2,size=120 --vnc
```

Finish the Installation

To finish the installation:

1. Login to your Kubernetes cluster:

```
# export KUBECONFIG=/root/ocp4/auth/kubeconfig
```

2. Your installation might be waiting for worker nodes to approve the certificate signing request (CSR). The machineconfig node approval operator typically handles CSR approval.

   CSR approval, however, sometimes has to be performed manually.

   To check pending CSRs:

   ```
   # oc get csr
   ```

   To approve all pending CSRs:

   ```
   # oc get csr -o go-template='{{range .items}}{{if not .status}}{{.metadata.name}}{{"\n"}}
   {{end}}{{end}}' | xargs oc adm certificate approve
   ```

   You may have to approve all pending CSRs multiple times, depending on the number of worker nodes in your environment and other factors.

   To monitor incoming CSRs:

   ```
   # watch -n5 oc get csr
   ```

   Do not move to the next step until incoming CSRs have stopped.

3. Set your cluster management state to *Managed*:

   ```
   # oc patch configs.imageregistry.operator.openshift.io cluster --type merge --patch '{"spec":
   {"managementState":"Managed"}}'
   ```

4. Setup your registry storage.

   For most environments, see Configuring registry storage for bare metal in the Red Hat Openshift documentation.

   For proof of concept labs and other smaller environments, you can set storage to *emptyDir*.

   ```
   # oc patch configs.imageregistry.operator.openshift.io cluster --type merge --patch '{"spec":
   {"storage":{"emptyDir":{}}}}'
   ```

5. If you need to make the registry accessible:

   ```
   # oc patch configs.imageregistry.operator.openshift.io/cluster --type merge -p '{"spec":
   {"defaultRoute":true}}'
   ```

6. Wait for the installation to finish:

```
# openshift-install wait-for install-complete
INFO Waiting up to 30m0s for the cluster at https://api.ocp4.example.com:6443 to initialize...
INFO Waiting up to 10m0s for the openshift-console route to be created...
INFO Install complete!
INFO To access the cluster as the system:admin user when using 'oc', run 'export KUBECONFIG=/
root/ocp4/auth/kubeconfig'
INFO Access the OpenShift web-console here: https://console-openshift-
console.apps.ocp4.example.com
INFO Login to the console with user: kubeadmin, password: XXX-XXXX-XXXX-XXXX
```

7. Add a user to the cluster. See "How to Add a User After Completing the Installation" on page 35.

RELATED DOCUMENTATION

Contrail Networking Supported Platforms

Installing a Standalone Red Hat OpenShift Container Platform 3.11 Cluster with Contrail Using Contrail OpenShift Deployer | 94

Installing a Nested Red Hat OpenShift Container Platform 3.11 Cluster Using Contrail Ansible Deployer | 106

How to Install Contrail Networking and Red Hat OpenShift 4.6 on Amazon Web Services

IN THIS SECTION

Follow these procedures to install Contrail Networking and Red Hat Openshift 4.6 on Amazon Web Services (AWS):

When to Use This Procedure

This procedure is used to install Contrail Networking and Red Hat OpenShift 4.6 orchestration in AWS. Support for Contrail Networking and Red Hat OpenShift 4.6 environments is introduced in Contrail Networking Release 2011.L1. See Contrail Networking Supported Platforms.

Prerequisites

This document makes the following assumptions about your environment:

- the server meets the platform requirements for the Contrail Networking installation. See Contrail Networking Supported Platforms.
- You have the Openshift binary version 4.4.8 files or later. See the Openshift Installation site if you need to update your binary files.
- You can access Openshift image pull secrets. See Using image pull secrets from Red Hat.
- You have an active AWS account.
- AWS CLI is installed. See Installing the AWS CLI from AWS.
- You have an SSH key that you can generate or provide on your local machine during the installation.

Configure DNS

A DNS zone must be created and available in Route 53 for your AWS account before starting this installation. You must also register a domain for your Contrail cluster in AWS Route 53. All entries created in AWS Route 53 are expected to be resolvable from the nodes in the Contrail cluster.

For information on configuring DNS zones in AWS Route 53, see the *Amazon Route 53 Developer Guide* from AWS.

Configure AWS Credentials

The installer used in this procedure creates multiple resources in AWS that are needed to run your cluster. These resources include Elastic Compute Cloud (EC2) instances, Virtual Private Clouds (VPCs), security groups, IAM roles, and other necessary network building blocks.

AWS credentials are needed to access these resources and should be configured before starting this installation.

To configure AWS credentials, see the Configuration and credential file settings section of the AWS Command Line Interface User Guide from AWS.

Download the OpenShift Installer and the Command Line Tools

To download the installer and the command line tools:

1. Check which versions of the OpenShift installer are available:

```
$ curl -s https://mirror.openshift.com/pub/openshift-v4/clients/ocp/ | \
  awk '{print $5}'| \
  grep -o '4.[0-9].[0-9]*' | \
  uniq | \
  sort | \
  column
```

2. Set the version and download the OpenShift installer and the CLI tool.

   In this example output, the Openshift version is 4.6.12.

```
$ VERSION=4.6.12
$ wget https://mirror.openshift.com/pub/openshift-v4/clients/ocp/$VERSION/openshift-install-
mac-$VERSION.tar.gz
$ wget https://mirror.openshift.com/pub/openshift-v4/clients/ocp/$VERSION/openshift-client-
mac-$VERSION.tar.gz

$ tar -xvzf openshift-install-mac-${VERSION}.tar.gz -C /usr/local/bin
$ tar -xvzf openshift-client-mac-${VERSION}.tar.gz -C /usr/local/bin

$ openshift-install version
$ oc version
$ kubectl version
```

Deploy the Cluster

To deploy the cluster:

1. Generate an SSH private key and add it to the agent:

```
$ ssh-keygen -b 4096 -t rsa -f ~/.ssh/id_rsa -N ""
```

2. Create a working folder:

In this example, a working folder named *aws-ocp4* is created and the user is then moved into the new directory.

```
$ mkdir ~/aws-ocp4 ; cd ~/aws-ocp4
```

3. Create an installation configuration file. See Creating the installation configuration file section of the Installing a cluster on AWS with customizations document from Red Hat OpenShift.

```
$ openshift-install create install-config
```

An *install-config.yaml* file needs to be created and added to the current directory. A sample *install-config.yaml* file is provided below.

Be aware of the following factors while creating the *install-config.yaml* file:

- The *networkType* field is usually set as *OpenShiftSDN* in the YAML file by default.

  For configuration pointing at Contrail cluster nodes, the *networkType* field needs to be configured as *Contrail*.

- OpenShift master nodes need larger instances. We recommend setting the type to *m5.2xlarge* or larger for OpenShift nodes.

- Most OpenShift worker nodes can use the default instance sizes. You should consider using larger instances, however, for high demand performance workloads.

- Many of the installation parameters in the YAML file are described in more detail in the Installation configuration parameters section of the Installing a cluster on AWS with customizations document from Red Hat OpenShift.

- You may want to add the credentials to the Contrail secured registry at *hub.juniper.net* at this point of the procedure.

A sample *install-config.yaml* file:

```
apiVersion: v1
baseDomain: ovsandbox.com
compute:
- architecture: amd64
  hyperthreading: Enabled
  name: worker
  platform:
    aws:
      rootVolume:
```

```
        iops: 2000
        size: 500
        type: io1
      type: m5.4xlarge
  replicas: 3
controlPlane:
  architecture: amd64
  hyperthreading: Enabled
  name: master
  platform:
    aws:
      rootVolume:
        iops: 4000
        size: 500
        type: io1
      type: m5.2xlarge
  replicas: 3
metadata:
  creationTimestamp: null
  name: w1
networking:
  clusterNetwork:
  - cidr: 10.128.0.0/14
    hostPrefix: 23
  machineNetwork:
  - cidr: 10.0.0.0/16
  networkType: Contrail
  serviceNetwork:
  - 172.30.0.0/16
platform:
  aws:
    region: eu-west-1
publish: External
pullSecret: '{"auths"...}'
sshKey: |
  ssh-rsa ...
```

4. Create the installation manifests:

```
# openshift-install create manifests
```

5. Download the tf-openshift installer (**tf-openshift-***release-tag***.tgz**) and the tf-operator (**tf-operator-***release-tag***.tgz**) for your release from the Contrail Networking Software Download Site.

   See the README Access to Contrail Registry 20XX to obtain the release tags for the installer for your version of Contrail Networking.

6. Install the YAML files to apply the Contrail configuration.

   Configure the YAML file for your environment, paying particular attention to the registry, container tag, cluster name, and domain fields.

   The container tag for any R2011 and R2011.L release can be retrieved from README Access to Contrail Registry 20XX.

```
yum -y install git jq python3
python3 -m pip install jinja2
export INSTALL_DIR=$PWD./tf-openshift/scripts/apply_install_manifests.sh $INSTALL_DIR

export CONTRAIL_CONTAINER_TAG="2011.L1.297"
export CONTAINER_REGISTRY="hub.juniper.net/contrail"
export DEPLOYER="openshift"
export KUBERNETES_CLUSTER_NAME="ocp4"
export KUBERNETES_CLUSTER_DOMAIN="example.com"
export CONTRAIL_REPLICAS=3
./tf-operator/contrib/render_manifests.sh
for i in $(ls ./tf-operator/deploy/crds/) ; do
  cp ./tf-operator/deploy/crds/$i $INSTALL_DIR/manifests/01_$i
done
for i in namespace service-account role cluster-role role-binding cluster-role-binding ; do
  cp ./tf-operator/deploy/kustomize/base/operator/$i.yaml $INSTALL_DIR/manifests/02-tf-
operator-$i.yaml
done
oc kustomize ./tf-operator/deploy/kustomize/operator/templates/ | sed -n 'H; /---/h; $
{g;p;}' > $INSTALL_DIR/manifests/02-tf-operator.yaml
oc kustomize ./tf-operator/deploy/kustomize/contrail/templates/ > $INSTALL_DIR/manifests/03-
tf.yaml
```

7. Modify the YAML files for your environment.

   The scope of each potential configuration changes is beyond the scope of this document.

   Common configuration changes include:

   - If you are using non-default network-CIDR subnets for your pods or services, open the *deploy/openshift/manifests/cluster-network-02-config.yml* file and update the CIDR values.

- The default number of master nodes in a Kubernetes cluster is 3. If you are using a different number of master nodes, modify the *deploy/openshift/manifests/00-contrail-09-manager.yaml* file and set the spec.commonConfiguration.replicas field to the number of master nodes.

8. Create the cluster:

```
$ openshift-install create cluster --log-level=debug
```

- Contrail Networking needs to open some networking ports for operation within AWS. These ports are opened by adding rules to security groups.

  Follow this procedure to add rules to security groups when AWS resources are manually created:

  a. Build the Contrail CLI tool for managing security group ports on AWS. This tool allows you to automatically open ports that are required for Contrail to manage security group ports on AWS that are attached to Contrail cluster resources.

     To build this tool:

     i. Clone the tool operator into AWS. In this sample output, the operator is cloned for Contrail Networking Release 2011:

     ```
     git clone https://github.com/tungstenfabric/tf-operator.git -b R2011
     ```

     ii. Build the operator tool:

     ```
     cd /root/tf-operator/contrib/aws/
         go build .
     ```

     iii. Start the tool:

     ```
     ./tf-sc-open -cluster-name name of your Openshift cluster -region AWS region
     where cluster is located
     ```

     After entering this command, you should be in the tf-sc-open tool in your directory. This interface is the compiled tool.

b. Verify that the service has been created:

```
oc -n openshift-ingress get service router-default
```

Proceed to the next step after confirming the service was created.

9. When the service router-default is created in openshift-ingress, use the following command to patch the configuration:

```
$ oc -n openshift-ingress patch service router-default --patch '{"spec":
{"externalTrafficPolicy": "Cluster"}}'
```

10. Monitor the screen messages.

Look for the *INFO Install complete!*.

The final messages from a sample successful installation:

```
INFO Waiting up to 10m0s for the openshift-console route to be created...
DEBUG Route found in openshift-console namespace: console
DEBUG Route found in openshift-console namespace: downloads
DEBUG OpenShift console route is created
INFO Install complete!
INFO To access the cluster as the system:admin user when using 'oc', run 'export
KUBECONFIG=/Users/ovaleanu/aws1-ocp4/auth/kubeconfig'
INFO Access the OpenShift web-console here: https://console-openshift-
console.apps.w1.ovsandbox.com
INFO Login to the console with user: kubeadmin, password: XXXxx-XxxXX-xxXXX-XxxxX
```

11. Access the cluster:

```
$ export KUBECONFIG=~/aws-ocp4/auth/kubeconfig
```

12. Add a user to the cluster. See "How to Add a User After Completing the Installation" on page 35.

How to Add a User After Completing the Installation

The process for adding an Openshift user is identical in KVM or on AWS.

Redhat OpenShift 4.6 supports a single kubeadmin user by default. This kubeadmin user is used to deploy the initial cluster configuration.

You can use this procedure to create a Custom Resource (CR) to define a HTTPasswd identity provider.

1. Generate a flat file that contains the user names and passwords for your cluster by using the HTPasswd identity provider:

```
$ htpasswd -c -B -b users.htpasswd testuser MyPassword
```

   A file called users.httpasswd is created.

2. Define a secret password that contains the HTPasswd user file:

```
$ oc create secret generic htpass-secret --from-file=htpasswd=/root/ocp4/users.htpasswd -n
openshift-config
```

   This custom resource shows the parameters and acceptable values for an HTPasswd identity provider.

```
$ cat htpasswdCR.yaml
apiVersion: config.openshift.io/v1
kind: OAuth
metadata:
  name: cluster
spec:
  identityProviders:
  - name: testuser
    mappingMethod: claim
    type: HTPasswd
    htpasswd:
      fileData:
        name: htpass-secret
```

3. Apply the defined custom resource:

```
$ oc create -f htpasswdCR.yaml
```

4. Add the user and assign the *cluster-admin* role:

```
$ oc adm policy add-cluster-role-to-user cluster-admin testuser
```

5. Login using the new user credentials:

```
oc login -u testuser
Authentication required for https://api.ocp4.example.com:6443 (openshift)
Username: testuser
Password:
Login successful.
```

The kubeadmin user can now safely be removed. See the Removing the kubeadmin user document from Red Hat OpenShift.

How to Install Earlier Releases of Contrail Networking and Red Hat OpenShift

If you have a need to install Contrail Networking with earlier versions of Red Hat Openshift, earlier versions of Contrail Networking are also supported with Red Hat Openshift versions 4.5, 4.4, and 3.11.

For information on installing Contrail Networking with Red Hat Openshift 4.5, see "How to Install Contrail Networking and Red Hat OpenShift 4.5" on page 38.

For information on installing Contrail Networking with Red Hat Openshift 4.4, see "How to Install Contrail Networking and Red Hat OpenShift 4.4" on page 69.

For information on installing Contrail Networking with Red Hat Openshift 3.11, see the following documentation:

- "Installing a Standalone Red Hat OpenShift Container Platform 3.11 Cluster with Contrail Using Contrail OpenShift Deployer" on page 94
- "Installing a Nested Red Hat OpenShift Container Platform 3.11 Cluster Using Contrail Ansible Deployer" on page 106

NOTE: The session affinity by client with ClusterIP service is not supported. Contrail Networking implementation of ClusterIP service uses ECMP load balancer and supports stickiness per flow, not per client IP address.

How to Install Contrail Networking and Red Hat OpenShift 4.5

IN THIS SECTION

NOTE: This topic covers Contrail Networking in Red Hat Openshift environments that are using Contrail Networking Release 21-based releases.

Starting in Release 22.1, Contrail Networking evolved into Cloud-Native Contrail Networking. Cloud-Native Contrail offers significant enhancements to optimize networking performance in Kubernetes-orchestrated environments. Cloud-Native Contrail supports Red Hat Openshift and we strongly recommend using Cloud-Native Contrail for networking in environments using Red Hat Openshift.

For general information about Cloud-Native Contrail, see the Cloud-Native Contrail Networking Techlibrary homepage.

Starting in Contrail Networking Release 2011, you can install –Contrail Networking with Red Hat Openshift 4.5 in multiple environments.

This document shows one method of installing Red Hat Openshift 4.5 with Contrail Networking in two separate contexts—on a VM running in a KVM module and within Amazon Web Services (AWS).

There are many implementation and configuration options available for installing and configuring Red Hat OpenShift 4.5 and the scope of all options is beyond this document. For additional information on Red Hat Openshift 4.5 implementation options, see the OpenShift Container Platform 4.5 Documentation from Red Hat.

This document includes the following sections:

How to Install Contrail Networking and Red Hat OpenShift 4.5 using a VM Running in a KVM Module

IN THIS SECTION

This section illustrates how to install Contrail Networking with Red Hat OpenShift 4.5 orchestration, where Contrail Networking and Red Hat Openshift are running on virtual machines (VMs) in a Kernel-based Virtual Machine (KVM) module.

This procedure can also be performed to configure an environment where Contrail Networking and Red Hat OpenShift 4.5 are running in an environment with bare metal servers. You can, for instance, use this procedure to establish an environment where the master nodes host the VMs that run the control plane on KVM while the worker nodes operate on physical bare metal servers.

When to Use This Procedure

This procedure is used to install Contrail Networking and Red Hat OpenShift 4.5 orchestration on a virtual machine (VM) running in a Kernel-based Virtual Machine (KVM) module. Support for Contrail Networking installations onto VMs in Red Hat OpenShift 4.5 environments is introduced in Contrail Networking Release 2011. See Contrail Networking Supported Platforms.

You can also use this procedure to install Contrail Networking and Red Hat OpenShift 4.5 orchestration on a bare metal server.

This procedure should work with all versions of Openshift 4.5.

Prerequisites

This document makes the following assumptions about your environment:

- the KVM environment is operational.
- the server meets the platform requirements for the Contrail Networking installation. See Contrail Networking Supported Platforms.
- Minimum server requirements:
  - Master nodes: 8 CPU, 40GB RAM, 250GB SSD storage

    NOTE: The term *master node* refers to the nodes that build the control plane in this document.

  - Worker nodes: 4 CPU, 16GB RAM, 120GB SSD storage

    NOTE: The term *worker node* refers to nodes running compute services using the data plane in this document.

  - Helper node: 4 CPU, 8GB RAM, 30GB SSD storage
- In single node deployments, do not use spinning disk arrays with low Input/Output Operations Per Second (IOPS) when using Contrail Networking with Red Hat Openshift. Higher IOPS disk arrays are required because the control plane always operates as a high availability setup in single node deployments.

  IOPS requirements vary by environment due to multiple factors beyond Contrail Networking and Red Hat Openshift. We, therefore, provide this guideline but do not provide direct guidance around IOPS requirements.

Install Contrail Networking and Red Hat Openshift 4.5

IN THIS SECTION

Perform these steps to install Contrail Networking and Red Hat OpenShift 4.5 using a VM running in a KVM module:

Create a Virtual Network or a Bridge Network for the Installation

To create a virtual network or a bridge network for the installation:

1. Log onto the server that will host the VM that will run Contrail Networking.

   Download the *virt-net.xml* virtual network configuration file from the Red Hat repository.

   ```
   # wget https://raw.githubusercontent.com/RedHatOfficial/ocp4-helpernode/master/docs/examples/
   virt-net.xml
   ```

2. Create a virtual network using the *virt-net.xml* file.

   You may need to modify your virtual network for your environment.

   Example:

   ```
   # virsh net-define --file virt-net.xml
   ```

3. Set the OpenShift 4 virtual network to autostart on bootup:

   ```
   # virsh net-autostart openshift4
   # virsh net-start openshift4
   ```

   NOTE: If the worker nodes are running on physical bare metal servers in your environment, this virtual network will be a bridge network with IP address allocations within the same subnet. This addressing scheme is similar to the scheme for the KVM server.

Create a Helper Node with a Virtual Machine Running CentOS 7 or 8

This procedure requires a helper node with a virtual machine that is running either CentOS 7 or 8.

To create this helper node:

1. Download the Kickstart file for the helper node from the Red Hat repository:

   CentOS 8

   ```
   # wget https://raw.githubusercontent.com/RedHatOfficial/ocp4-helpernode/master/docs/examples/
   helper-ks8.cfg -O helper-ks.cfg
   ```

   CentOS 7

   ```
   # wget https://raw.githubusercontent.com/RedHatOfficial/ocp4-helpernode/master/docs/examples/
   helper-ks.cfg -O helper-ks.cfg
   ```

2. If you haven't already configured a root password and the NTP server on the helper node, enter the following commands:

   Example Root Password

   ```
   rootpw --plaintext password
   ```

   Example NTP Configuration

   ```
   timezone America/Los_Angeles --isUtc --
   ntpservers=0.centos.pool.ntp.org,1.centos.pool.ntp.org,2.centos.pool.ntp.org,3.centos.pool.ntp
   .org
   ```

3. Edit the *helper-ks.cfg* file for your environment and use it to install the helper node.

   The following examples show how to install the helper node without having to take further actions:

   CentOS 8

   ```
   # virt-install --name="ocp4-aHelper" --vcpus=2 --ram=4096 \
   --disk path=/var/lib/libvirt/images/ocp4-aHelper.qcow2,bus=virtio,size=50 \
   --os-variant centos8 --network network=openshift4,model=virtio \
   --boot hd,menu=on --location /var/lib/libvirt/iso/CentOS-8.2.2004-x86_64-dvd1.iso \
   --initrd-inject helper-ks.cfg --extra-args "inst.ks=file:/helper-ks.cfg" --noautoconsole
   ```

CentOS 7

```
# virt-install --name="ocp4-aHelper" --vcpus=2 --ram=4096 \
--disk path=/var/lib/libvirt/images/ocp4-aHelper.qcow2,bus=virtio,size=30 \
--os-variant centos7.0 --network network=openshift4,model=virtio \
--boot hd,menu=on --location /var/lib/libvirt/iso/CentOS-7-x86_64-Minimal-2003.iso \
--initrd-inject helper-ks.cfg --extra-args "inst.ks=file:/helper-ks.cfg" --noautoconsole
```

The helper node is installed with the following settings, which are pulled from the *virt-net.xml* file:

- **HELPER_IP**: 192.168.7.77
- **NetMask**: 255.255.255.0
- **Default Gateway**: 192.168.7.1
- **DNS Server**: 8.8.8.8

4. Monitor the helper node installation progress in the viewer:

```
# virt-viewer --domain-name ocp4-aHelper
```

When the installation process is complete, the helper node shuts off.

5. Start the helper node:

```
# virsh start ocp4-aHelper
```

Prepare the Helper Node

To prepare the helper node after the helper node installation:

1. Login to the helper node:

```
# ssh -l root HELPER_IP
```

NOTE: The default *HELPER_IP*, which was pulled from the *virt-net.xml* file, is 192.168.7.77.

2. Install Enterprise Linux and update CentOS.

```
# yum -y install https://dl.fedoraproject.org/pub/epel/epel-release-latest-$(rpm -E
%rhel).noarch.rpm
# yum -y update
```

3. Install Ansible and Git and clone the *helpernode* repository onto the helper node.

```
# yum -y install ansible git
# git clone https://github.com/RedHatOfficial/ocp4-helpernode
# cd ocp4-helpernode
```

4. Copy the vars.yaml file into the top-level directory:

```
# cp docs/examples/vars.yaml .
```

Review the vars.yml file. Consider changing any value that requires changing in your environment.

The following values should be reviewed especially carefully:

- The domain name, which is defined using the *domain:* parameter in the *dns:* hierarchy. If you are using local DNS servers, modify the forwarder parameters—*forwarder1:* and *forwarder2:* are used in this example—to connect to these DNS servers.
- Hostnames for master and worker nodes. Hostnames are defined using the *name:* parameter in either the *primaries:* or *workers:* hierarchies.
- IP and DHCP settings. If you are using a custom bridge network, modify the IP and DHCP settings accordingly.
- VM and BMS settings.

  If you are using a VM, set the *disk:* parameter as *disk: vda*.

  If you are using a BMS, set the *disk:* parameter as *disk: sda*.

A sample *vars.yml* file:

```
disk: vda
helper:
  name: "helper"
  ipaddr: "192.168.7.77"
dns:
```

```
  domain: "example.com"
  clusterid: "ocp4"
  forwarder1: "8.8.8.8"
  forwarder2: "8.8.4.4"
dhcp:
  router: "192.168.7.1"
  bcast: "192.168.7.255"
  netmask: "255.255.255.0"
  poolstart: "192.168.7.10"
  poolend: "192.168.7.30"
  ipid: "192.168.7.0"
  netmaskid: "255.255.255.0"
bootstrap:
  name: "bootstrap"
  ipaddr: "192.168.7.20"
  macaddr: "52:54:00:60:72:67"
masters:
  - name: "master0"
    ipaddr: "192.168.7.21"
    macaddr: "52:54:00:e7:9d:67"
  - name: "master1"
    ipaddr: "192.168.7.22"
    macaddr: "52:54:00:80:16:23"
  - name: "master2"
    ipaddr: "192.168.7.23"
    macaddr: "52:54:00:d5:1c:39"
workers:
  - name: "worker0"
    ipaddr: "192.168.7.11"
    macaddr: "52:54:00:f4:26:a1"
  - name: "worker1"
    ipaddr: "192.168.7.12"
    macaddr: "52:54:00:82:90:00"
```

NOTE: If you are using physical servers to host worker nodes, change the provisioning interface for the worker nodes to the mac address.

5. Review the *vars/main.yml* file to ensure the file reflects the correct version of Red Hat OpenShift. If you need to change the Red Hat Openshift version in the file, change it.

In the following sample *main.yml* file, Red Hat Openshift 4.5 is installed:

```
ssh_gen_key: true
install_filetranspiler: false
staticips: false
force_ocp_download: false
remove_old_config_files: false
ocp_bios: "https://mirror.openshift.com/pub/openshift-v4/dependencies/rhcos/4.5/4.5.6/
rhcos-4.5.6-x86_64-metal.x86_64.raw.gz"
ocp_initramfs: "https://mirror.openshift.com/pub/openshift-v4/dependencies/rhcos/4.5/4.5.6/
rhcos-4.5.6-x86_64-installer-initramfs.x86_64.img"
ocp_install_kernel: "https://mirror.openshift.com/pub/openshift-v4/dependencies/rhcos/
4.5/4.5.6/rhcos-4.5.6-x86_64-installer-kernel-x86_64"
ocp_client: "https://mirror.openshift.com/pub/openshift-v4/clients/ocp/4.5.21/openshift-
client-linux-4.5.21.tar.gz"
ocp_installer: "https://mirror.openshift.com/pub/openshift-v4/clients/ocp/4.5.21/openshift-
install-linux-4.5.21.tar.gz"
helm_source: "https://get.helm.sh/helm-v3.2.4-linux-amd64.tar.gz"
chars: (\\_|\\$|\\|\\/|\\=|\\)|\\(|\\&|\\^|\\%|\\$|\\#|\\@|\\!|\\*)
ppc64le: false
chronyconfig:
  enabled: false
setup_registry:
  deploy: false
  autosync_registry: false
  registry_image: docker.io/library/registry:2
  local_repo: "ocp4/openshift4"
  product_repo: "openshift-release-dev"
  release_name: "ocp-release"
  release_tag: "4.5.21-x86_64"
```

6. Run the playbook to setup the helper node:

```
# ansible-playbook -e @vars.yaml tasks/main.yml
```

7. After the playbook is run, gather information about your environment and confirm that all services are active and running:

```
# /usr/local/bin/helpernodecheck services
Status of services:
===================
```

```
Status of dhcpd svc         ->    Active: active (running) since Mon 2020-09-28 05:40:10 EDT;
33min ago
Status of named svc         ->    Active: active (running) since Mon 2020-09-28 05:40:08 EDT;
33min ago
Status of haproxy svc     ->    Active: active (running) since Mon 2020-09-28 05:40:08 EDT;
33min ago
Status of httpd svc         ->    Active: active (running) since Mon 2020-09-28 05:40:10 EDT;
33min ago
Status of tftp svc         ->    Active: active (running) since Mon 2020-09-28 06:13:34 EDT;
1s ago
Unit local-registry.service could not be found.
Status of local-registry svc         ->
```

Create the Ignition Configurations

To create Ignition configurations:

1. On your hypervisor and helper nodes, check that your NTP server is properly configured in the */etc/chrony.conf* file:

   ```
   chronyc tracking
   ```

   The installation fails with a *X509: certificate has expired or is not yet valid* message when NTP is not properly configured.
2. Create a location to store your pull secret objects:

   ```
   # mkdir -p ~/.openshift
   ```
3. From Get Started with Openshift website, download your pull secret and save it in the *~/.openshift/pull-secret* directory.

   ```
   # ls -1 ~/.openshift/pull-secret
   /root/.openshift/pull-secret
   ```
4. An SSH key is created for you in the *~/.ssh/helper_rsa* directory after completing the previous step. You can use this key or create a unique key for authentication.

   ```
   # ls -1 ~/.ssh/helper_rsa
   /root/.ssh/helper_rsa
   ```

5. Create an installation directory.

```
# mkdir ~/ocp4
# cd ~/ocp4
```

6. Create an install-config.yaml file.

   An example file:

```
# cat <<EOF > install-config.yaml
apiVersion: v1
baseDomain: example.com
compute:
- hyperthreading: Enabled
  name: worker
  replicas: 0
controlPlane:
  hyperthreading: Enabled
  name: master
  replicas: 3
metadata:
  name: ocp4
networking:
  clusterNetworks:
  - cidr: 10.254.0.0/16
    hostPrefix: 24
  networkType: Contrail
  serviceNetwork:
  - 172.30.0.0/16
platform:
  none: {}
pullSecret: '$(< ~/.openshift/pull-secret)'
sshKey: '$(< ~/.ssh/helper_rsa.pub)'
EOF
```

7. Create the installation manifests:

```
# openshift-install create manifests
```

8. Set the **mastersSchedulable:** variable to **false** in the *manifests/cluster-scheduler-02-config.yml* file.

```
# sed -i 's/mastersSchedulable: true/mastersSchedulable: false/g' manifests/cluster-
scheduler-02-config.yml
```

A sample **cluster-scheduler-02-config.yml** file after this configuration change:

```
# cat manifests/cluster-scheduler-02-config.yml
apiVersion: config.openshift.io/v1
kind: Scheduler
metadata:
  creationTimestamp: null
  name: cluster
spec:
  mastersSchedulable: false
  policy:
    name: ""
status: {}
```

This configuration change is needed to prevent pods from being scheduled on control plane machines.

9. Install the YAML files to apply the Contrail configuration:

```
bash <<EOF
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/openshift/
manifests/00-contrail-01-namespace.yaml -o manifests/00-contrail-01-namespace.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/openshift/
manifests/00-contrail-02-admin-password.yaml -o manifests/00-contrail-02-admin-
password.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/openshift/
manifests/00-contrail-02-rbac-auth.yaml -o manifests/00-contrail-02-rbac-auth.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/openshift/
manifests/00-contrail-02-registry-secret.yaml -o manifests/00-contrail-02-registry-
secret.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/openshift/
manifests/00-contrail-03-cluster-role.yaml -o manifests/00-contrail-03-cluster-role.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/openshift/
manifests/00-contrail-04-serviceaccount.yaml -o manifests/00-contrail-04-
serviceaccount.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/openshift/
```

```
manifests/00-contrail-05-rolebinding.yaml -o manifests/00-contrail-05-rolebinding.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/openshift/
manifests/00-contrail-06-clusterrolebinding.yaml -o manifests/00-contrail-06-
clusterrolebinding.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/crds/
contrail.juniper.net_cassandras_crd.yaml -o manifests/00-contrail-07-
contrail.juniper.net_cassandras_crd.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/crds/
contrail.juniper.net_commands_crd.yaml -o manifests/00-contrail-07-
contrail.juniper.net_commands_crd.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/crds/
contrail.juniper.net_configs_crd.yaml -o manifests/00-contrail-07-
contrail.juniper.net_configs_crd.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/crds/
contrail.juniper.net_contrailcnis_crd.yaml -o manifests/00-contrail-07-
contrail.juniper.net_contrailcnis_crd.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/crds/
contrail.juniper.net_fernetkeymanagers_crd.yaml -o manifests/00-contrail-07-
contrail.juniper.net_fernetkeymanagers_crd.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/crds/
contrail.juniper.net_contrailmonitors_crd.yaml -o manifests/00-contrail-07-
contrail.juniper.net_contrailmonitors_crd.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/crds/
contrail.juniper.net_contrailstatusmonitors_crd.yaml -o manifests/00-contrail-07-
contrail.juniper.net_contrailstatusmonitors_crd.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/crds/
contrail.juniper.net_controls_crd.yaml -o manifests/00-contrail-07-
contrail.juniper.net_controls_crd.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/crds/
contrail.juniper.net_keystones_crd.yaml -o manifests/00-contrail-07-
contrail.juniper.net_keystones_crd.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/crds/
contrail.juniper.net_kubemanagers_crd.yaml -o manifests/00-contrail-07-
contrail.juniper.net_kubemanagers_crd.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/crds/
contrail.juniper.net_managers_crd.yaml -o manifests/00-contrail-07-
contrail.juniper.net_managers_crd.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/crds/
contrail.juniper.net_memcacheds_crd.yaml -o manifests/00-contrail-07-
contrail.juniper.net_memcacheds_crd.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/crds/
contrail.juniper.net_postgres_crd.yaml -o manifests/00-contrail-07-
contrail.juniper.net_postgres_crd.yaml;\
```

```
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/crds/
contrail.juniper.net_provisionmanagers_crd.yaml -o manifests/00-contrail-07-
contrail.juniper.net_provisionmanagers_crd.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/crds/
contrail.juniper.net_rabbitmqs_crd.yaml -o manifests/00-contrail-07-
contrail.juniper.net_rabbitmqs_crd.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/crds/
contrail.juniper.net_swiftproxies_crd.yaml -o manifests/00-contrail-07-
contrail.juniper.net_swiftproxies_crd.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/crds/
contrail.juniper.net_swifts_crd.yaml -o manifests/00-contrail-07-
contrail.juniper.net_swifts_crd.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/crds/
contrail.juniper.net_swiftstorages_crd.yaml -o manifests/00-contrail-07-
contrail.juniper.net_swiftstorages_crd.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/crds/
contrail.juniper.net_vrouters_crd.yaml -o manifests/00-contrail-07-
contrail.juniper.net_vrouters_crd.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/crds/
contrail.juniper.net_webuis_crd.yaml -o manifests/00-contrail-07-
contrail.juniper.net_webuis_crd.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/crds/
contrail.juniper.net_zookeepers_crd.yaml -o manifests/00-contrail-07-
contrail.juniper.net_zookeepers_crd.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/openshift/
releases/R2011/manifests/00-contrail-08-operator.yaml -o manifests/00-contrail-08-
operator.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/openshift/
releases/R2011/manifests/00-contrail-09-manager.yaml -o manifests/00-contrail-09-
manager.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/openshift/
manifests/cluster-network-02-config.yml -o manifests/cluster-network-02-config.yml
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/openshift/
openshift/99_master-iptables-machine-config.yaml -o openshift/99_master-iptables-machine-
config.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/openshift/
openshift/99_master-kernel-modules-overlay.yaml -o openshift/99_master-kernel-modules-
overlay.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/openshift/
openshift/99_master_network_functions.yaml -o openshift/99_master_network_functions.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/openshift/
openshift/99_master_network_manager_stop_service.yaml -o openshift/
99_master_network_manager_stop_service.yaml;\
```

```
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/openshift/
openshift/99_master-pv-mounts.yaml -o openshift/99_master-pv-mounts.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/openshift/
openshift/99_worker-iptables-machine-config.yaml -o openshift/99_worker-iptables-machine-
config.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/openshift/
openshift/99_worker-kernel-modules-overlay.yaml -o openshift/99_worker-kernel-modules-
overlay.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/openshift/
openshift/99_worker_network_functions.yaml -o openshift/99_worker_network_functions.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/openshift/
openshift/99_worker_network_manager_stop_service.yaml -o openshift/
99_worker_network_manager_stop_service.yaml;
EOF
```

10. If your environment has to use a specific NTP server, set the environment using the steps in the Openshift 4.x Chrony Configuration document.
11. Generate the Ignition configurations:

```
# openshift-install create ignition-configs
```

12. Copy the Ignition files in the Ignition directory for the webserver:

```
# cp ~/ocp4/*.ign /var/www/html/ignition/
# restorecon -vR /var/www/html/
# restorecon -vR /var/lib/tftpboot/
# chmod o+r /var/www/html/ignition/*.ign
```

Launch the Virtual Machines

To launch the virtual machines:

1. From the hypervisor, use PXE booting to launch the virtual machine or machines. If you are using a bare metal server, use PXE booting to boot the servers.
2. Launch the bootstrap virtual machine:

```
# virt-install --pxe --network bridge=openshift4 --mac=52:54:00:60:72:67 --name ocp4-
bootstrap --ram=8192 --vcpus=4 --os-variant rhel8.0 --disk path=/var/lib/libvirt/images/ocp4-
bootstrap.qcow2,size=120 --vnc
```

The following actions occur as a result of this step:

- a bootstrap node virtual machine is created.
- the bootstrap node VM is connected to the PXE server. The PXE server is our helper node.
- an IP address is assigned from DHCP.
- A Red Hat Enterprise Linux CoreOS (RHCOS) image is downloaded from the HTTP server.

The ignition file is embedded at the end of the installation process.

3. Use SSH to run the helper RSA:

```
# ssh -i ~/.ssh/helper_rsa core@192.168.7.20
```

4. Review the logs:

```
journalctl -f
```

5. On the bootstrap node, a temporary etcd and bootkube is created.

   You can monitor these services when they are running by entering the **sudo crictl ps** command.

```
[core@bootstrap ~]$ sudo crictl ps
CONTAINER      IMAGE         CREATED             STATE    NAME                              POD
ID
33762f4a23d7d  976cc3323...  54 seconds ago      Running  manager
29a...
ad6f2453d7a16  86694d2cd...  About a minute ago  Running  kube-apiserver-insecure-readyz
4cd...
3bbdf4176882f  quay.io/...   About a minute ago  Running  kube-scheduler
b3e...
57ad52023300e  quay.io/...   About a minute ago  Running  kube-controller-manager
596...
a1dbe7b8950da  quay.io/...   About a minute ago  Running  kube-apiserver
4cd...
5aa7a59a06feb  quay.io/...   About a minute ago  Running  cluster-version-operator
3ab...
ca45790f4a5f6  099c2a...     About a minute ago  Running  etcd-metrics
081...
e72fb8aaa1606  quay.io/...   About a minute ago  Running  etcd-member
081...
```

```
ca56bbf2708f7  1ac19399...   About a minute ago  Running  machine-config-server
c11...
```

NOTE: Output modified for readability.

6. From the hypervisor, launch the VMs on the master nodes:

```
# virt-install --pxe --network bridge=openshift4 --mac=52:54:00:e7:9d:67 --name ocp4-master0
--ram=40960 --vcpus=8 --os-variant rhel8.0 --disk path=/var/lib/libvirt/images/ocp4-
master0.qcow2,size=250 --vnc
# virt-install --pxe --network bridge=openshift4 --mac=52:54:00:80:16:23 --name ocp4-master1
--ram=40960 --vcpus=8 --os-variant rhel8.0 --disk path=/var/lib/libvirt/images/ocp4-
master1.qcow2,size=250 --vnc
# virt-install --pxe --network bridge=openshift4 --mac=52:54:00:d5:1c:39 --name ocp4-master2
--ram=40960 --vcpus=8 --os-variant rhel8.0 --disk path=/var/lib/libvirt/images/ocp4-
master2.qcow2,size=250 --vnc
```

You can login to the master nodes from the helper node after the master nodes have been provisioned:

```
# ssh -i ~/.ssh/helper_rsa core@192.168.7.21
# ssh -i ~/.ssh/helper_rsa core@192.168.7.22
# ssh -i ~/.ssh/helper_rsa core@192.168.7.23
```

Enter the **sudo crictl ps** at any point to monitor pod creation as the VMs are launching.

Monitor the Installation Process and Delete the Bootstrap Virtual Machine

To monitor the installation process:

1. From the helper node, navigate to the *~/ocp4* directory.
2. Track the install process log:

```
# openshift-install wait-for bootstrap-complete --log-level debug
```

Look for the *DEBUG Bootstrap status: complete* and the *INFO It is now safe to remove the bootstrap resources* messages to confirm that the installation is complete.

```
INFO Waiting up to 30m0s for the Kubernetes API at https://api.ocp4.example.com:6443...
INFO API v1.13.4+838b4fa up
INFO Waiting up to 30m0s for bootstrapping to complete...
DEBUG Bootstrap status: complete
INFO It is now safe to remove the bootstrap resources
```

Do not proceed to the next step until you see these messages.

3. From the hypervisor, delete the bootstrap VM and launch the worker nodes.

> **NOTE**: If you are using physical bare metal servers as worker nodes, skip this step.
> Boot the bare metal servers using PXE instead.

```
# virt-install --pxe --network bridge=openshift4 --mac=52:54:00:f4:26:a1 --name ocp4-worker0
--ram=16384 --vcpus=4 --os-variant rhel8.0 --disk path=/var/lib/libvirt/images/ocp4-
worker0.qcow2,size=120 --vnc

# virt-install --pxe --network bridge=openshift4 --mac=52:54:00:82:90:00 --name ocp4-worker1
--ram=16384 --vcpus=4 --os-variant rhel8.0 --disk path=/var/lib/libvirt/images/ocp4-
worker1.qcow2,size=120 --vnc
```

Finish the Installation

To finish the installation:

1. Login to your Kubernetes cluster:

```
# export KUBECONFIG=/root/ocp4/auth/kubeconfig
```

2. Your installation might be waiting for worker nodes to approve the certificate signing request (CSR). The machineconfig node approval operator typically handles CSR approval.

   CSR approval, however, sometimes has to be performed manually.

To check pending CSRs:

```
# oc get csr
```

To approve all pending CSRs:

```
# oc get csr -o go-template='{{range .items}}{{if not .status}}{{.metadata.name}}{{"\n"}}
{{end}}{{end}}' | xargs oc adm certificate approve
```

You may have to approve all pending CSRs multiple times, depending on the number of worker nodes in your environment and other factors.

To monitor incoming CSRs:

```
# watch -n5 oc get csr
```

Do not move to the next step until incoming CSRs have stopped.

3. Set your cluster management state to *Managed*:

```
# oc patch configs.imageregistry.operator.openshift.io cluster --type merge --patch '{"spec":
{"managementState":"Managed"}}'
```

4. Setup your registry storage.

   For most environments, see Configuring registry storage for bare metal in the Red Hat Openshift documentation.

   For proof of concept labs and other smaller environments, you can set storage to *emptyDir*.

```
# oc patch configs.imageregistry.operator.openshift.io cluster --type merge --patch '{"spec":
{"storage":{"emptyDir":{}}}}'
```

5. If you need to make the registry accessible:

```
# oc patch configs.imageregistry.operator.openshift.io/cluster --type merge -p '{"spec":
{"defaultRoute":true}}'
```

6. Wait for the installation to finish:

```
# openshift-install wait-for install-complete
INFO Waiting up to 30m0s for the cluster at https://api.ocp4.example.com:6443 to initialize...
INFO Waiting up to 10m0s for the openshift-console route to be created...
INFO Install complete!
INFO To access the cluster as the system:admin user when using 'oc', run 'export KUBECONFIG=/
root/ocp4/auth/kubeconfig'
INFO Access the OpenShift web-console here: https://console-openshift-
console.apps.ocp4.example.com
INFO Login to the console with user: kubeadmin, password: XXX-XXXX-XXXX-XXXX
```

7. Add a user to the cluster. See "How to Add a User After Completing the Installation" on page 66.

RELATED DOCUMENTATION

Contrail Networking Supported Platforms

Installing a Standalone Red Hat OpenShift Container Platform 3.11 Cluster with Contrail Using Contrail OpenShift Deployer | 94

Installing a Nested Red Hat OpenShift Container Platform 3.11 Cluster Using Contrail Ansible Deployer | 106

How to Install Contrail Networking and Red Hat OpenShift 4.5 on Amazon Web Services

IN THIS SECTION

Follow these procedures to install Contrail Networking and Red Hat Openshift 4.5 on Amazon Web Services (AWS):

When to Use This Procedure

This procedure is used to install Contrail Networking and Red Hat OpenShift 4.5 orchestration in AWS. Support for Contrail Networking and Red Hat OpenShift 4.5 environments is introduced in Contrail Networking Release 2011. See Contrail Networking Supported Platforms.

Prerequisites

This document makes the following assumptions about your environment:

- the server meets the platform requirements for the Contrail Networking installation. See Contrail Networking Supported Platforms.
- You have the Openshift binary version 4.4.8 files or later. See the Openshift Installation site if you need to update your binary files.
- You can access Openshift image pull secrets. See Using image pull secrets from Red Hat.
- You have an active AWS account.
- AWS CLI is installed. See Installing the AWS CLI from AWS.
- You have an SSH key that you can generate or provide on your local machine during the installation.

Configure DNS

A DNS zone must be created and available in Route 53 for your AWS account before starting this installation. You must also register a domain for your Contrail cluster in AWS Route 53. All entries created in AWS Route 53 are expected to be resolvable from the nodes in the Contrail cluster.

For information on configuring DNS zones in AWS Route 53, see the *Amazon Route 53 Developer Guide* from AWS.

Configure AWS Credentials

The installer used in this procedure creates multiple resources in AWS that are needed to run your cluster. These resources include Elastic Compute Cloud (EC2) instances, Virtual Private Clouds (VPCs), security groups, IAM roles, and other necessary network building blocks.

AWS credentials are needed to access these resources and should be configured before starting this installation.

To configure AWS credentials, see the Configuration and credential file settings section of the AWS Command Line Interface User Guide from AWS.

Download the OpenShift Installer and the Command Line Tools

To download the installer and the command line tools:

1. Check which versions of the OpenShift installer are available:

```
$ curl -s https://mirror.openshift.com/pub/openshift-v4/clients/ocp/ | \
  awk '{print $5}'| \
  grep -o '4.[0-9].[0-9]*' | \
  uniq | \
  sort | \
  column
```

2. Set the version and download the OpenShift installer and the CLI tool.

   In this example output, the Openshift version is 4.5.21.

```
$ VERSION=4.5.21
$ wget https://mirror.openshift.com/pub/openshift-v4/clients/ocp/$VERSION/openshift-install-
mac-$VERSION.tar.gz
$ wget https://mirror.openshift.com/pub/openshift-v4/clients/ocp/$VERSION/openshift-client-
mac-$VERSION.tar.gz

$ tar -xvzf openshift-install-mac-${VERSION}.tar.gz -C /usr/local/bin
$ tar -xvzf openshift-client-mac-${VERSION}.tar.gz -C /usr/local/bin

$ openshift-install version
$ oc version
$ kubectl version
```

Deploy the Cluster

To deploy the cluster:

1. Generate an SSH private key and add it to the agent:

```
$ ssh-keygen -b 4096 -t rsa -f ~/.ssh/id_rsa -N ""
```

2. Create a working folder:

In this example, a working folder named *aws-ocp4* is created and the user is then moved into the new directory.

```
$ mkdir ~/aws-ocp4 ; cd ~/aws-ocp4
```

3. Create an installation configuration file. See Creating the installation configuration file section of the Installing a cluster on AWS with customizations document from Red Hat OpenShift.

```
$ openshift-install create install-config
```

An *install-config.yaml* file needs to be created and added to the current directory. A sample *install-config.yaml* file is provided below.

Be aware of the following factors while creating the *install-config.yaml* file:

- The *networkType* field is usually set as *OpenShiftSDN* in the YAML file by default.

  For configuration pointing at Contrail cluster nodes, the *networkType* field needs to be configured as *Contrail*.

- OpenShift master nodes need larger instances. We recommend setting the type to *m5.2xlarge* or larger for OpenShift nodes.

- Most OpenShift worker nodes can use the default instance sizes. You should consider using larger instances, however, for high demand performance workloads.

- Many of the installation parameters in the YAML file are described in more detail in the Installation configuration parameters section of the Installing a cluster on AWS with customizations document from Red Hat OpenShift.

A sample *install-config.yaml* file:

```
apiVersion: v1
baseDomain: ovsandbox.com
compute:
- architecture: amd64
  hyperthreading: Enabled
  name: worker
  platform:
    aws:
      rootVolume:
        iops: 2000
        size: 500
```

```
        type: io1
      type: m5.4xlarge
  replicas: 3
controlPlane:
  architecture: amd64
  hyperthreading: Enabled
  name: master
  platform:
    aws:
      rootVolume:
        iops: 4000
        size: 500
        type: io1
      type: m5.2xlarge
  replicas: 3
metadata:
  creationTimestamp: null
  name: w1
networking:
  clusterNetwork:
  - cidr: 10.128.0.0/14
    hostPrefix: 23
  machineNetwork:
  - cidr: 10.0.0.0/16
  networkType: Contrail
  serviceNetwork:
  - 172.30.0.0/16
platform:
  aws:
    region: eu-west-1
publish: External
pullSecret: '{"auths"...}'
sshKey: |
  ssh-rsa ...
```

4. Create the installation manifests:

```
# openshift-install create manifests
```

5. Install the YAML files to apply the Contrail configuration:

```
bash <<EOF
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/openshift/
manifests/00-contrail-01-namespace.yaml -o manifests/00-contrail-01-namespace.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/openshift/
manifests/00-contrail-02-admin-password.yaml -o manifests/00-contrail-02-admin-
password.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/openshift/
manifests/00-contrail-02-rbac-auth.yaml -o manifests/00-contrail-02-rbac-auth.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/openshift/
manifests/00-contrail-02-registry-secret.yaml -o manifests/00-contrail-02-registry-
secret.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/openshift/
manifests/00-contrail-03-cluster-role.yaml -o manifests/00-contrail-03-cluster-role.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/openshift/
manifests/00-contrail-04-serviceaccount.yaml -o manifests/00-contrail-04-
serviceaccount.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/openshift/
manifests/00-contrail-05-rolebinding.yaml -o manifests/00-contrail-05-rolebinding.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/openshift/
manifests/00-contrail-06-clusterrolebinding.yaml -o manifests/00-contrail-06-
clusterrolebinding.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/crds/
contrail.juniper.net_cassandras_crd.yaml -o manifests/00-contrail-07-
contrail.juniper.net_cassandras_crd.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/crds/
contrail.juniper.net_commands_crd.yaml -o manifests/00-contrail-07-
contrail.juniper.net_commands_crd.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/crds/
contrail.juniper.net_configs_crd.yaml -o manifests/00-contrail-07-
contrail.juniper.net_configs_crd.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/crds/
contrail.juniper.net_contrailcnis_crd.yaml -o manifests/00-contrail-07-
contrail.juniper.net_contrailcnis_crd.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/crds/
contrail.juniper.net_fernetkeymanagers_crd.yaml -o manifests/00-contrail-07-
contrail.juniper.net_fernetkeymanagers_crd.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/crds/
contrail.juniper.net_contrailmonitors_crd.yaml -o manifests/00-contrail-07-
contrail.juniper.net_contrailmonitors_crd.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/crds/
```

```
contrail.juniper.net_contrailstatusmonitors_crd.yaml -o manifests/00-contrail-07-
contrail.juniper.net_contrailstatusmonitors_crd.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/crds/
contrail.juniper.net_controls_crd.yaml -o manifests/00-contrail-07-
contrail.juniper.net_controls_crd.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/crds/
contrail.juniper.net_keystones_crd.yaml -o manifests/00-contrail-07-
contrail.juniper.net_keystones_crd.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/crds/
contrail.juniper.net_kubemanagers_crd.yaml -o manifests/00-contrail-07-
contrail.juniper.net_kubemanagers_crd.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/crds/
contrail.juniper.net_managers_crd.yaml -o manifests/00-contrail-07-
contrail.juniper.net_managers_crd.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/crds/
contrail.juniper.net_memcacheds_crd.yaml -o manifests/00-contrail-07-
contrail.juniper.net_memcacheds_crd.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/crds/
contrail.juniper.net_postgres_crd.yaml -o manifests/00-contrail-07-
contrail.juniper.net_postgres_crd.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/crds/
contrail.juniper.net_provisionmanagers_crd.yaml -o manifests/00-contrail-07-
contrail.juniper.net_provisionmanagers_crd.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/crds/
contrail.juniper.net_rabbitmqs_crd.yaml -o manifests/00-contrail-07-
contrail.juniper.net_rabbitmqs_crd.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/crds/
contrail.juniper.net_swiftproxies_crd.yaml -o manifests/00-contrail-07-
contrail.juniper.net_swiftproxies_crd.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/crds/
contrail.juniper.net_swifts_crd.yaml -o manifests/00-contrail-07-
contrail.juniper.net_swifts_crd.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/crds/
contrail.juniper.net_swiftstorages_crd.yaml -o manifests/00-contrail-07-
contrail.juniper.net_swiftstorages_crd.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/crds/
contrail.juniper.net_vrouters_crd.yaml -o manifests/00-contrail-07-
contrail.juniper.net_vrouters_crd.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/crds/
contrail.juniper.net_webuis_crd.yaml -o manifests/00-contrail-07-
contrail.juniper.net_webuis_crd.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/crds/
contrail.juniper.net_zookeepers_crd.yaml -o manifests/00-contrail-07-
```

```
contrail.juniper.net_zookeepers_crd.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/openshift/
releases/R2011/manifests/00-contrail-08-operator.yaml -o manifests/00-contrail-08-
operator.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/openshift/
releases/R2011/manifests/00-contrail-09-manager.yaml -o manifests/00-contrail-09-
manager.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/openshift/
manifests/cluster-network-02-config.yml -o manifests/cluster-network-02-config.yml
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/openshift/
openshift/99_master-iptables-machine-config.yaml -o openshift/99_master-iptables-machine-
config.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/openshift/
openshift/99_master-kernel-modules-overlay.yaml -o openshift/99_master-kernel-modules-
overlay.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/openshift/
openshift/99_master_network_functions.yaml -o openshift/99_master_network_functions.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/openshift/
openshift/99_master_network_manager_stop_service.yaml -o openshift/
99_master_network_manager_stop_service.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/openshift/
openshift/99_master-pv-mounts.yaml -o openshift/99_master-pv-mounts.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/openshift/
openshift/99_worker-iptables-machine-config.yaml -o openshift/99_worker-iptables-machine-
config.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/openshift/
openshift/99_worker-kernel-modules-overlay.yaml -o openshift/99_worker-kernel-modules-
overlay.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/openshift/
openshift/99_worker_network_functions.yaml -o openshift/99_worker_network_functions.yaml;\
curl https://raw.githubusercontent.com/Juniper/contrail-operator/R2011/deploy/openshift/
openshift/99_worker_network_manager_stop_service.yaml -o openshift/
99_worker_network_manager_stop_service.yaml;
EOF
```

6. Modify the YAML files for your environment.

   The scope of each potential configuration changes is beyond the scope of this document.

   Common configuration changes include:

   - Modify the 00-contrail-02-registry-secret.yaml file to providing proper configuration with credentials to a registry. The most commonly used registry is the Contrail repository at hub.juniper.net.

> **NOTE**: You can create a base64 encoded value for configuration with the script provided in this directory. If you want to use this value for security, copy the output of the script and paste it into the Contrail registry secret configuration by replacing the *DOCKER_CONFIG* variable with the generated base64 encoded value string.

- If you are using non-default network-CIDR subnets for your pods or services, open the *deploy/openshift/manifests/cluster-network-02-config.yml* file and update the CIDR values.
- The default number of master nodes in a Kubernetes cluster is 3. If you are using a different number of master nodes, modify the *deploy/openshift/manifests/00-contrail-09-manager.yaml* file and set the spec.commonConfiguration.replicas field to the number of master nodes.

7. Create the cluster:

```
$ openshift-install create cluster --log-level=debug
```

- Contrail Networking needs to open some networking ports for operation within AWS. These ports are opened by adding rules to security groups.

  Follow this procedure to add rules to security groups when AWS resources are manually created:

  a. Build the Contrail CLI tool for managing security group ports on AWS. This tool allows you to automatically open ports that are required for Contrail to manage security group ports on AWS that are attached to Contrail cluster resources.

     To build this tool:

     ```
     go build .
     ```

     After entering this command, you should be in the binary contrail-sc-open in your directory. This interface is the compiled tool.

  b. Start the tool:

     ```
     ./contrail-sc-open -cluster-name name of your Openshift cluster -region AWS region where cluster is located
     ```

c. Verify that the service has been created:

```
oc -n openshift-ingress get service router-default
```

Proceed to the next step after confirming the service was created.

8. When the service router-default is created in openshift-ingress, use the following command to patch the configuration:

```
$ oc -n openshift-ingress patch service router-default --patch '{"spec":
{"externalTrafficPolicy": "Cluster"}}'
```

9. Monitor the screen messages.

   Look for the *INFO Install complete!*.

   The final messages from a sample successful installation:

```
INFO Waiting up to 10m0s for the openshift-console route to be created...
DEBUG Route found in openshift-console namespace: console
DEBUG Route found in openshift-console namespace: downloads
DEBUG OpenShift console route is created
INFO Install complete!
INFO To access the cluster as the system:admin user when using 'oc', run 'export
KUBECONFIG=/Users/ovaleanu/aws1-ocp4/auth/kubeconfig'
INFO Access the OpenShift web-console here: https://console-openshift-
console.apps.w1.ovsandbox.com
INFO Login to the console with user: kubeadmin, password: XXXxx-XxxXX-xxXXX-XxxxX
```

10. Access the cluster:

```
$ export KUBECONFIG=~/aws-ocp4/auth/kubeconfig
```

11. Add a user to the cluster. See "How to Add a User After Completing the Installation" on page 66.

How to Add a User After Completing the Installation

The process for adding an Openshift user is identical in KVM or on AWS.

Redhat OpenShift 4.5 supports a single kubeadmin user by default. This kubeadmin user is used to deploy the initial cluster configuration.

You can use this procedure to create a Custom Resource (CR) to define a HTTPasswd identity provider.

1. Generate a flat file that contains the user names and passwords for your cluster by using the HTPasswd identity provider:

```
$ htpasswd -c -B -b users.htpasswd testuser MyPassword
```

   A file called users.httpasswd is created.

2. Define a secret password that contains the HTPasswd user file:

```
$ oc create secret generic htpass-secret --from-file=htpasswd=/root/ocp4/users.htpasswd -n
openshift-config
```

   This custom resource shows the parameters and acceptable values for an HTPasswd identity provider.

```
$ cat htpasswdCR.yaml
apiVersion: config.openshift.io/v1
kind: OAuth
metadata:
  name: cluster
spec:
  identityProviders:
  - name: testuser
    mappingMethod: claim
    type: HTPasswd
    htpasswd:
      fileData:
        name: htpass-secret
```

3. Apply the defined custom resource:

```
$ oc create -f htpasswdCR.yaml
```

4. Add the user and assign the *cluster-admin* role:

```
$ oc adm policy add-cluster-role-to-user cluster-admin testuser
```

5. Login using the new user credentials:

```
oc login -u testuser
Authentication required for https://api.ocp4.example.com:6443 (openshift)
Username: testuser
Password:
Login successful.
```

The kubeadmin user can now safely be removed. See the Removing the kubeadmin user document from Red Hat OpenShift.

How to Install Earlier Releases of Contrail Networking and Red Hat OpenShift

If you have a need to install Contrail Networking with earlier versions of Red Hat Openshift, Contrail Networking is also supported with Red Hat Openshift versions 4.4 and 3.11.

For information on installing Contrail Networking with Red Hat Openshift 4.4, see "How to Install Contrail Networking and Red Hat OpenShift 4.4" on page 69.

For information on installing Contrail Networking with Red Hat Openshift 3.11, see the following documentation:

- "Installing a Standalone Red Hat OpenShift Container Platform 3.11 Cluster with Contrail Using Contrail OpenShift Deployer" on page 94
- "Installing a Nested Red Hat OpenShift Container Platform 3.11 Cluster Using Contrail Ansible Deployer" on page 106

How to Install Contrail Networking and Red Hat OpenShift 4.4

IN THIS SECTION

NOTE: This topic covers Contrail Networking in Red Hat Openshift environments that are using Contrail Networking Release 21-based releases.

Starting in Release 22.1, Contrail Networking evolved into Cloud-Native Contrail Networking. Cloud-Native Contrail offers significant enhancements to optimize networking performance in Kubernetes-orchestrated environments. Cloud-Native Contrail supports Red Hat Openshift and we strongly recommend using Cloud-Native Contrail for networking in environments using Red Hat Openshift.

For general information about Cloud-Native Contrail, see the Cloud-Native Contrail Networking Techlibrary homepage.

You can install Contrail Networking with Red Hat Openshift 4.4 in multiple environments.

This document shows one method of installing Red Hat Openshift 4.4 with Contrail Networking in two separate contexts—on a VM running in a KVM module and within Amazon Web Services (AWS). There are many implementation and configuration options available for installing and configuring Red Hat OpenShift 4.4 and the scope of all options is beyond this document. For additional information on Red Hat Openshift 4.4 implementation options, see the OpenShift Container Platform 4.4 Documentation from Red Hat.

This document includes the following sections:

How to Install Contrail Networking and Red Hat OpenShift 4.4 using a VM Running in a KVM Module

IN THIS SECTION

This section illustrates how to install Contrail Networking with Red Hat OpenShift 4.4 orchestration, where Contrail Networking and Red Hat Openshift are running on virtual machines (VMs) in a Kernel-based Virtual Machine (KVM) module. This procedure can also be performed to configure an environment where Contrail Networking and Red Hat OpenShift 4.4 are running on a bare metal server.

When to Use This Procedure

This procedure is used to install Contrail Networking and Red Hat OpenShift 4.4 orchestration on a virtual machine (VM) running in a Kernel-based Virtual Machine (KVM) module. Support for Contrail Networking installations onto VMs in Red Hat OpenShift 4.4 environments is introduced in Contrail Networking Release 2008. See Contrail Networking Supported Platforms.

You can also use this procedure to install Contrail Networking and Red Hat OpenShift 4.4 orchestration on a bare metal server.

This procedure should work with all versions of Openshift 4.4.

Prerequisites

This document makes the following assumptions about your environment:

- the KVM environment is operational.
- the server meets the platform requirements for the installation. See Contrail Networking Supported Platforms.
- Minimum server requirements:
  - Primary nodes: 8 CPU, 40GB RAM, 250GB SSD storage
  - Backup nodes: 4 CPU, 16GB RAM, 120GB SSD storage
  - Helper node: 4 CPU, 8GB RAM, 30GB SSD storage

- In single node deployments, do not use spinning disk arrays with low Input/Output Operations Per Second (IOPS) when using Contrail Networking with Red Hat Openshift. Higher IOPS disk arrays are required because the control plane always operates as a high availability setup in single node deployments.

  IOPS requirements vary by environment due to multiple factors beyond Contrail Networking and Red Hat Openshift. We, therefore, provide this guideline but do not provide direct guidance around IOPS requirements.

Install Contrail Networking and Red Hat Openshift 4.4

IN THIS SECTION

Perform these steps to install Contrail Networking and Red Hat OpenShift 4.4 using a VM running in a KVM module:

Create a Virtual Network or a Bridge Network for the Installation

To create a virtual network or a bridge network for the installation:

1. Log onto the server that will host the VM that will run Contrail Networking.

   Download the *virt-net.xml* virtual network configuration file from the Red Hat repository.

   ```
   # wget https://raw.githubusercontent.com/RedHatOfficial/ocp4-helpernode/master/docs/examples/
   virt-net.xml
   ```

2. Create a virtual network using the *virt-net.xml* file.

   You may need to modify your virtual network for your environment.

Example:

```
# virsh net-define --file virt-net.xml
```

3. Set the OpenShift 4.4 virtual network to autostart on bootup:

```
# virsh net-autostart openshift4
# virsh net-start openshift4
```

Create a Helper Node with a Virtual Machine Running CentOS 7 or 8

This procedure requires a helper node with a virtual machine that is running either CentOS 7 or 8.

To create this helper node:

1. Download the Kickstart file for the helper node from the Red Hat repository:
   CentOS 8

```
# wget https://raw.githubusercontent.com/RedHatOfficial/ocp4-helpernode/master/docs/examples/
helper-ks8.cfg -O helper-ks.cfg
```

   CentOS 7

```
# wget https://raw.githubusercontent.com/RedHatOfficial/ocp4-helpernode/master/docs/examples/
helper-ks.cfg -O helper-ks.cfg
```

2. If you haven't already configured a root password and the NTP server on the helper node, enter the following commands:
   Example Root Password

```
rootpw --plaintext password
```

   Example NTP Configuration

```
timezone America/Los_Angeles --isUtc --
ntpservers=0.centos.pool.ntp.org,1.centos.pool.ntp.org,2.centos.pool.ntp.org,3.centos.pool.ntp
.org
```

3. Edit the *helper-ks.cfg* file for your environment and use it to install the helper node.

The following examples show how to install the helper node without having to take further actions:

CentOS 8

```
# virt-install --name="ocp4-aHelper" --vcpus=2 --ram=4096 \
--disk path=/var/lib/libvirt/images/ocp4-aHelper.qcow2,bus=virtio,size=50 \
--os-variant centos8 --network network=openshift4,model=virtio \
--boot hd,menu=on --location /var/lib/libvirt/iso/CentOS-8.2.2004-x86_64-dvd1.iso \
--initrd-inject helper-ks.cfg --extra-args "inst.ks=file:/helper-ks.cfg" --noautoconsole
```

CentOS 7

```
# virt-install --name="ocp4-aHelper" --vcpus=2 --ram=4096 \
--disk path=/var/lib/libvirt/images/ocp4-aHelper.qcow2,bus=virtio,size=30 \
--os-variant centos7.0 --network network=openshift4,model=virtio \
--boot hd,menu=on --location /var/lib/libvirt/iso/CentOS-7-x86_64-Minimal-2003.iso \
--initrd-inject helper-ks.cfg --extra-args "inst.ks=file:/helper-ks.cfg" --noautoconsole
```

The helper node is installed with the following settings, which are pulled from the *virt-net.xml* file:

- **HELPER_IP**: 192.168.7.77
- **NetMask**: 255.255.255.0
- **Default Gateway**: 192.168.7.1
- **DNS Server**: 8.8.8.8

4. Monitor the helper node installation progress in the viewer:

```
# virt-viewer --domain-name ocp4-aHelper
```

When the installation process is complete, the helper node shuts off.

5. Start the helper node:

```
# virsh start ocp4-aHelper
```

Prepare the Helper Node

To prepare the helper node after the helper node installation:

1. Login to the helper node:

```
# ssh -l root HELPER_IP
```

> **NOTE**: The default *HELPER_IP*, which was pulled from the *virt-net.xml* file, is 192.168.7.77.

2. Install Enterprise Linux and update CentOS.

```
# yum -y install https://dl.fedoraproject.org/pub/epel/epel-release-latest-$(rpm -E
%rhel).noarch.rpm
# yum -y update
```

3. Install Ansible and Git and clone the *helpernode* repository onto the helper node.

```
# yum -y install ansible git
# git clone https://github.com/RedHatOfficial/ocp4-helpernode
# cd ocp4-helpernode
```

4. Copy the vars.yaml file into the top-level directory:

```
# cp docs/examples/vars.yaml .
```

Review the vars.yml file. Consider changing any value that requires changing in your environment.

The following values should be reviewed especially carefully:

- The domain name, which is defined using the *domain:* parameter in the *dns:* hierarchy. If you are using local DNS servers, modify the forwarder parameters—*forwarder1:* and *forwarder2:* are used in this example—to connect to these DNS servers.
- Hostnames for primary and worker nodes. Hostnames are defined using the *name:* parameter in either the *primaries:* or *workers:* hierarchies.
- IP and DHCP settings. If you are using a custom bridge network, modify the IP and DHCP settings accordingly.
- VM and BMS settings.

  If you are using a VM, set the *disk:* parameter as *disk: vda*.

  If you are using a BMS, set the *disk:* parameter as *disk: sda*.

A sample vars.yml file:

```
disk: vda
helper:
  name: "helper"
  ipaddr: "192.168.7.77"
dns:
  domain: "example.com"
  clusterid: "ocp4"
  forwarder1: "8.8.8.8"
  forwarder2: "8.8.4.4"
dhcp:
  router: "192.168.7.1"
  bcast: "192.168.7.255"
  netmask: "255.255.255.0"
  poolstart: "192.168.7.10"
  poolend: "192.168.7.30"
  ipid: "192.168.7.0"
  netmaskid: "255.255.255.0"
bootstrap:
  name: "bootstrap"
  ipaddr: "192.168.7.20"
  macaddr: "52:54:00:60:72:67"
masters:
  - name: "master0"
    ipaddr: "192.168.7.21"
    macaddr: "52:54:00:e7:9d:67"
  - name: "master1"
    ipaddr: "192.168.7.22"
    macaddr: "52:54:00:80:16:23"
  - name: "master2"
    ipaddr: "192.168.7.23"
    macaddr: "52:54:00:d5:1c:39"
workers:
  - name: "worker0"
    ipaddr: "192.168.7.11"
    macaddr: "52:54:00:f4:26:a1"
  - name: "worker1"
    ipaddr: "192.168.7.12"
    macaddr: "52:54:00:82:90:00"
```

5. Review the *vars/main.yml* file to ensure the file reflects the correct version of Red Hat OpenShift. If you need to change the Red Hat Openshift version in the file, change it.

   In the following sample *main.yml* file, Red Hat Openshift 4.4.21 is installed:

```
ssh_gen_key: true
install_filetranspiler: false
staticips: false
force_ocp_download: false
ocp_bios: "https://mirror.openshift.com/pub/openshift-v4/dependencies/rhcos/4.4/latest/
rhcos-4.4.17-x86_64-metal.x86_64.raw.gz"
ocp_initramfs: "https://mirror.openshift.com/pub/openshift-v4/dependencies/rhcos/4.4/latest/
rhcos-4.4.17-x86_64-installer-initramfs.x86_64.img"
ocp_install_kernel: "https://mirror.openshift.com/pub/openshift-v4/dependencies/rhcos/4.4/
latest/rhcos-4.4.17-x86_64-installer-kernel-x86_64"
ocp_client: "https://mirror.openshift.com/pub/openshift-v4/clients/ocp/stable-4.4/openshift-
client-linux.tar.gz"
ocp_installer: "https://mirror.openshift.com/pub/openshift-v4/clients/ocp/stable-4.4/
openshift-install-linux.tar.gz"
helm_source: "https://get.helm.sh/helm-v3.2.4-linux-amd64.tar.gz"
chars: (\\_|\\$|\\|\\/|\\=|\\)|\\(|\\&|\\^|\\%|\\$|\\#|\\@|\\!|\\*)
ppc64le: false
chronyconfig:
  enabled: false
setup_registry:
  deploy: false
  autosync_registry: false
  registry_image: docker.io/library/registry:2
  local_repo: "ocp4/openshift4"
  product_repo: "openshift-release-dev"
  release_name: "ocp-release"
  release_tag: "4.4.21-x86_64"
```

6. Run the playbook to setup the helper node:

```
# ansible-playbook -e @vars.yaml tasks/main.yml
```

7. After the playbook is run, gather information about your environment and confirm that all services are active and running:

```
# /usr/local/bin/helpernodecheck services
Status of services:
```

```
==================
Status of dhcpd svc         ->    Active: active (running) since Mon 2020-09-28 05:40:10 EDT;
33min ago
Status of named svc         ->    Active: active (running) since Mon 2020-09-28 05:40:08 EDT;
33min ago
Status of haproxy svc     ->    Active: active (running) since Mon 2020-09-28 05:40:08 EDT;
33min ago
Status of httpd svc         ->    Active: active (running) since Mon 2020-09-28 05:40:10 EDT;
33min ago
Status of tftp svc          ->    Active: active (running) since Mon 2020-09-28 06:13:34 EDT;
1s ago
Unit local-registry.service could not be found.
Status of local-registry svc         ->
```

Create the Ignition Configurations

To create Ignition configurations:

1. On your hypervisor and helper nodes, check that your NTP server is properly configured in the */etc/chrony.conf* file:

   ```
   chronyc tracking
   ```

   The installation fails with a *X509: certificate has expired or is not yet valid* message when NTP is not properly configured.
2. Create a location to store your pull secret objects:

   ```
   # mkdir -p ~/.openshift
   ```
3. From Get Started with Openshift website, download your pull secret and save it in the *~/.openshift/pull-secret* directory.

   ```
   # ls -1 ~/.openshift/pull-secret
   /root/.openshift/pull-secret
   ```

4. An SSH key is created for you in the *~/.ssh/helper_rsa* directory after completing the previous step. You can use this key or create a unique key for authentication.

```
# ls -1 ~/.ssh/helper_rsa
/root/.ssh/helper_rsa
```

5. Create an installation directory.

```
# mkdir ~/ocp4
# cd ~/ocp4
```

6. Create an install-config.yaml file.

   An example file:

```
# cat <<EOF > install-config.yaml
apiVersion: v1
baseDomain: example.com
compute:
- hyperthreading: Disabled
  name: worker
  replicas: 0
controlPlane:
  hyperthreading: Disabled
  name: master
  replicas: 3
metadata:
  name: ocp4
networking:
  clusterNetworks:
  - cidr: 10.254.0.0/16
    hostPrefix: 24
  networkType: Contrail
  serviceNetwork:
  - 172.30.0.0/16
platform:
  none: {}
pullSecret: '$(< ~/.openshift/pull-secret)'
sshKey: '$(< ~/.ssh/helper_rsa.pub)'
EOF
```

7. Create the installation manifests:

```
# openshift-install create manifests
```

8. Set the **mastersSchedulable:** variable to **false** in the *manifests/cluster-scheduler-02-config.yml* file.

```
# sed -i 's/mastersSchedulable: true/mastersSchedulable: false/g' manifests/cluster-
scheduler-02-config.yml
```

A sample **cluster-scheduler-02-config.yml** file after this configuration change:

```
# cat manifests/cluster-scheduler-02-config.yml
apiVersion: config.openshift.io/v1
kind: Scheduler
metadata:
  creationTimestamp: null
  name: cluster
spec:
  mastersSchedulable: false
  policy:
    name: ""
status: {}
```

This configuration change is needed to prevent pods from being scheduled on control plane machines.

9. Clone the contrail operator repository:

```
# git clone https://github.com/Juniper/contrail-operator.git
# git checkout R2008
```

10. Create the Contrail operator configuration file.

Example:

```
# cat <<EOF > config_contrail_operator.yaml
CONTRAIL_VERSION=2008.121
CONTRAIL_REGISTRY=hub.juniper.net/contrail
DOCKER_CONFIG=<this_needs_to_be_generated>
EOF
```

where:

- *CONTRAIL_VERSION* is the Contrail Networking container tag of the version of Contrail Networking that you are downloading.

  This procedure is initially supported in Contrail Networking Release 2008. You can obtain the Contrail Networking container tags for all Contrail Networking 20 releases in README Access to Contrail Networking Registry 20XX.

- *CONTRAIL_REGISTRY* is the path to the container registry. The default Juniper Contrail Container Registry contains the files needed for this installation and is located at *hub.juniper.net/contrail*.

  If needed, email mailto:contrail-registry@juniper.net to obtain your username and password credentials to access the Contrail Container registry.

- *DOCKER_CONFIG* is the registry secret credential. Set the *DOCKER_CONFIG* to registry secret with proper data in base64.

  NOTE: You can create base64 encoded values using a script. See DOCKER_CONFIG generate.
  To start the script:

  ```
  # ./contrail-operator/deploy/openshift/tools/docker-config-generate/generate-docker-config.sh
  ```

  You can copy output generated from the script and use it as the *DOCKER_CONFIG* value in this file.

11. Install Contrail manifests:

    ```
    # ./contrail-operator/deploy/openshift/install-manifests.sh --dir ./ --config ./config_contrail_operator.yaml
    ```

12. If your environment has to use a specific NTP server, set the environment using the steps in the Openshift 4.x Chrony Configuration document.
13. Generate the Ignition configurations:

    ```
    # openshift-install create ignition-configs
    ```

14. Copy the Ignition files in the Ignition directory for the webserver:

```
# cp ~/ocp4/*.ign /var/www/html/ignition/
# restorecon -vR /var/www/html/
# restorecon -vR /var/lib/tftpboot/
# chmod o+r /var/www/html/ignition/*.ign
```

Launch the Virtual Machines

To launch the virtual machines:

1. From the hypervisor, use PXE booting to launch the virtual machine or machines. If you are using a bare metal server, use PXE booting to boot the servers.
2. Launch the bootstrap virtual machine:

```
# virt-install --pxe --network bridge=openshift4 --mac=52:54:00:60:72:67 --name ocp4-
bootstrap --ram=8192 --vcpus=4 --os-variant rhel8.0 --disk path=/var/lib/libvirt/images/ocp4-
bootstrap.qcow2,size=120 --vnc
```

   The following actions occur as a result of this step:

   - a bootstrap node virtual machine is created.
   - the bootstrap node VM is connected to the PXE server. The PXE server is our helper node.
   - an IP address is assigned from DHCP.
   - A Red Hat Enterprise Linux CoreOS (RHCOS) image is downloaded from the HTTP server.

   The ignition file is embedded at the end of the installation process.
3. Use SSH to run the helper RSA:

```
# ssh -i ~/.ssh/helper_rsa core@192.168.7.20
```

4. Review the logs:

```
journalctl -f
```

5. On the bootstrap node, a temporary etcd and bootkube is created.

You can monitor these services when they are running by entering the **sudo crictl ps** command.

```
[core@bootstrap ~]$ sudo crictl ps
CONTAINER      IMAGE         CREATED             STATE    NAME                              POD
ID
33762f4a23d7d  976cc3323...  54 seconds ago      Running  manager
29a...
ad6f2453d7a16  86694d2cd...  About a minute ago  Running  kube-apiserver-insecure-readyz
4cd...
3bbdf4176882f  quay.io/...   About a minute ago  Running  kube-scheduler
b3e...
57ad52023300e  quay.io/...   About a minute ago  Running  kube-controller-manager
596...
a1dbe7b8950da  quay.io/...   About a minute ago  Running  kube-apiserver
4cd...
5aa7a59a06feb  quay.io/...   About a minute ago  Running  cluster-version-operator
3ab...
ca45790f4a5f6  099c2a...     About a minute ago  Running  etcd-metrics
081...
e72fb8aaa1606  quay.io/...   About a minute ago  Running  etcd-member
081...
ca56bbf2708f7  1ac19399...   About a minute ago  Running  machine-config-server
c11...
```

NOTE: Output modified for readability.

6. From the hypervisor, launch the VMs on the primary nodes:

```
# virt-install --pxe --network bridge=openshift4 --mac=52:54:00:e7:9d:67 --name ocp4-master0
--ram=40960 --vcpus=8 --os-variant rhel8.0 --disk path=/var/lib/libvirt/images/ocp4-
master0.qcow2,size=250 --vnc
# virt-install --pxe --network bridge=openshift4 --mac=52:54:00:80:16:23 --name ocp4-master1
--ram=40960 --vcpus=8 --os-variant rhel8.0 --disk path=/var/lib/libvirt/images/ocp4-
master1.qcow2,size=250 --vnc
# virt-install --pxe --network bridge=openshift4 --mac=52:54:00:d5:1c:39 --name ocp4-master2
--ram=40960 --vcpus=8 --os-variant rhel8.0 --disk path=/var/lib/libvirt/images/ocp4-
master2.qcow2,size=250 --vnc
```

You can login to the primary nodes from the helper node after the primary nodes have been provisioned:

```
# ssh -i ~/.ssh/helper_rsa core@192.168.7.21
# ssh -i ~/.ssh/helper_rsa core@192.168.7.22
# ssh -i ~/.ssh/helper_rsa core@192.168.7.23
```

Enter the **sudo crictl ps** at any point to monitor pod creation as the VMs are launching.

Monitor the Installation Process and Delete the Bootstrap Virtual Machine

To monitor the installation process:

1. From the helper node, navigate to the *~/ocp4* directory.
2. Track the install process log:

```
# openshift-install wait-for bootstrap-complete --log-level debug
```

Look for the *DEBUG Bootstrap status: complete* and the *INFO It is now safe to remove the bootstrap resources* messages to confirm that the installation is complete.

```
INFO Waiting up to 30m0s for the Kubernetes API at https://api.ocp4.example.com:6443...
INFO API v1.13.4+838b4fa up
INFO Waiting up to 30m0s for bootstrapping to complete...
DEBUG Bootstrap status: complete
INFO It is now safe to remove the bootstrap resources
```

Do not proceed to the next step until you see these messages.

3. From the hypervisor, delete the bootstrap VM and launch the worker nodes.

```
# virt-install --pxe --network bridge=openshift4 --mac=52:54:00:f4:26:a1 --name ocp4-worker0
--ram=16384 --vcpus=4 --os-variant rhel8.0 --disk path=/var/lib/libvirt/images/ocp4-
worker0.qcow2,size=120 --vnc

# virt-install --pxe --network bridge=openshift4 --mac=52:54:00:82:90:00 --name ocp4-worker1
--ram=16384 --vcpus=4 --os-variant rhel8.0 --disk path=/var/lib/libvirt/images/ocp4-
worker1.qcow2,size=120 --vnc
```

Finish the Installation

To finish the installation:

1. Login to your Kubernetes cluster:

```
# export KUBECONFIG=/root/ocp4/auth/kubeconfig
```

2. Your installation might be waiting for worker nodes to approve the certificate signing request (CSR). The machineconfig node approval operator typically handles CSR approval.

   CSR approval, however, sometimes has to be performed manually.

   To check pending CSRs:

```
# oc get csr
```

   To approve all pending CSRs:

```
# oc get csr -o go-template='{{range .items}}{{if not .status}}{{.metadata.name}}{{"\n"}}
{{end}}{{end}}' | xargs oc adm certificate approve
```

   You may have to approve all pending CSRs multiple times, depending on the number of worker nodes in your environment and other factors.

   To monitor incoming CSRs:

```
# watch -n5 oc get csr
```

   Do not move to the next step until incoming CSRs have stopped.

3. Set your cluster management state to *Managed*:

```
# oc patch configs.imageregistry.operator.openshift.io cluster --type merge --patch '{"spec":
{"managementState":"Managed"}}'
```

4. Setup your registry storage.

   For most environments, see Configuring registry storage for bare metal in the Red Hat Openshift documentation.

For proof of concept labs and other smaller environments, you can set storage to *emptyDir*.

```
# oc patch configs.imageregistry.operator.openshift.io cluster --type merge --patch '{"spec":
{"storage":{"emptyDir":{}}}}'
```

5. If you need to make the registry accessible:

```
# oc patch configs.imageregistry.operator.openshift.io/cluster --type merge -p '{"spec":
{"defaultRoute":true}}'
```

6. Wait for the installation to finish:

```
# openshift-install wait-for install-complete
INFO Waiting up to 30m0s for the cluster at https://api.ocp4.example.com:6443 to initialize...
INFO Waiting up to 10m0s for the openshift-console route to be created...
INFO Install complete!
INFO To access the cluster as the system:admin user when using 'oc', run 'export KUBECONFIG=/
root/ocp4/auth/kubeconfig'
INFO Access the OpenShift web-console here: https://console-openshift-
console.apps.ocp4.example.com
INFO Login to the console with user: kubeadmin, password: XXX-XXXX-XXXX-XXXX
```

7. Add a user to the cluster. See "How to Add a User After Completing the Installation" on page 92.

RELATED DOCUMENTATION

Contrail Networking Supported Platforms

Installing a Standalone Red Hat OpenShift Container Platform 3.11 Cluster with Contrail Using Contrail OpenShift Deployer | 94

Installing a Nested Red Hat OpenShift Container Platform 3.11 Cluster Using Contrail Ansible Deployer | 106

How to Install Contrail Networking and Red Hat OpenShift 4.4 on Amazon Web Services

IN THIS SECTION

Follow these procedures to install Contrail Networking and Red Hat Openshift 4.4 on Amazon Web Services (AWS):

When to Use This Procedure

This procedure is used to install Contrail Networking and Red Hat OpenShift 4.4 orchestration in AWS. Support for Contrail Networking and Red Hat OpenShift 4.4 environments is introduced in Contrail Networking Release 2008. See Contrail Networking Supported Platforms.

Prerequisites

This document makes the following assumptions about your environment:

- the server meets the platform requirements for the installation. See Contrail Networking Supported Platforms.

Configure DNS

A DNS zone must be created and available in Route 53 for your AWS account before starting this installation. You must also register a domain for your Contrail cluster in AWS Route 53. All entries created in AWS Route 53 are expected to be resolvable from the nodes in the Contrail cluster.

For information on configuring DNS zones in AWS Route 53, see the *Amazon Route 53 Developer Guide* from AWS.

Configure AWS Credentials

The installer used in this procedure creates multiple resources in AWS that are needed to run your cluster. These resources include Elastic Compute Cloud (EC2) instances, Virtual Private Clouds (VPCs), security groups, IAM roles, and other necessary network building blocks.

AWS credentials are needed to access these resources and should be configured before starting this installation.

To configure AWS credentials, see the Configuration and credential file settings section of the AWS Command Line Interface User Guide from AWS.

Download the OpenShift Installer and the Command Line Tools

To download the installer and the command line tools:

1. Check which versions of the OpenShift installer are available:

```
$ curl -s https://mirror.openshift.com/pub/openshift-v4/clients/ocp/ | \
  awk '{print $5}'| \
  grep -o '4.[0-9].[0-9]*' | \
  uniq | \
  sort | \
  column
```

2. Set the version and download the OpenShift installer and the CLI tool.

   In this example output, the Openshift version is 4.4.20.

```
$ VERSION=4.4.20
$ wget https://mirror.openshift.com/pub/openshift-v4/clients/ocp/$VERSION/openshift-install-
mac-$VERSION.tar.gz
$ wget https://mirror.openshift.com/pub/openshift-v4/clients/ocp/$VERSION/openshift-client-
mac-$VERSION.tar.gz

$ tar -xvzf openshift-install-mac-4.4.20.tar.gz -C /usr/local/bin
$ tar -xvzf openshift-client-mac-4.4.20.tar.gz -C /usr/local/bin

$ openshift-install version
$ oc version
$ kubectl version
```

Deploy the Cluster

To deploy the cluster:

1. Generate an SSH private key and add it to the agent:

```
$ ssh-keygen -b 4096 -t rsa -f ~/.ssh/id_rsa -N ""
```

2. Create a working folder:

   In this example, a working folder named *aws-ocp4* is created and the user is then moved into the new directory.

```
$ mkdir ~/aws-ocp4 ; cd ~/aws-ocp4
```

3. Create an installation configuration file. See Creating the installation configuration file section of the Installing a cluster on AWS with customizations document from Red Hat OpenShift.

```
$ openshift-install create install-config
```

   An *install-config.yaml* file needs to be created and added to the current directory. A sample *install-config.yaml* file is provided below.

   Be aware of the following factors while creating the *install-config.yaml* file:

   - The *networkType* field is usually set as *OpenShiftSDN* in the YAML file by default.

     For configuration pointing at Contrail cluster nodes, the *networkType* field needs to be configured as *Contrail*.

   - OpenShift primary nodes need larger instances. We recommend setting the type to *m5.2xlarge* or larger for OpenShift primary nodes.

   - Most OpenShift worker nodes can use the default instance sizes. You should consider using larger instances, however, for high demand performance workloads.

   - Many of the installation parameters in the YAML file are described in more detail in the Installation configuration parameters section of the Installing a cluster on AWS with customizations document from Red Hat OpenShift.

   A sample *install-config.yaml* file:

```
apiVersion: v1
baseDomain: ovsandbox.com
```

```
compute:
- architecture: amd64
  hyperthreading: Enabled
  name: worker
  platform:
    aws:
      rootVolume:
        iops: 2000
        size: 500
        type: io1
      type: m5.4xlarge
  replicas: 3
controlPlane:
  architecture: amd64
  hyperthreading: Enabled
  name: master
  platform:
    aws:
      rootVolume:
        iops: 4000
        size: 500
        type: io1
      type: m5.2xlarge
  replicas: 3
metadata:
  creationTimestamp: null
  name: w1
networking:
  clusterNetwork:
  - cidr: 10.128.0.0/14
    hostPrefix: 23
  machineNetwork:
  - cidr: 10.0.0.0/16
  networkType: Contrail
  serviceNetwork:
  - 172.30.0.0/16
platform:
  aws:
    region: eu-west-1
publish: External
pullSecret: '{"auths"...}'
```

```
sshKey: |
  ssh-rsa ...
```

4. Create the installation manifests:

```
# openshift-install create manifests
```

5. Clone the Contrail operator repository:

```
$ git clone https://github.com/Juniper/contrail-operator.git
$ git checkout R2008
```

6. Create the Contrail operator configuration file.

   Example:

```
# cat <<EOF > config_contrail_operator.yaml
CONTRAIL_VERSION=2008.121
CONTRAIL_REGISTRY=hub.juniper.net/contrail
DOCKER_CONFIG=<this_needs_to_be_generated>
EOF
```

   where:

   - *CONTRAIL_VERSION* is the Contrail Networking container tag of the version of Contrail Networking that you are downloading.

     This procedure is initially supported in Contrail Networking Release 2008. You can obtain the Contrail Networking container tags for all Contrail Networking 20 releases in README Access to Contrail Networking Registry 20XX.

   - *CONTRAIL_REGISTRY* is the path to the container registry. The default Juniper Contrail Container Registry contains the files needed for this installation and is located at *hub.juniper.net/contrail*.

     If needed, email mailto:contrail-registry@juniper.net to obtain your username and password credentials to access the Contrail Container registry.

   - *DOCKER_CONFIG* is the registry secret credential. Set the *DOCKER_CONFIG* to registry secret with proper data in base64.

NOTE: You can create base64 encoded values using a script. See DOCKER_CONFIG generate.

To start the script:

```
# ./contrail-operator/deploy/openshift/tools/docker-config-generate/generate-docker-config.sh
```

You can copy output generated from the script and use it as the *DOCKER_CONFIG* value in this file.

7. Install Contrail manifests:

```
# ./contrail-operator/deploy/openshift/install-manifests.sh --dir ./ --config ./config_contrail_operator.yaml
```

8. Create the cluster:

```
$ openshift-install create cluster --log-level=debug
```

- Contrail Networking needs to open some networking ports for operation within AWS. These ports are opened by adding rules to security groups.

  Follow this procedure to add rules to security groups when AWS resources are manually created:

  a. Build the Contrail CLI tool for managing security group ports on AWS. This tool allows you to automatically open ports that are required for Contrail to manage security group ports on AWS that are attached to Contrail cluster resources.

     To build this tool:

     ```
     go build .
     ```

     After entering this command, you should be in the binary contrail-sc-open in your directory. This interface is the compiled tool.

b. Start the tool:

```
./contrail-sc-open -cluster-name name of your Openshift cluster -region AWS region
where cluster is located
```

9. When the service router-default is created in openshift-ingress, use the following command to patch the configuration:

```
$ oc -n openshift-ingress patch service router-default --patch '{"spec":
{"externalTrafficPolicy": "Cluster"}}'
```

10. Monitor the screen messages.

    Look for the *INFO Install complete!*.

    The final messages from a sample successful installation:

```
INFO Waiting up to 10m0s for the openshift-console route to be created...
DEBUG Route found in openshift-console namespace: console
DEBUG Route found in openshift-console namespace: downloads
DEBUG OpenShift console route is created
INFO Install complete!
INFO To access the cluster as the system:admin user when using 'oc', run 'export
KUBECONFIG=/Users/ovaleanu/aws1-ocp4/auth/kubeconfig'
INFO Access the OpenShift web-console here: https://console-openshift-
console.apps.w1.ovsandbox.com
INFO Login to the console with user: kubeadmin, password: XXXxx-XxxXX-xxXXX-XxxxX
```

11. Access the cluster:

```
$ export KUBECONFIG=~/aws-ocp4/auth/kubeconfig
```

12. Add a user to the cluster. See "How to Add a User After Completing the Installation" on page 92.

How to Add a User After Completing the Installation

The process for adding an Openshift user is identical in KVM or on AWS.

Redhat OpenShift 4.4 supports a single kubeadmin user by default. This kubeadmin user is used to deploy the initial cluster configuration.

You can use this procedure to create a Custom Resource (CR) to define a HTTPasswd identity provider.

1. Generate a flat file that contains the user names and passwords for your cluster by using the HTPasswd identity provider:

```
$ htpasswd -c -B -b users.htpasswd testuser MyPassword
```

   A file called users.httpasswd is created.

2. Define a secret password that contains the HTPasswd user file:

```
$ oc create secret generic htpass-secret --from-file=htpasswd=/root/ocp4/users.htpasswd -n
openshift-config
```

   This custom resource shows the parameters and acceptable values for an HTPasswd identity provider.

```
$ cat htpasswdCR.yaml
apiVersion: config.openshift.io/v1
kind: OAuth
metadata:
  name: cluster
spec:
  identityProviders:
  - name: testuser
    mappingMethod: claim
    type: HTPasswd
    htpasswd:
      fileData:
        name: htpass-secret
```

3. Apply the defined custom resource:

```
$ oc create -f htpasswdCR.yaml
```

4. Add the user and assign the *cluster-admin* role:

```
$ oc adm policy add-cluster-role-to-user cluster-admin testuser
```

5. Login using the new user credentials:

```
oc login -u testuser
Authentication required for https://api.ocp4.example.com:6443 (openshift)
Username: testuser
Password:
Login successful.
```

The kubeadmin user can now safely be removed. See the Removing the kubeadmin user document from Red Hat OpenShift.

How to Install Earlier Releases of Contrail Networking and Red Hat OpenShift

If you have a need to install Contrail Networking with earlier versions of Red Hat Openshift, Contrail Networking is also supported with Red Hat Openshift 3.11.

For information on installing Contrail Networking with Red Hat Openshift 3.11, see the following documentation:

- "Installing a Standalone Red Hat OpenShift Container Platform 3.11 Cluster with Contrail Using Contrail OpenShift Deployer" on page 94
- "Installing a Nested Red Hat OpenShift Container Platform 3.11 Cluster Using Contrail Ansible Deployer" on page 106

Installing a Standalone Red Hat OpenShift Container Platform 3.11 Cluster with Contrail Using Contrail OpenShift Deployer

NOTE: This topic covers Contrail Networking in Red Hat Openshift environments that are using Contrail Networking Release 21-based releases.

Starting in Release 22.1, Contrail Networking evolved into Cloud-Native Contrail Networking. Cloud-Native Contrail offers significant enhancements to optimize networking performance in Kubernetes-orchestrated environments. Cloud-Native Contrail supports Red Hat Openshift and we strongly recommend using Cloud-Native Contrail for networking in environments using Red Hat Openshift.

For general information about Cloud-Native Contrail, see the Cloud-Native Contrail Networking Techlibrary homepage.

You can install Contrail Networking together with a standalone Red Hat OpenShift Container Platform 3.11 cluster using Contrail OpenShift deployer. Consider the topology illustrated here.

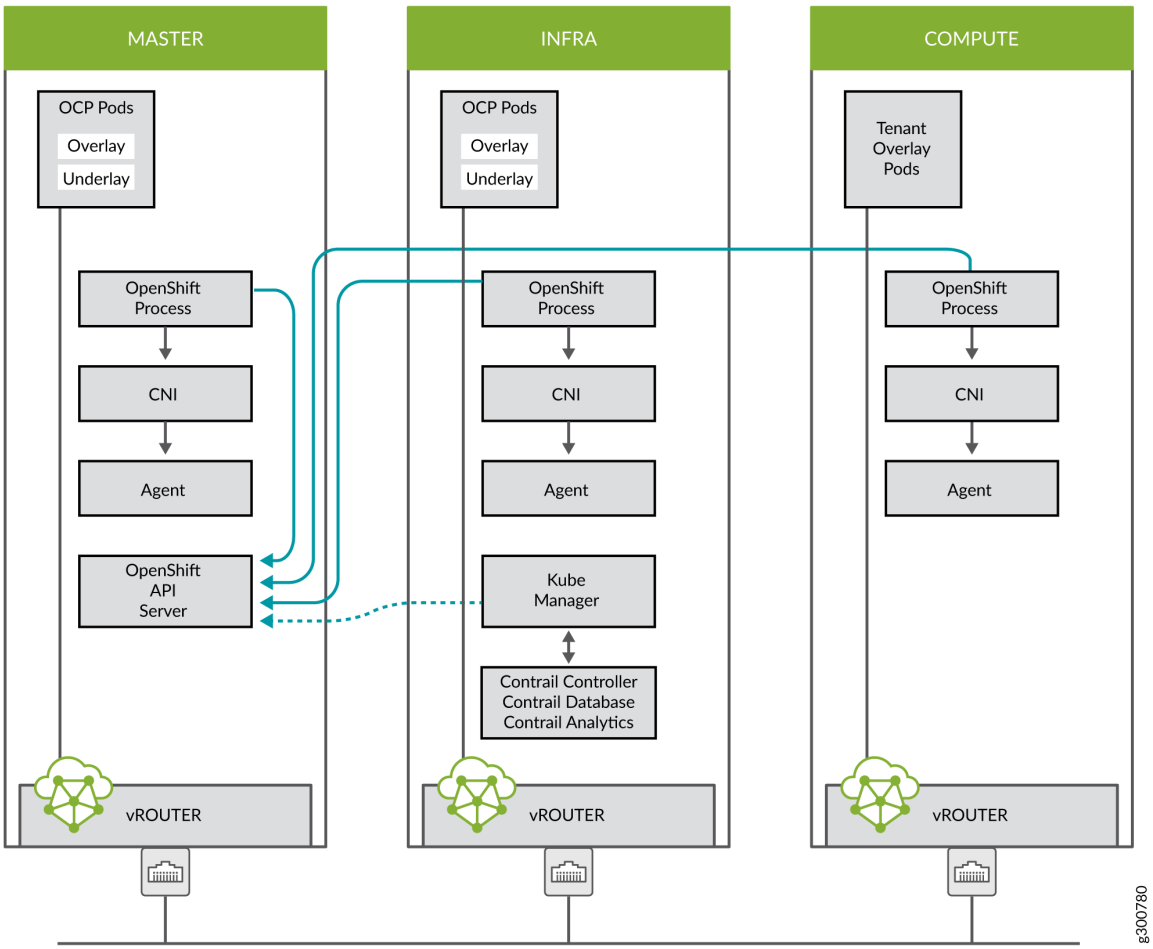
Prerequisites

The recommended system requirements are:

System Requirements	Primary Node	Infrastructure Node	Compute Node
CPU/RAM	8 vCPU, 16 GB RAM	16 vCPU, 64 GB RAM	As per OpenShift recommendations.
Disk	100 GB	250 GB	

NOTE: If you use NFS mount volumes, check disk capacity and mounts. Also, openshift-logging with NFS is not recommended.

Figure 1: Sample installation topology



Perform the following steps to install a standalone OpenShift 3.11 cluster along with Contrail Networking using contrail-openshift-deployer.

1. Set up environment nodes for RHEL OpenShift enterprise installations:

   a. Subscribe to RHEL.

   ```
   (all-nodes)# subscription-manager register --username <> --password <> --force
   ```

   b. From the list of available subscriptions, find and attach the pool ID for the OpenShift Container Platform subscription.

   ```
   (all-nodes)# subscription-manager attach --pool=pool-ID
   ```

   c. Disable all yum repositories.

   ```
   (all-nodes)# subscription-manager repos --disable="*"
   ```

d. Enable only the required repositories.

```
(all-nodes)# subscription-manager repos \
   --enable="rhel-7-server-rpms" \
   --enable="rhel-7-server-extras-rpms" \
   --enable="rhel-7-server-ose-3.11-rpms" \
   --enable=rhel-7-fast-datapath-rpms \
   --enable="rhel-7-server-ansible-2.6-rpms"
```

e. Install required packages, such as python-netaddr, iptables-services, and so on.

```
(all-nodes)# yum install -y tcpdump wget git net-tools bind-utils yum-utils iptables-services bridge-utils
bash-completion kexec-tools sos psacct python-netaddr openshift-ansible
```

NOTE: CentOS OpenShift Origin installations are not supported.

2. Get the files from the latest tar ball. Download the OpenShift Container Platform install package from Juniper software download site and modify the contents of the `openshift-ansible` inventory file.

   a. Download the Openshift Deployer (**contrail-openshift-deployer-*release-tag*.tgz**) installer from the Juniper software download site, https://www.juniper.net/support/downloads/?p=contrail#sw. See README Access for Contrail Networking Registry 19xx for appropriate release tags.

   b. Copy the install package to the node from where Ansible is deployed. Ensure that the node has password-free access to the OpenShift primary and slave nodes.

   ```
   scp contrail-openshift-deployer-release-tag.tgz openshift-ansible-node:/root/
   ```

   c. Log in to the Ansible node and untar the `contrail-openshift-deployer-release-tag.tgz` package.

   ```
   tar -xzvf contrail-openshift-deployer-release-tag.tgz -C /root/
   ```

   d. Verify the contents of the **openshift-ansible** directory.

   ```
   cd /root/openshift-ansible/
   ```

   e. Modify the **inventory/ose-install** file to match your OpenShift environment.

   Populate the **inventory/ose-install** file with Contrail configuration parameters specific to your system. The following mandatory parameters must be set. For example:

```
contrail_version=5.1
contrail_container_tag=<>
contrail_registry="hub.juniper.net/contrail-nightly"
contrail_registry_username=<>
contrail_registry_password=<>
```

```
openshift_use_openshift_sdn=false
os_sdn_network_plugin_name='cni'
openshift_use_contrail=true
```

NOTE: The `contrail_container_tag` value for this release can be found in the README Access to Contrail Registry 19XX file.

Juniper Networks recommends that you obtain the Ansible source files from the latest release.

This procedure assumes that there is one primary node, one infrastructure node, and one compute node.

```
master : server1 (1x.xx.xx.11)
infrastructure : server2 (1x.xx.xx.22)
compute : server3 (1x.xx.xx.33)
```

3. Edit **/etc/hosts** to include all the nodes information.

```
[root@server1]# cat /etc/hosts
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6
1x.xx.xx.100 puppet
1x.xx.xx.11 server1.contrail.juniper.net server1
1x.xx.xx.22 server2.contrail.juniper.net server2
1x.xx.xx.33 server3.contrail.juniper.net server3
```

4. Set up password-free SSH access to the Ansible node and all the nodes.

```
ssh-keygen -t rsa
ssh-copy-id root@1x.xx.xx.11
ssh-copy-id root@1x.xx.xx.22
ssh-copy-id root@1x.xx.xx.33
```

5. Run Ansible playbook to install OpenShift Container Platform with Contrail. Before you run Ansible playbook, ensure that you have edited **inventory/ose-install** file.

```
(ansible-node)# cd /root/openshift-ansible
(ansible-node)# ansible-playbook -i inventory/ose-install playbooks/prerequisites.yml
(ansible-node)# ansible-playbook -i inventory/ose-install playbooks/deploy_cluster.yml
```

   For a sample **inventory/ose-install** file, see "No Link Title" on page 99.

6. Create a password for the admin user to log in to the UI from the primary node.

```
(master-node)# htpasswd /etc/origin/master/htpasswd admin
```

   NOTE: If you are using a load balancer, you must manually copy the htpasswd file into all your primary nodes.

7. Assign cluster-admin role to admin user.

```
(master-node)# oc adm policy add-cluster-role-to-user cluster-admin admin
(master-node)# oc login -u admin
```

8. Open a Web browser and type the entire fqdn name of your primary node or load balancer node, followed by `:8443/console`.

```
https://<your host name from your ose-install inventory>:8443/console
```

   Use the user name and password created in step 6 to log in to the Web console.

   Your DNS should resolve the host name for access. If the host name is not resolved, modify the /etc/hosts file to route to the above host.

NOTE: OpenShift 3.11 cluster upgrades are not supported.

Sample inventory/ose-install File

```
[OSEv3:vars]

###########################################################################
```

```
### OpenShift Basic Vars
###############################################################################
openshift_deployment_type=openshift-enterprise
deployment_type=openshift-enterprise
containerized=false
openshift_disable_check=docker_image_availability,memory_availability,package_availability,disk_a
vailability,package_version,docker_storage

# Default node selectors
openshift_hosted_infra_selector="node-role.kubernetes.io/infra=true"

oreg_auth_user=<>
oreg_auth_password=<>

###############################################################################
### OpenShift Master Vars
###############################################################################

openshift_master_api_port=8443
openshift_master_console_port=8443
openshift_master_cluster_method=native

# Set this line to enable NFS
openshift_enable_unsupported_configurations=True


###############################################################################
### OpenShift Network Vars
###############################################################################

openshift_use_openshift_sdn=false
os_sdn_network_plugin_name='cni'
openshift_use_contrail=true

###############################################################################
### OpenShift Authentication Vars
###############################################################################

# htpasswd Authentication
openshift_master_identity_providers=[{'name': 'htpasswd_auth', 'login': 'true', 'challenge':
'true', 'kind': 'HTPasswdPasswordIdentityProvider'}]

###############################################################################
```

```
### OpenShift Router and Registry Vars
#############################################################################

openshift_hosted_router_replicas=1
openshift_hosted_registry_replicas=1

openshift_hosted_registry_storage_kind=nfs
openshift_hosted_registry_storage_access_modes=['ReadWriteMany']
openshift_hosted_registry_storage_nfs_directory=/export
openshift_hosted_registry_storage_nfs_options='*(rw,root_squash)'
openshift_hosted_registry_storage_volume_name=registry
openshift_hosted_registry_storage_volume_size=10Gi
openshift_hosted_registry_pullthrough=true
openshift_hosted_registry_acceptschema2=true
openshift_hosted_registry_enforcequota=true
openshift_hosted_router_selector="node-role.kubernetes.io/infra=true"
openshift_hosted_registry_selector="node-role.kubernetes.io/infra=true"

#############################################################################
### OpenShift Service Catalog Vars
#############################################################################

openshift_enable_service_catalog=True

template_service_broker_install=True
openshift_template_service_broker_namespaces=['openshift']

ansible_service_broker_install=True

openshift_hosted_etcd_storage_kind=nfs
openshift_hosted_etcd_storage_nfs_options="*(rw,root_squash,sync,no_wdelay)"
openshift_hosted_etcd_storage_nfs_directory=/export
openshift_hosted_etcd_storage_labels={'storage': 'etcd-asb'}
openshift_hosted_etcd_storage_volume_name=etcd-asb
openshift_hosted_etcd_storage_access_modes=['ReadWriteOnce']
openshift_hosted_etcd_storage_volume_size=2G




#############################################################################
### OpenShift Metrics and Logging Vars
```

```
#############################################################################
# Enable cluster metrics
openshift_metrics_install_metrics=True

openshift_metrics_storage_kind=nfs
openshift_metrics_storage_access_modes=['ReadWriteOnce']
openshift_metrics_storage_nfs_directory=/export
openshift_metrics_storage_nfs_options='*(rw,root_squash)'
openshift_metrics_storage_volume_name=metrics
openshift_metrics_storage_volume_size=2Gi
openshift_metrics_storage_labels={'storage': 'metrics'}

openshift_metrics_cassandra_nodeselector={"node-role.kubernetes.io/infra":"true"}
openshift_metrics_hawkular_nodeselector={"node-role.kubernetes.io/infra":"true"}
openshift_metrics_heapster_nodeselector={"node-role.kubernetes.io/infra":"true"}

# Enable cluster logging. ((
####openshift_logging_install_logging=True
openshift_logging_install_logging=False
#openshift_logging_storage_kind=nfs
#openshift_logging_storage_access_modes=['ReadWriteOnce']
#openshift_logging_storage_nfs_directory=/export
#openshift_logging_storage_nfs_options='*(rw,root_squash)'
#openshift_logging_storage_volume_name=logging
#openshift_logging_storage_volume_size=5Gi
#openshift_logging_storage_labels={'storage': 'logging'}
#openshift_logging_es_cluster_size=1
#openshift_logging_es_nodeselector={"node-role.kubernetes.io/infra":"true"}
#openshift_logging_kibana_nodeselector={"node-role.kubernetes.io/infra":"true"}
#openshift_logging_curator_nodeselector={"node-role.kubernetes.io/infra":"true"}

#############################################################################
### OpenShift Prometheus Vars
#############################################################################

## Add Prometheus Metrics:
openshift_hosted_prometheus_deploy=True
openshift_prometheus_node_selector={"node-role.kubernetes.io/infra":"true"}
openshift_prometheus_namespace=openshift-metrics

# Prometheus
openshift_prometheus_storage_kind=nfs
openshift_prometheus_storage_access_modes=['ReadWriteOnce']
```

```
openshift_prometheus_storage_nfs_directory=/export
openshift_prometheus_storage_nfs_options='*(rw,root_squash)'
openshift_prometheus_storage_volume_name=prometheus
openshift_prometheus_storage_volume_size=1Gi
openshift_prometheus_storage_labels={'storage': 'prometheus'}
openshift_prometheus_storage_type='pvc'

# For prometheus-alertmanager
openshift_prometheus_alertmanager_storage_kind=nfs
openshift_prometheus_alertmanager_storage_access_modes=['ReadWriteOnce']
openshift_prometheus_alertmanager_storage_nfs_directory=/export
openshift_prometheus_alertmanager_storage_nfs_options='*(rw,root_squash)'
openshift_prometheus_alertmanager_storage_volume_name=prometheus-alertmanager
openshift_prometheus_alertmanager_storage_volume_size=1Gi
openshift_prometheus_alertmanager_storage_labels={'storage': 'prometheus-alertmanager'}
openshift_prometheus_alertmanager_storage_type='pvc'

# For prometheus-alertbuffer
openshift_prometheus_alertbuffer_storage_kind=nfs
openshift_prometheus_alertbuffer_storage_access_modes=['ReadWriteOnce']
openshift_prometheus_alertbuffer_storage_nfs_directory=/export
openshift_prometheus_alertbuffer_storage_nfs_options='*(rw,root_squash)'
openshift_prometheus_alertbuffer_storage_volume_name=prometheus-alertbuffer
openshift_prometheus_alertbuffer_storage_volume_size=1Gi
openshift_prometheus_alertbuffer_storage_labels={'storage': 'prometheus-alertbuffer'}
openshift_prometheus_alertbuffer_storage_type='pvc'


#############################################################################
### Openshift HA
#############################################################################

# Openshift HA
openshift_master_cluster_hostname=load-balancer-0-3eba0c20dc494dfc93d5d50d06bbde89
openshift_master_cluster_public_hostname=load-balancer-0-3eba0c20dc494dfc93d5d50d06bbde89


#############################################################################
### Contrail Variables
#############################################################################

service_subnets="172.30.0.0/16"
pod_subnets="10.128.0.0/14"
```

```
# Below are Contrail variables. Comment them out if you don't want to install Contrail through
ansible-playbook
contrail_version=5.1
contrail_container_tag=<>
contrail_registry=hub.juniper.net/contrail
contrail_registry_username=<>
contrail_registry_password=<>
openshift_docker_insecure_registries=hub.juniper.net/contrail
contrail_nodes=[10.0.0.5,10.0.0.3,10.0.0.4]
vrouter_physical_interface=eth0


##############################################################################
### OpenShift Hosts
##############################################################################
[OSEv3:children]
masters
etcd
nodes
lb
nfs
openshift_ca

[masters]
kube-master-2-3eba0c20dc494dfc93d5d50d06bbde89
kube-master-1-3eba0c20dc494dfc93d5d50d06bbde89
kube-master-0-3eba0c20dc494dfc93d5d50d06bbde89

[etcd]
kube-master-2-3eba0c20dc494dfc93d5d50d06bbde89
kube-master-1-3eba0c20dc494dfc93d5d50d06bbde89
kube-master-0-3eba0c20dc494dfc93d5d50d06bbde89

[lb]
load-balancer-0-3eba0c20dc494dfc93d5d50d06bbde89

[nodes]
kube-master-2-3eba0c20dc494dfc93d5d50d06bbde89 openshift_node_group_name='node-config-master'
controller-0-3eba0c20dc494dfc93d5d50d06bbde89 openshift_node_group_name='node-config-infra'
compute-1-3eba0c20dc494dfc93d5d50d06bbde89 openshift_node_group_name='node-config-compute'
controller-2-3eba0c20dc494dfc93d5d50d06bbde89 openshift_node_group_name='node-config-infra'
kube-master-1-3eba0c20dc494dfc93d5d50d06bbde89 openshift_node_group_name='node-config-master'
```

```
kube-master-0-3eba0c20dc494dfc93d5d50d06bbde89 openshift_node_group_name='node-config-master'
compute-0-3eba0c20dc494dfc93d5d50d06bbde89 openshift_node_group_name='node-config-compute'
controller-1-3eba0c20dc494dfc93d5d50d06bbde89 openshift_node_group_name='node-config-infra'

[nfs]
load-balancer-0-3eba0c20dc494dfc93d5d50d06bbde89

[openshift_ca]
kube-master-2-3eba0c20dc494dfc93d5d50d06bbde89
kube-master-1-3eba0c20dc494dfc93d5d50d06bbde89
kube-master-0-3eba0c20dc494dfc93d5d50d06bbde89
```

NOTE: The /etc/resolv.conf must have write permissions.

Caveats and Troubleshooting Instructions

- If a Java error occurs, install the `yum install java-1.8.0-openjdk-devel.x86_64` package and rerun `deploy_cluster`.
- If the service_catalog parameter does not pass but the cluster is operational, check whether the **/etc/resolv.conf** has `cluster.local` in its search line, and the nameserver as host IP address.
- NTP is installed by OpenShift and must be synchronized by the user. This does not affect any Contrail functionality but is displayed in the `contrail-status` output.
- If the `ansible_service_broker` component of OpenShift is not up and its `ansible_service_broker_deploy` displays an error, it means that the `ansible_service_broker` pod did not come up properly. The most likely reason is that the `ansible_service_broker` pod failed its liveliness and readiness checks. Modify the liveliness and readiness checks of this pod when it's brought online to make it operational. Also, verify that the `ansible_service_broker` pod uses the correct URL from Red Hat.

RELATED DOCUMENTATION

Installing a Nested Red Hat OpenShift Container Platform 3.11 Cluster Using Contrail Ansible Deployer | **106**

Installing a Nested Red Hat OpenShift Container Platform 3.11 Cluster Using Contrail Ansible Deployer

NOTE: This topic covers Contrail Networking in Red Hat Openshift environments that are using Contrail Networking Release 21-based releases.

Starting in Release 22.1, Contrail Networking evolved into Cloud-Native Contrail Networking. Cloud-Native Contrail offers significant enhancements to optimize networking performance in Kubernetes-orchestrated environments. Cloud-Native Contrail supports Red Hat Openshift and we strongly recommend using Cloud-Native Contrail for networking in environments using Red Hat Openshift.

For general information about Cloud-Native Contrail, see the Cloud-Native Contrail Networking Techlibrary homepage.

You can install a nested Red Hat OpenShift Container Platform 3.11 cluster along with Contrail Networking using Contrail Ansible deployer.

Prerequisites

Ensure that the following prerequisites are met for a successful provisioning of a nested Contrail-OpenShift cluster.

- The recommended system requirements are:

System Requirements	Primary Node	Infrastructure Node	Compute Node
CPU/RAM	8 vCPU, 16 GB RAM	16 vCPU, 64 GB RAM	As per OpenShift recommendations.
Disk	100 GB	250 GB	

- A running Red Hat OpenStack Platform Director (RHOSPD) 13 cluster with Contrail. OpenShift Contrail release must be same as RHOSPD 13 Contrail release.

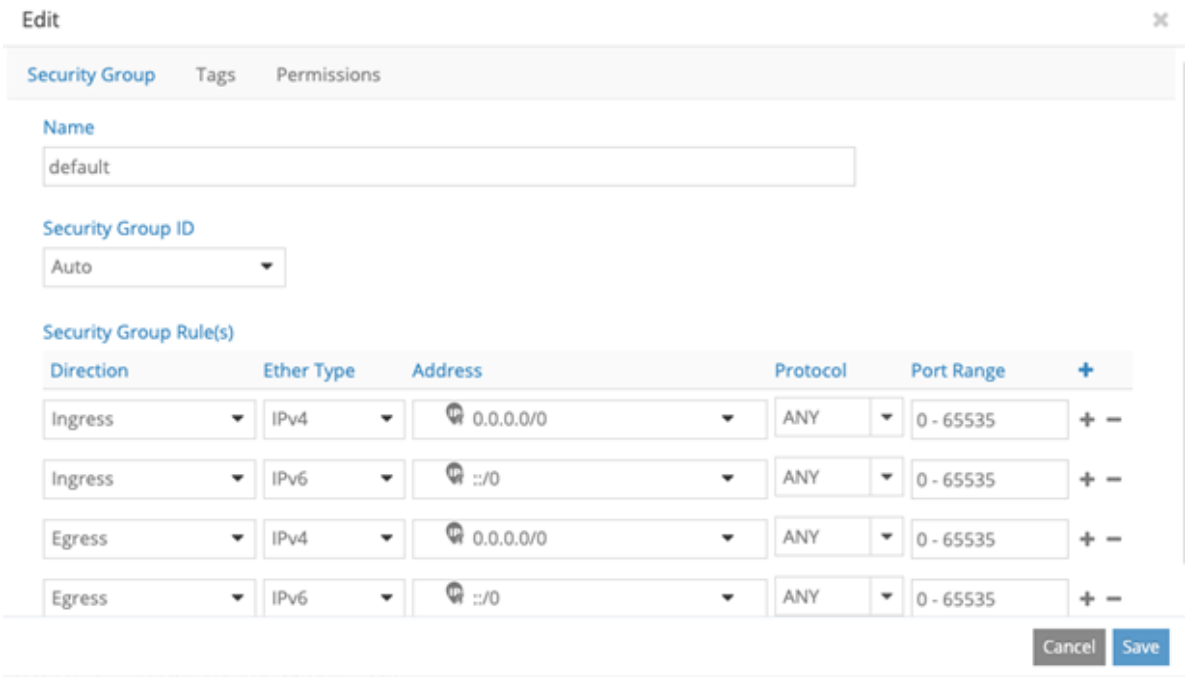
- RHOSPD environments require that the Contrail vrouter, Contrail config and OpenStack keystone are in “internal-api” network. Modify the ServiceNetMap parameters in the **contrail-services.yaml** file to configure in “internal-api” network.

```
parameter_defaults:
  ServiceNetMap:
    ContrailDatabaseNetwork: internal_api
    ContrailAnalyticsNetwork: internal_api
    ContrailAnalyticsAlarmNetwork: internal_api
    ContrailAnalyticsDatabaseNetwork: internal_api
    ContrailAnalyticsSnmpNetwork: internal_api
    ContrailConfigNetwork: internal_api
    ContrailControlNetwork: internal_api
    ContrailWebuiNetwork: internal_api
    ContrailVrouterNetwork: internal_api
    ContrailCertmongerUserNetwork: internal_api
    KeystoneAdminApiNetwork: internal_api
```

- Ensure that the vRouter gateway in the **contrail-services.yaml** file is part of “internal-api” network.

```
# Custom Contrail container configuration settings
  ContrailSettings:
    VROUTER_GATEWAY: 10.1.0.254
```

- OpenShift nodes (VMs) must have Internet connectivity.
- Default security group of the virtual-network where OpenShift nodes are launched must be modified to allow all ingress traffic to communicate with OpenShift networks provided in the OpenShift inventory file.



Provisioning Nested OpenShift Cluster

Provisioning a nested OpenShift cluster is a two-step process.

1. Create link-local services in the Contrail-OpenStack cluster.

   A nested OpenShift cluster is managed by the same Contrail controller that manages the underlying OpenStack cluster. Hence, the nested Openshift cluster needs IP reachability to the Contrail controller and OpenStack keystone service. Since the OpenShift cluster is actually an overlay on the OpenStack cluster, we use the Link Local Service feature of Contrail to provide IP reachability to and from the overlay OpenShift cluster and OpenStack cluster.

   To configure a Link Local Service, we need a Fabric IP and Service IP. Fabric IP is the node IP on which the Contrail Controller and OpenStack services are running. Service IP is a unique and unused IP in the entire OpenStack cluster and is shared with the OpenShift cluster to reach Contrail Controller and OpenStack services. Service IP (along with port number) is used by the data plane to identify the fabric IP. For each node of the OpenStack cluster, one service IP must be identified.

   You must configure the following Link Local Services in Contrail.

Contrail Controller and OpenStack Process	Service IP	Service Port	Fabric IP	Fabric Port

Contrail Config	<Service IP for the running node>	8082	<Node IP of running node>	8082
Contrail Analytics	<Service IP for the running node>	8086	<Node IP of running node>	8086
Contrail Msg Queue	<Service IP for the running node>	5673	<Node IP of running node>	5673
Contrail VNC DB	<Service IP for the running node>	9161	<Node IP of running node>	9161
Keystone	<Service IP for the running node>	35357	<Node IP of running node>	35357
K8s-cni-to-agent	<Service IP for the running node>	9091	<Node IP of running node>	9091

For example, consider a sample cluster of seven nodes.

```
Contrail Config : 192.168.1.100
Contrail Analytics : 192.168.1.100, 192.168.1.101
Contrail Msg Queue : 192.168.1.100
Contrail VNC DB : 192.168.1.100, 192.168.1.101, 192.168.1.102
Keystone: 192.168.1.200
Vrouter: 192.168.1.201, 192.168.1.202, 192.168.1.203
```

Allocate seven unused IP addresses for the seven nodes.

```
192.168.1.100  --> 10.10.10.1
192.168.1.101  --> 10.10.10.2
192.168.1.102  --> 10.10.10.3
192.168.1.200  --> 10.10.10.4
192.168.1.201/192.168.1.202/192.168.1.203  --> 10.10.10.5
```

NOTE: One Service IP address can represent all vRouter nodes.

The following link-local services must be created:

Contrail controller and OpenStack process	Service IP	Service Port	Fabric IP	Fabric Port
Contrail Config	10.10.10.1	8082	192.168.1.100	8082
Contrail Analytics 1	10.10.10.1	8086	192.168.1.100	8086
Contrail Analytics 2	10.10.10.1	8086	192.168.1.101	8086
Contrail Msg Queue	10.10.10.2	5673	192.168.1.100	5673
Contrail VNC DB 1	10.10.10.1	9161	192.168.1.100	9161
Contrail VNC DB 2	10.10.10.2	9161	192.168.1.101	9161
Contrail VNC DB 3	10.10.10.2	9161	192.168.1.102	9161
Keystone	10.10.10.4	35357	192.168.1.200	35357
K8s-cni-to-agent	10.10.10.5	9091	127.0.0.1	9091

2. Install OpenShift using OpenShift Ansible deployer.

   Perform the following steps to install the nested OpenShift 3.11 cluster along with Contrail Networking using OpenShift Ansible deployer.

   a. Set up environment nodes for RHEL OpenShift enterprise installations:

      i. Subscribe to RHEL.

      ```
      (all-nodes)# subscription-manager register --username <> --password <> --force
      ```

      ii. From the list of available subscriptions, find and attach the pool ID for the OpenShift Container Platform subscription.

```
(all-nodes)# subscription-manager attach --pool=pool-ID
```

iii. Disable all yum repositories.

```
(all-nodes)# subscription-manager repos --disable="*"
```

iv. Enable only the required repositories.

```
(all-nodes)# subscription-manager repos \
       --enable="rhel-7-server-rpms" \
       --enable="rhel-7-server-extras-rpms" \
       --enable="rhel-7-server-ose-3.11-rpms" \
       --enable=rhel-7-fast-datapath-rpms \
       --enable="rhel-7-server-ansible-2.6-rpms"
```

v. Install required packages, such as python-netaddr, iptables-services, and so on.

```
(all-nodes)# yum install -y tcpdump wget git net-tools bind-utils yum-utils iptables-services
bridge-utils bash-completion kexec-tools sos psacct python-netaddr openshift-ansible
```

NOTE: CentOS OpenShift Origin installations are not supported.

b. Get the files from the latest tar ball. Download the OpenShift Container Platform install package from Juniper software download site and modify the contents of the `openshift-ansible` inventory file.

i.

i. Download Openshift Ansible (**contrail-ansible-deployer-*release-tag*.tgz**) installer from the Juniper software download site, https://www.juniper.net/support/downloads/?p=contrail#sw. See README Access to Contrail Networking Registry 20xx for appropriate release tags.

ii. Copy the install package to the node from where Ansible is deployed. Ensure that the node has password-free access to the OpenShift primary and slave nodes.

```
scp contrail-ansible-deployer-release-tag.tgz openshift-ansible-node:/root/
```

iii. Log in to the Ansible node and untar the `contrail-ansible-deployer-*release-tag*.tgz` package.

```
tar -xzvf contrail-ansible-deployer-release-tag.tgz -C /root/
```

iv. Verify the contents of the **openshift-ansible** directory.

```
cd /root/openshift-ansible/
```

v. Modify the **inventory/ose-install** file to match your OpenShift environment.

Populate the **inventory/ose-install** file with Contrail configuration parameters specific to your system. The following mandatory parameters must be set.

```
contrail_version=1907
            contrail_container_tag=<>
            contrail_registry="hub.juniper.net/contrail"
            contrail_registry_username=<>
            contrail_registry_password=<>
            openshift_use_openshift_sdn=false
            os_sdn_network_plugin_name='cni'
            openshift_use_contrail=true
```

NOTE: The `contrail_container_tag` value for this release can be found in the README Access to Contrail Networking Registry 20xx file.

NOTE: Juniper Networks recommends that you obtain the Ansible source files from the latest release.

This procedure assumes that there is one primary node, one infrastructure node, and one compute node.

```
master : server1 (1x.xx.xx.11)
infrastructure : server2 (1x.xx.xx.22)
compute : server3 (1x.xx.xx.33)
```

c. Edit **/etc/hosts** to include all the nodes information.

```
[root@server1]# cat /etc/hosts
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6
1x.xx.xx.100 puppet
```

```
1x.xx.xx.11 server1.contrail.juniper.net server1
1x.xx.xx.22 server2.contrail.juniper.net server2
1x.xx.xx.33 server3.contrail.juniper.net server3
```

d. Set up password-free SSH access to the Ansible node and all the nodes.

```
ssh-keygen -t rsa
ssh-copy-id root@1x.xx.xx.11
ssh-copy-id root@1x.xx.xx.22
ssh-copy-id root@1x.xx.xx.33
```

e. Run Ansible playbook to install OpenShift Container Platform with Contrail. Before you run Ansible playbook, ensure that you have edited **inventory/ose-install** file.

```
(ansible-node)# cd /root/openshift-ansible
(ansible-node)# ansible-playbook -i inventory/ose-install playbooks/prerequisites.yml
(ansible-node)# ansible-playbook -i inventory/ose-install playbooks/deploy_cluster.yml
```

For a sample **inventory/ose-install** file, see "No Link Title" on page 114.

f. Create a password for the admin user to log in to the UI from the primary node.

```
(master-node)# htpasswd /etc/origin/master/htpasswd admin
```

NOTE: If you are using a load balancer, you must manually copy the htpasswd file into all your primary nodes.

g. Assign cluster-admin role to admin user.

```
(master-node)# oc adm policy add-cluster-role-to-user cluster-admin admin
(master-node)# oc login -u admin
```

h. Open a Web browser and type the entire fqdn name of your primary node or load balancer node, followed by :8443/console.

```
https://<your host name from your ose-install inventory>:8443/console
```

Use the user name and password created in step "2.f" on page 113 to log in to the Web console.

Your DNS should resolve the host name for access. If the host name is not resolved, modify the /etc/hosts file to route to the above host.

NOTE: OpenShift 3.11 cluster upgrades are not supported.

Sample inventory/ose-install File

```
[OSEv3:vars]


###############################################################################
### OpenShift Nested mode vars
###############################################################################
nested_mode_contrail=true
rabbitmq_node_port=5673
contrail_nested_masters_ip="1.1.1.1 2.2.2.2 3.3.3.3"          <---  ips of contrail controllers
auth_mode=keystone
keystone_auth_host=<w.x.y.z>        <--- This should be the IP where Keystone service is running.
keystone_auth_admin_tenant=admin
keystone_auth_admin_user=admin
keystone_auth_admin_password=MAYffWrX7ZpPrV2AMAa9zAUvG     <-- Keystone admin password.
keystone_auth_admin_port=35357
keystone_auth_url_version=/v3
#k8s_nested_vrouter_vip is a service IP for the running node which we configured above
k8s_nested_vrouter_vip=10.10.10.5   <-- Service IP configured for CNI to Agent communication.
(K8s-cni-to-agent in above examples)
#k8s_vip is kubernetes api server ip
k8s_vip=<W.X.Y.Z>                   <-- IP of the Openshift Master Node.
#cluster_network is the one which vm network belongs to
cluster_network="{'domain': 'default-domain', 'project': 'admin', 'name': 'net1'}" <-- FQName of
the Virtual Network where Virtual Machines are running. The VMs in which Openshift cluster is
being installed in nested mode.
#config_nodes="x.x.x.x,y.y.y.y.y"
#analytics_nodes="x.x.x.x,y.y.y.y.y"
#config_api_vip=x.x.x.x
#analytics_api_vip=x.x.x.x


###############################################################################
```

```
### OpenShift Basic Vars
#############################################################################
openshift_deployment_type=openshift-enterprise
deployment_type=openshift-enterprise
containerized=false
openshift_disable_check=docker_image_availability,memory_availability,package_availability,disk_a
vailability,package_version,docker_storage

# Default node selectors
openshift_hosted_infra_selector="node-role.kubernetes.io/infra=true"

oreg_auth_user=<>
oreg_auth_password=<>

#############################################################################
### OpenShift Master Vars
#############################################################################

openshift_master_api_port=8443
openshift_master_console_port=8443
openshift_master_cluster_method=native

# Set this line to enable NFS
openshift_enable_unsupported_configurations=True


#############################################################################
### OpenShift Network Vars
#############################################################################

openshift_use_openshift_sdn=false
os_sdn_network_plugin_name='cni'
openshift_use_contrail=true

#############################################################################
### OpenShift Authentication Vars
#############################################################################

# htpasswd Authentication
openshift_master_identity_providers=[{'name': 'htpasswd_auth', 'login': 'true', 'challenge':
'true', 'kind': 'HTPasswdPasswordIdentityProvider'}]

#############################################################################
```

```
### OpenShift Router and Registry Vars
#############################################################################

openshift_hosted_router_replicas=1
openshift_hosted_registry_replicas=1

openshift_hosted_registry_storage_kind=nfs
openshift_hosted_registry_storage_access_modes=['ReadWriteMany']
openshift_hosted_registry_storage_nfs_directory=/export
openshift_hosted_registry_storage_nfs_options='*(rw,root_squash)'
openshift_hosted_registry_storage_volume_name=registry
openshift_hosted_registry_storage_volume_size=10Gi
openshift_hosted_registry_pullthrough=true
openshift_hosted_registry_acceptschema2=true
openshift_hosted_registry_enforcequota=true
openshift_hosted_router_selector="node-role.kubernetes.io/infra=true"
openshift_hosted_registry_selector="node-role.kubernetes.io/infra=true"

#############################################################################
### OpenShift Service Catalog Vars
#############################################################################

openshift_enable_service_catalog=True

template_service_broker_install=True
openshift_template_service_broker_namespaces=['openshift']

ansible_service_broker_install=True

openshift_hosted_etcd_storage_kind=nfs
openshift_hosted_etcd_storage_nfs_options="*(rw,root_squash,sync,no_wdelay)"
openshift_hosted_etcd_storage_nfs_directory=/export
openshift_hosted_etcd_storage_labels={'storage': 'etcd-asb'}
openshift_hosted_etcd_storage_volume_name=etcd-asb
openshift_hosted_etcd_storage_access_modes=['ReadWriteOnce']
openshift_hosted_etcd_storage_volume_size=2G




#############################################################################
### OpenShift Metrics and Logging Vars
```

```
#############################################################################
# Enable cluster metrics
openshift_metrics_install_metrics=True

openshift_metrics_storage_kind=nfs
openshift_metrics_storage_access_modes=['ReadWriteOnce']
openshift_metrics_storage_nfs_directory=/export
openshift_metrics_storage_nfs_options='*(rw,root_squash)'
openshift_metrics_storage_volume_name=metrics
openshift_metrics_storage_volume_size=2Gi
openshift_metrics_storage_labels={'storage': 'metrics'}

openshift_metrics_cassandra_nodeselector={"node-role.kubernetes.io/infra":"true"}
openshift_metrics_hawkular_nodeselector={"node-role.kubernetes.io/infra":"true"}
openshift_metrics_heapster_nodeselector={"node-role.kubernetes.io/infra":"true"}

# Enable cluster logging. ((
####openshift_logging_install_logging=True
openshift_logging_install_logging=False
#openshift_logging_storage_kind=nfs
#openshift_logging_storage_access_modes=['ReadWriteOnce']
#openshift_logging_storage_nfs_directory=/export
#openshift_logging_storage_nfs_options='*(rw,root_squash)'
#openshift_logging_storage_volume_name=logging
#openshift_logging_storage_volume_size=5Gi
#openshift_logging_storage_labels={'storage': 'logging'}
#openshift_logging_es_cluster_size=1
#openshift_logging_es_nodeselector={"node-role.kubernetes.io/infra":"true"}
#openshift_logging_kibana_nodeselector={"node-role.kubernetes.io/infra":"true"}
#openshift_logging_curator_nodeselector={"node-role.kubernetes.io/infra":"true"}

#############################################################################
### OpenShift Prometheus Vars
#############################################################################

## Add Prometheus Metrics:
openshift_hosted_prometheus_deploy=True
openshift_prometheus_node_selector={"node-role.kubernetes.io/infra":"true"}
openshift_prometheus_namespace=openshift-metrics

# Prometheus
openshift_prometheus_storage_kind=nfs
openshift_prometheus_storage_access_modes=['ReadWriteOnce']
```

```
openshift_prometheus_storage_nfs_directory=/export
openshift_prometheus_storage_nfs_options='*(rw,root_squash)'
openshift_prometheus_storage_volume_name=prometheus
openshift_prometheus_storage_volume_size=1Gi
openshift_prometheus_storage_labels={'storage': 'prometheus'}
openshift_prometheus_storage_type='pvc'

# For prometheus-alertmanager
openshift_prometheus_alertmanager_storage_kind=nfs
openshift_prometheus_alertmanager_storage_access_modes=['ReadWriteOnce']
openshift_prometheus_alertmanager_storage_nfs_directory=/export
openshift_prometheus_alertmanager_storage_nfs_options='*(rw,root_squash)'
openshift_prometheus_alertmanager_storage_volume_name=prometheus-alertmanager
openshift_prometheus_alertmanager_storage_volume_size=1Gi
openshift_prometheus_alertmanager_storage_labels={'storage': 'prometheus-alertmanager'}
openshift_prometheus_alertmanager_storage_type='pvc'

# For prometheus-alertbuffer
openshift_prometheus_alertbuffer_storage_kind=nfs
openshift_prometheus_alertbuffer_storage_access_modes=['ReadWriteOnce']
openshift_prometheus_alertbuffer_storage_nfs_directory=/export
openshift_prometheus_alertbuffer_storage_nfs_options='*(rw,root_squash)'
openshift_prometheus_alertbuffer_storage_volume_name=prometheus-alertbuffer
openshift_prometheus_alertbuffer_storage_volume_size=1Gi
openshift_prometheus_alertbuffer_storage_labels={'storage': 'prometheus-alertbuffer'}
openshift_prometheus_alertbuffer_storage_type='pvc'


#############################################################################
### Openshift HA
#############################################################################

# Openshift HA
openshift_master_cluster_hostname=load-balancer-0-3eba0c20dc494dfc93d5d50d06bbde89
openshift_master_cluster_public_hostname=load-balancer-0-3eba0c20dc494dfc93d5d50d06bbde89


#############################################################################
### Contrail Variables
#############################################################################

service_subnets="172.30.0.0/16"
pod_subnets="10.128.0.0/14"
```

```
# Below are Contrail variables. Comment them out if you don't want to install Contrail through
ansible-playbook
contrail_version=1907
contrail_container_tag=<>
contrail_registry=hub.juniper.net/contrail
contrail_registry_username=<>
contrail_registry_password=<>
openshift_docker_insecure_registries=hub.juniper.net/contrail
contrail_nodes=[10.0.0.5,10.0.0.3,10.0.0.4]
vrouter_physical_interface=eth0


##############################################################################
### OpenShift Hosts
##############################################################################
[OSEv3:children]
masters
etcd
nodes
lb
nfs
openshift_ca

[masters]
kube-master-2-3eba0c20dc494dfc93d5d50d06bbde89
kube-master-1-3eba0c20dc494dfc93d5d50d06bbde89
kube-master-0-3eba0c20dc494dfc93d5d50d06bbde89

[etcd]
kube-master-2-3eba0c20dc494dfc93d5d50d06bbde89
kube-master-1-3eba0c20dc494dfc93d5d50d06bbde89
kube-master-0-3eba0c20dc494dfc93d5d50d06bbde89

[lb]
load-balancer-0-3eba0c20dc494dfc93d5d50d06bbde89

[nodes]
kube-master-2-3eba0c20dc494dfc93d5d50d06bbde89 openshift_node_group_name='node-config-master'
controller-0-3eba0c20dc494dfc93d5d50d06bbde89 openshift_node_group_name='node-config-infra'
compute-1-3eba0c20dc494dfc93d5d50d06bbde89 openshift_node_group_name='node-config-compute'
controller-2-3eba0c20dc494dfc93d5d50d06bbde89 openshift_node_group_name='node-config-infra'
kube-master-1-3eba0c20dc494dfc93d5d50d06bbde89 openshift_node_group_name='node-config-master'
```

```
kube-master-0-3eba0c20dc494dfc93d5d50d06bbde89 openshift_node_group_name='node-config-master'
compute-0-3eba0c20dc494dfc93d5d50d06bbde89 openshift_node_group_name='node-config-compute'
controller-1-3eba0c20dc494dfc93d5d50d06bbde89 openshift_node_group_name='node-config-infra'

[nfs]
load-balancer-0-3eba0c20dc494dfc93d5d50d06bbde89

[openshift_ca]
kube-master-2-3eba0c20dc494dfc93d5d50d06bbde89
kube-master-1-3eba0c20dc494dfc93d5d50d06bbde89
kube-master-0-3eba0c20dc494dfc93d5d50d06bbde89
```

NOTE: The /etc/resolv.conf must have write permissions.

Release History Table

Release	Description
1907	You can install a nested Red Hat OpenShift Container Platform 3.11 cluster along with Contrail Networking using Contrail Ansible deployer.

RELATED DOCUMENTATION

Installing a Standalone Red Hat OpenShift Container Platform 3.11 Cluster with Contrail Using Contrail OpenShift Deployer | 94

3

CHAPTER

Contrail Networking with the Elastic Kubernetes Service (EKS)in Amazon Web Services (AWS)

How to Install Contrail Networking within an Amazon Elastic Kubernetes Service (EKS) Environment in AWS

IN THIS SECTION

NOTE: This topic covers Contrail Networking in Kubernetes-orchestrated environments that are using Contrail Networking Release 21-based releases.

Starting in Release 22.1, Contrail Networking evolved into Cloud-Native Contrail Networking. Cloud-Native Contrail Networking offers significant enhancements to optimize networking performance in Kubernetes-orchestrated environments. We recommend using Cloud-Native Contrail for networking in most Kubernetes-orchestrated environments.

For general information about Cloud-Native Contrail, see the Cloud-Native Contrail Networking Techlibrary homepage.

The Elastic Kubernetes Service (EKS) runs Kubernetes-orchestrated environments within Amazon Web Services (AWS).

Kubernetes supports a pluggable framework—called the Container Networking Interface (CNI)—for networking. See Pod networking (CNI) from AWS for information on how the CNI framework is implemented by EKS.

Contrail Networking is supported as a custom CNI in Kubernetes-orchestrated environments. This document show you how to install Contrail Networking as the CNI when a Kubernetes environment is running in EKS on AWS.

It includes the following sections:

When to Use This Procedure

Use this procedure to enable Contrail Networking as the CNI in a Kubernetes-orchestrated environment running on AWS. Contrail Networking is used in this procedure to enable an MPLS data plane and a BGP control plane within the environment.

The procedure in this document was validated for Contrail Networking 2008 running in EKS 1.16. This procedure should work in EKS 1.16 and all later EKS releases.

Prerequisites

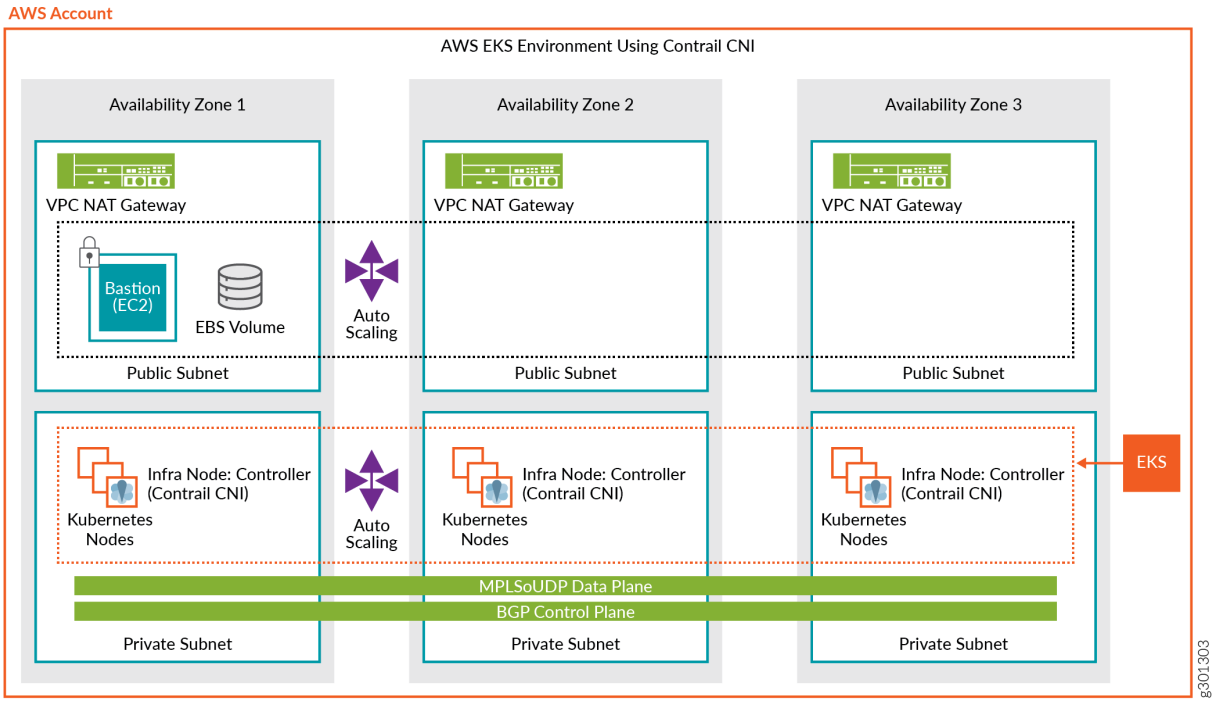
This procedure makes the following assumptions about your environment:

- A Kubernetes client is installed.
- The aws-iam-authenticator is installed to allow authentication into your EKS cluster. See Installing aws-iam-authenticator from AWS.
- AWS CLI is installed. See Installing the AWS CLI from AWS.
- You have obtained the login credentials to the Juniper Networks Contrail docker private secure registry at *hub.juniper.net*. If you need to obtain these credentials, email mailto:contrail-registry@juniper.net.

Install Contrail Networking as the CNI for EKS

This procedure installs Contrail Networking as the CNI in a Kubernetes orchestrated environment in the EKS service within AWS.

The procedure uses the following sample topology:



To install Contrail Networking as the CNI in a Kubernetes-orchestrated environment running in EKS:

1. (Recommended) Review the video procedure of this installation. See the Deep Dive: Contrail SDN and AWS EKS channel on Youtube.
2. Download the EKS deployer:

```
wget https://s3-eu-central-1.amazonaws.com/contrail-one-click-deployers/EKS-Scripts.zip -O
EKS-Scripts.zip
unzip EKS-Scripts.zip
cd contrail-as-the-cni-for-aws-eks/
```

   We recommend running this procedure in the *eu-central-1* default region during your first attempt.

   The procedure supports most AWS regions. You can run the procedure in other regions by updating the *variables.sh* file after familiarizing yourself with the steps.

3. Modify the *variables.sh* file to fit your environment.

   The following fields must be updated:

   - *CLOUDFORMATIONREGION*—the AWS region that your client is configured to use. Cloudformation deploys EKS into this region using the quickstart. The default region is *eu-west-1*.

- *JUNIPERREPONAME*—username to access the Contrail repository. You can email mailto:contrail-registry@juniper.net to obtain your username and password credentials, if needed.
- *JUNIPERREPOPASS*—password to access the Contrail repository. You can email mailto:contrail-registry@juniper.net to obtain your username and password credentials, if needed.
- *RELEASE*—Contrail Networking Release container tag. The container tag is used to identify images in the Contrail repository. The container tag for any Contrail Release 20xx image can be found in README Access to Contrail Registry 20XX.
- *EC2KEYNAME*—an existing keyname in your specified AWS region.
- *BASTIONSSHKEYPATH*—the local path, which is usually the path on your PC, to the private key file for the AWS EC2 key.

Example file:

```
###############################################################################
#complete the below variables for your setup and run the script
###############################################################################
#this is the aws region you are connected to and want to deploy EKS and Contrail into
export CLOUDFORMATIONREGION="eu-west-1"
#this is the region for my quickstart, only change if you plan to deploy your own quickstart
export S3QUICKSTARTREGION="eu-west-1"
export LOGLEVEL="SYS_NOTICE"
#example Juniper docker login, change to yours
export JUNIPERREPONAME="JNPR-FieldUserxxx"
export JUNIPERREPOPASS="Exxxxxxxxxxu"
export RELEASE="2008.121"
export K8SAPIPORT="443"
export PODSN="10.20.0.0/24"
export SERVICESN="10.100.0.0/16"
export FABRICSN="10.20.2.0/24"
export ASN="64513"
export MYEMAIL="example@mail.com"
#example key, change these two to your existing ec2 ssh key name and private key file for
the region
#also don't forget to chmod 0400 [your private key]
export EC2KEYNAME="ContrailKey"
export BASTIONSSHKEYPATH="/Users/user1/Downloads/ContrailKey-1.pem"
```

4. Deploy the *cloudformation-resources.sh* file:

```
. ./cloudformation-resources.sh
```

This step is needed to prepare the environment in some AWS regions.

5. From the AWS CLI, deploy the EKS quickstart stack:

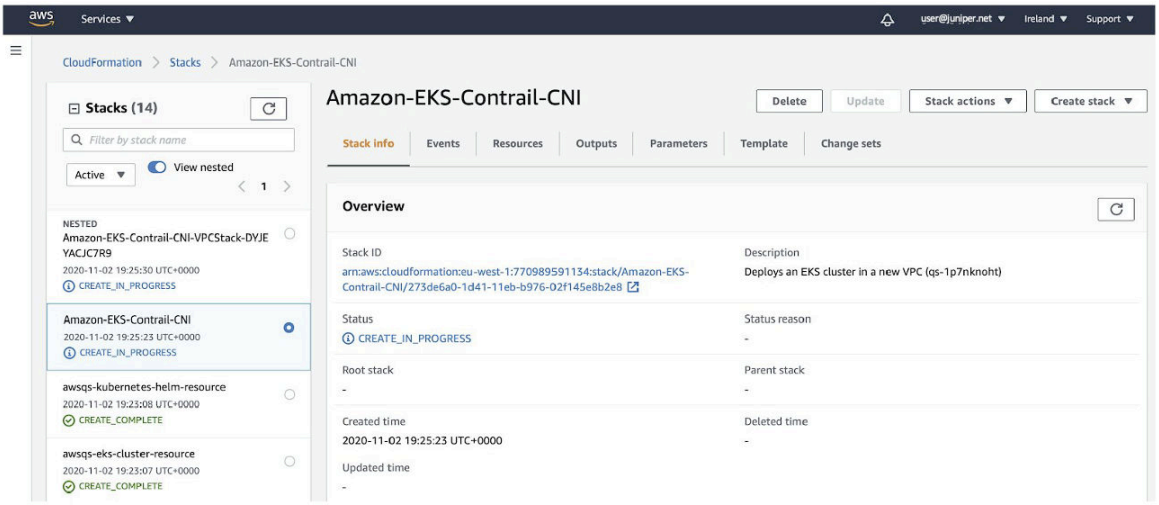
```
. ./eks-ubuntu.sh
```

This step can take 45 minutes or longer to deploy.

> **NOTE**: You can also use the Cloudformation user interface to deploy this stack. You will have to manually complete all parameters if you use the Cloudformation user interface. See this document from AWS.

You can monitor the status of the deployment using this command:

```
aws cloudformation describe-stacks --stack-name Amazon-EKS-Contrail-CNI --output table |
grep StackStatus
||  StackStatus                |  CREATE_COMPLETE
```



6. Return to your PC.

Install the *aws-iam-authenticator* and the register:

```
aws sts get-caller-identity
export CLUSTER=$(aws eks list-clusters --output text | awk -F ' ' '{print $2}')
export REGION=$CLOUDFORMATIONREGION
aws eks --region $REGION update-kubeconfig --name $CLUSTER
```

7. From the Kubernetes CLI, verify your cluster parameters:

```
$ kubectl get nodes
NAME                                          STATUS   ROLES   AGE   VERSION
ip-100-72-0-19.eu-west-1.compute.internal     Ready    (none)  19m   v1.14.8
ip-100-72-0-210.eu-west-1.compute.internal    Ready    (none)  19m   v1.14.8
ip-100-72-0-44.eu-west-1.compute.internal     Ready    (none)  19m   v1.14.8
ip-100-72-1-124.eu-west-1.compute.internal    Ready    (none)  19m   v1.14.8
ip-100-72-1-53.eu-west-1.compute.internal     Ready    (none)  19m   v1.14.8

$ kubectl get pods -A -o wide
NAMESPACE    NAME                       READY   STATUS    AGE  IP             NODE
kube-system  aws-node-7gh94             1/1     Running   21m  100.72.1.124
ip-100-72-1-124.eu-west-1.compute.internal
kube-system  aws-node-bq2x9             1/1     Running   21m  100.72.1.53
ip-100-72-1-53.eu-west-1.compute.internal
kube-system  aws-node-gtdz7             1/1     Running   21m  100.72.0.44
ip-100-72-0-44.eu-west-1.compute.internal
kube-system  aws-node-jr4gn             1/1     Running   21m  100.72.0.19
ip-100-72-0-19.eu-west-1.compute.internal
kube-system  aws-node-zlrbj             1/1     Running   21m  100.72.0.210
ip-100-72-0-210.eu-west-1.compute.internal
kube-system  coredns-6987776bbd-ggsjt   1/1     Running   33m  100.72.0.5
ip-100-72-0-44.eu-west-1.compute.internal
kube-system  coredns-6987776bbd-v7ckc   1/1     Running   33m  100.72.1.77
ip-100-72-1-53.eu-west-1.compute.internal
kube-system  kube-proxy-k6hdc           1/1     Running   21m  100.72.0.210
ip-100-72-0-210.eu-west-1.compute.internal
kube-system  kube-proxy-m59sb           1/1     Running   21m  100.72.0.44
ip-100-72-0-44.eu-west-1.compute.internal
kube-system  kube-proxy-qrrqn           1/1     Running   21m  100.72.0.19
ip-100-72-0-19.eu-west-1.compute.internal
kube-system  kube-proxy-r2vqw           1/1     Running   21m  100.72.1.53
ip-100-72-1-53.eu-west-1.compute.internal
```

```
kube-system  kube-proxy-vzkcd          1/1     Running   21m  100.72.1.124
ip-100-72-1-124.eu-west-1.compute.internal
```

NOTE: Some command output fields removed for readability.

8. Upgrade the worker nodes to the latest EKS version:

```
kubectl apply -f upgrade-nodes.yaml
```

After a few minutes, confirm that the EKS version has updated on all nodes.

In this sample output, the EKS version was updated to 1.16.15.

```
$ kubectl get nodes
NAME                                        STATUS AGE VERSION
ip-100-72-0-174.eu-west-1.compute.internal Ready  19m v1.16.15
ip-100-72-0-93.eu-west-1.compute.internal  Ready  19m v1.16.15
ip-100-72-0-95.eu-west-1.compute.internal  Ready  19m v1.16.15
ip-100-72-1-23.eu-west-1.compute.internal  Ready  19m v1.16.15
ip-100-72-1-85.eu-west-1.compute.internal  Ready  19m v1.16.15
```

NOTE: Command output slightly modified for readability.

After confirming that the EKS version is updated on all nodes, delete the upgrade pods:

```
kubectl delete -f upgrade-nodes.yaml
```

9. Apply the OS fixes for the EC2 worker nodes for Contrail Networking:

```
kubectl apply -f cni-patches.yaml
```

10. Deploy Contrail Networking as the CNI for EKS:

```
. ./deploy-me.sh
```

This step typically takes about 5 minutes to complete.

11. Deploy the setup bastion to provide SSH access for worker nodes:

```
. ./setup-bastion.sh
```

12. Run the Contrail setup file to provide a base Contrail Networking configuration:

```
. ./setup-contrail.sh
```

13. Check Contrail status:

```
$ . ./contrail-status.sh

**************************************
******node is 100.72.0.19
**************************************

################################################################################
#     ___        ______     ___        _      _      ____  _             _    #
#    / \ \      / / ___|   / _ \ _   _(_) ___| | __ / ___|| |_ __ _ _ __| |_  #
#   / _ \ \ /\ / /\___ \  | | | | | | | |/ __| |/ / \___ \| __/ _` | '__| __| #
#  / ___ \ V  V /  ___) | | |_| | |_| | | (__|   <   ___) | || (_| | |  | |_  #
# /_/   \_\_/\_/  |____/   \__\_\\__,_|_|\___|_|\_\ |____/ \__\__,_|_|   \__| #
#--------------------------------------------------------------------------------#
#                      Amazon EKS Quick Start bastion host                       #
#    https://docs.aws.amazon.com/quickstart/latest/amazon-eks-architecture/    #
################################################################################
Unable to find image 'hub.juniper.net/contrail/contrail-status:2008.121' locally
2008.121: Pulling from contrail/contrail-status
f34b00c7da20: Already exists
5a390a7d68be: Already exists
07ca884ff4ba: Already exists
0d7531696e74: Already exists
eda9dec1319f: Already exists
c52247bf208e: Already exists
a5dc1d3a1a1f: Already exists
0297580c16ad: Already exists
e341bea3e3e5: Pulling fs layer
12584a95f49f: Pulling fs layer
367eed12f241: Pulling fs layer
367eed12f241: Download complete
12584a95f49f: Download complete
```

```
e341bea3e3e5: Verifying Checksum
e341bea3e3e5: Download complete
e341bea3e3e5: Pull complete
12584a95f49f: Pull complete
367eed12f241: Pull complete
Digest: sha256:54ba0b280811a45f846d673addd38d4495eec0e7c3a7156e5c0cd556448138a7
Status: Downloaded newer image for hub.juniper.net/contrail/contrail-status:2008.121
Pod              Service         Original Name                          Original Version
State    Id            Status
                 redis           contrail-external-redis                2008-121
running  bf3a68e58446  Up 9 minutes
analytics        api             contrail-analytics-api                 2008-121
running  4d394a8fa343  Up 9 minutes
analytics        collector       contrail-analytics-collector           2008-121
running  1772e258b8b4  Up 9 minutes
analytics        nodemgr         contrail-nodemgr                       2008-121
running  f7cb3d64ff2d  Up 9 minutes
analytics        provisioner     contrail-provisioner                   2008-121
running  4f73934a4744  Up 7 minutes
analytics-alarm  alarm-gen       contrail-analytics-alarm-gen           2008-121
running  472b5d2fd7dd  Up 9 minutes
analytics-alarm  kafka           contrail-external-kafka                2008-121
running  88641415d540  Up 9 minutes
analytics-alarm  nodemgr         contrail-nodemgr                       2008-121
running  35e75ddd5b6e  Up 9 minutes
analytics-alarm  provisioner     contrail-provisioner                   2008-121
running  e82526c4d835  Up 7 minutes
analytics-snmp   nodemgr         contrail-nodemgr                       2008-121
running  6883986527fa  Up 9 minutes
analytics-snmp   provisioner     contrail-provisioner                   2008-121
running  91c7be2f4ac9  Up 7 minutes
analytics-snmp   snmp-collector  contrail-analytics-snmp-collector      2008-121
running  342a11ca471e  Up 9 minutes
analytics-snmp   topology        contrail-analytics-snmp-topology       2008-121
running  f4fa7aa0d980  Up 9 minutes
config           api             contrail-controller-config-api         2008-121
running  17093d75ec93  Up 9 minutes
config           device-manager  contrail-controller-config-devicemgr   2008-121
running  f2c11a305851  Up 6 minutes
config           nodemgr         contrail-nodemgr                       2008-121
running  8322869eaf34  Up 9 minutes
config           provisioner     contrail-provisioner                   2008-121
running  3d2618f9a20b  Up 7 minutes
```

```
config           schema          contrail-controller-config-schema      2008-121
running  e3b7cbff4ef7  Up 6 minutes
config           svc-monitor     contrail-controller-config-svcmonitor  2008-121
running  49c3a0f44466  Up 6 minutes
config-database  cassandra       contrail-external-cassandra            2008-121
running  0eb7d5c56612  Up 9 minutes
config-database  nodemgr         contrail-nodemgr                       2008-121
running  8f1bb252f002  Up 9 minutes
config-database  provisioner     contrail-provisioner                   2008-121
running  4b23ff9ad2bc  Up 7 minutes
config-database  rabbitmq        contrail-external-rabbitmq             2008-121
running  22ab5777e1fa  Up 9 minutes
config-database  zookeeper       contrail-external-zookeeper            2008-121
running  5d1e33e545ae  Up 9 minutes
control          control         contrail-controller-control-control    2008-121
running  05e3ac0e4de3  Up 9 minutes
control          dns             contrail-controller-control-dns        2008-121
running  ea24d045f221  Up 9 minutes
control          named           contrail-controller-control-named      2008-121
running  977ddeb4a636  Up 9 minutes
control          nodemgr         contrail-nodemgr                       2008-121
running  248ae2888c15  Up 9 minutes
control          provisioner     contrail-provisioner                   2008-121
running  c666bd178d29  Up 9 minutes
database         cassandra       contrail-external-cassandra            2008-121
running  9e840c1a5034  Up 9 minutes
database         nodemgr         contrail-nodemgr                       2008-121
running  355984d1689c  Up 9 minutes
database         provisioner     contrail-provisioner                   2008-121
running  60d472efb042  Up 7 minutes
database         query-engine    contrail-analytics-query-engine        2008-121
running  fa56e2c7c765  Up 9 minutes
kubernetes       kube-manager    contrail-kubernetes-kube-manager       2008-121
running  584013153ef8  Up 9 minutes
vrouter          agent           contrail-vrouter-agent                 2008-121
running  7bc5b164ed44  Up 8 minutes
vrouter          nodemgr         contrail-nodemgr                       2008-121
running  5c9201f4308e  Up 8 minutes
vrouter          provisioner     contrail-provisioner                   2008-121
running  ce9d14aaba89  Up 8 minutes
webui            job             contrail-controller-webui-job          2008-121
running  d92079688dda  Up 9 minutes
webui            web             contrail-controller-webui-web          2008-121
```

```
running  8efed46b98d6  Up 9 minutes

vrouter kernel module is PRESENT
== Contrail control ==
control: active
nodemgr: active
named: active
dns: active

== Contrail analytics-alarm ==
nodemgr: active
kafka: active
alarm-gen: active

== Contrail kubernetes ==
kube-manager: active

== Contrail database ==
nodemgr: active
query-engine: active
cassandra: active

== Contrail analytics ==
nodemgr: active
api: active
collector: active

== Contrail config-database ==
nodemgr: active
zookeeper: active
rabbitmq: active
cassandra: active

== Contrail webui ==
web: active
job: active

== Contrail vrouter ==
nodemgr: active
agent: timeout

== Contrail analytics-snmp ==
snmp-collector: active
```

```
nodemgr: active
topology: active

== Contrail config ==
svc-monitor: backup
nodemgr: active
device-manager: backup
api: active
schema: backup
```

NOTE: A vRouter agent timeout might appear in the output. In most cases, the vRouter is working fine and this is a cosmetic issue.

14. Confirm that the pods are running:

```
$ kubectl get pods -A -o wide
NAMESPACE     NAME                                READY   STATUS    RESTARTS   AGE
IP             NODE
kube-system   cni-patches-dgjnc                   1/1     Running   0          44s
100.72.0.210   ip-100-72-0-210.eu-west-1.compute.internal
kube-system   cni-patches-krss8                   1/1     Running   0          44s
100.72.1.53    ip-100-72-1-53.eu-west-1.compute.internal
kube-system   cni-patches-r9vgj                   1/1     Running   0          44s
100.72.1.124   ip-100-72-1-124.eu-west-1.compute.internal
kube-system   cni-patches-wcc9p                   1/1     Running   0          44s
100.72.0.44    ip-100-72-0-44.eu-west-1.compute.internal
kube-system   cni-patches-xqrw8                   1/1     Running   0          44s
100.72.0.19    ip-100-72-0-19.eu-west-1.compute.internal
kube-system   config-zookeeper-2mspv              1/1     Running   0          16m
100.72.0.19    ip-100-72-0-19.eu-west-1.compute.internal
kube-system   config-zookeeper-k65hk              1/1     Running   0          16m
100.72.0.210   ip-100-72-0-210.eu-west-1.compute.internal
kube-system   config-zookeeper-nj2qb              1/1     Running   0          16m
100.72.0.44    ip-100-72-0-44.eu-west-1.compute.internal
kube-system   contrail-agent-2cqbz                3/3     Running   0          16m
100.72.0.44    ip-100-72-0-44.eu-west-1.compute.internal
kube-system   contrail-agent-kbd7v                3/3     Running   0          16m
100.72.1.53    ip-100-72-1-53.eu-west-1.compute.internal
kube-system   contrail-agent-kc4gk                3/3     Running   0          16m
100.72.0.210   ip-100-72-0-210.eu-west-1.compute.internal
```

```
kube-system   contrail-agent-n7shj                3/3     Running   0          16m
100.72.0.19    ip-100-72-0-19.eu-west-1.compute.internal
kube-system   contrail-agent-vckdh                3/3     Running   0          16m
100.72.1.124   ip-100-72-1-124.eu-west-1.compute.internal
kube-system   contrail-analytics-9llmv            4/4     Running   1          16m
100.72.0.19    ip-100-72-0-19.eu-west-1.compute.internal
kube-system   contrail-analytics-alarm-27x47      4/4     Running   1          16m
100.72.0.210   ip-100-72-0-210.eu-west-1.compute.internal
kube-system   contrail-analytics-alarm-rzxgv      4/4     Running   1          16m
100.72.0.44    ip-100-72-0-44.eu-west-1.compute.internal
kube-system   contrail-analytics-alarm-z6w9k      4/4     Running   1          16m
100.72.0.19    ip-100-72-0-19.eu-west-1.compute.internal
kube-system   contrail-analytics-jmjzk            4/4     Running   1          16m
100.72.0.44    ip-100-72-0-44.eu-west-1.compute.internal
kube-system   contrail-analytics-snmp-4prpn       4/4     Running   1          16m
100.72.0.44    ip-100-72-0-44.eu-west-1.compute.internal
kube-system   contrail-analytics-snmp-s4r4g       4/4     Running   1          16m
100.72.0.19    ip-100-72-0-19.eu-west-1.compute.internal
kube-system   contrail-analytics-snmp-z8gxh       4/4     Running   1          16m
100.72.0.210   ip-100-72-0-210.eu-west-1.compute.internal
kube-system   contrail-analytics-xbbfz            4/4     Running   1          16m
100.72.0.210   ip-100-72-0-210.eu-west-1.compute.internal
kube-system   contrail-analyticsdb-gkcnw          4/4     Running   1          16m
100.72.0.210   ip-100-72-0-210.eu-west-1.compute.internal
kube-system   contrail-analyticsdb-k89fl          4/4     Running   1          16m
100.72.0.19    ip-100-72-0-19.eu-west-1.compute.internal
kube-system   contrail-analyticsdb-txkb4          4/4     Running   1          16m
100.72.0.44    ip-100-72-0-44.eu-west-1.compute.internal
kube-system   contrail-configdb-6hp6v             3/3     Running   1          16m
100.72.0.44    ip-100-72-0-44.eu-west-1.compute.internal
kube-system   contrail-configdb-w7sf8             3/3     Running   1          16m
100.72.0.19    ip-100-72-0-19.eu-west-1.compute.internal
kube-system   contrail-configdb-wkcpp             3/3     Running   1          16m
100.72.0.210   ip-100-72-0-210.eu-west-1.compute.internal
kube-system   contrail-controller-config-h4g7l    6/6     Running   4          16m
100.72.0.19    ip-100-72-0-19.eu-west-1.compute.internal
kube-system   contrail-controller-config-pmlcb    6/6     Running   3          16m
100.72.0.210   ip-100-72-0-210.eu-west-1.compute.internal
kube-system   contrail-controller-config-vvklq    6/6     Running   3          16m
100.72.0.44    ip-100-72-0-44.eu-west-1.compute.internal
kube-system   contrail-controller-control-56d46   5/5     Running   0          16m
100.72.0.210   ip-100-72-0-210.eu-west-1.compute.internal
kube-system   contrail-controller-control-t4mrf   5/5     Running   0          16m
```

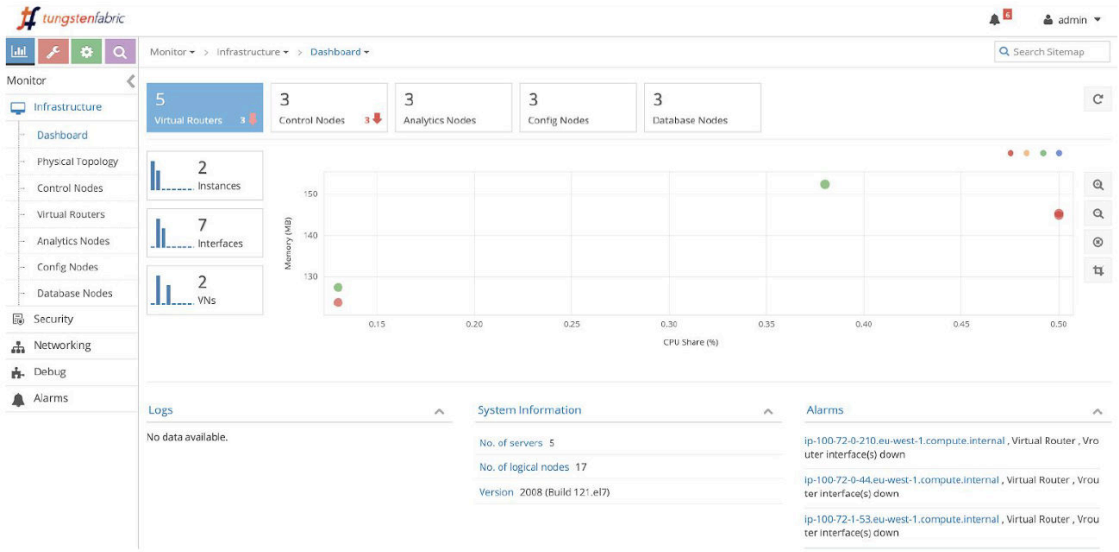
```
100.72.0.19    ip-100-72-0-19.eu-west-1.compute.internal
kube-system   contrail-controller-control-wlhzq   5/5     Running   0          16m
100.72.0.44    ip-100-72-0-44.eu-west-1.compute.internal
kube-system   contrail-controller-webui-t4bzd     2/2     Running   0          16m
100.72.0.19    ip-100-72-0-19.eu-west-1.compute.internal
kube-system   contrail-controller-webui-wkqzz     2/2     Running   0          16m
100.72.0.44    ip-100-72-0-44.eu-west-1.compute.internal
kube-system   contrail-controller-webui-wnf4z     2/2     Running   0          16m
100.72.0.210   ip-100-72-0-210.eu-west-1.compute.internal
kube-system   contrail-kube-manager-fd6mr         1/1     Running   0          3m23s
100.72.0.44    ip-100-72-0-44.eu-west-1.compute.internal
kube-system   contrail-kube-manager-jhl2l         1/1     Running   0          3m33s
100.72.0.210   ip-100-72-0-210.eu-west-1.compute.internal
kube-system   contrail-kube-manager-wnmxt         1/1     Running   0          3m23s
100.72.0.19    ip-100-72-0-19.eu-west-1.compute.internal
kube-system   coredns-6987776bbd-8vzv9            1/1     Running   0          12m
10.20.0.250    ip-100-72-0-19.eu-west-1.compute.internal
kube-system   coredns-6987776bbd-w8h8d            1/1     Running   0          12m
10.20.0.249    ip-100-72-1-124.eu-west-1.compute.internal
kube-system   kube-proxy-k6hdc                    1/1     Running   1          50m
100.72.0.210   ip-100-72-0-210.eu-west-1.compute.internal
kube-system   kube-proxy-m59sb                    1/1     Running   1          50m
100.72.0.44    ip-100-72-0-44.eu-west-1.compute.internal
kube-system   kube-proxy-qrrqn                    1/1     Running   1          50m
100.72.0.19    ip-100-72-0-19.eu-west-1.compute.internal
kube-system   kube-proxy-r2vqw                    1/1     Running   1          50m
100.72.1.53    ip-100-72-1-53.eu-west-1.compute.internal
kube-system   kube-proxy-vzkcd                    1/1     Running   1          50m
100.72.1.124   ip-100-72-1-124.eu-west-1.compute.internal
kube-system   rabbitmq-754b8                      1/1     Running   0          16m
100.72.0.44    ip-100-72-0-44.eu-west-1.compute.internal
kube-system   rabbitmq-bclkx                      1/1     Running   0          16m
100.72.0.210   ip-100-72-0-210.eu-west-1.compute.internal
kube-system   rabbitmq-mk76f                      1/1     Running   0          16m
100.72.0.19    ip-100-72-0-19.eu-west-1.compute.internal
kube-system   redis-8wr29                         1/1     Running   0          16m
100.72.0.19    ip-100-72-0-19.eu-west-1.compute.internal
kube-system   redis-kbtmd                         1/1     Running   0          16m
100.72.0.44    ip-100-72-0-44.eu-west-1.compute.internal
kube-system   redis-rmr8h                         1/1     Running   0          16m
100.72.0.210   ip-100-72-0-210.eu-west-1.compute.internal
```

15. Setup Contrail user interface access.

```
. ./setup-contrail-ui.sh
```

To view the Contrail user interface after performing this step:

a. In your web browser, enter https:// *bastion-public-ip-address*:8143 as the address.



b. Enter your credentials.

The default credentials use *admin* as the user and *contrail123* as the password. We recommend changing these credentials to maximize security.

> **NOTE**: You may get some BGP alarm messages upon login. These messages occur because sample BGP peering relationships are established with gateway devices and federated clusters. Delete the BGP peers in your environment if you want to clear the alarms.

16. Modify the auto scaling groups so that you can stop instances that are not in use.

```
export SCALINGGROUPS=( $(aws autoscaling describe-auto-scaling-groups --query
"AutoScalingGroups[].AutoScalingGroupName" --output text) )
    aws autoscaling suspend-processes --auto-scaling-group-name ${SCALINGGROUPS[0]}
    aws autoscaling suspend-processes --auto-scaling-group-name ${SCALINGGROUPS[1]}
```

NOTE: If you plan on deleting stacks at a later time, you will have to reset this configuration and use the *resume-processes* option before deleting the primary stack:

```
export SCALINGGROUPS=( $(aws autoscaling describe-auto-scaling-groups --query
"AutoScalingGroups[].AutoScalingGroupName" --output text) )
    aws autoscaling resume-processes --auto-scaling-group-name ${SCALINGGROUPS[0]}
    aws autoscaling resume-processes --auto-scaling-group-name ${SCALINGGROUPS[1]}
```

17. (Optional) If you have a public network that you'd like to use for ingress via a gateway, perform the following configuration steps:

    a. Enter https:// *bastion-public-ip-address*:8143 to connect to the web user interface.

    b. Navigate to **Configure** > **Networks** > **k8s-default** > **networks** (left side of page) > **Add network (+)**

    c. In the Add network box, enter the following parameters:

    - Name: **k8s-public**
    - Subnet: Select **ipv4**, then enter the IP address of your public service network.

      Leave all other fields in subnet as default.
    - advanced: **External=tick**
    - advanced: **Share-tick**
    - route target: Click **+**. Enter a route target for your public network. For example, 64512:1000.

    Click **Save**.

18. Deploy a test application on each node:

```
cd TestDaemonSet
./Create.sh
kubectl get pods -A -o wide
```

19. Deploy a multitier test application:

```
cd ../TestApp
./Create.sh
kubectl get deployments -n justlikenetflix
kubectl get pods -o wide -n justlikenetflix
```

```
kubectl get services -n justlikenetflix
kubectl get ingress -n justlikenetflix
```

4

CHAPTER

Contrail Networking with Google Anthos

How to Integrate Kubernetes Clusters using Contrail Networking into Google Cloud Anthos

IN THIS SECTION

NOTE: This topic covers Contrail Networking in Kubernetes-orchestrated environments that are using Contrail Networking Release 21-based releases.

Starting in Release 22.1, Contrail Networking evolved into Cloud-Native Contrail Networking. Cloud-Native Contrail Networking offers significant enhancements to optimize networking performance in Kubernetes-orchestrated environments. We recommend using Cloud-Native Contrail for networking in most Kubernetes-orchestrated environments.

For general information about Cloud-Native Contrail, see the Cloud-Native Contrail Networking Techlibrary homepage.

Anthos is an application management platform developed by Google that provides a consistent development and operations experience for users working in cloud networking clusters that were created in Google Cloud or on third-party cloud platforms. For additional information on Anthos, see the Anthos technical overview from Google Cloud.

The purpose of this document is to illustrate how cloud environments using Kubernetes for orchestration and Contrail Networking for networking can be integrated into the Anthos management platform. This document shows how to create clusters in three separate cloud environments—a private on-premises cloud, a cloud created using the Elastic Kubernetes Service (EKS) in Amazon Web Services (AWS), and a cloud created using the Google Kubernetes Engine (GKE) in the Google Cloud Platform—and add those clusters into Anthos.

This document also provides instructions on introductory configuration and usage tasks after the clouds have been integrated into Anthos. It includes a section on Anthos Configuration management and a

section showing how to load applications from the Google Marketplace into third-party cloud environments.

This document covers the following topics:

Prerequisites

The procedures in this document make the following assumptions about your environment:

- *All Environments*

  The following CLI tools have been downloaded:

  - kubectl. See Install and Set Up kubectl.
  - (Recommend for management) kubectx and kubens. See kubectx + kubens: Power tools for kubectl in Github.

- *Google Cloud Platform*

  - The GCP CLI tools from the Cloud SDK package are operational. See Getting Started with Cloud SDK from Google.

- *Amazon Web Services*

  - This procedure assumes that you have an active AWS account with operating credentials and that the AWS CLI is working on your system. See the Configuring the AWS CLI document from AWS.
  - the eksctl CLI tool is running. See eksctl from the eksctl website.

Creating Kubernetes Clusters

IN THIS SECTION

This sections shows how to create the following Kubernetes clusters:

On-Premises: Creating the Private Kubernetes Cluster

Create an on-premises Kubernetes cluster that includes Contrail Networking. See Installing Kubernetes with Contrail.

The procedure used in this document installs Kubernetes 1.18.9 on a server node running Ubuntu 18.04.5:

```
$ kubectl get nodes -o wide
NAME          STATUS   ROLES    VERSION   OS-IMAGE             KERNEL-VERSION
k8s-master1   Ready    master   v1.18.9   Ubuntu 18.04.5 LTS   4.15.0-118-generic
k8s-master2   Ready    master   v1.18.9   Ubuntu 18.04.5 LTS   4.15.0-118-generic
k8s-master3   Ready    master   v1.18.9   Ubuntu 18.04.5 LTS   4.15.0-118-generic
k8s-node1     Ready    <none>   v1.18.9   Ubuntu 18.04.5 LTS   4.15.0-112-generic
k8s-node2     Ready    <none>   v1.18.9   Ubuntu 18.04.5 LTS   4.15.0-112-generic
```

NOTE: Some output fields removed for readability.

After deploying the Kubernetes cluster, Contrail is installed using a single YAML file.

```
$ kubectl get po -n kube-system
NAME                                  READY   STATUS    RESTARTS   AGE
config-zookeeper-4klts                1/1     Running   0          19h
config-zookeeper-cs2fk                1/1     Running   0          19h
config-zookeeper-wgrtb                1/1     Running   0          19h
contrail-agent-ch8kv                  3/3     Running   2          19h
contrail-agent-kh9cf                  3/3     Running   1          19h
contrail-agent-kqtmz                  3/3     Running   0          19h
contrail-agent-m6nrz                  3/3     Running   1          19h
contrail-agent-qgzxt                  3/3     Running   3          19h
contrail-analytics-6666s              4/4     Running   1          19h
contrail-analytics-jrl5x              4/4     Running   4          19h
contrail-analytics-x756g              4/4     Running   4          19h
contrail-configdb-2h7kd               3/3     Running   4          19h
contrail-configdb-d57tb               3/3     Running   4          19h
contrail-configdb-zpmsq               3/3     Running   4          19h
contrail-controller-config-c2226      6/6     Running   9          19h
contrail-controller-config-pbbmz      6/6     Running   5          19h
```

```
contrail-controller-config-zqkm6      6/6     Running   4          19h
contrail-controller-control-2kz4c     5/5     Running   2          19h
contrail-controller-control-k522d     5/5     Running   0          19h
contrail-controller-control-nr54m     5/5     Running   2          19h
contrail-controller-webui-5vxl7       2/2     Running   0          19h
contrail-controller-webui-mzpdv       2/2     Running   1          19h
contrail-controller-webui-p8rc2       2/2     Running   1          19h
contrail-kube-manager-88c4f           1/1     Running   0          19h
contrail-kube-manager-fsz2z           1/1     Running   0          19h
contrail-kube-manager-qc27b           1/1     Running   0          19h
coredns-684f7f6cb4-4mmgc              1/1     Running   0          93m
coredns-684f7f6cb4-dvpjk              1/1     Running   0          107m
coredns-684f7f6cb4-m6sj7              1/1     Running   0          84m
coredns-684f7f6cb4-nfkfh              1/1     Running   0          84m
coredns-684f7f6cb4-tk48d              1/1     Running   0          86m
etcd-k8s-master1                      1/1     Running   0          94m
etcd-k8s-master2                      1/1     Running   0          95m
etcd-k8s-master3                      1/1     Running   0          92m
kube-apiserver-k8s-master1            1/1     Running   0          94m
kube-apiserver-k8s-master2            1/1     Running   0          95m
kube-apiserver-k8s-master3            1/1     Running   0          92m
kube-controller-manager-k8s-master1   1/1     Running   0          94m
kube-controller-manager-k8s-master2   1/1     Running   0          95m
kube-controller-manager-k8s-master3   1/1     Running   0          92m
kube-proxy-975tn                      1/1     Running   0          108m
kube-proxy-9qzc9                      1/1     Running   0          108m
kube-proxy-fgwqt                      1/1     Running   0          109m
kube-proxy-n6nnq                      1/1     Running   0          109m
kube-proxy-wf289                      1/1     Running   0          108m
kube-scheduler-k8s-master1            1/1     Running   0          94m
kube-scheduler-k8s-master2            1/1     Running   0          95m
kube-scheduler-k8s-master3            1/1     Running   0          90m
rabbitmq-82lmk                        1/1     Running   0          19h
rabbitmq-b2lz8                        1/1     Running   0          19h
rabbitmq-f2nfc                        1/1     Running   0          19h
redis-42tkr                           1/1     Running   0          19h
redis-bj76v                           1/1     Running   0          19h
redis-ctzhg                           1/1     Running   0          19h
```

You should also configure user roles using role-based access control (RBAC). This example shows you how to grant the customer-admin RBAC role to all Kubernetes namespaces:

```
$ kubectl create clusterrolebinding permissive-binding \
  --clusterrole=cluster-admin \
  --user=admin \
  --user=kubelet \
  --group=system:serviceaccounts

kubectl auth can-i '*' '*' --all-namespaces
```

Amazon Web Services (AWS): Install Contrail Networking in an Elastic Kubernetes Service (EKS) Environment

To create a Kubernetes cluster within the Elastic Kubernetes Service (EKS) in AWS, perform following procedure using the *eksctl* CLI tool :

1. Create the cluster. To create a cluster that includes Contrail running in Kubernetes within EKS, follow the instructions in "How to Install Contrail Networking within an Amazon Elastic Kubernetes Service (EKS) Environment in AWS" on page 122.
2. View the nodes:

```
$ kubectl get nodes -o wide
NAME                                          STATUS  ROLES   VERSION   OS-IMAGE
KERNEL-VERSION
ip-100-72-0-119.eu-central-1.compute.internal  Ready   infra   v1.16.15  Ubuntu 18.04.3 LTS
4.15.0-1054-aws
ip-100-72-0-220.eu-central-1.compute.internal  Ready   <none>  v1.16.15  Ubuntu 18.04.3 LTS
4.15.0-1054-aws
ip-100-72-0-245.eu-central-1.compute.internal  Ready   infra   v1.16.15  Ubuntu 18.04.3 LTS
4.15.0-1054-aws
ip-100-72-1-116.eu-central-1.compute.internal  Ready   infra   v1.16.15  Ubuntu 18.04.3 LTS
4.15.0-1054-aws
ip-100-72-1-67.eu-central-1.compute.internal   Ready   <none>  v1.16.15  Ubuntu 18.04.3 LTS
4.15.0-1054-aws
```

3. View the pods.

Note the Contrail pods to confirm that Contrail is running in the environment.

```
$ kubectl get pods --all-namespaces
NAME                                        READY   STATUS    RESTARTS   AGE
cni-patches-2jm8n                           1/1     Running   0          4d21h
cni-patches-2svt6                           1/1     Running   0          4d21h
cni-patches-9mpss                           1/1     Running   0          4d21h
cni-patches-fdbws                           1/1     Running   0          4d21h
cni-patches-ggdph                           1/1     Running   0          4d21h
config-management-operator-5994858fbb-9xvmx 1/1     Running   0          2d20h
config-zookeeper-fz5zv                      1/1     Running   0          4d21h
config-zookeeper-n7wgk                      1/1     Running   0          4d21h
config-zookeeper-pjffv                      1/1     Running   0          4d21h
contrail-agent-69zpn                        3/3     Running   0          4d21h
contrail-agent-gqtfv                        3/3     Running   0          4d21h
contrail-agent-lb8tj                        3/3     Running   0          4d21h
contrail-agent-lrrp8                        3/3     Running   0          4d21h
contrail-agent-z4qjc                        3/3     Running   0          4d21h
contrail-analytics-2bv7c                    4/4     Running   0          4d21h
contrail-analytics-4jgq6                    4/4     Running   0          4d21h
contrail-analytics-sn6cj                    4/4     Running   0          4d21h
contrail-configdb-bhvlw                     3/3     Running   0          4d21h
contrail-configdb-kvvk4                     3/3     Running   0          4d21h
contrail-configdb-vbczf                     3/3     Running   0          4d21h
contrail-controller-config-8vrrm            6/6     Running   1          4d21h
contrail-controller-config-lxsms            6/6     Running   3          4d21h
contrail-controller-config-r7ncm            6/6     Running   4          4d21h
contrail-controller-control-5795l           5/5     Running   0          4d21h
contrail-controller-control-dz6pl           5/5     Running   0          4d21h
contrail-controller-control-qznf9           5/5     Running   0          4d21h
contrail-controller-webui-2g5jx             2/2     Running   0          4d21h
contrail-controller-webui-7kg48             2/2     Running   0          4d21h
contrail-controller-webui-ww5z9             2/2     Running   0          4d21h
contrail-kube-manager-2jhzc                 1/1     Running   2          4d21h
contrail-kube-manager-8psh9                 1/1     Running   0          4d21h
contrail-kube-manager-m8zg7                 1/1     Running   1          4d21h
coredns-5fdf64ff8-bf2fc                     1/1     Running   0          4d21h
<additional output removed for readability>
```

4. Use role-based access control (RBAC) to define access roles for users accessing cluster resources.

This sample configuration illustrates how to configure RBAC to set the cluster admin role to all namespaces in the cluster. The remaining procedures in this document assume that the user has cluster admin access to all cluster resources.

```
$ kubectl create clusterrolebinding permissive-binding \
  --clusterrole=cluster-admin \
  --user=admin \
  --user=kubelet \
  --group=system:serviceaccounts

kubectl auth can-i '*' '*' --all-namespaces
```

Other RBAC options are available and the discussion of those options is beyond the scope of this document. See Using RBAC Authorization from Kubernetes.

Google Cloud Platform (GCP): Creating a Kubernetes Cluster in Google Kubernetes Engine (GKE)

To create a Kubernetes cluster in Google Cloud using the Google Kubernetes Engine (GKE):

1. Create a project by entering the following command:

   ```
   $ gcloud init
   ```

   Follow the onscreen process to create the project.
2. Verify that the project was created:

   ```
   $ gcloud projects list
   ```
3. Select a project:

   ```
   $ gcloud config set project contrail-k8s-289615
   ```
4. Assign the required IAM user roles.

   In this sample configuration, IAM user roles are set so that users have complete control of all registration tasks. For more information on IAM user role options, see Grant the required IAM roles to the user registering the cluster document from Google Cloud.

   ```
   PROJECT_ID=contrail-k8s-289615
   $ gcloud projects add-iam-policy-binding ${PROJECT_ID} \
   ```

```
 --member user:[GCP_EMAIL_ADDRESS] \
 --role=roles/gkehub.admin \
 --role=roles/iam.serviceAccountAdmin \
 --role=roles/iam.serviceAccountKeyAdmin \
 --role=roles/resourcemanager.projectIamAdmin
```

5. APIs are required to access resources in Google Cloud. See the Enable the required APIs in your project content in Google Cloud.

   To enable the APIs required for this project:

```
gcloud services enable \
 --project=${PROJECT_ID} \
 container.googleapis.com \
 compute.googleapis.com \
 gkeconnect.googleapis.com \
 gkehub.googleapis.com \
 cloudresourcemanager.googleapis.com \
 cloudtrace.googleapis.com \
 anthos.googleapis.com \
 iamcredentials.googleapis.com \
 meshca.googleapis.com \
 meshconfig.googleapis.com \
 meshtelemetry.googleapis.com \
 monitoring.googleapis.com \
 logging.googleapis.com \
 runtimeconfig.googleapis.com
```

6. Create the Kubernetes cluster:

```
$ export KUBECONFIG=gke-config
$ gcloud container clusters create gke-cluster-1 \
--zone "europe-west2-b" \
--disk-type "pd-ssd" \
--disk-size "150GB" \
--machine-type "n2-standard-4" \
--num-nodes=3 \
--image-type "COS" \
--enable-stackdriver-kubernetes \
--addons HorizontalPodAutoscaling,HttpLoadBalancing,Istio,CloudRun \
--istio-config auth=MTLS_PERMISSIVE \
--cluster-version "1.17.9-gke.1504"
```

```
kubectl create clusterrolebinding cluster-admin-binding \
  --clusterrole cluster-admin \
  --user $(gcloud config get-value account)
```

7. To assist with later management tasks, merge the cloud configurations into a single configuration.

   In this example, the on-premises, EKS, and GKE configuration directories are copied into the same directory:

```
$ cp *-config ~/.kube
$ KUBECONFIG=$HOME/.kube/eks-config:$HOME/.kube/contrail-config:$HOME/.kube/gke-config
kubectl config view --merge --flatten > $HOME/.kube/config

$ kubectx gke_contrail-k8s-289615_europe-west2-b_gke-cluster-1
$ kubectx gke=.

$ kubectx arn:aws:eks:eu-central-1:927874460243:cluster/EKS-YC0U0TU5
$ kubectx eks-contrail=.

$ kubectx kubernetes-admin@kubernetes
$ kubectx onprem-k8s-contrail=.
```

8. Confirm the contexts representing the Kubernetes clusters.

   This output illustrates an environment where an on-premises and an EKS cluster were created using the procedures in this document.

```
$ kubectx
eks-contrail
gke
onprem-k8s-contrail
```

Preparing Your Clusters for Anthos

IN THIS SECTION

This section describes how to prepare your Google Cloud Platform account and your clusters for Anthos.

It includes the following sections:

Configure Your Google Cloud Platform Account for Anthos

You need to create a service account in GCP and provision a JSON file with the Google Cloud service account credentials for external clusters—in this example, the external clusters are the on-premises cloud and the AWS cloud networks—before you can connect the clusters created by third-party providers into Google Anthos.

To configure your Google Cloud Platform for Anthos:

1. Create the Google Cloud service account.

   This step includes creating a project ID and creating an IAM profile for the account:

```
$ PROJECT_ID=contrail-k8s-289615
$ SERVICE_ACCOUNT_NAME=anthos-connect

$ gcloud iam service-accounts create ${SERVICE_ACCOUNT_NAME} --project=${PROJECT_ID}
```

2. Bind the gkehub.connect IAM role to the service account:

```
$ gcloud projects add-iam-policy-binding ${PROJECT_ID} \
 --member="serviceAccount:${SERVICE_ACCOUNT_NAME}@${PROJECT_ID}.iam.gserviceaccount.com" \
 --role="roles/gkehub.connect"
```

3. Create a private key JSON file for the service account in the current directory. This JSON file is required to register the clusters.

```
$ gcloud iam service-accounts keys create ./${SERVICE_ACCOUNT_NAME}-svc.json \
  --iam-account=${SERVICE_ACCOUNT_NAME}@${PROJECT_ID}.iam.gserviceaccount.com \
  --project=${PROJECT_ID}
```

How to Register an External Kubernetes Cluster to Google Connect

The Google Connect feature is part of Anthos and it allows you to connect your Kubernetes clusters—including clusters created outside Google Cloud—into Google Cloud. This support within Google Connect provides the external Kubernetes clusters with the ability to use many cluster and workload management features from Google Cloud, including the Cloud Console unified user interface. See Connect Overview from Google for additional information on Google Connect and Cloud Console from Google for additional information on Google Cloud Console.

To register external Kubernetes clusters into Google connect:

1. Connect the cluster to the Google Kubernetes Engine (GKE). A GKE agent which is responsible for allowing the cloud network to communicate with the GKE hub is installed in the cloud network during this step.

   - To add an on-premises cluster:

     ```
     gcloud container hub memberships register onpremk8s-contrail-cluster-1 \
        --project=${PROJECT_ID} \
        --context=onprem-k8s-contrail \
        --kubeconfig=$HOME/.kube/config \
        --service-account-key-file=./anthos-connect-svc.json
     ```

     To confirm that the GKE connect agent is running after the command is executed:

     ```
     $ kubectx onprem-k8s-contrail
     Switched to context "onprem-k8s-contrail".

     $ kubectl get pods -n gke-connect
     NAMESPACE      NAME                                          READY   STATUS
     gke-connect    gke-connect-agent-20200918-01-00-7bc77884d-st4r2   1/1     Running
     ```

     NOTE: SNAT usually needs to be enabled in Contrail Networking to allow the GKE connect agent to connect to the Internet.

   - To add a cluster running in Elastic Kubernetes Service (EKS) on Amazon Web Services (AWS):

     ```
     gcloud container hub memberships register eks-contrail-cluster-1 \
        --project=${PROJECT_ID} \
        --context=eks-contrail \
     ```

```
   --kubeconfig=$HOME/.kube/config \
   --service-account-key-file=./anthos-connect-svc.json
```

To confirm that the GKE connect agent is running after executing the command:

```
$ kubectx eks-contrail
Switched to context "eks-contrail".

$ kubectl get pods -n gke-connect
NAME                                                 READY   STATUS
gke-connect-agent-20201002-01-00-5749bfc847-qhvft    1/1     Running
```

- To add a cluster running in GKE on Google Cloud Platform:

```
gcloud container hub memberships register gke-cluster-1 \
--project=${PROJECT_ID} \
--gke-cluster=europe-west2-b/gke-cluster-1 \
--service-account-key-file=./anthos-connect-svc.json
```

To confirm that the GKE connect agent is running in the cluster after executing the command.

Note that the on-premises and AWS EKS clusters that were connected to the GKE hub in the earlier bulletpoints are also visible in the command output.

```
$ gcloud container hub memberships list
NAME                          EXTERNAL_ID
onpremk8s-contrail-cluster-1  78f7890b-3a43-4bc7-8fd9-44c76953781b
eks-contrail-cluster-1        42e532ba-a0d9-4087-baed-647be8bca7e9
gke-cluster-1                 6671599e-87af-461b-aff9-7105ebda5c66
```

2. A bearer token will be used in this procedure to login to the external clusters from the Google Anthos Console. A Kubernetes service account (KSA) will be created in the cluster to generate this bearer token.

   To create and apply this bearer token for an on-premises cluster:

   a. Create and apply the node-reader role in role-based access control (RBAC) using the *node-reader* role in the *node-reader.yaml* file:

```
$ cat <<EOF > node-reader.yaml
kind: ClusterRole
```

```
apiVersion: rbac.authorization.k8s.io/v1
metadata:
 name: node-reader
rules:
- apiGroups: [""]
  resources: ["nodes"]
  verbs: ["get", "list", "watch"]
EOF
F

$ kubectx onpremk8s-contrail-cluster-1

$ kubectl apply -f node-reader.yaml
```

b. Create and authorize a Kubernetes service account (KSA):

```
$ KSA_NAME=anthos-sa
$ kubectl create serviceaccount ${KSA_NAME}
$ kubectl create clusterrolebinding anthos-view --clusterrole view --serviceaccount
default:${KSA_NAME}
$ kubectl create clusterrolebinding anthos-node-reader --clusterrole node-reader --
serviceaccount default:${KSA_NAME}
$ kubectl create clusterrolebinding anthos-cluster-admin --clusterrole cluster-admin --
serviceaccount default:${KSA_NAME}
```

c. Acquire the bearer token for the KSA:

```
$ SECRET_NAME=$(kubectl get serviceaccount ${KSA_NAME} -o jsonpath='{$.secrets[0].name}')
$ kubectl get secret ${SECRET_NAME} -o jsonpath='{$.data.token}' | base64 --decode
```

d. Use the output token in the Cloud Console to login to the cluster.

To create and apply this bearer token for an EKS cluster in AWS:

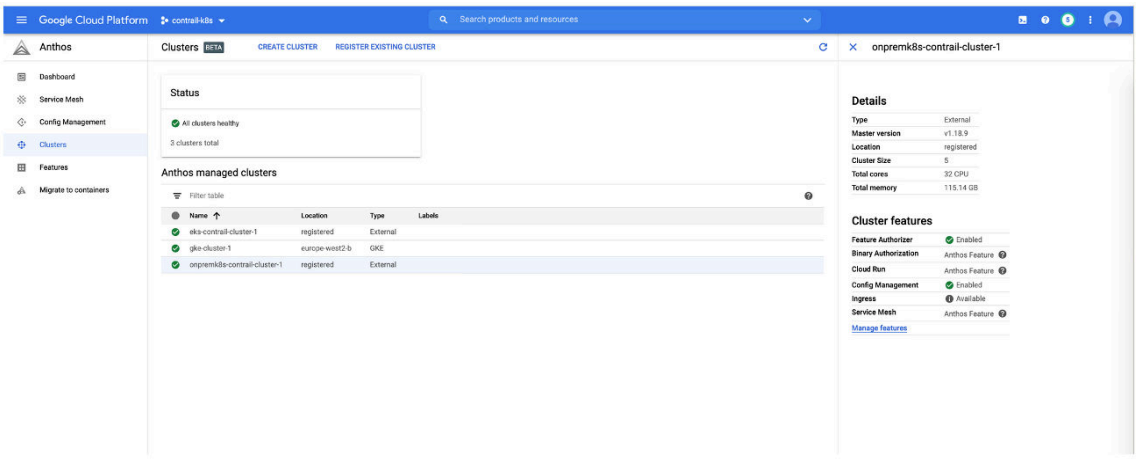
a. Perform the parallel steps for an on-premises cluster for an AWS EKS cluster:

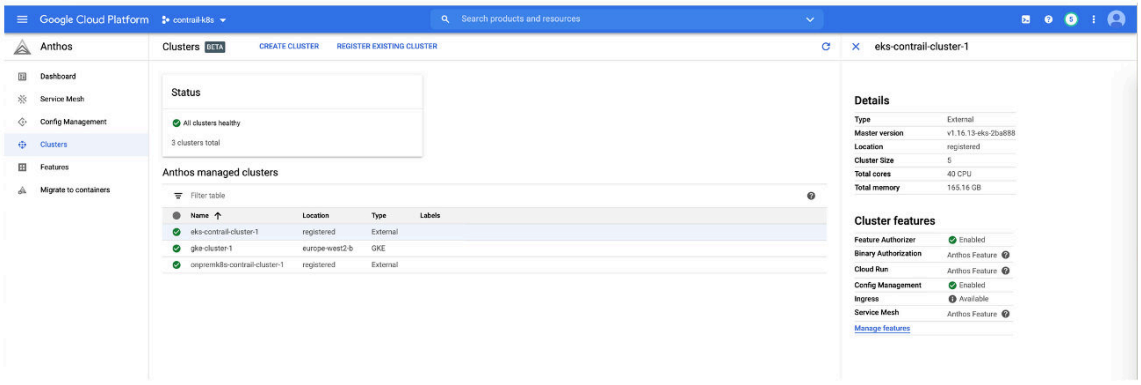
```
$ kubectx eks-contrail
$ $ kubectl apply -f node-reader.yaml

$ kubectl create serviceaccount ${KSA_NAME}
```

```
$ kubectl create clusterrolebinding anthos-view --clusterrole view --serviceaccount
default:${KSA_NAME}
$ kubectl create clusterrolebinding anthos-node-reader --clusterrole node-reader --
serviceaccount default:${KSA_NAME}
$ kubectl create clusterrolebinding anthos-cluster-admin --clusterrole cluster-admin --
serviceaccount default:${KSA_NAME}

$ SECRET_NAME=$(kubectl get serviceaccount ${KSA_NAME} -o jsonpath='{$.secrets[0].name}')
$ kubectl get secret ${SECRET_NAME} -o jsonpath='{$.data.token}' | base64 --decode
```

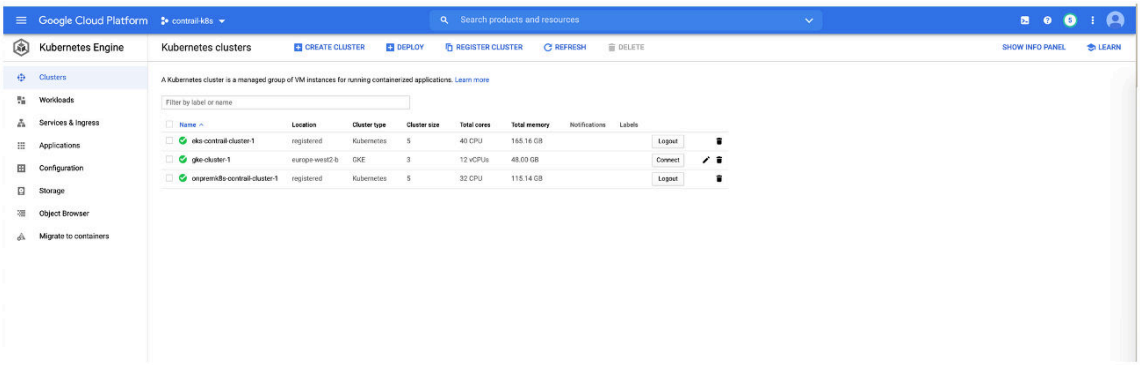
3. Verify the clusters.

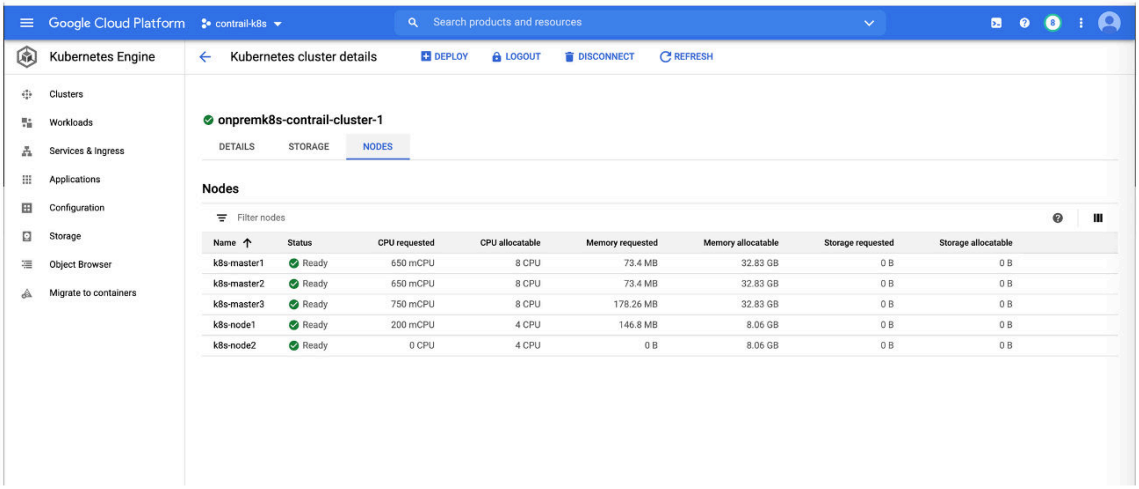
   a. Verify that the clusters are visible in Anthos:

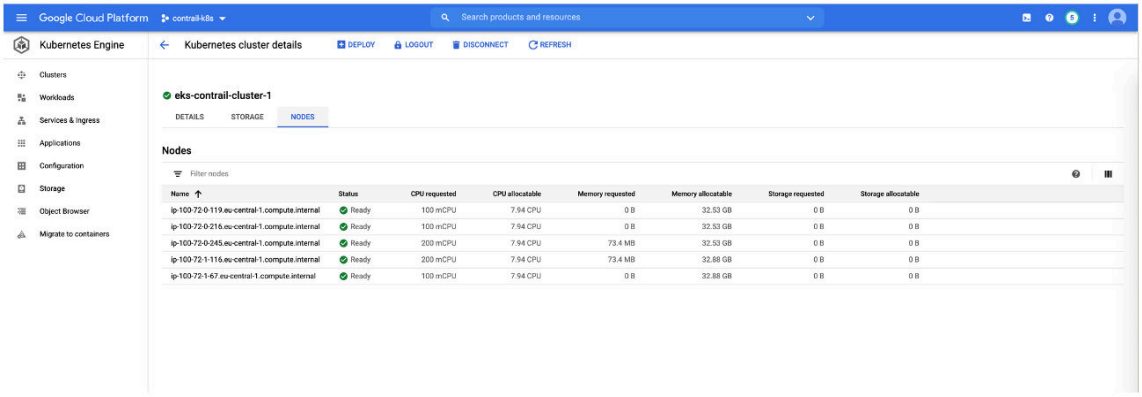




   b. Verify that cluster details are visible from the Kubernetes Engine tab:


Google Cloud Platform
contrail-k8s
Search products and resources
Kubernetes Engine
Kubernetes clusters
CREATE CLUSTER
DEPLOY
REGISTER CLUSTER
REFRESH
DELETE
SHOW INFO PANEL
LEARN
Clusters
Workloads
Services & Ingress
Applications
Configuration
Storage
Object Browser
Migrate to containers
A Kubernetes cluster is a managed group of VM instances for running containerized applications. Learn more
Filter by label or name
Name
Location
Cluster type
Cluster size
Total cores
Total memory
Notifications
Labels
eks-contrail-cluster-1
registered
Kubernetes
5
40 CPU
165.16 GB
Logout
gke-cluster-1
europe-west2-b
GKE
3
12 vCPUs
48.00 GB
Connect
onpremk8s-contrail-cluster-1
registered
Kubernetes
5
32 CPU
115.14 GB
Logout


Google Cloud Platform
contrail-k8s
Search products and resources
Kubernetes Engine
Kubernetes cluster details
DEPLOY
LOGOUT
DISCONNECT
REFRESH
Clusters
Workloads
Services & Ingress
Applications
Configuration
Storage
Object Browser
Migrate to containers
onpremk8s-contrail-cluster-1
DETAILS
STORAGE
NODES
Nodes
Filter nodes
Name
Status
CPU requested
CPU allocatable
Memory requested
Memory allocatable
Storage requested
Storage allocatable
k8s-master1
Ready
650 mCPU
8 CPU
73.4 MB
32.83 GB
0 B
0 B
k8s-master2
Ready
650 mCPU
8 CPU
73.4 MB
32.83 GB
0 B
0 B
k8s-master3
Ready
750 mCPU
8 CPU
178.26 MB
32.83 GB
0 B
0 B
k8s-node1
Ready
200 mCPU
4 CPU
146.8 MB
8.06 GB
0 B
0 B
k8s-node2
Ready
0 CPU
4 CPU
0 B
8.06 GB
0 B
0 B


Google Cloud Platform
contrail-k8s
Search products and resources
Kubernetes Engine
Kubernetes cluster details
DEPLOY
LOGOUT
DISCONNECT
REFRESH
Clusters
Workloads
Services & Ingress
Applications
Configuration
Storage
Object Browser
Migrate to containers
eks-contrail-cluster-1
DETAILS
STORAGE
NODES
Nodes
Filter nodes
Name
Status
CPU requested
CPU allocatable
Memory requested
Memory allocatable
Storage requested
Storage allocatable
ip-100-72-0-119.eu-central-1.compute.internal
Ready
100 mCPU
7.94 CPU
0 B
32.53 GB
0 B
0 B
ip-100-72-0-216.eu-central-1.compute.internal
Ready
100 mCPU
7.94 CPU
0 B
32.53 GB
0 B
0 B
ip-100-72-0-245.eu-central-1.compute.internal
Ready
200 mCPU
7.94 CPU
73.4 MB
32.53 GB
0 B
0 B
ip-100-72-1-116.eu-central-1.compute.internal
Ready
200 mCPU
7.94 CPU
73.4 MB
32.88 GB
0 B
0 B
ip-100-72-1-67.eu-central-1.compute.internal
Ready
100 mCPU
7.94 CPU
0 B
32.88 GB
0 B
0 B

Deploying GCP Applications into Third Party Clusters That are Integrated Into Anthos

IN THIS SECTION

This section shows how to deploy an application from Google Marketplace onto clusters created outside GCP and integrated into Anthos.

It includes the following sections:

On-premises Kubernetes cluster: How to Deploy Applications from the GCP Marketplace Onto an On-premises Cloud

This procedure shows how to add an application—illustrated using the PostgreSQL application—from the Google Cloud Marketplace into an on-premises cluster that was built outside of Google Cloud and integrated into Anthos.

Perform the following steps to deploy the application:

1. Create a namespace called *application-system* for Google Cloud Marketplace components.

   You must create this namespace to deploy applications to Google Anthos in an on-premises cluster. The namespace must be called *application-system* and must apply an imagePullSecret credential to the default service account for the namespace.

   ```
   $ kubectl create ns application-system

   $ kubens application-system
   Context "kubernetes-admin@kubernetes" modified.
   Active namespace is "application-system".
   ```

2. Create a service account and download an associated JSON token.

This step is required to pull images from the Google Cloud Repository.

```
$ PROJECT_ID=contrail-k8s-289615

$ gcloud iam service-accounts create gcr-sa \
    --project=${PROJECT_ID}

$ gcloud iam service-accounts list \
    --project=${PROJECT_ID}

$ gcloud projects add-iam-policy-binding ${PROJECT_ID} \
 --member="serviceAccount:gcr-sa@${PROJECT_ID}.iam.gserviceaccount.com" \
 --role="roles/storage.objectViewer"

$ gcloud iam service-accounts keys create ./gcr-sa.json \
  --iam-account="gcr-sa@${PROJECT_ID}.iam.gserviceaccount.com" \
  --project=${PROJECT_ID}
```

3. Create a secret credential with the contents of the token:

```
$ kubectl create secret docker-registry gcr-json-key \
--docker-server=https://marketplace.gcr.io \
--docker-username=_json_key \
--docker-password="$(cat ./gcr-sa.json)" \
--docker-email=[GCP_EMAIL_ADDRESS]
```

4. Patch the default service account within the namespace to use the secret credential for pulling images from the Google Cloud Repository instead of the Docker Hub.

```
$ kubectl patch serviceaccount default -p '{"imagePullSecrets": [{"name": "gcr-json-key"}]}'
```

5. Annotate the *application-system* namespace to enable the deployment of Kubernetes Applications from the GCP Marketplace:

```
$ kubectl annotate namespace application-system marketplace.cloud.google.com/
imagePullSecret=gcr-json-key
```

6. Create a default storage class named *standard* by either renaming your storage class to *standard* or creating a new storage class. This step is necessary because the GCP Marketplace expects a storage class named *standard* as the default storage class.

To rename your storage class:

```
$ cat sc.yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: standard
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
provisioner: kubernetes.io/no-provisioner
reclaimPolicy: Delete
volumeBindingMode: WaitForFirstConsumer

$ kubectl get sc
NAME                 PROVISIONER                    AGE
standard (default)   kubernetes.io/no-provisioner   6m14s
```

To create a new storage class, see Setup a Local Persistent Volume for a Kubernetes cluster.

This namespace will be utilized by the GCP Marketplace Apps to dynamically provision Persistent Volume (PV) and Persistent Volume Claim (PVC).

7. Create and configure a namespace for an app that will be deployed from the GCP Marketplace.

   We'll illustrate how to deploy PostgreSQL in this document.

```
$ kubectl create ns pgsql

$ kubens pgsql

$ kubectl create secret docker-registry gcr-json-key \
 --docker-server=https://gcr.io \
--docker-username=_json_key \
--docker-password="$(cat ./gcr-sa.json)" \
--docker-email=[GCP_EMAIL_ADDRESS]
```

8. Patch the default service account within the namespace to use the secret credential to pull images from the Google Cloud repository instead of Docker Hub.

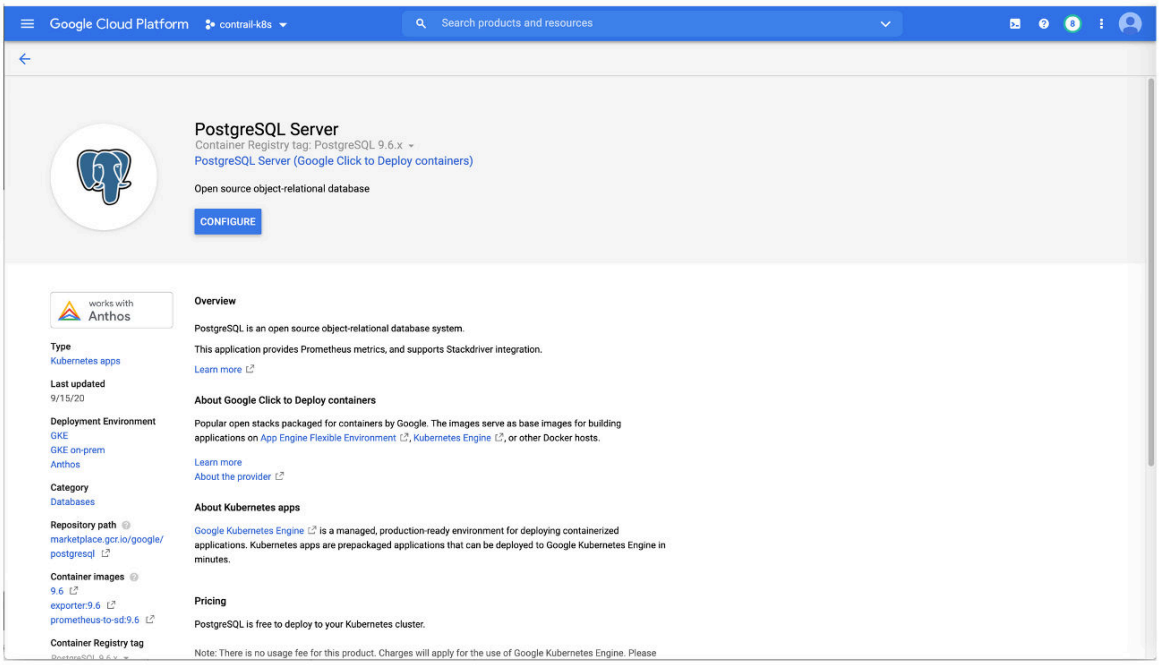
   In this sample case, the default service account is within the pgsql namespace.

```
$ kubectl patch serviceaccount default -p '{"imagePullSecrets": [{"name": "gcr-json-key"}]}'
```

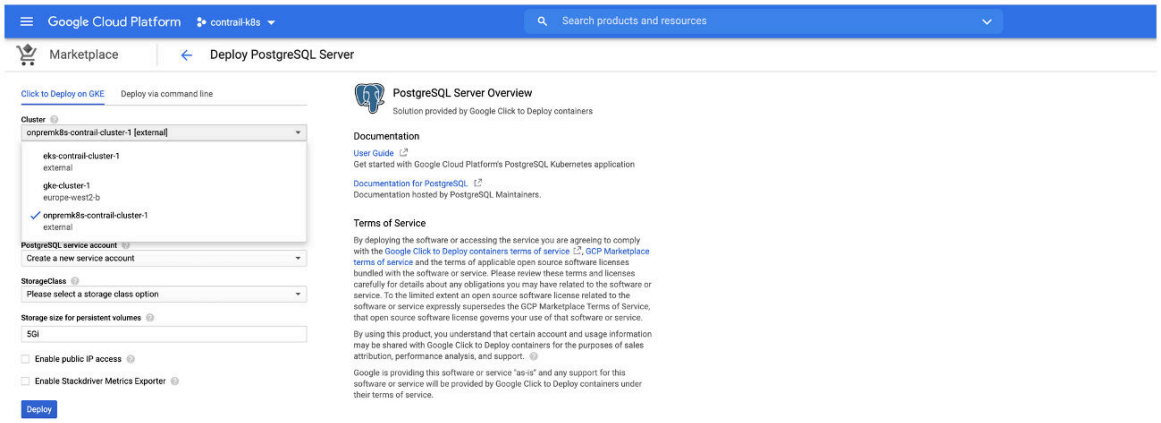
9. Annotate the namespace—in this case, the *pgsql* namespace—to enable the deployment of Kubernetes Apps from the GCP Marketplace:

```
$ kubectl annotate namespace pgsql marketplace.cloud.google.com/imagePullSecret=gcr-json-key
```

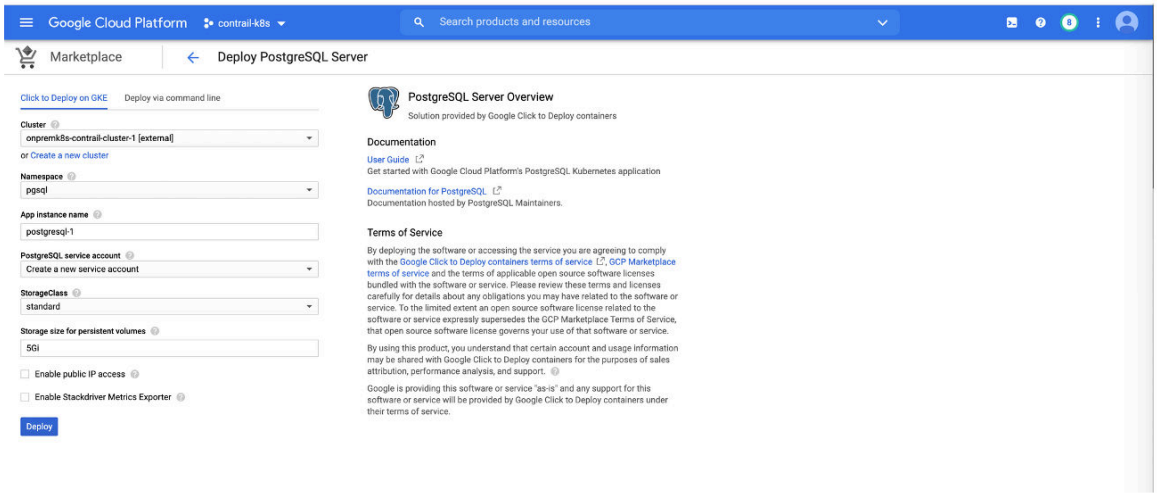
10. Choose the app—in this case, PostgresSQL Server—from GCP Marketplace and click on Configure to start the deployment procedure.



11. Choose the *contrail-cluster-1* external cluster from the **Cluster** drop-down menu:



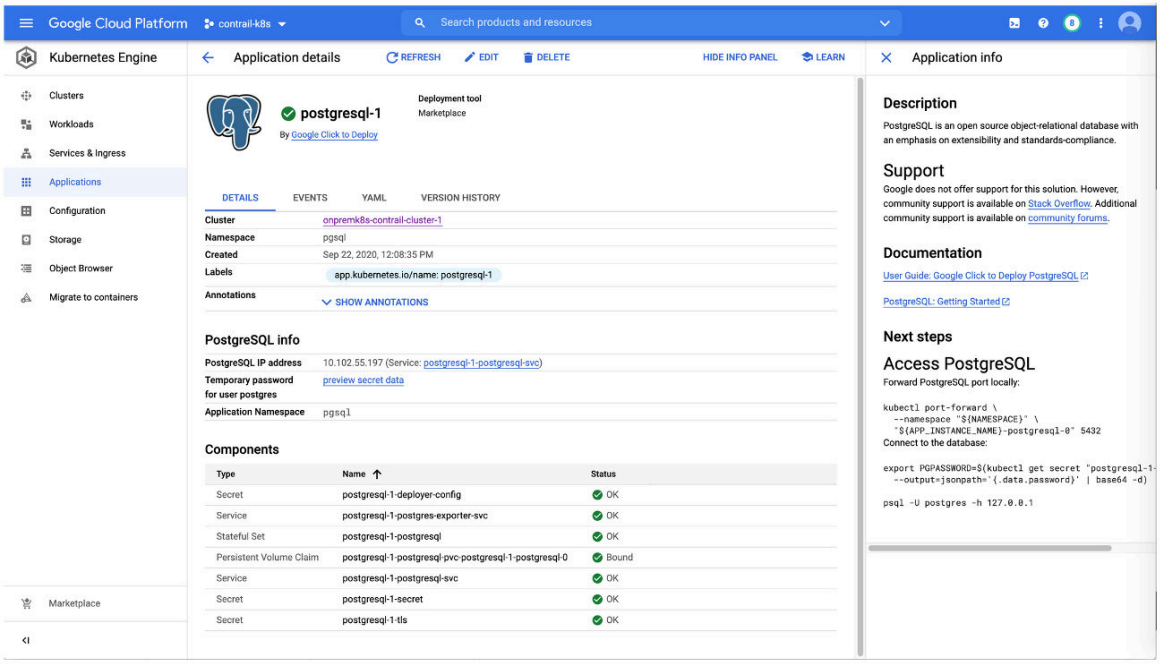
12. Select the namespace that you previously created from the **Namespace** drop-down menu and set the **StorageClass** as *standard*.



Click **Deploy**. Wait a couple of minutes.

The **Application details** screen appears.

Review the **Status** row in the **Components** table to confirm that all components successfully deployed.

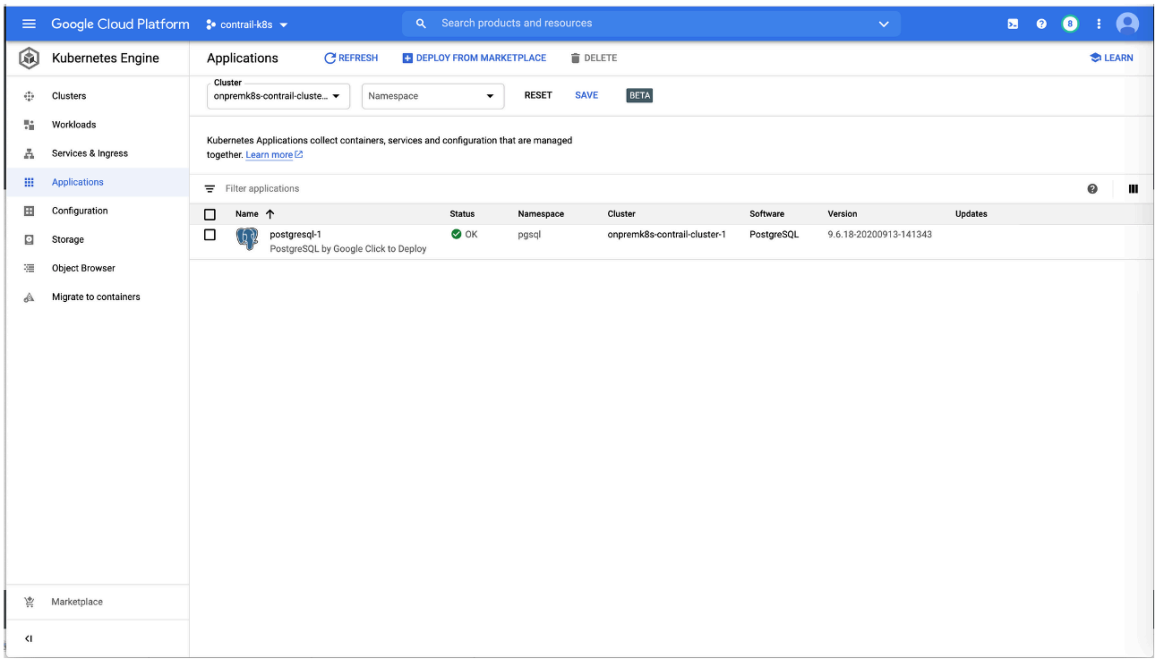


You can also verify that the app is running from the CLI:

```
$ kubectl get po -n pgsql
NAME                          READY   STATUS      RESTARTS   AGE
postgresql-1-deployer-nzpfn   0/1     Completed   0          91s
postgresql-1-postgresql-0     2/2     Running     0          46s
```

```
$ kubectl get pvc
NAME                                                    STATUS    VOLUME
CAPACITY   ACCESS MODES   STORAGECLASS   AGE
postgresql-1-postgresql-pvc-postgresql-1-postgresql-0   Bound     local-pv-e00b14f6
62Gi       RWO            standard       91s
```

13. Use filtering within the GKE Console to see the applications deployed in the on-premises cluster.



14. To access the application:

- Forward the PostgreSQL port locally:

```
$ export NAMESPACE=pgsql
$ export APP_INSTANCE_NAME="postgresql-1"
$ kubectl port-forward --namespace "${NAMESPACE}" "${APP_INSTANCE_NAME}-postgresql-0"
5432
Forwarding from 127.0.0.1:5432 -> 5432
Forwarding from [::1]:5432 -> 5432
```

- Connect to the database

```
$ apt -y install postgresql-client-10 postgresql-client-common
$ export PGPASSWORD=$(kubectl get secret "postgresql-1-secret" --
output=jsonpath='{.data.password}' | base64 -d)
```

```
$ psql (10.12 (Ubuntu 10.12-0ubuntu0.18.04.1), server 9.6.18)
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256,
compression: off)
Type "help" for help.

postgres=#
```

AWS Elastic Kubernetes Service Cluster: How to Deploy an Application from Google Marketplace

You can deploy an application from the Google Marketplace into an EKS cluster that is using Contrail Networking in AWS after the cluster is enabled in Anthos. This procedure will illustrate this process by deploying Prometheus and Grafana from Google Marketplace

Perform the following steps to deploy an application from Google Marketplace onto an EKS cluster in AWS that is using Contrail Networking.

1. Enable credentials within the *eks-contrail* context:

```
$ kubectx eks-contrail
Switched to context "eks-contrail"

$ kubectl create ns application-system

$ kubens application-system
Context "kubernetes-admin@kubernetes" modified.
Active namespace is "application-system".

$ kubectl create secret docker-registry gcr-json-key \
--docker-server=https://marketplace.gcr.io \
--docker-username=_json_key \
--docker-password="$(cat ./gcr-sa.json)" \
--docker-email=[GCP_EMAIL_ADDRESS]

$ kubectl patch serviceaccount default -p '{"imagePullSecrets": [{"name": "gcr-json-key"}]}'

$ kubectl annotate namespace application-system marketplace.cloud.google.com/
imagePullSecret=gcr-json-key
```

2. The GCP Marketplace expects a storage class named *standard* to be configured in a context. The default story class name in EKS, however, is *gp2*.

   To change the storage class name:

a. Remove the default flag from the *gp2* storage class using the patch command:

```
$  kubectl patch storageclass gp2 -p '{"metadata": {"annotations":
{"storageclass.kubernetes.io/is-default-class":"false"}}}'
```

b. Create a new storage class for the Amazon EKS context and mark it as the default storage class:

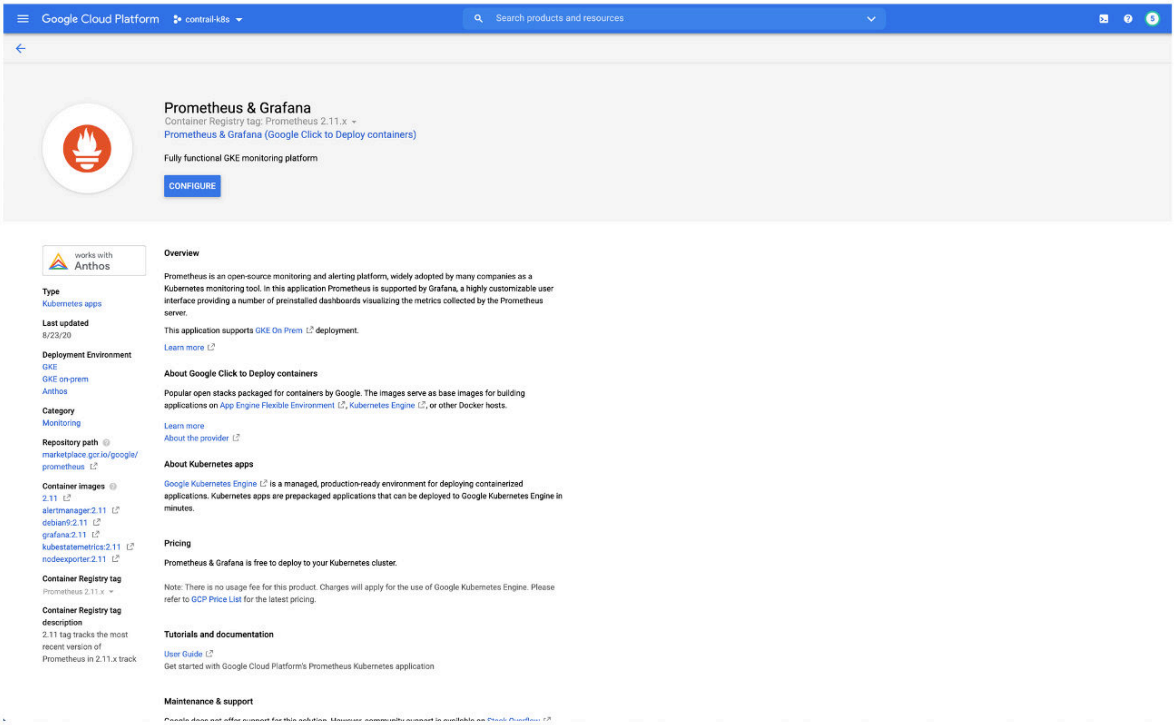
```
$ cat <<EOF > eks-sc.yaml
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: standard
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
provisioner: kubernetes.io/aws-ebs
parameters:
  type: gp2
  fsType: ext4
EOF

$ kubectl create -f eks-sc.yaml
storageclass.storage.k8s.io/standard created

$ kubectl get sc
NAME                 PROVISIONER             AGE
gp2                  kubernetes.io/aws-ebs   2d
standard (default)   kubernetes.io/aws-ebs   5s
```

3. Create a namespace for the applications:

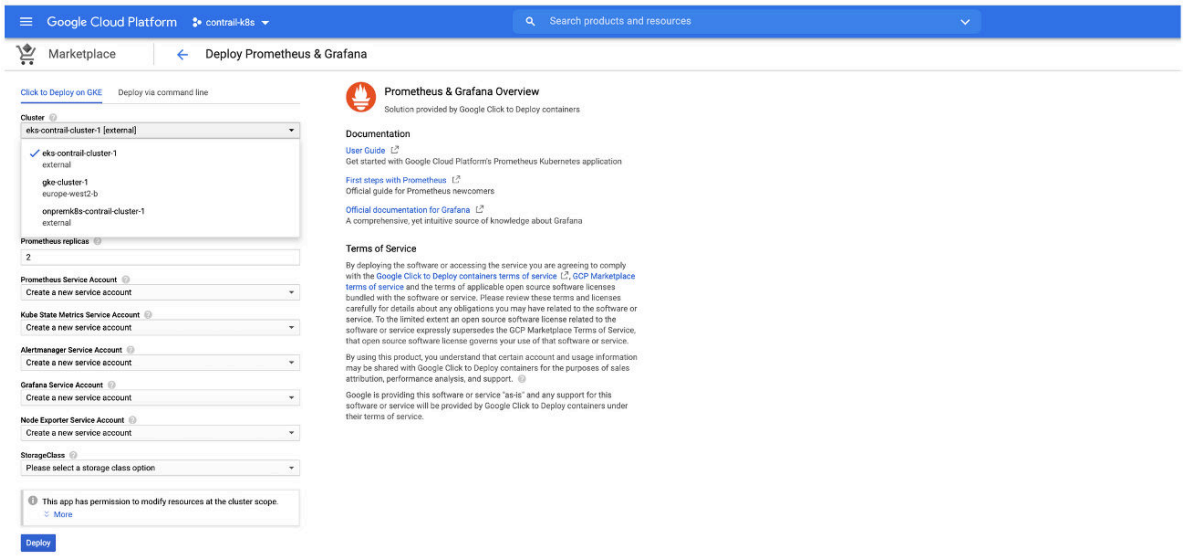
```
$ kubectl create ns monitoring

$ kubens monitoring

kubectl create secret docker-registry gcr-json-key \
 --docker-server=https://gcr.io \
--docker-username=_json_key \
--docker-password="$(cat ./gcr-sa.json)" \
--docker-email=[GCP_EMAIL_ADDRESS]

$ kubectl patch serviceaccount default -p '{"imagePullSecrets": [{"name": "gcr-json-key"}]}'
```

```
$ kubectl annotate namespace monitoring marketplace.cloud.google.com/imagePullSecret=gcr-json-key
```

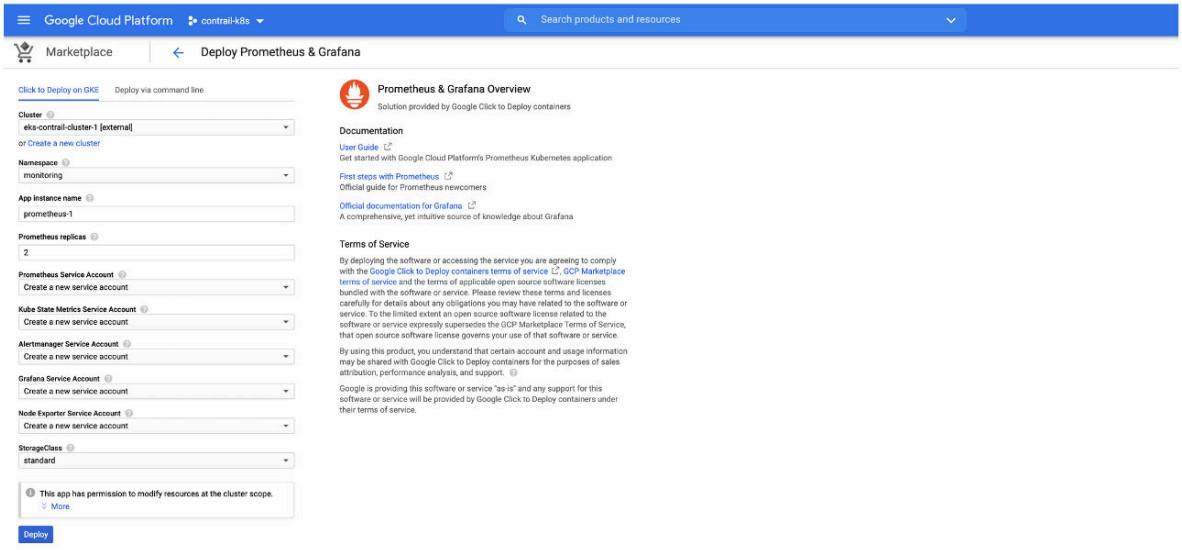
4. Choose Prometheus and Grafana from GCP Marketplace. Click the **Configure** button to start the deployment procedure.



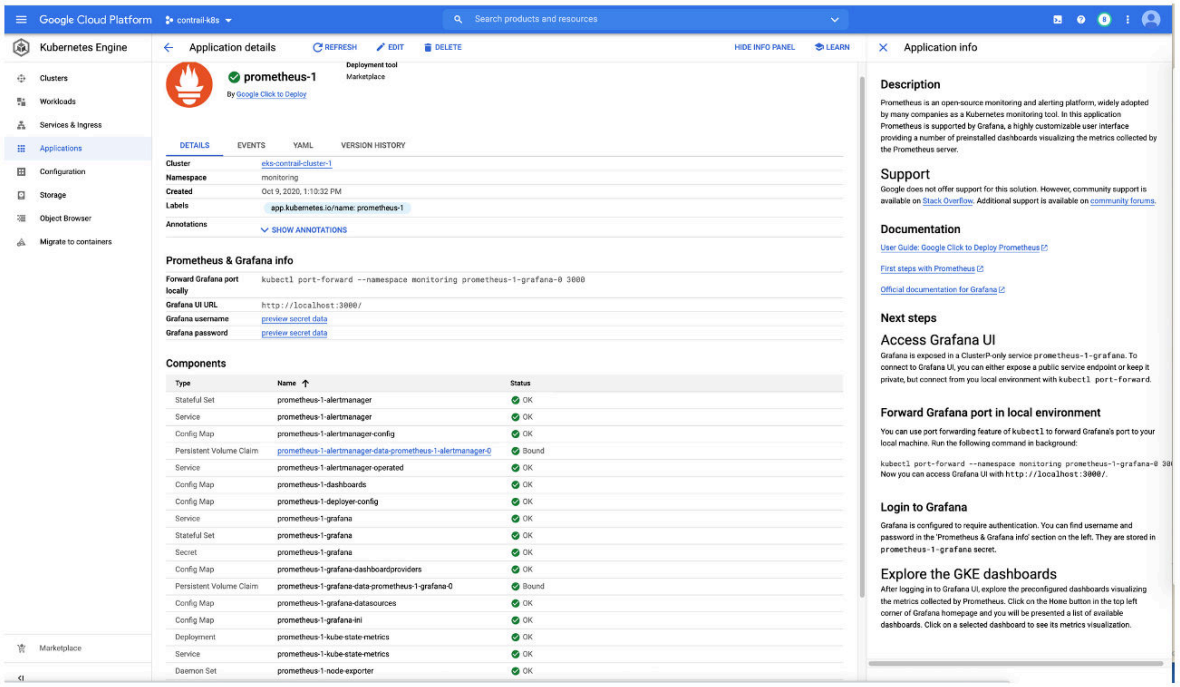
5. Choose the EKS cluster from the cluster drop-down menu.



6. Select the namespace and storage class. Click Deploy.



Wait several minutes for the application to deploy.



You can also verify that the application has deployed using the CLI:

```
$ kubectl get pods -n monitoring
NAME                                  READY   STATUS      RESTARTS   AGE
prometheus-1-alertmanager-0           1/1     Running     0          2m36s
prometheus-1-alertmanager-1           1/1     Running     0          88s
prometheus-1-deployer-blm5f           0/1     Completed   0          3m20s
prometheus-1-grafana-0                1/1     Running     0          2m36s
```

```
prometheus-1-kube-state-metrics-6f64b67684-shtdg   2/2     Running     0          2m37s
prometheus-1-node-exporter-5scf4                   1/1     Running     0          2m36s
prometheus-1-node-exporter-gdp77                   1/1     Running     0          2m36s
prometheus-1-node-exporter-k8vfn                   1/1     Running     0          2m36s
prometheus-1-node-exporter-v6w7g                   1/1     Running     0          2m36s
prometheus-1-node-exporter-zffs9                   1/1     Running     0          2m36s
prometheus-1-prometheus-0                          1/1     Running     0          2m36s
prometheus-1-prometheus-1                          1/1     Running     0          2m36s
```

7. If you have a private service, consider how your going to make it accessible.

   In this case, the Grafana user interface is exposed in the ClusterP-only service named *prometheus-1-grafana*. To connect to the Grafana user interface, either change the service to a public service endpoint or keep the service private and access it from your local environment.

```
kubectl get svc -n monitoring
NAME                                TYPE        CLUSTER-IP       EXTERNAL-IP
PORT(S)             AGE
prometheus-1-alertmanager           ClusterIP   10.100.92.6      <none>        9093/
TCP            10m
prometheus-1-alertmanager-operated   ClusterIP   None             <none>        6783/TCP,9093/
TCP   10m
prometheus-1-grafana                ClusterIP   10.100.126.78    <none>        80/
TCP              10m
prometheus-1-kube-state-metrics     ClusterIP   10.100.46.18     <none>        8080/TCP,8081/
TCP   10m
prometheus-1-prometheus             ClusterIP   10.100.214.104   <none>        9090/
TCP            10m
```

   You can use the kubectl port forwarding feature to forward Graffana traffic to your local machine by running the following command:

```
$ kubectl port-forward --namespace monitoring prometheus-1-grafana-0 3000
Now you can access Grafana UI with http://localhost:3000/.
```

Configuration Management in Anthos

IN THIS SECTION

This section covers Configuration Management in Anthos.

It includes the following sections:

Overview: Anthos Configuration Management

Google Cloud uses a tool called Config Sync that acts as the bridge between an external source code repository and the Kubernetes API server. See Config Sync overview from Google Cloud for additional information.

Anthos Configuration Management (ACM) uses Config Sync to extend configuration to non-GCP clusters that are connected using Anthos.

In the following sections, a GitHub repository is used as a single source for deployments and configuration. An ACM component is installed onto each of the clusters that are included with Anthos to monitor the external repositories for changes and synchronizing them across Anthos.

GitOps-style deployments are used in the following procedures to push workloads across all registered clusters through Anthos Config Management. GitOps provides a method of performing Kubernetes cluster management and application delivery. It works by using Git as a single source of truth for declarative infrastructure and applications and using the YAML or JSON files used in Kubernetes to combine with Anthos for code.

Installing the Configuration Management Operator

The Configuration Management Operator is a controller that manages installation of the Anthos Configuration Manager. The operator will be installed on all three clusters using these instructions.

To install the Configuration Management Operator:

1. Download the Configuration Management Operator and apply it to each cluster:

```
gsutil cp gs://config-management-release/released/latest/config-management-operator.yaml
config-management-operator.yaml

$ kubectl create -f config-management-operator.yaml
customresourcedefinition.apiextensions.k8s.io/configmanagements.configmanagement.gke.io
configured
clusterrolebinding.rbac.authorization.k8s.io/config-management-operator configured
clusterrole.rbac.authorization.k8s.io/config-management-operator configured
serviceaccount/config-management-operator configured
deployment.apps/config-management-operator configured
namespace/config-management-system configured
```

   Run this command in each cluster.

2. Confirm that the operator was created:

```
$ kubectl describe crds configmanagements.configmanagement.gke.io
Name:         configmanagements.configmanagement.gke.io
Namespace:
Labels:       controller-tools.k8s.io=1.0
Annotations:  <none>
API Version:  apiextensions.k8s.io/v1
Kind:         CustomResourceDefinition
Metadata:
  Creation Timestamp:  2020-10-09T13:13:17Z
  Generation:          1
  Resource Version:    363244
  Self Link:           /apis/apiextensions.k8s.io/v1/customresourcedefinitions/
configmanagements.configmanagement.gke.io
  UID:                 a088edbc-8232-419f-8f42-365fa36de110
Spec:
  Conversion:
    Strategy:  None
  Group:       configmanagement.gke.io
  Names:
    Kind:                   ConfigManagement
    List Kind:              ConfigManagementList
    Plural:                 configmanagements
```

```
    Singular:              configmanagement
....
```

Configuring the Clusters for Anthos Configuration Management

To configure the clusters for Anthos Configuration Management:

1. Create an SSH keypair to allow the Operator to authenticate to your Git repository:

```
$ ssh-keygen -t rsa -b 4096 -C "git-user1" -N '' -f "~/.ssh/gke-github"
```

2. Configure your repository to recognize the newly-created public key. See Adding a new SSH key to your GitHub account from GitHub.

   Add a private key to a new secret in the cluster:

```
$ kubectl create secret generic git-creds \
  --namespace=config-management-system \
  --from-file=ssh="/Users/user1/.ssh/gke-github"
```

   Repeat this step for each individual cluster

3. (Optional) Gather the name of each cluster, if needed:

```
$ gcloud container hub memberships list
NAME                          EXTERNAL_ID
onpremk8s-contrail-cluster-1  78f7890b-3a43-4bc7-8fd9-44c76953781b
eks-contrail-cluster-1        42e532ba-a0d9-4087-baed-647be8bca7e9
gke-cluster-1                 6671599e-87af-461b-aff9-7105ebda5c66
```

4. Create a config-management.yaml file for each cluster. Replace the clusterName with the registered clustered name in Anthos in each file.

```
$ cat config-management.yaml
apiVersion: configmanagement.gke.io/v1
kind: ConfigManagement
metadata:
  name: config-management
spec:
  # clusterName is required and must be unique among all managed clusters
  clusterName:
  git:
```

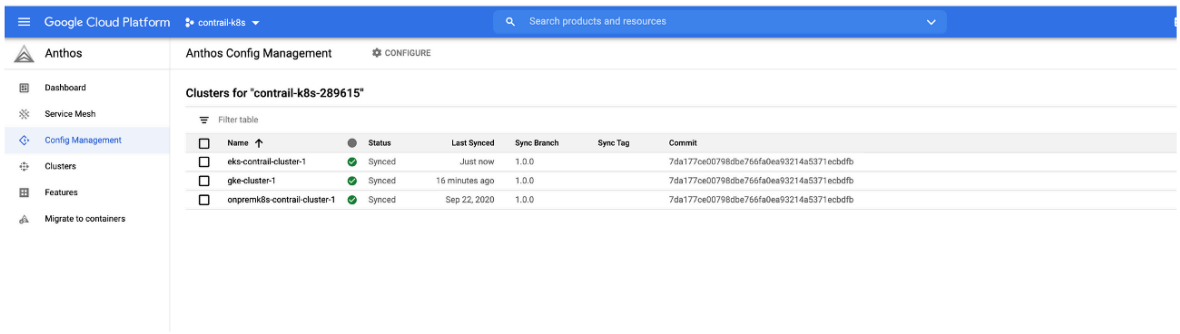
```
    syncRepo: git@github.com:git-user1/csp-config-management.git
    syncBranch: 1.0.0
    secretType: ssh
    policyDir: foo-corp
    proxy: {}

$ kubectx eks-contrail
$ kubectl apply -f config-management.yaml

$ kubectx onprem-k8s-contrail
$ kubectl apply -f config-management.yaml

$ kubectx gke
$ kubectl apply -f config-management.yaml
```

5. Verify that the pods are running on each cluster.

   To verify in the CLI:

```
$ kubectl get pods -n config-management-system
NAME                              READY   STATUS    RESTARTS   AGE
git-importer-584bd49676-46bjq     3/3     Running   0          4m23s
monitor-c8c68d5ff-bdhzl           1/1     Running   0          4m25s
syncer-7dbbc8868c-gtp8d           1/1     Running   0          4m25s
```

   To verify on the Anthos dashboard:



Using Nomos to Manage the Anthos Configuration Manager

The Google Cloud Platform offers a utility called Nomos which can be used to manage the Anthos Configuration Manager (ACM). See Using the nomos command from Google Cloud for more information on Nomos.

To enable Nomos:

1. Get the utility and copy it into a local directory:

```
$ gsutil cp gs://config-management-release/released/latest/darwin_amd64/nomos nomos

$ cp ./nomos /usr/local/bin
$ chmod +x /usr/local/bin/nomos
```

2. Verify that nomos is running in the clusters connected using Anthos:

```
$ nomos status
Connecting to clusters...
Current   Context              Sync Status      Last Synced Token   Sync Branch   Resource
Status
-------   -------              -----------      -----------------   -----------
---------------
*         eks-contrail         SYNCED           7da177ce            1.0.0         Healthy
          gke                  SYNCED           7da177ce            1.0.0         Healthy
          onprem-k8s-contrail  SYNCED           7da177ce            1.0.0         Healthy
```

3. List the namespaces that are currently managed by Anthos Configuration Management.

   In this sample output, configurations are stored in the cluster/ and namespace/ directories. All objects managed by Anthos Config Management have the app.kubernetes.io/managed-by label set to configmanagement.gke.io.

```
$ kubectl get ns -l app.kubernetes.io/managed-by=configmanagement.gke.io
NAME              STATUS   AGE
audit             Active   13m
shipping-dev      Active   13m
shipping-prod     Active   13m
shipping-staging  Active   13m
```

4. In the following sequence, we'll validate that nomos and Anthos Configuration Management are efficiently managing the configuration of configuration in a third-party cluster by deleting a namespace in EKS and confirming that a new namespace is quickly recreated.

```
$ kubectx eks-contrail

$ kubectl delete ns audit
namespace "audit" deleted
```

```
$ kubectl get ns audit
NAME    STATUS   AGE
audit   Active   5s
```

The output shows that a new audit workspace was created 5 seconds ago, confirming that Anthos Configuration Management is working.

5
CHAPTER

Using KubeVirt

How to Integrate Kubernetes Clusters using Contrail Networking into Google Cloud Anthos

IN THIS SECTION

NOTE: This topic covers Contrail Networking in Kubernetes-orchestrated environments that are using Contrail Networking Release 21-based releases.

Starting in Release 22.1, Contrail Networking evolved into Cloud-Native Contrail Networking. Cloud-Native Contrail Networking offers significant enhancements to optimize networking performance in Kubernetes-orchestrated environments. We recommend using Cloud-Native Contrail for networking in most Kubernetes-orchestrated environments.

For general information about Cloud-Native Contrail, see the Cloud-Native Contrail Networking Techlibrary homepage.

Anthos is an application management platform developed by Google that provides a consistent development and operations experience for users working in cloud networking clusters that were created in Google Cloud or on third-party cloud platforms. For additional information on Anthos, see the Anthos technical overview from Google Cloud.

The purpose of this document is to illustrate how cloud environments using Kubernetes for orchestration and Contrail Networking for networking can be integrated into the Anthos management platform. This document shows how to create clusters in three separate cloud environments—a private on-premises cloud, a cloud created using the Elastic Kubernetes Service (EKS) in Amazon Web Services (AWS), and a cloud created using the Google Kubernetes Engine (GKE) in the Google Cloud Platform—and add those clusters into Anthos.

This document also provides instructions on introductory configuration and usage tasks after the clouds have been integrated into Anthos. It includes a section on Anthos Configuration management and a

section showing how to load applications from the Google Marketplace into third-party cloud environments.

This document covers the following topics:

Prerequisites

The procedures in this document make the following assumptions about your environment:

- *All Environments*

  The following CLI tools have been downloaded:

  - kubectl. See Install and Set Up kubectl.
  - (Recommend for management) kubectx and kubens. See kubectx + kubens: Power tools for kubectl in Github.

- *Google Cloud Platform*

  - The GCP CLI tools from the Cloud SDK package are operational. See Getting Started with Cloud SDK from Google.

- *Amazon Web Services*

  - This procedure assumes that you have an active AWS account with operating credentials and that the AWS CLI is working on your system. See the Configuring the AWS CLI document from AWS.
  - the eksctl CLI tool is running. See eksctl from the eksctl website.

Creating Kubernetes Clusters

IN THIS SECTION

This sections shows how to create the following Kubernetes clusters:

On-Premises: Creating the Private Kubernetes Cluster

Create an on-premises Kubernetes cluster that includes Contrail Networking. See Installing Kubernetes with Contrail.

The procedure used in this document installs Kubernetes 1.18.9 on a server node running Ubuntu 18.04.5:

```
$ kubectl get nodes -o wide
NAME          STATUS   ROLES    VERSION   OS-IMAGE             KERNEL-VERSION
k8s-master1   Ready    master   v1.18.9   Ubuntu 18.04.5 LTS   4.15.0-118-generic
k8s-master2   Ready    master   v1.18.9   Ubuntu 18.04.5 LTS   4.15.0-118-generic
k8s-master3   Ready    master   v1.18.9   Ubuntu 18.04.5 LTS   4.15.0-118-generic
k8s-node1     Ready    <none>   v1.18.9   Ubuntu 18.04.5 LTS   4.15.0-112-generic
k8s-node2     Ready    <none>   v1.18.9   Ubuntu 18.04.5 LTS   4.15.0-112-generic
```

NOTE: Some output fields removed for readability.

After deploying the Kubernetes cluster, Contrail is installed using a single YAML file.

```
$ kubectl get po -n kube-system
NAME                                  READY   STATUS    RESTARTS   AGE
config-zookeeper-4klts                1/1     Running   0          19h
config-zookeeper-cs2fk                1/1     Running   0          19h
config-zookeeper-wgrtb                1/1     Running   0          19h
contrail-agent-ch8kv                  3/3     Running   2          19h
contrail-agent-kh9cf                  3/3     Running   1          19h
contrail-agent-kqtmz                  3/3     Running   0          19h
contrail-agent-m6nrz                  3/3     Running   1          19h
contrail-agent-qgzxt                  3/3     Running   3          19h
contrail-analytics-6666s              4/4     Running   1          19h
contrail-analytics-jrl5x              4/4     Running   4          19h
contrail-analytics-x756g              4/4     Running   4          19h
contrail-configdb-2h7kd               3/3     Running   4          19h
contrail-configdb-d57tb               3/3     Running   4          19h
contrail-configdb-zpmsq               3/3     Running   4          19h
contrail-controller-config-c2226      6/6     Running   9          19h
contrail-controller-config-pbbmz      6/6     Running   5          19h
```

```
contrail-controller-config-zqkm6      6/6     Running   4          19h
contrail-controller-control-2kz4c     5/5     Running   2          19h
contrail-controller-control-k522d     5/5     Running   0          19h
contrail-controller-control-nr54m     5/5     Running   2          19h
contrail-controller-webui-5vxl7       2/2     Running   0          19h
contrail-controller-webui-mzpdv       2/2     Running   1          19h
contrail-controller-webui-p8rc2       2/2     Running   1          19h
contrail-kube-manager-88c4f           1/1     Running   0          19h
contrail-kube-manager-fsz2z           1/1     Running   0          19h
contrail-kube-manager-qc27b           1/1     Running   0          19h
coredns-684f7f6cb4-4mmgc              1/1     Running   0          93m
coredns-684f7f6cb4-dvpjk              1/1     Running   0          107m
coredns-684f7f6cb4-m6sj7              1/1     Running   0          84m
coredns-684f7f6cb4-nfkfh              1/1     Running   0          84m
coredns-684f7f6cb4-tk48d              1/1     Running   0          86m
etcd-k8s-master1                      1/1     Running   0          94m
etcd-k8s-master2                      1/1     Running   0          95m
etcd-k8s-master3                      1/1     Running   0          92m
kube-apiserver-k8s-master1            1/1     Running   0          94m
kube-apiserver-k8s-master2            1/1     Running   0          95m
kube-apiserver-k8s-master3            1/1     Running   0          92m
kube-controller-manager-k8s-master1   1/1     Running   0          94m
kube-controller-manager-k8s-master2   1/1     Running   0          95m
kube-controller-manager-k8s-master3   1/1     Running   0          92m
kube-proxy-975tn                      1/1     Running   0          108m
kube-proxy-9qzc9                      1/1     Running   0          108m
kube-proxy-fgwqt                      1/1     Running   0          109m
kube-proxy-n6nnq                      1/1     Running   0          109m
kube-proxy-wf289                      1/1     Running   0          108m
kube-scheduler-k8s-master1            1/1     Running   0          94m
kube-scheduler-k8s-master2            1/1     Running   0          95m
kube-scheduler-k8s-master3            1/1     Running   0          90m
rabbitmq-82lmk                        1/1     Running   0          19h
rabbitmq-b2lz8                        1/1     Running   0          19h
rabbitmq-f2nfc                        1/1     Running   0          19h
redis-42tkr                           1/1     Running   0          19h
redis-bj76v                           1/1     Running   0          19h
redis-ctzhg                           1/1     Running   0          19h
```

You should also configure user roles using role-based access control (RBAC). This example shows you how to grant the customer-admin RBAC role to all Kubernetes namespaces:

```
$ kubectl create clusterrolebinding permissive-binding \
  --clusterrole=cluster-admin \
  --user=admin \
  --user=kubelet \
  --group=system:serviceaccounts

kubectl auth can-i '*' '*' --all-namespaces
```

Amazon Web Services (AWS): Install Contrail Networking in an Elastic Kubernetes Service (EKS) Environment

To create a Kubernetes cluster within the Elastic Kubernetes Service (EKS) in AWS, perform following procedure using the *eksctl* CLI tool :

1. Create the cluster. To create a cluster that includes Contrail running in Kubernetes within EKS, follow the instructions in "How to Install Contrail Networking within an Amazon Elastic Kubernetes Service (EKS) Environment in AWS" on page 122.
2. View the nodes:

```
$ kubectl get nodes -o wide
NAME                                          STATUS  ROLES   VERSION   OS-IMAGE
KERNEL-VERSION
ip-100-72-0-119.eu-central-1.compute.internal  Ready   infra   v1.16.15  Ubuntu 18.04.3 LTS
4.15.0-1054-aws
ip-100-72-0-220.eu-central-1.compute.internal  Ready   <none>  v1.16.15  Ubuntu 18.04.3 LTS
4.15.0-1054-aws
ip-100-72-0-245.eu-central-1.compute.internal  Ready   infra   v1.16.15  Ubuntu 18.04.3 LTS
4.15.0-1054-aws
ip-100-72-1-116.eu-central-1.compute.internal  Ready   infra   v1.16.15  Ubuntu 18.04.3 LTS
4.15.0-1054-aws
ip-100-72-1-67.eu-central-1.compute.internal   Ready   <none>  v1.16.15  Ubuntu 18.04.3 LTS
4.15.0-1054-aws
```

3. View the pods.

Note the Contrail pods to confirm that Contrail is running in the environment.

```
$ kubectl get pods --all-namespaces
NAME                                        READY   STATUS    RESTARTS   AGE
cni-patches-2jm8n                           1/1     Running   0          4d21h
cni-patches-2svt6                           1/1     Running   0          4d21h
cni-patches-9mpss                           1/1     Running   0          4d21h
cni-patches-fdbws                           1/1     Running   0          4d21h
cni-patches-ggdph                           1/1     Running   0          4d21h
config-management-operator-5994858fbb-9xvmx 1/1     Running   0          2d20h
config-zookeeper-fz5zv                      1/1     Running   0          4d21h
config-zookeeper-n7wgk                      1/1     Running   0          4d21h
config-zookeeper-pjffv                      1/1     Running   0          4d21h
contrail-agent-69zpn                        3/3     Running   0          4d21h
contrail-agent-gqtfv                        3/3     Running   0          4d21h
contrail-agent-lb8tj                        3/3     Running   0          4d21h
contrail-agent-lrrp8                        3/3     Running   0          4d21h
contrail-agent-z4qjc                        3/3     Running   0          4d21h
contrail-analytics-2bv7c                    4/4     Running   0          4d21h
contrail-analytics-4jgq6                    4/4     Running   0          4d21h
contrail-analytics-sn6cj                    4/4     Running   0          4d21h
contrail-configdb-bhvlw                     3/3     Running   0          4d21h
contrail-configdb-kvvk4                     3/3     Running   0          4d21h
contrail-configdb-vbczf                     3/3     Running   0          4d21h
contrail-controller-config-8vrrm            6/6     Running   1          4d21h
contrail-controller-config-lxsms            6/6     Running   3          4d21h
contrail-controller-config-r7ncm            6/6     Running   4          4d21h
contrail-controller-control-5795l           5/5     Running   0          4d21h
contrail-controller-control-dz6pl           5/5     Running   0          4d21h
contrail-controller-control-qznf9           5/5     Running   0          4d21h
contrail-controller-webui-2g5jx             2/2     Running   0          4d21h
contrail-controller-webui-7kg48             2/2     Running   0          4d21h
contrail-controller-webui-ww5z9             2/2     Running   0          4d21h
contrail-kube-manager-2jhzc                 1/1     Running   2          4d21h
contrail-kube-manager-8psh9                 1/1     Running   0          4d21h
contrail-kube-manager-m8zg7                 1/1     Running   1          4d21h
coredns-5fdf64ff8-bf2fc                     1/1     Running   0          4d21h
<additional output removed for readability>
```

4. Use role-based access control (RBAC) to define access roles for users accessing cluster resources.

This sample configuration illustrates how to configure RBAC to set the cluster admin role to all namespaces in the cluster. The remaining procedures in this document assume that the user has cluster admin access to all cluster resources.

```
$ kubectl create clusterrolebinding permissive-binding \
  --clusterrole=cluster-admin \
  --user=admin \
  --user=kubelet \
  --group=system:serviceaccounts

kubectl auth can-i '*' '*' --all-namespaces
```

Other RBAC options are available and the discussion of those options is beyond the scope of this document. See Using RBAC Authorization from Kubernetes.

Google Cloud Platform (GCP): Creating a Kubernetes Cluster in Google Kubernetes Engine (GKE)

To create a Kubernetes cluster in Google Cloud using the Google Kubernetes Engine (GKE):

1. Create a project by entering the following command:

   ```
   $ gcloud init
   ```

   Follow the onscreen process to create the project.
2. Verify that the project was created:

   ```
   $ gcloud projects list
   ```
3. Select a project:

   ```
   $ gcloud config set project contrail-k8s-289615
   ```
4. Assign the required IAM user roles.

   In this sample configuration, IAM user roles are set so that users have complete control of all registration tasks. For more information on IAM user role options, see Grant the required IAM roles to the user registering the cluster document from Google Cloud.

   ```
   PROJECT_ID=contrail-k8s-289615
   $ gcloud projects add-iam-policy-binding ${PROJECT_ID} \
   ```

```
 --member user:[GCP_EMAIL_ADDRESS] \
 --role=roles/gkehub.admin \
 --role=roles/iam.serviceAccountAdmin \
 --role=roles/iam.serviceAccountKeyAdmin \
 --role=roles/resourcemanager.projectIamAdmin
```

5. APIs are required to access resources in Google Cloud. See the Enable the required APIs in your project content in Google Cloud.

   To enable the APIs required for this project:

```
gcloud services enable \
 --project=${PROJECT_ID} \
 container.googleapis.com \
 compute.googleapis.com \
 gkeconnect.googleapis.com \
 gkehub.googleapis.com \
 cloudresourcemanager.googleapis.com \
 cloudtrace.googleapis.com \
 anthos.googleapis.com \
 iamcredentials.googleapis.com \
 meshca.googleapis.com \
 meshconfig.googleapis.com \
 meshtelemetry.googleapis.com \
 monitoring.googleapis.com \
 logging.googleapis.com \
 runtimeconfig.googleapis.com
```

6. Create the Kubernetes cluster:

```
$ export KUBECONFIG=gke-config
$ gcloud container clusters create gke-cluster-1 \
--zone "europe-west2-b" \
--disk-type "pd-ssd" \
--disk-size "150GB" \
--machine-type "n2-standard-4" \
--num-nodes=3 \
--image-type "COS" \
--enable-stackdriver-kubernetes \
--addons HorizontalPodAutoscaling,HttpLoadBalancing,Istio,CloudRun \
--istio-config auth=MTLS_PERMISSIVE \
--cluster-version "1.17.9-gke.1504"
```

```
kubectl create clusterrolebinding cluster-admin-binding \
  --clusterrole cluster-admin \
  --user $(gcloud config get-value account)
```

7. To assist with later management tasks, merge the cloud configurations into a single configuration.

   In this example, the on-premises, EKS, and GKE configuration directories are copied into the same directory:

```
$ cp *-config ~/.kube
$ KUBECONFIG=$HOME/.kube/eks-config:$HOME/.kube/contrail-config:$HOME/.kube/gke-config
kubectl config view --merge --flatten > $HOME/.kube/config

$ kubectx gke_contrail-k8s-289615_europe-west2-b_gke-cluster-1
$ kubectx gke=.

$ kubectx arn:aws:eks:eu-central-1:927874460243:cluster/EKS-YC0U0TU5
$ kubectx eks-contrail=.

$ kubectx kubernetes-admin@kubernetes
$ kubectx onprem-k8s-contrail=.
```

8. Confirm the contexts representing the Kubernetes clusters.

   This output illustrates an environment where an on-premises and an EKS cluster were created using the procedures in this document.

```
$ kubectx
eks-contrail
gke
onprem-k8s-contrail
```

Preparing Your Clusters for Anthos

IN THIS SECTION

This section describes how to prepare your Google Cloud Platform account and your clusters for Anthos.

It includes the following sections:

Configure Your Google Cloud Platform Account for Anthos

You need to create a service account in GCP and provision a JSON file with the Google Cloud service account credentials for external clusters—in this example, the external clusters are the on-premises cloud and the AWS cloud networks—before you can connect the clusters created by third-party providers into Google Anthos.

To configure your Google Cloud Platform for Anthos:

1. Create the Google Cloud service account.

   This step includes creating a project ID and creating an IAM profile for the account:

```
$ PROJECT_ID=contrail-k8s-289615
$ SERVICE_ACCOUNT_NAME=anthos-connect

$ gcloud iam service-accounts create ${SERVICE_ACCOUNT_NAME} --project=${PROJECT_ID}
```

2. Bind the gkehub.connect IAM role to the service account:

```
$ gcloud projects add-iam-policy-binding ${PROJECT_ID} \
 --member="serviceAccount:${SERVICE_ACCOUNT_NAME}@${PROJECT_ID}.iam.gserviceaccount.com" \
 --role="roles/gkehub.connect"
```

3. Create a private key JSON file for the service account in the current directory. This JSON file is required to register the clusters.

```
$ gcloud iam service-accounts keys create ./${SERVICE_ACCOUNT_NAME}-svc.json \
  --iam-account=${SERVICE_ACCOUNT_NAME}@${PROJECT_ID}.iam.gserviceaccount.com \
  --project=${PROJECT_ID}
```

How to Register an External Kubernetes Cluster to Google Connect

The Google Connect feature is part of Anthos and it allows you to connect your Kubernetes clusters—including clusters created outside Google Cloud—into Google Cloud. This support within Google Connect provides the external Kubernetes clusters with the ability to use many cluster and workload management features from Google Cloud, including the Cloud Console unified user interface. See Connect Overview from Google for additional information on Google Connect and Cloud Console from Google for additional information on Google Cloud Console.

To register external Kubernetes clusters into Google connect:

1. Connect the cluster to the Google Kubernetes Engine (GKE). A GKE agent which is responsible for allowing the cloud network to communicate with the GKE hub is installed in the cloud network during this step.

   - To add an on-premises cluster:

     ```
     gcloud container hub memberships register onpremk8s-contrail-cluster-1 \
        --project=${PROJECT_ID} \
        --context=onprem-k8s-contrail \
        --kubeconfig=$HOME/.kube/config \
        --service-account-key-file=./anthos-connect-svc.json
     ```

     To confirm that the GKE connect agent is running after the command is executed:

     ```
     $ kubectx onprem-k8s-contrail
     Switched to context "onprem-k8s-contrail".

     $ kubectl get pods -n gke-connect
     NAMESPACE      NAME                                          READY   STATUS
     gke-connect    gke-connect-agent-20200918-01-00-7bc77884d-st4r2   1/1     Running
     ```

     NOTE: SNAT usually needs to be enabled in Contrail Networking to allow the GKE connect agent to connect to the Internet.

   - To add a cluster running in Elastic Kubernetes Service (EKS) on Amazon Web Services (AWS):

     ```
     gcloud container hub memberships register eks-contrail-cluster-1 \
        --project=${PROJECT_ID} \
        --context=eks-contrail \
     ```

```
   --kubeconfig=$HOME/.kube/config \
   --service-account-key-file=./anthos-connect-svc.json
```

To confirm that the GKE connect agent is running after executing the command:

```
$ kubectx eks-contrail
Switched to context "eks-contrail".

$ kubectl get pods -n gke-connect
NAME                                                 READY   STATUS
gke-connect-agent-20201002-01-00-5749bfc847-qhvft    1/1     Running
```

- To add a cluster running in GKE on Google Cloud Platform:

```
gcloud container hub memberships register gke-cluster-1 \
--project=${PROJECT_ID} \
--gke-cluster=europe-west2-b/gke-cluster-1 \
--service-account-key-file=./anthos-connect-svc.json
```

To confirm that the GKE connect agent is running in the cluster after executing the command.

Note that the on-premises and AWS EKS clusters that were connected to the GKE hub in the earlier bulletpoints are also visible in the command output.

```
$ gcloud container hub memberships list
NAME                          EXTERNAL_ID
onpremk8s-contrail-cluster-1  78f7890b-3a43-4bc7-8fd9-44c76953781b
eks-contrail-cluster-1        42e532ba-a0d9-4087-baed-647be8bca7e9
gke-cluster-1                 6671599e-87af-461b-aff9-7105ebda5c66
```

2. A bearer token will be used in this procedure to login to the external clusters from the Google Anthos Console. A Kubernetes service account (KSA) will be created in the cluster to generate this bearer token.

   To create and apply this bearer token for an on-premises cluster:

   a. Create and apply the node-reader role in role-based access control (RBAC) using the *node-reader* role in the *node-reader.yaml* file:

```
$ cat <<EOF > node-reader.yaml
kind: ClusterRole
```

```
apiVersion: rbac.authorization.k8s.io/v1
metadata:
 name: node-reader
rules:
- apiGroups: [""]
  resources: ["nodes"]
  verbs: ["get", "list", "watch"]
EOF
F

$ kubectx onpremk8s-contrail-cluster-1

$ kubectl apply -f node-reader.yaml
```

b. Create and authorize a Kubernetes service account (KSA):

```
$ KSA_NAME=anthos-sa
$ kubectl create serviceaccount ${KSA_NAME}
$ kubectl create clusterrolebinding anthos-view --clusterrole view --serviceaccount
default:${KSA_NAME}
$ kubectl create clusterrolebinding anthos-node-reader --clusterrole node-reader --
serviceaccount default:${KSA_NAME}
$ kubectl create clusterrolebinding anthos-cluster-admin --clusterrole cluster-admin --
serviceaccount default:${KSA_NAME}
```

c. Acquire the bearer token for the KSA:

```
$ SECRET_NAME=$(kubectl get serviceaccount ${KSA_NAME} -o jsonpath='{$.secrets[0].name}')
$ kubectl get secret ${SECRET_NAME} -o jsonpath='{$.data.token}' | base64 --decode
```

d. Use the output token in the Cloud Console to login to the cluster.

To create and apply this bearer token for an EKS cluster in AWS:

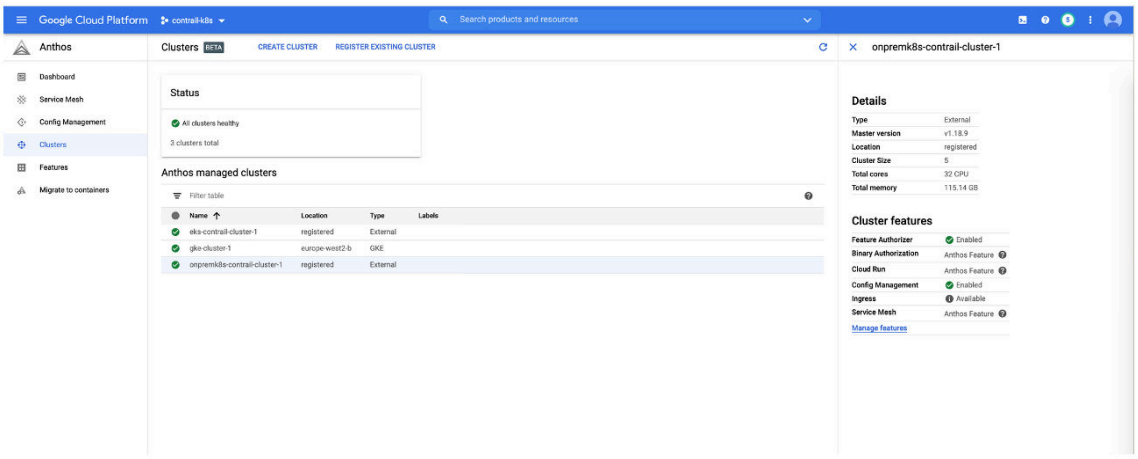
a. Perform the parallel steps for an on-premises cluster for an AWS EKS cluster:

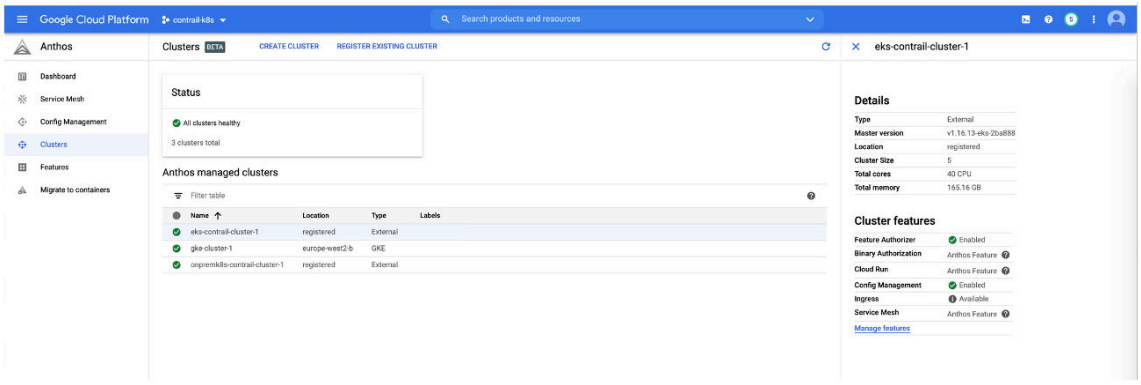
```
$ kubectx eks-contrail
$ $ kubectl apply -f node-reader.yaml

$ kubectl create serviceaccount ${KSA_NAME}
```

```
$ kubectl create clusterrolebinding anthos-view --clusterrole view --serviceaccount
default:${KSA_NAME}
$ kubectl create clusterrolebinding anthos-node-reader --clusterrole node-reader --
serviceaccount default:${KSA_NAME}
$ kubectl create clusterrolebinding anthos-cluster-admin --clusterrole cluster-admin --
serviceaccount default:${KSA_NAME}

$ SECRET_NAME=$(kubectl get serviceaccount ${KSA_NAME} -o jsonpath='{$.secrets[0].name}')
$ kubectl get secret ${SECRET_NAME} -o jsonpath='{$.data.token}' | base64 --decode
```

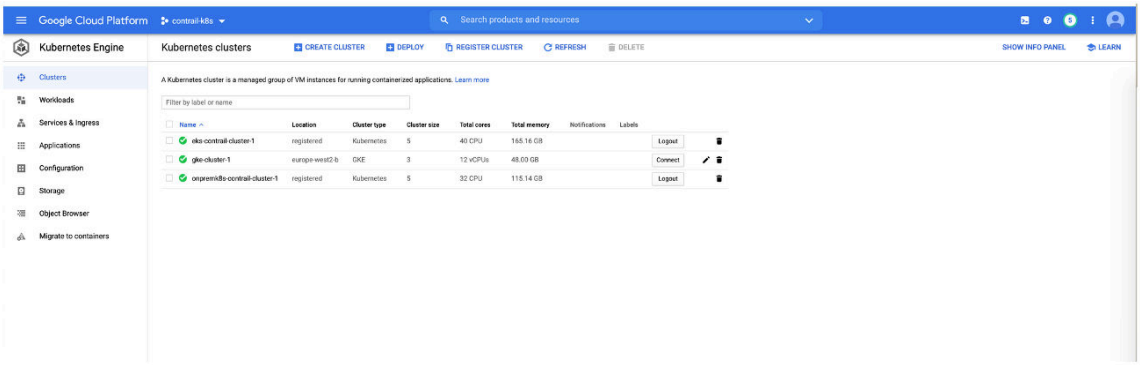
3. Verify the clusters.

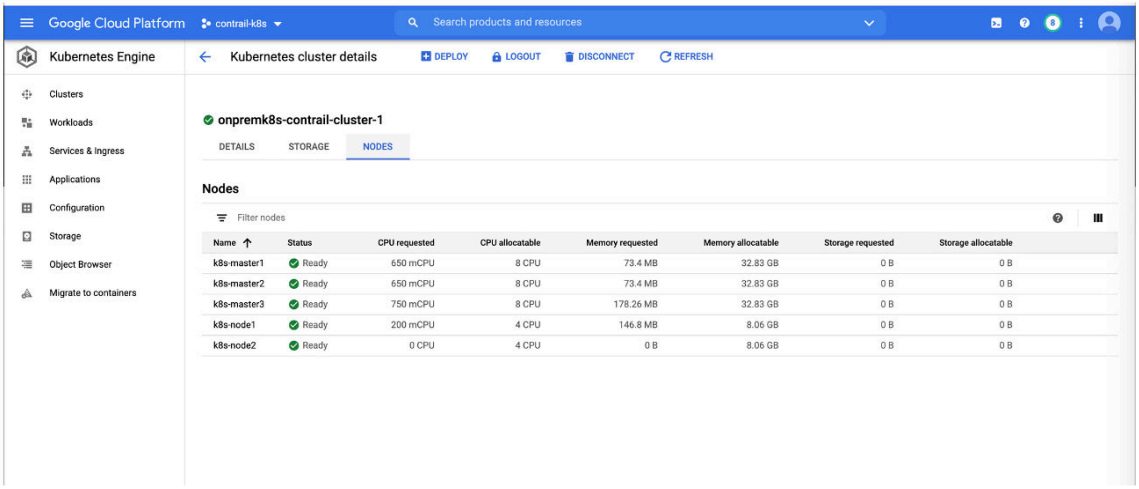
   a. Verify that the clusters are visible in Anthos:

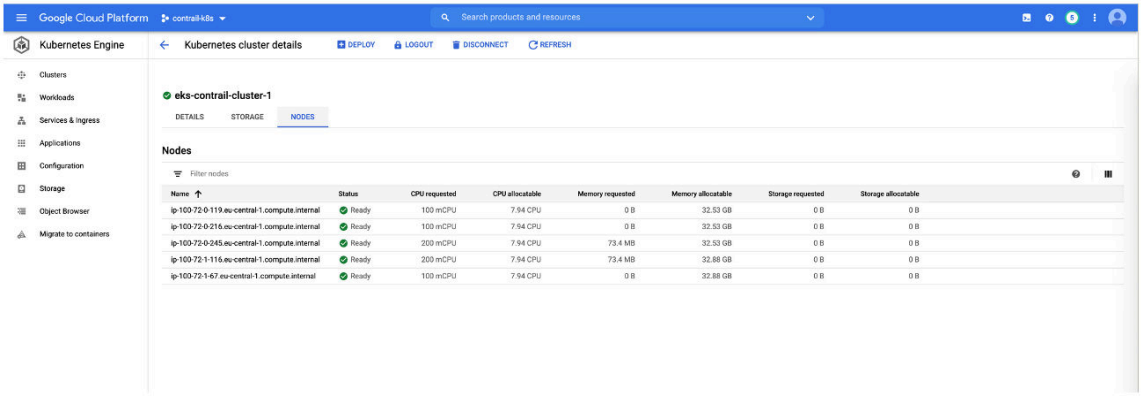




   b. Verify that cluster details are visible from the Kubernetes Engine tab:


Google Cloud Platform
contrail-k8s
Search products and resources
Kubernetes Engine
Kubernetes clusters
CREATE CLUSTER
DEPLOY
REGISTER CLUSTER
REFRESH
DELETE
SHOW INFO PANEL
LEARN
Clusters
Workloads
Services & Ingress
Applications
Configuration
Storage
Object Browser
Migrate to containers
A Kubernetes cluster is a managed group of VM instances for running containerized applications. Learn more
Filter by label or name
Name
Location
Cluster type
Cluster size
Total cores
Total memory
Notifications
Labels
eks-contrail-cluster-1
registered
Kubernetes
5
40 CPU
165.16 GB
Logout
gke-cluster-1
europe-west2-b
GKE
3
12 vCPUs
48.00 GB
Connect
onpremk8s-contrail-cluster-1
registered
Kubernetes
5
32 CPU
115.14 GB
Logout


Google Cloud Platform
contrail-k8s
Search products and resources
Kubernetes Engine
Kubernetes cluster details
DEPLOY
LOGOUT
DISCONNECT
REFRESH
Clusters
Workloads
Services & Ingress
Applications
Configuration
Storage
Object Browser
Migrate to containers
onpremk8s-contrail-cluster-1
DETAILS
STORAGE
NODES
Nodes
Filter nodes
Name
Status
CPU requested
CPU allocatable
Memory requested
Memory allocatable
Storage requested
Storage allocatable
k8s-master1 Ready 650 mCPU 8 CPU 73.4 MB 32.83 GB 0 B 0 B
k8s-master2 Ready 650 mCPU 8 CPU 73.4 MB 32.83 GB 0 B 0 B
k8s-master3 Ready 750 mCPU 8 CPU 178.26 MB 32.83 GB 0 B 0 B
k8s-node1 Ready 200 mCPU 4 CPU 146.8 MB 8.06 GB 0 B 0 B
k8s-node2 Ready 0 CPU 4 CPU 0 B 8.06 GB 0 B 0 B


Google Cloud Platform
contrail-k8s
Search products and resources
Kubernetes Engine
Kubernetes cluster details
DEPLOY
LOGOUT
DISCONNECT
REFRESH
Clusters
Workloads
Services & Ingress
Applications
Configuration
Storage
Object Browser
Migrate to containers
eks-contrail-cluster-1
DETAILS
STORAGE
NODES
Nodes
Filter nodes
Name
Status
CPU requested
CPU allocatable
Memory requested
Memory allocatable
Storage requested
Storage allocatable
ip-100-72-0-119.eu-central-1.compute.internal Ready 100 mCPU 7.94 CPU 0 B 32.53 GB 0 B 0 B
ip-100-72-0-216.eu-central-1.compute.internal Ready 100 mCPU 7.94 CPU 0 B 32.53 GB 0 B 0 B
ip-100-72-0-245.eu-central-1.compute.internal Ready 200 mCPU 7.94 CPU 73.4 MB 32.53 GB 0 B 0 B
ip-100-72-1-116.eu-central-1.compute.internal Ready 200 mCPU 7.94 CPU 73.4 MB 32.88 GB 0 B 0 B
ip-100-72-1-67.eu-central-1.compute.internal Ready 100 mCPU 7.94 CPU 0 B 32.88 GB 0 B 0 B

Deploying GCP Applications into Third Party Clusters That are Integrated Into Anthos

IN THIS SECTION

This section shows how to deploy an application from Google Marketplace onto clusters created outside GCP and integrated into Anthos.

It includes the following sections:

On-premises Kubernetes cluster: How to Deploy Applications from the GCP Marketplace Onto an On-premises Cloud

This procedure shows how to add an application—illustrated using the PostgreSQL application—from the Google Cloud Marketplace into an on-premises cluster that was built outside of Google Cloud and integrated into Anthos.

Perform the following steps to deploy the application:

1. Create a namespace called *application-system* for Google Cloud Marketplace components.

   You must create this namespace to deploy applications to Google Anthos in an on-premises cluster. The namespace must be called *application-system* and must apply an imagePullSecret credential to the default service account for the namespace.

   ```
   $ kubectl create ns application-system

   $ kubens application-system
   Context "kubernetes-admin@kubernetes" modified.
   Active namespace is "application-system".
   ```

2. Create a service account and download an associated JSON token.

This step is required to pull images from the Google Cloud Repository.

```
$ PROJECT_ID=contrail-k8s-289615

$ gcloud iam service-accounts create gcr-sa \
    --project=${PROJECT_ID}

$ gcloud iam service-accounts list \
    --project=${PROJECT_ID}

$ gcloud projects add-iam-policy-binding ${PROJECT_ID} \
 --member="serviceAccount:gcr-sa@${PROJECT_ID}.iam.gserviceaccount.com" \
 --role="roles/storage.objectViewer"

$ gcloud iam service-accounts keys create ./gcr-sa.json \
  --iam-account="gcr-sa@${PROJECT_ID}.iam.gserviceaccount.com" \
  --project=${PROJECT_ID}
```

3. Create a secret credential with the contents of the token:

```
$ kubectl create secret docker-registry gcr-json-key \
--docker-server=https://marketplace.gcr.io \
--docker-username=_json_key \
--docker-password="$(cat ./gcr-sa.json)" \
--docker-email=[GCP_EMAIL_ADDRESS]
```

4. Patch the default service account within the namespace to use the secret credential for pulling images from the Google Cloud Repository instead of the Docker Hub.

```
$ kubectl patch serviceaccount default -p '{"imagePullSecrets": [{"name": "gcr-json-key"}]}'
```

5. Annotate the *application-system* namespace to enable the deployment of Kubernetes Applications from the GCP Marketplace:

```
$ kubectl annotate namespace application-system marketplace.cloud.google.com/
imagePullSecret=gcr-json-key
```

6. Create a default storage class named *standard* by either renaming your storage class to *standard* or creating a new storage class. This step is necessary because the GCP Marketplace expects a storage class named *standard* as the default storage class.

To rename your storage class:

```
$ cat sc.yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: standard
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
provisioner: kubernetes.io/no-provisioner
reclaimPolicy: Delete
volumeBindingMode: WaitForFirstConsumer

$ kubectl get sc
NAME                 PROVISIONER                    AGE
standard (default)   kubernetes.io/no-provisioner   6m14s
```

To create a new storage class, see Setup a Local Persistent Volume for a Kubernetes cluster.

This namespace will be utilized by the GCP Marketplace Apps to dynamically provision Persistent Volume (PV) and Persistent Volume Claim (PVC).

7. Create and configure a namespace for an app that will be deployed from the GCP Marketplace.

   We'll illustrate how to deploy PostgreSQL in this document.

```
$ kubectl create ns pgsql

$ kubens pgsql

$ kubectl create secret docker-registry gcr-json-key \
 --docker-server=https://gcr.io \
--docker-username=_json_key \
--docker-password="$(cat ./gcr-sa.json)" \
--docker-email=[GCP_EMAIL_ADDRESS]
```

8. Patch the default service account within the namespace to use the secret credential to pull images from the Google Cloud repository instead of Docker Hub.

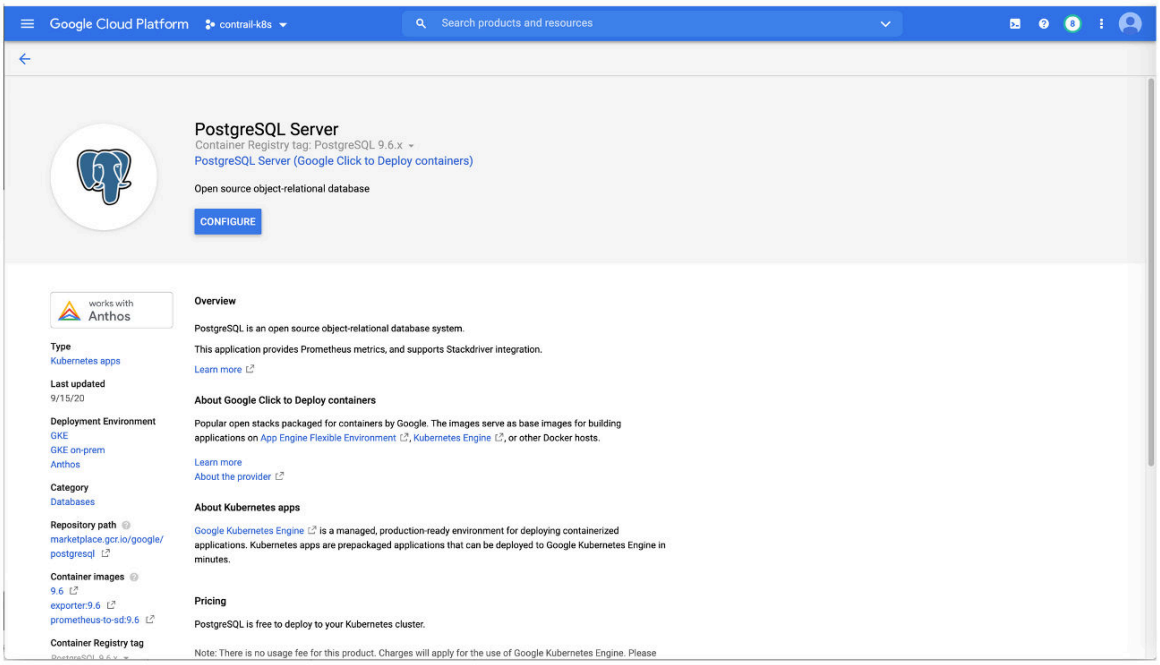
   In this sample case, the default service account is within the pgsql namespace.

```
$ kubectl patch serviceaccount default -p '{"imagePullSecrets": [{"name": "gcr-json-key"}]}'
```

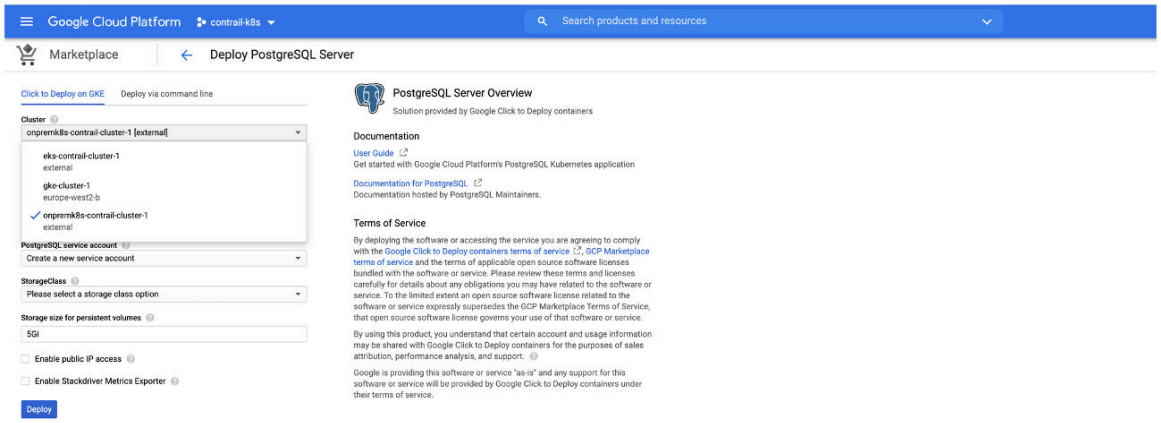
9. Annotate the namespace—in this case, the *pgsql* namespace—to enable the deployment of Kubernetes Apps from the GCP Marketplace:

```
$ kubectl annotate namespace pgsql marketplace.cloud.google.com/imagePullSecret=gcr-json-key
```

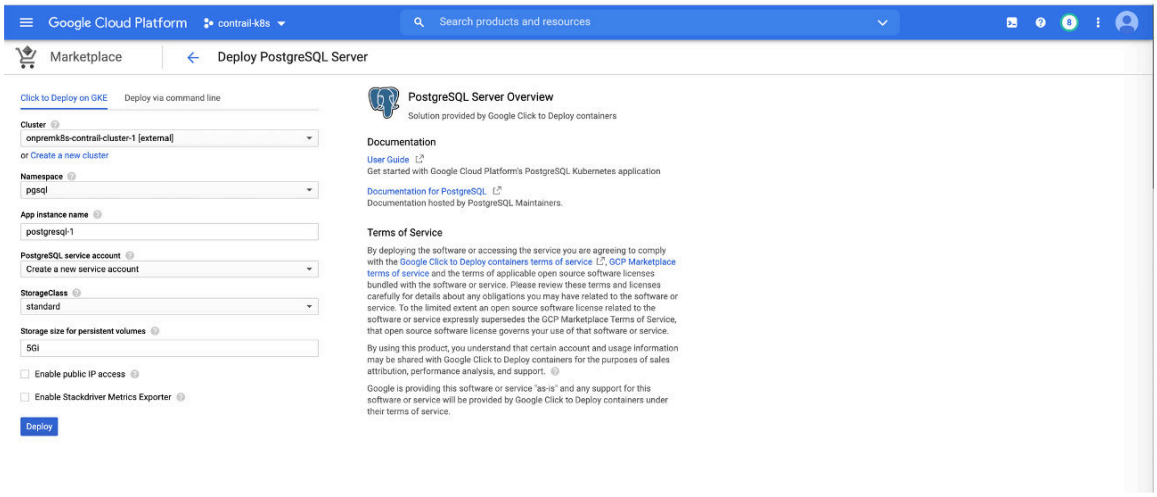
10. Choose the app—in this case, PostgresSQL Server—from GCP Marketplace and click on Configure to start the deployment procedure.



11. Choose the *contrail-cluster-1* external cluster from the **Cluster** drop-down menu:



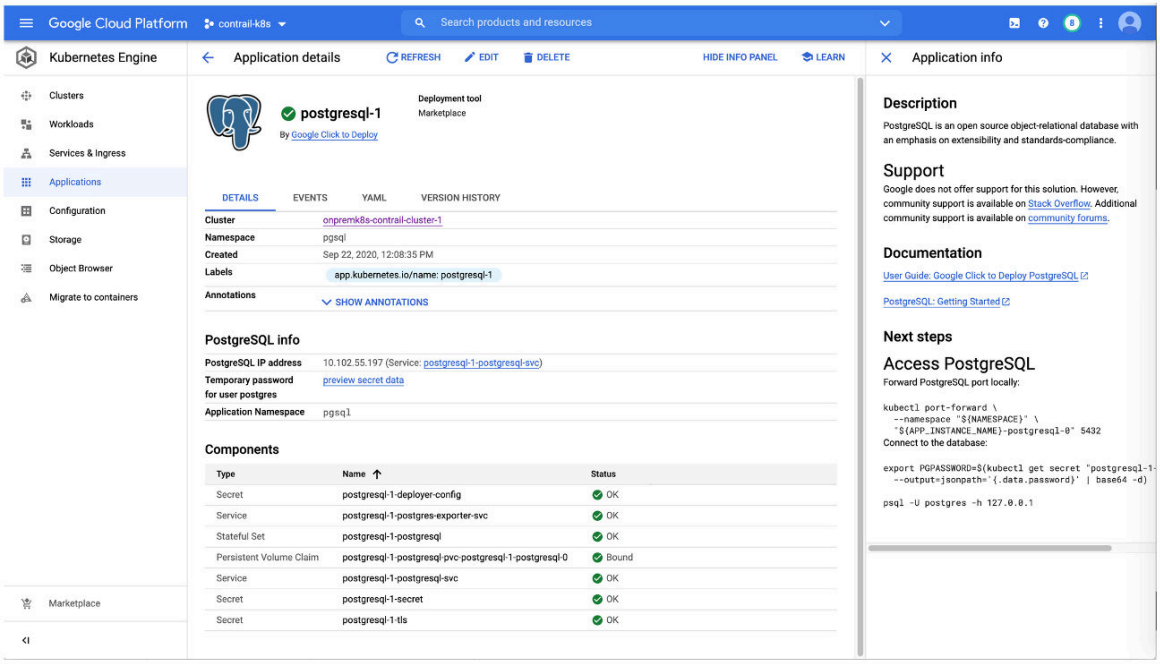
12. Select the namespace that you previously created from the **Namespace** drop-down menu and set the **StorageClass** as *standard*.



Click **Deploy**. Wait a couple of minutes.

The **Application details** screen appears.

Review the **Status** row in the **Components** table to confirm that all components successfully deployed.

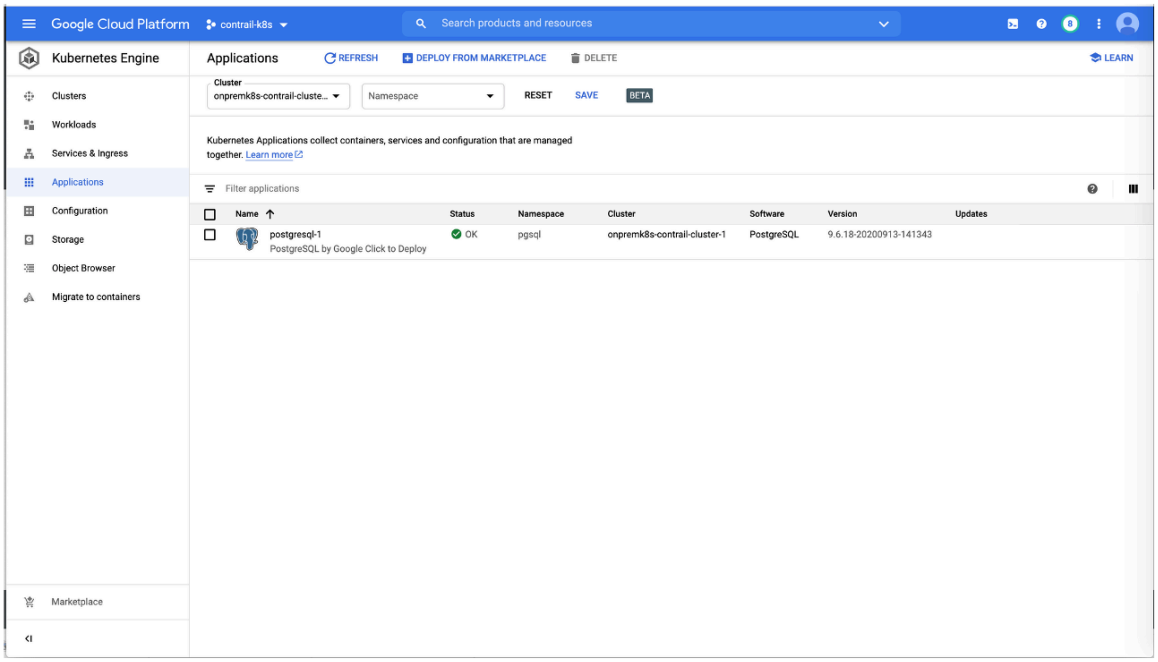


You can also verify that the app is running from the CLI:

```
$ kubectl get po -n pgsql
NAME                          READY   STATUS      RESTARTS   AGE
postgresql-1-deployer-nzpfn   0/1     Completed   0          91s
postgresql-1-postgresql-0     2/2     Running     0          46s
```

```
$ kubectl get pvc
NAME                                                    STATUS    VOLUME
CAPACITY   ACCESS MODES   STORAGECLASS   AGE
postgresql-1-postgresql-pvc-postgresql-1-postgresql-0   Bound     local-pv-e00b14f6
62Gi       RWO            standard       91s
```

13. Use filtering within the GKE Console to see the applications deployed in the on-premises cluster.



14. To access the application:

- Forward the PostgreSQL port locally:

```
$ export NAMESPACE=pgsql
$ export APP_INSTANCE_NAME="postgresql-1"
$ kubectl port-forward --namespace "${NAMESPACE}" "${APP_INSTANCE_NAME}-postgresql-0"
5432
Forwarding from 127.0.0.1:5432 -> 5432
Forwarding from [::1]:5432 -> 5432
```

- Connect to the database

```
$ apt -y install postgresql-client-10 postgresql-client-common
$ export PGPASSWORD=$(kubectl get secret "postgresql-1-secret" --
output=jsonpath='{.data.password}' | base64 -d)
```

```
$ psql (10.12 (Ubuntu 10.12-0ubuntu0.18.04.1), server 9.6.18)
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256,
compression: off)
Type "help" for help.

postgres=#
```

AWS Elastic Kubernetes Service Cluster: How to Deploy an Application from Google Marketplace

You can deploy an application from the Google Marketplace into an EKS cluster that is using Contrail Networking in AWS after the cluster is enabled in Anthos. This procedure will illustrate this process by deploying Prometheus and Grafana from Google Marketplace

Perform the following steps to deploy an application from Google Marketplace onto an EKS cluster in AWS that is using Contrail Networking.

1. Enable credentials within the *eks-contrail* context:

```
$ kubectx eks-contrail
Switched to context "eks-contrail"

$ kubectl create ns application-system

$ kubens application-system
Context "kubernetes-admin@kubernetes" modified.
Active namespace is "application-system".

$ kubectl create secret docker-registry gcr-json-key \
--docker-server=https://marketplace.gcr.io \
--docker-username=_json_key \
--docker-password="$(cat ./gcr-sa.json)" \
--docker-email=[GCP_EMAIL_ADDRESS]

$ kubectl patch serviceaccount default -p '{"imagePullSecrets": [{"name": "gcr-json-key"}]}'

$ kubectl annotate namespace application-system marketplace.cloud.google.com/
imagePullSecret=gcr-json-key
```

2. The GCP Marketplace expects a storage class named *standard* to be configured in a context. The default story class name in EKS, however, is *gp2*.

   To change the storage class name:

a. Remove the default flag from the *gp2* storage class using the patch command:

```
$  kubectl patch storageclass gp2 -p '{"metadata": {"annotations":
{"storageclass.kubernetes.io/is-default-class":"false"}}}'
```

b. Create a new storage class for the Amazon EKS context and mark it as the default storage class:

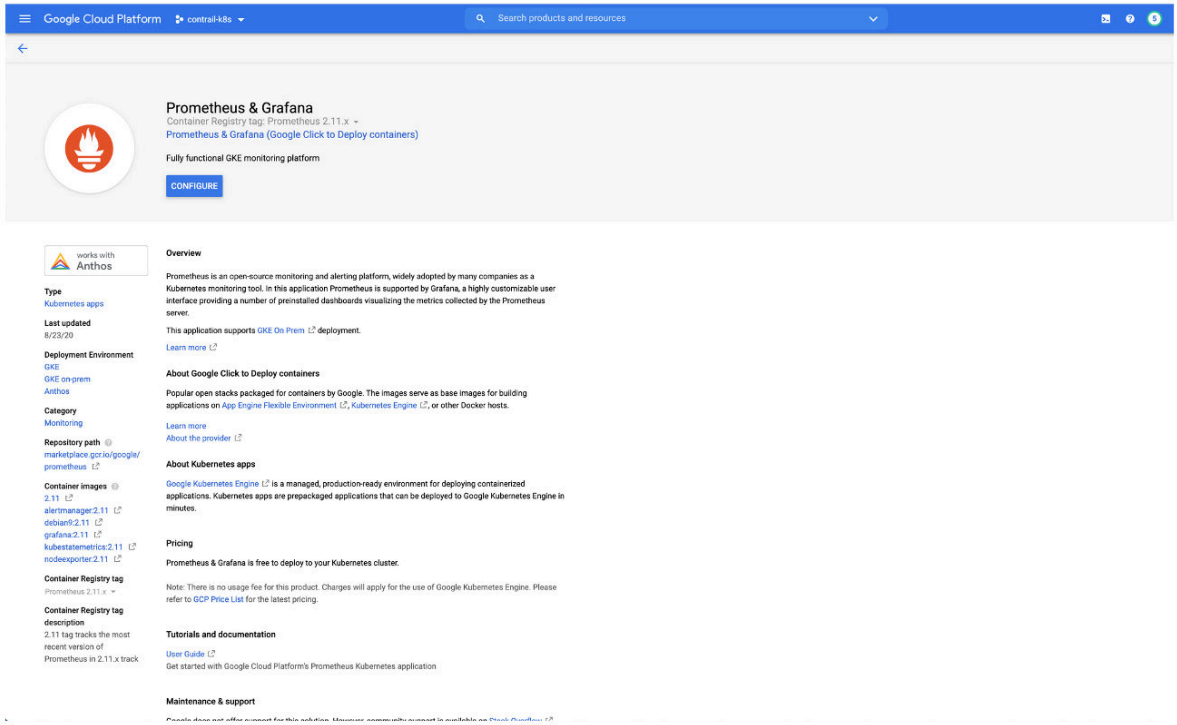
```
$ cat <<EOF > eks-sc.yaml
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: standard
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
provisioner: kubernetes.io/aws-ebs
parameters:
  type: gp2
  fsType: ext4
EOF

$ kubectl create -f eks-sc.yaml
storageclass.storage.k8s.io/standard created

$ kubectl get sc
NAME                 PROVISIONER             AGE
gp2                  kubernetes.io/aws-ebs   2d
standard (default)   kubernetes.io/aws-ebs   5s
```

3. Create a namespace for the applications:

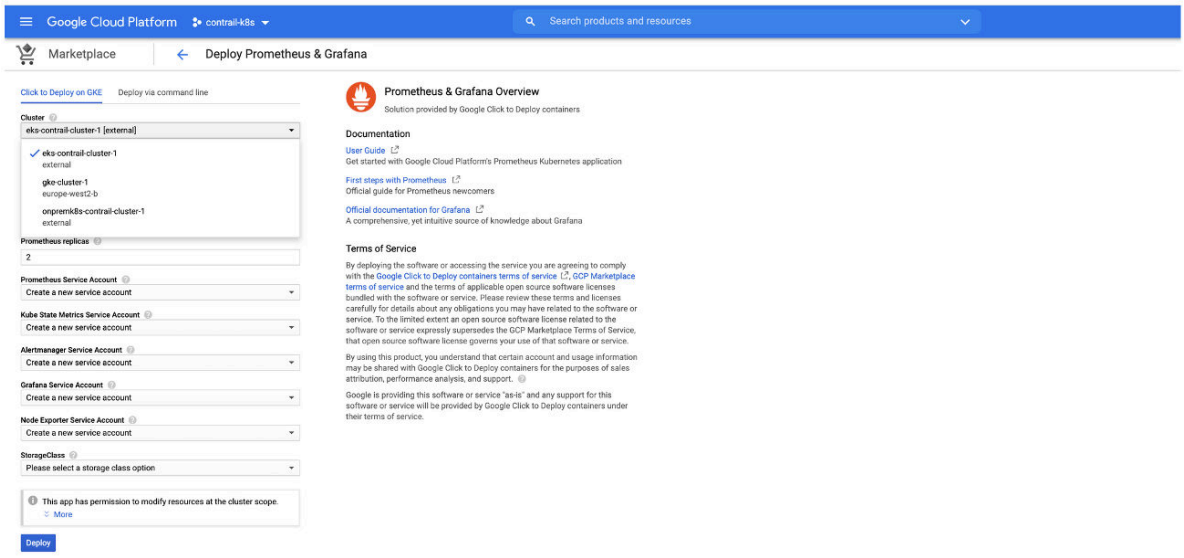
```
$ kubectl create ns monitoring

$ kubens monitoring

kubectl create secret docker-registry gcr-json-key \
 --docker-server=https://gcr.io \
--docker-username=_json_key \
--docker-password="$(cat ./gcr-sa.json)" \
--docker-email=[GCP_EMAIL_ADDRESS]

$ kubectl patch serviceaccount default -p '{"imagePullSecrets": [{"name": "gcr-json-key"}]}'
```

```
$ kubectl annotate namespace monitoring marketplace.cloud.google.com/imagePullSecret=gcr-json-key
```

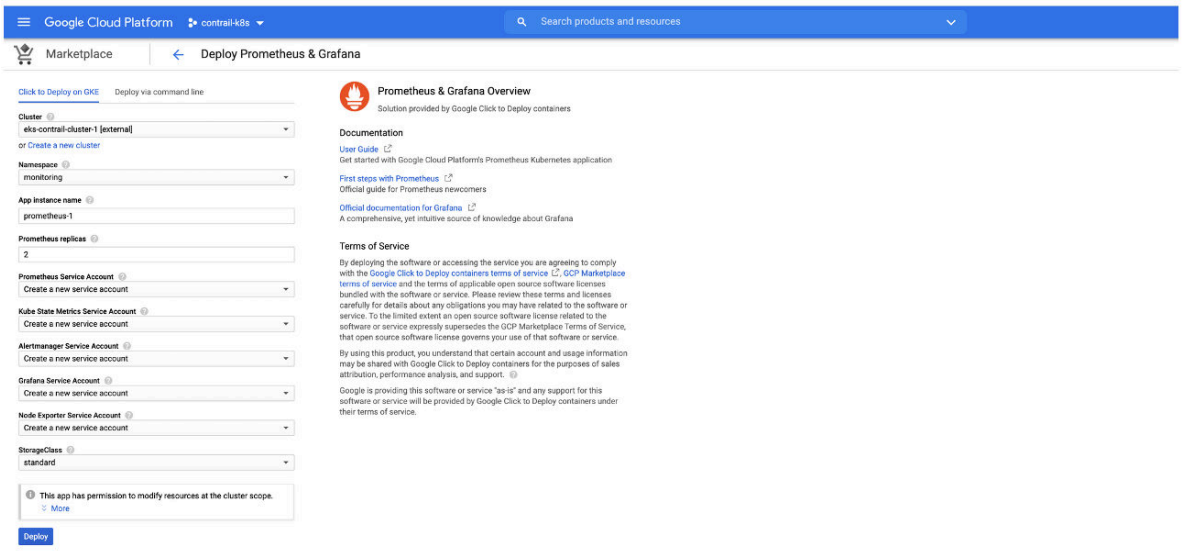
4. Choose Prometheus and Grafana from GCP Marketplace. Click the **Configure** button to start the deployment procedure.



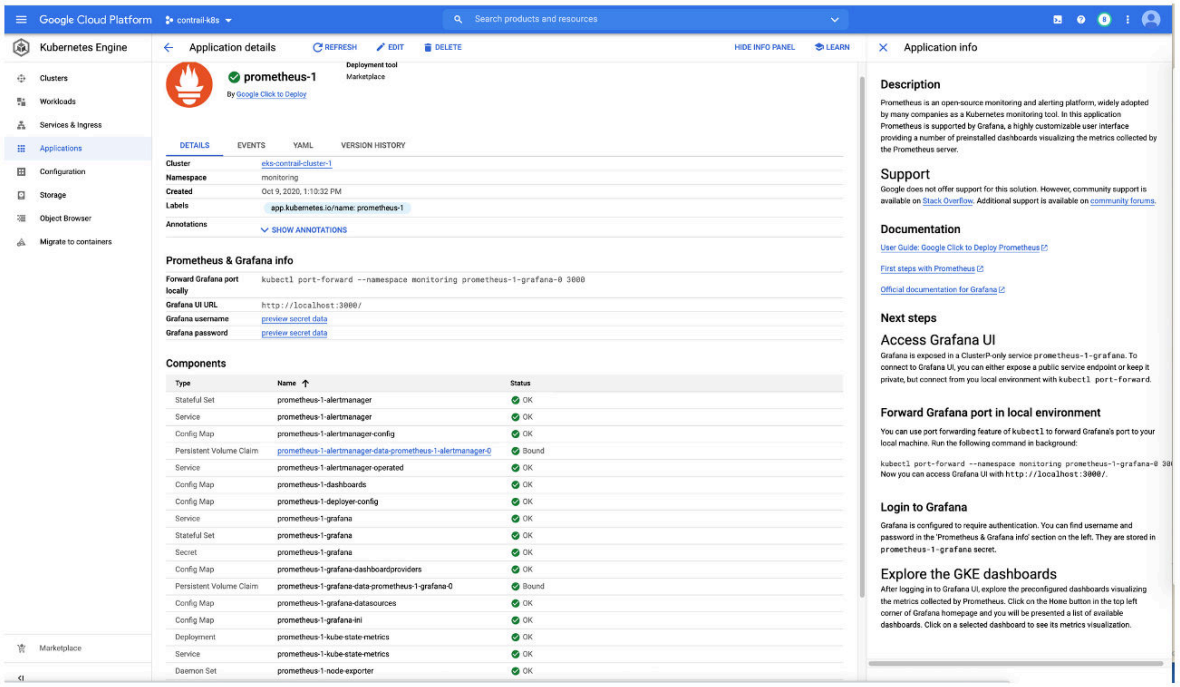
5. Choose the EKS cluster from the cluster drop-down menu.



6. Select the namespace and storage class. Click Deploy.



Wait several minutes for the application to deploy.



You can also verify that the application has deployed using the CLI:

```
$ kubectl get pods -n monitoring
NAME                                  READY   STATUS      RESTARTS   AGE
prometheus-1-alertmanager-0           1/1     Running     0          2m36s
prometheus-1-alertmanager-1           1/1     Running     0          88s
prometheus-1-deployer-blm5f           0/1     Completed   0          3m20s
prometheus-1-grafana-0                1/1     Running     0          2m36s
```

```
prometheus-1-kube-state-metrics-6f64b67684-shtdg   2/2     Running     0          2m37s
prometheus-1-node-exporter-5scf4                   1/1     Running     0          2m36s
prometheus-1-node-exporter-gdp77                   1/1     Running     0          2m36s
prometheus-1-node-exporter-k8vfn                   1/1     Running     0          2m36s
prometheus-1-node-exporter-v6w7g                   1/1     Running     0          2m36s
prometheus-1-node-exporter-zffs9                   1/1     Running     0          2m36s
prometheus-1-prometheus-0                          1/1     Running     0          2m36s
prometheus-1-prometheus-1                          1/1     Running     0          2m36s
```

7. If you have a private service, consider how your going to make it accessible.

   In this case, the Grafana user interface is exposed in the ClusterP-only service named *prometheus-1-grafana*. To connect to the Grafana user interface, either change the service to a public service endpoint or keep the service private and access it from your local environment.

```
kubectl get svc -n monitoring
NAME                                TYPE        CLUSTER-IP       EXTERNAL-IP
PORT(S)             AGE
prometheus-1-alertmanager           ClusterIP   10.100.92.6      <none>        9093/
TCP            10m
prometheus-1-alertmanager-operated  ClusterIP   None             <none>        6783/TCP,9093/
TCP   10m
prometheus-1-grafana                ClusterIP   10.100.126.78    <none>        80/
TCP              10m
prometheus-1-kube-state-metrics     ClusterIP   10.100.46.18     <none>        8080/TCP,8081/
TCP   10m
prometheus-1-prometheus             ClusterIP   10.100.214.104   <none>        9090/
TCP            10m
```

   You can use the kubectl port forwarding feature to forward Graffana traffic to your local machine by running the following command:

```
$ kubectl port-forward --namespace monitoring prometheus-1-grafana-0 3000
Now you can access Grafana UI with http://localhost:3000/.
```

Configuration Management in Anthos

IN THIS SECTION

This section covers Configuration Management in Anthos.

It includes the following sections:

Overview: Anthos Configuration Management

Google Cloud uses a tool called Config Sync that acts as the bridge between an external source code repository and the Kubernetes API server. See Config Sync overview from Google Cloud for additional information.

Anthos Configuration Management (ACM) uses Config Sync to extend configuration to non-GCP clusters that are connected using Anthos.

In the following sections, a GitHub repository is used as a single source for deployments and configuration. An ACM component is installed onto each of the clusters that are included with Anthos to monitor the external repositories for changes and synchronizing them across Anthos.

GitOps-style deployments are used in the following procedures to push workloads across all registered clusters through Anthos Config Management. GitOps provides a method of performing Kubernetes cluster management and application delivery. It works by using Git as a single source of truth for declarative infrastructure and applications and using the YAML or JSON files used in Kubernetes to combine with Anthos for code.

Installing the Configuration Management Operator

The Configuration Management Operator is a controller that manages installation of the Anthos Configuration Manager. The operator will be installed on all three clusters using these instructions.

To install the Configuration Management Operator:

1. Download the Configuration Management Operator and apply it to each cluster:

```
gsutil cp gs://config-management-release/released/latest/config-management-operator.yaml
config-management-operator.yaml

$ kubectl create -f config-management-operator.yaml
customresourcedefinition.apiextensions.k8s.io/configmanagements.configmanagement.gke.io
configured
clusterrolebinding.rbac.authorization.k8s.io/config-management-operator configured
clusterrole.rbac.authorization.k8s.io/config-management-operator configured
serviceaccount/config-management-operator configured
deployment.apps/config-management-operator configured
namespace/config-management-system configured
```

   Run this command in each cluster.

2. Confirm that the operator was created:

```
$ kubectl describe crds configmanagements.configmanagement.gke.io
Name:         configmanagements.configmanagement.gke.io
Namespace:
Labels:       controller-tools.k8s.io=1.0
Annotations:  <none>
API Version:  apiextensions.k8s.io/v1
Kind:         CustomResourceDefinition
Metadata:
  Creation Timestamp:  2020-10-09T13:13:17Z
  Generation:          1
  Resource Version:    363244
  Self Link:           /apis/apiextensions.k8s.io/v1/customresourcedefinitions/
configmanagements.configmanagement.gke.io
  UID:                 a088edbc-8232-419f-8f42-365fa36de110
Spec:
  Conversion:
    Strategy:  None
  Group:       configmanagement.gke.io
  Names:
    Kind:                   ConfigManagement
    List Kind:              ConfigManagementList
    Plural:                 configmanagements
```

```
    Singular:               configmanagement
....
```

Configuring the Clusters for Anthos Configuration Management

To configure the clusters for Anthos Configuration Management:

1. Create an SSH keypair to allow the Operator to authenticate to your Git repository:

```
$ ssh-keygen -t rsa -b 4096 -C "git-user1" -N '' -f "~/.ssh/gke-github"
```

2. Configure your repository to recognize the newly-created public key. See Adding a new SSH key to your GitHub account from GitHub.

   Add a private key to a new secret in the cluster:

```
$ kubectl create secret generic git-creds \
  --namespace=config-management-system \
  --from-file=ssh="/Users/user1/.ssh/gke-github"
```

   Repeat this step for each individual cluster

3. (Optional) Gather the name of each cluster, if needed:

```
$ gcloud container hub memberships list
NAME                          EXTERNAL_ID
onpremk8s-contrail-cluster-1  78f7890b-3a43-4bc7-8fd9-44c76953781b
eks-contrail-cluster-1        42e532ba-a0d9-4087-baed-647be8bca7e9
gke-cluster-1                 6671599e-87af-461b-aff9-7105ebda5c66
```

4. Create a config-management.yaml file for each cluster. Replace the clusterName with the registered clustered name in Anthos in each file.

```
$ cat config-management.yaml
apiVersion: configmanagement.gke.io/v1
kind: ConfigManagement
metadata:
  name: config-management
spec:
  # clusterName is required and must be unique among all managed clusters
  clusterName:
  git:
```

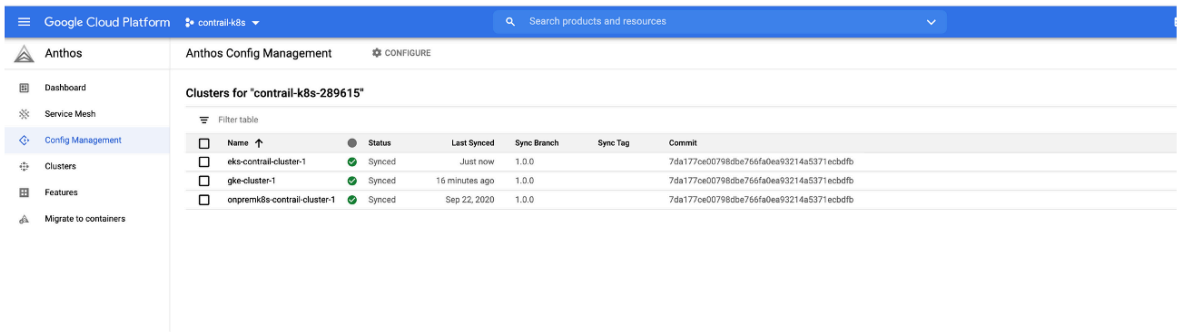
```
    syncRepo: git@github.com:git-user1/csp-config-management.git
    syncBranch: 1.0.0
    secretType: ssh
    policyDir: foo-corp
    proxy: {}

$ kubectx eks-contrail
$ kubectl apply -f config-management.yaml

$ kubectx onprem-k8s-contrail
$ kubectl apply -f config-management.yaml

$ kubectx gke
$ kubectl apply -f config-management.yaml
```

5. Verify that the pods are running on each cluster.

   To verify in the CLI:

```
$ kubectl get pods -n config-management-system
NAME                              READY   STATUS    RESTARTS   AGE
git-importer-584bd49676-46bjq     3/3     Running   0          4m23s
monitor-c8c68d5ff-bdhzl           1/1     Running   0          4m25s
syncer-7dbbc8868c-gtp8d           1/1     Running   0          4m25s
```

   To verify on the Anthos dashboard:



Using Nomos to Manage the Anthos Configuration Manager

The Google Cloud Platform offers a utility called Nomos which can be used to manage the Anthos Configuration Manager (ACM). See Using the nomos command from Google Cloud for more information on Nomos.

To enable Nomos:

1. Get the utility and copy it into a local directory:

```
$ gsutil cp gs://config-management-release/released/latest/darwin_amd64/nomos nomos

$ cp ./nomos /usr/local/bin
$ chmod +x /usr/local/bin/nomos
```

2. Verify that nomos is running in the clusters connected using Anthos:

```
$ nomos status
Connecting to clusters...
Current   Context              Sync Status      Last Synced Token   Sync Branch   Resource
Status
-------   -------              -----------      -----------------   -----------
---------------
*         eks-contrail         SYNCED           7da177ce            1.0.0         Healthy
          gke                  SYNCED           7da177ce            1.0.0         Healthy
          onprem-k8s-contrail  SYNCED           7da177ce            1.0.0         Healthy
```

3. List the namespaces that are currently managed by Anthos Configuration Management.

   In this sample output, configurations are stored in the cluster/ and namespace/ directories. All objects managed by Anthos Config Management have the app.kubernetes.io/managed-by label set to configmanagement.gke.io.

```
$ kubectl get ns -l app.kubernetes.io/managed-by=configmanagement.gke.io
NAME              STATUS   AGE
audit             Active   13m
shipping-dev      Active   13m
shipping-prod     Active   13m
shipping-staging  Active   13m
```

4. In the following sequence, we'll validate that nomos and Anthos Configuration Management are efficiently managing the configuration of configuration in a third-party cluster by deleting a namespace in EKS and confirming that a new namespace is quickly recreated.

```
$ kubectx eks-contrail

$ kubectl delete ns audit
namespace "audit" deleted
```

```
$ kubectl get ns audit
NAME    STATUS   AGE
audit   Active   5s
```

The output shows that a new audit workspace was created 5 seconds ago, confirming that Anthos Configuration Management is working.

CHAPTER 6

Using Contrail Networking with Kubernetes

Provisioning of Kubernetes Clusters

IN THIS SECTION

NOTE: This topic covers Contrail Networking in Kubernetes-orchestrated environments that are using Contrail Networking Release 21-based releases.

Starting in Release 22.1, Contrail Networking evolved into Cloud-Native Contrail Networking. Cloud-Native Contrail Networking offers significant enhancements to optimize networking performance in Kubernetes-orchestrated environments. We recommend using Cloud-Native Contrail for networking in most Kubernetes-orchestrated environments.

For general information about Cloud-Native Contrail, see the Cloud-Native Contrail Networking Techlibrary homepage.

Contrail Networking supports the following ways of provisioning Kubernetes clusters:

Provisioning of a Standalone Kubernetes Cluster

You can provision a standalone Kubernetes cluster using contrail-ansible-deployer.

Perform the following steps to install one Kubernetes cluster and one Contrail cluster and integrate them together.

1. See the Supported Platforms document for a list of supported Contrail Networking version and orchestration combinations.
2. Install the necessary tools.

   ```
   yum -y install epel-release git ansible net-tools
   ```

3. Download the `contrail-ansible-deployer-19<xx>.<NN>.tgz` Ansible Deployer application tool package onto your provisioning host from Contrail Downloads page and extract the package.

   ```
   - tar xvf contrail-ansible-deployer-19<xx>.<NN>.tgz
   ```

4. Navigate to the **contrail-ansible-deployer** directory.

   ```
   cd contrail-ansible-deployer
   ```

5. Edit the **config/instances.yaml** and enter the necessary values. See *Understanding contrail-ansible-deployer used in Contrail Command* for a sample **config/instances.yaml** file.

6. Turn off the `swap` functionality on all nodes.

   ```
   swapoff -a
   ```

7. Configure the nodes.

   ```
   ansible-playbook -e orchestrator=kubernetes -i inventory/ playbooks/configure_instances.yml
   ```

8. Install Kubernetes and Contrail.

   ```
   ansible-playbook -e orchestrator=kubernetes -i inventory/ playbooks/install_k8s.yml

   ansible-playbook -e orchestrator=kubernetes -i inventory/ playbooks/install_contrail.yml
   ```

9. Turn on the `swap` functionality on all nodes.

   ```
   swapon -a
   ```

Provisioning of Nested Contrail Kubernetes Clusters

IN THIS SECTION

When Contrail provides networking for a Kubernetes cluster that is provisioned on the workloads of a Contrail-OpenStack cluster, it is called a nested Kubernetes cluster. Contrail components are shared between the two clusters.

Prerequisites

Ensure that the following prerequisites are met before provisioning a nested Kubernetes cluster:

1. Ensure that you have an operational Contrail-OpenStack cluster based on Contrail Networking Release 19<xx>..

2. Ensure that you have an operational Kubernetes v1.12.9 cluster on virtual machines created on an Contrail-OpenStack cluster.

3. Update the **/etc/hosts** file on the Kubernetes primary node with entries for each node of the cluster.

   For example, if the Kubernetes cluster is made up of three nodes such as master1 (IP: x.x.x.x), minion1 (IP: y.y.y.y), and minion2 (IP: z.z.z.z). The **/etc/hosts** on the Kubernetes primary node must have the following entries:

   ```
   x.x.x.x master1
   y.y.y.y minion1
   z.z.z.z minion2
   ```

4. If Contrail container images are stored in a secure docker registry, a Kubernetes secret must be created and referenced during "Generate a single yaml file to create a Contrail-k8s cluster" on page 210, with credentials of the private docker registry.

   ```
   kubectl create secret docker-registry name --docker-server=registry --docker-
   username=username --docker-password=password --docker-email=email -n namespace
   ```

   Command options:

   - *name*—Name of the secret.
   - *registry*—Name of the registry. Example: hub.juniper.net/contrail.
   - *username*—Username to log in to the registry.
   - *password*—Password to log in to the registry.
   - *email*—Registered email of the registry account.
   - *namespace*—Kubernetes namespace where the secret must be created. This should be the namespace where you intend to create the Contrail pods.

The following steps describe how to provision a nested Contrail Kubernetes cluster.

Configure network connectivity to Contrail configuration and data plane functions.

A nested Kubernetes cluster is managed by the same Contrail control processes that manage the underlying OpenStack cluster.

The kube-manager is essentially a part of the Contrail Config function. In a nested deployment, one kube-manager instance will is provisioned in each overlay cluster. This necessitates the need The kube-manager running in the overlay must have network reachability to Contrail config functions of the underlay OpenStack cluster.

Network connectivity for the following Contrail config functions are required:

- Contrail Config
- Contrail Analytics
- Contrail Msg Queue
- Contrail VNC DB
- Keystone

In addition to config connectivity, the CNI for the Kubernetes cluster needs network reachability to the vRouter on its Compute node. Network connectivity for the vRouter data plane function is also required.

You can use the link local service feature or a combination of link local service with fabric Source Network Address Translation (SNAT) feature of Contrail to provide IP reachability to and from the overlay Kubernetes cluster config and data components to corresponding config and data compoenents of the underlay OpenStack cluster.

To provide IP reachability to and from the Kubernetes cluster using the fabric SNAT with link local service, perform the following steps.

1. Enable fabric SNAT on the virtual network of the VMs.

   The fabric SNAT feature must be enabled on the virtual network of the virtual machines on which the Kubernetes primary and minions are running.
2. Create a link local service for the Container Network Interface (CNI) to communicate with its vRouter Agent. This link local service should be configured using the Contrail GUI, in the following example:

Contrail Process	Service IP	Service Port	Fabric IP	Fabric Port
vRouter	*Service-IP for the active node*	9091	127.0.0.1	9091

NOTE: Fabric IP address is 127.0.0.1 since you must make the CNI communicate with the vRouter on its underlay node.

For example, the following link local services must be created:

Link Local Service Name	Service IP	Service Port	Fabric IP	Fabric Port
K8s-cni-to-agent	10.10.10.5	9091	127.0.0.1	9091

NOTE: Here 10.10.10.5 is the Service IP address that you chose. This can be any unused IP in the cluster. This IP address is primarily used to identify link local traffic and has no other significance.

Generate a single yaml file to create a Contrail-k8s cluster

Contrail components are installed on the Kubernetes cluster as pods. The configuration to create these pods in Kubernetes is encoded in a yaml file.

This file can be generated as follows:

1. Download the `contrail-ansible-deployer-19<xx>.<NN>.tgz` Ansible Deployer application tool package onto your provisioning host from Juniper Networks and extract the package.

   `- tar xvf contrail-ansible-deployer-19<xx>.<NN>.tgz`

2. Navigate to the **contrail-container-builder** directory.

   ```
   cd contrail-container-builder
   ```

3. Populate the **common.env** file located in the top directory of the cloned contrail-container-builder repo with information corresponding to your cluster and environment.

   For a sample **common.env** file with the required bare minimum configurations, see the common.env.sample.nested_mode sample configuration file.

   NOTE: If Contrail container images are stored in a secure docker registry, a Kubernetes secret must be created and referenced as documented in "4" on page 208 of Prerequisites. Populate the variable KUBERNETES_SECRET_CONTRAIL_REPO=<*secret-name*> with the name of the generated Kubernetes secret, in the **common.env** file.

4. Generate the yaml file as following in your shell:

   ```
   cd contrail-container-build-repo/kubernetes/manifests

   ./resolve-manifest.sh contrail-kubernetes-nested.yaml  > nested-contrail.yml
   ```

5. Copy the output (or file) generated from 4 to the primary node in your Kubernetes cluster.

Instantiate the Contrail-k8s cluster

Create contrail components as pods on the Kubernetes cluster.

```
root@k8s:~# kubectl get pods -n kube-system
NAME                                READY     STATUS    RESTARTS   AGE
contrail-kube-manager-lcjbc         1/1       Running   0          3d
contrail-kubernetes-cni-agent-w8shc   1/1       Running   0          3d
```

You will see the following pods running in the kube-system namespace:

contrail-kube-manager-xxxxxx—This is the manager that acts as conduit between Kubernetes and OpenStack clusters

contrail-kubernetes-cni-agent-xxxxx—This installs and configures Contrail CNI on Kubernetes nodes

Provisioning of Non-Nested Contrail Kubernetes Clusters

Prerequisites

Ensure that the following prerequisites are met before provisioning a non-nested Kubernetes cluster:

1. You must have an installed and operational Contrail OpenStack cluster based on the Contrail Networking Release 19*xx* release.
2. You must have an installed and operational Kubernetes cluster on the server where you want to install the non-nested Contrail Kubernetes cluster.
3. Label the Kubernetes primary node with the Contrail controller label:

   ```
   kubectl label node node node-role.opencontrail.org/config=true
   ```

4. Ensure that the Kubelet running on the Kubernetes primary node is not run with network plugin options. If kubelet is running with network plugin option, then disable or comment out the KUBELET_NETWORK_ARGS option in the **/etc/systemd/system/kubelet.service.d/10-kubeadm.conf** configuration file.

   NOTE: It is recommended that the Kubernetes primary should not be configured with a network plugin, so as to not install vRouter kernel module on the control node. However, this is optional.

5. Restart the kubelet service:

```
systemctl daemon-reload;
systemctl restart kubelet.service
```

In non-nested mode, a Kubernetes cluster is provisioned side by side with an OpenStack cluster with networking provided by the same Contrail components of the OpenStack cluster.

Provisioning a Contrail Kubernetes Cluster

Follow these steps to provision Contrail Kubernetes cluster.

1. Download the `contrail-ansible-deployer-19<xx>.<NN>.tgz` Ansible Deployer application tool package onto your provisioning host from Juniper Networks and extract the package.

   `- tar xvf contrail-ansible-deployer-19<xx>.<NN>.tgz`

2. Navigate to the **contrail-container-builder** directory.

   `cd contrail-container-builder`

3. Populate the **common.env** file located in the top directory of the cloned contrail-container-builder repo with information corresponding to your cluster and environment.

   For a sample **common.env** file with required bare minimum configurations, see the common.env.sample.non_nested_mode sample configuration file.

   NOTE: If Config API is not secured by keystone, ensure that *AUTH_MODE* and *KEYSTONE_** variables are not configured or present while populating the **common.env** file.

4. Generate the yaml file as shown below:

```
cd contrail-container-build-repo/kubernetes/manifests

./resolve-manifest.sh contrail-kubernetes-nested.yaml  > non-nested-contrail.yml
```

5. Copy the file generated from 4 to the primary node in your Kubernetes cluster.
6. Create contrail components as pods on the Kubernetes cluster as follows:

```
kubectl apply -f non-nested-contrail.yml
```

7. Create the following Contrail pods on the Kubernetes cluster. Ensure that contrail-agent pod is created only on the worker node.

```
[root@b4s403 manifests]# kubectl get pods --all-namespaces -o wide
       NAMESPACE     NAME                              READY     STATUS    RESTARTS
AGE       IP            NODE
       kube-system   contrail-agent-mxkcq              2/2       Running   0
1m        <x.x.x.x>     b4s402
       kube-system   contrail-kube-manager-glw5m       1/1       Running   0
1m        <x.x.x.x>     b4s403
```

RELATED DOCUMENTATION

Contrail Integration with Kubernetes | 2

How to Enable Multi-Interface Pods in a Kubernetes Environment

NOTE: This topic covers Contrail Networking in Kubernetes-orchestrated environments that are using Contrail Networking Release 21-based releases.

Starting in Release 22.1, Contrail Networking evolved into Cloud-Native Contrail Networking. Cloud-Native Contrail Networking offers significant enhancements to optimize networking performance in Kubernetes-orchestrated environments. We recommend using Cloud-Native Contrail for networking in most Kubernetes-orchestrated environments.

For general information about Cloud-Native Contrail, see the Cloud-Native Contrail Networking Techlibrary homepage.

For information about enabling a multi-interface pod using Cloud-Native Contrail, see Enable Pods with Multiple Network Interfaces.

Contrail Networking, when used as the CNI in a Kubernetes environment, natively has the capability to create a Kubernetes pod with multiple interfaces.

This procedure demonstrates how to configure a multi-interface pod in Kubernetes running Contrail Networking Release 2008. In this example, two virtual networks are created and a Kubernetes pod has interfaces in each virtual network.

To configure a multi-interface pod:

1. Create two virtual networks in Contrail.

   In this example, two virtual networks—*red-net* and *green-net*—are created.

```
$ cat red-green-net.yaml
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
 name: red-net
 annotations: {
   "opencontrail.org/cidr" : "20.20.20.0/24",
   "opencontrail.org/ip_fabric_snat": "true"
   }
spec:
 config: '{
   "cniVersion": "0.3.1",
   "type": "contrail-k8s-cni"
}'

---
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
 name: green-net
 annotations: {
   "opencontrail.org/cidr" : "30.30.30.0/24",
   "opencontrail.org/ip_fabric_snat": "true"
  }
spec:
 config: '{
   "cniVersion": "0.3.1",
   "type": "contrail-k8s-cni"
}'

$ kubectl create -f red-green-net.yaml
```

2. Create a pod with network interfaces in both networks.

In this example, a pod with interfaces in *red-net* and *green-net* is created.

```
$ cat ubuntu-multi-nic.yaml
apiVersion: v1
kind: Pod
metadata:
 name: multi-intf-pod
 annotations:
   k8s.v1.cni.cncf.io/networks: '[
     { "name": "red-net" },
     { "name": "green-net" }
   ]'
spec:
 containers:
   - name: ubuntuapp
     image: ubuntu-upstart

$ kubectl create -f ubuntu-multi-nic.yaml
$ kubectl get pods | grep multi
multi-intf-pod      1/1     Running   0          5m10s
```

3. Connect to the pod to check both network interfaces.

   In this sample output, the pod has a network interface in each virtual network as well as an interface in the default pod network.

```
$ kubectl describe pod/multi-intf-pod
Name:         multi-intf-pod
Namespace:    default
Priority:     0
Node:         ru16-k8s-node2/172.16.133.155
Start Time:   Tue, 20 Oct 2020 09:00:05 -0400
Labels:       <none>
Annotations:  k8s.v1.cni.cncf.io/network-status:
                [
                    {
                        "ips": "20.20.20.252",
                        "mac": "02:2c:6f:b2:38:12",
                        "name": "red-net"
                    },
                    {
                        "ips": "30.30.30.252",
```

```
                        "mac": "02:2c:88:4b:18:12",
                        "name": "green-net"
                    },
                    {
                        "ips": "10.47.255.224",
                        "mac": "02:2c:59:66:f4:12",
                        "name": "cluster-wide-default"
                    }
                ]
              k8s.v1.cni.cncf.io/networks: [ { "name": "red-net" }, { "name": "green-net" } ]
Status:       Running
IP:           10.47.255.224


$ kubectl exec -it multi-intf-pod -- bash
root@multi-intf-pod:/# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
2: ip_vti0@NONE: <NOARP> mtu 1480 qdisc noop state DOWN group default qlen 1000
    link/ipip 0.0.0.0 brd 0.0.0.0
48: eth0@if49: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:2c:59:66:f4:12 brd ff:ff:ff:ff:ff:ff
    inet 10.47.255.224/12 scope global eth0
       valid_lft forever preferred_lft forever
50: eth1@if51: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:2c:6f:b2:38:12 brd ff:ff:ff:ff:ff:ff
    inet 20.20.20.252/24 scope global eth1
       valid_lft forever preferred_lft forever
52: eth2@if53: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:2c:88:4b:18:12 brd ff:ff:ff:ff:ff:ff
    inet 30.30.30.252/24 scope global eth2
       valid_lft forever preferred_lft forever
```

Installing Standalone Kubernetes Contrail Cluster using the Contrail Command UI

IN THIS SECTION

NOTE: This topic covers Contrail Networking in Kubernetes-orchestrated environments that are using Contrail Networking Release 21-based releases.

Starting in Release 22.1, Contrail Networking evolved into Cloud-Native Contrail Networking. Cloud-Native Contrail Networking offers significant enhancements to optimize networking performance in Kubernetes-orchestrated environments. We recommend using Cloud-Native Contrail for networking in most Kubernetes-orchestrated environments.

For general information about Cloud-Native Contrail, see the Cloud-Native Contrail Networking Techlibrary homepage.

Starting with Contrail Release 5.1, you can use Contrail Command to initiate Kubernetes Contrail cluster deployment. This example topic describes how to use the Contrail Command User interface (UI) to deploy a standalone Kubernetes Contrail cluster.

Requirements

- Contrail Controller — 8 vCPU, 64G memory, 300G storage.
- Contrail Server Node (CSN) — 4 vCPU, 16G memory, 100G storage.
- Compute nodes— Dependent on the workloads.

Overview

You can use Contrail Command to initiate a standalone Kubernetes Contrail cluster deployment. You must install the controller and compute nodes first. When the host nodes are operational, Contrail Command uses the underlying Ansible deployer to install a standalone Kubernetes Contrail cluster. Contrail Command supports the management and provisioning of Contrail components. To provision Kubernetes resources, such as pods, services, and so on, use the Kubernetes API server or the `kubectl` CLI on the Kubernetes master node.

Configuration

IN THIS SECTION

Deploying a Kubernetes Contrail Cluster

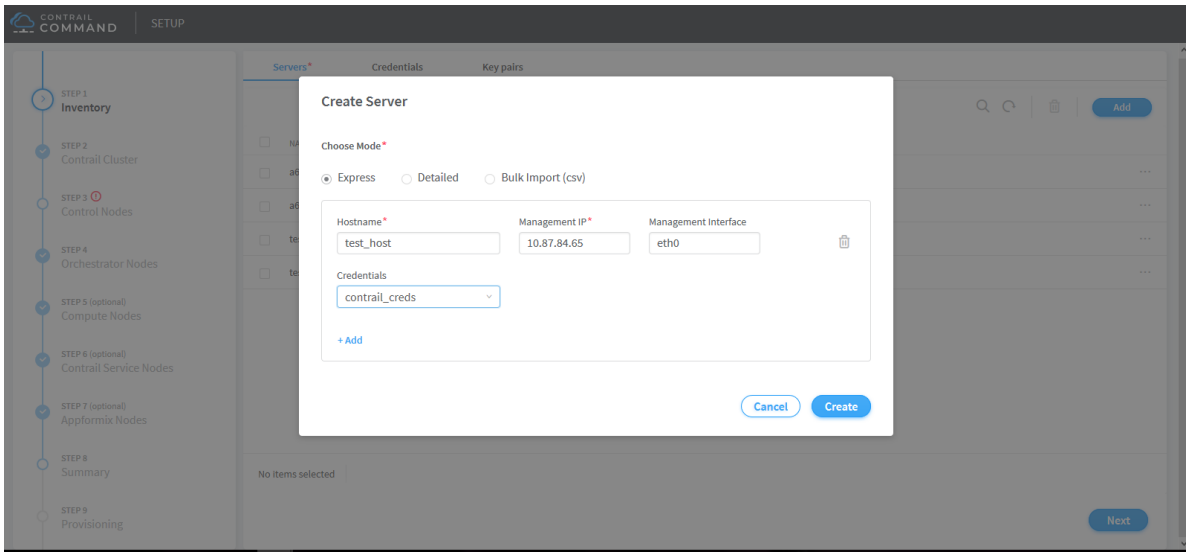
Step-by-Step Procedure

To deploy a Kubernetes Contrail cluster using Contrail Command, perform the following steps.

1. Click the **Create** button on the **Setup > Servers** tab to add physical servers. The **Create Server** page is displayed. You can add a server in the following ways:
   - Express
   - Detailed
   - Bulk Import (csv)

   NOTE: Create server login credentials before adding the servers.

Figure 2: Create Server



Click **Create** to create the server. The list of servers is displayed in the **Inventory** page. Click **Next** to continue creating a cluster. The **Contrail Cluster** page appears.

2. Create a Contrail cluster.

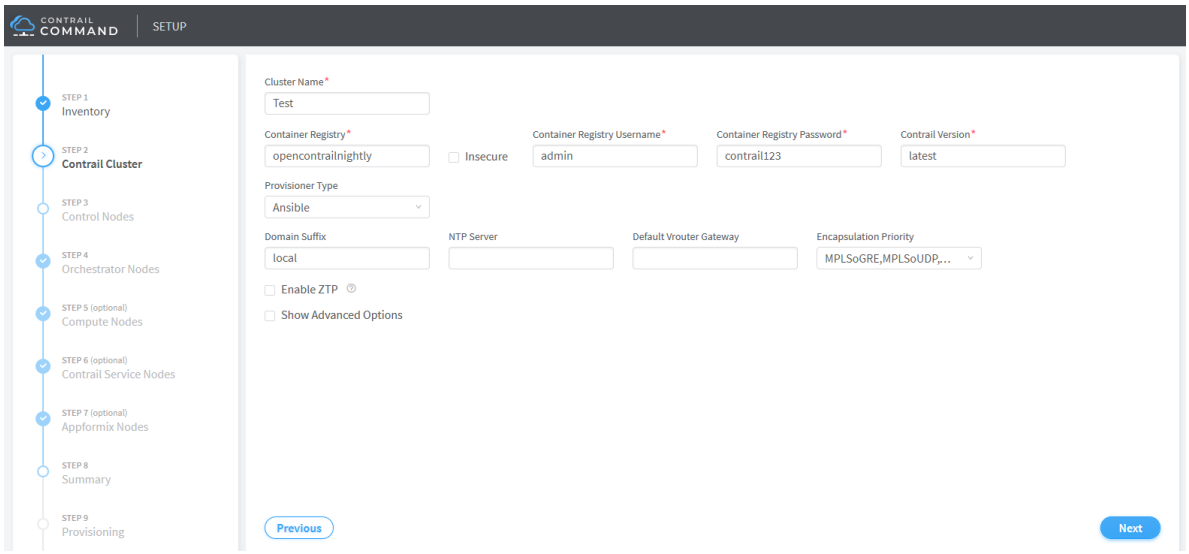
   If **Container registry** = hub.juniper.net/contrail . This registry is secure. Unselect the **Insecure** box. Also, **Contrail version** = *contrail_container_tag* for your release of Contrail as listed in README Access to Contrail Registry 20XX.

   Default vRouter Gateway = Default gateway for the compute nodes. If any one of the compute nodes has a different default gateway than the one provided here, enter that gateway in "5" on page 221 and "6" on page 222 for service nodes.

   Set the order of **Encapsulation Priority** for the EVPN supported methods - MPLS over UDP, MPLS over GRE And VxLAN.

```
VXLAN, MPLSoUDP, MPLSoGRE
```

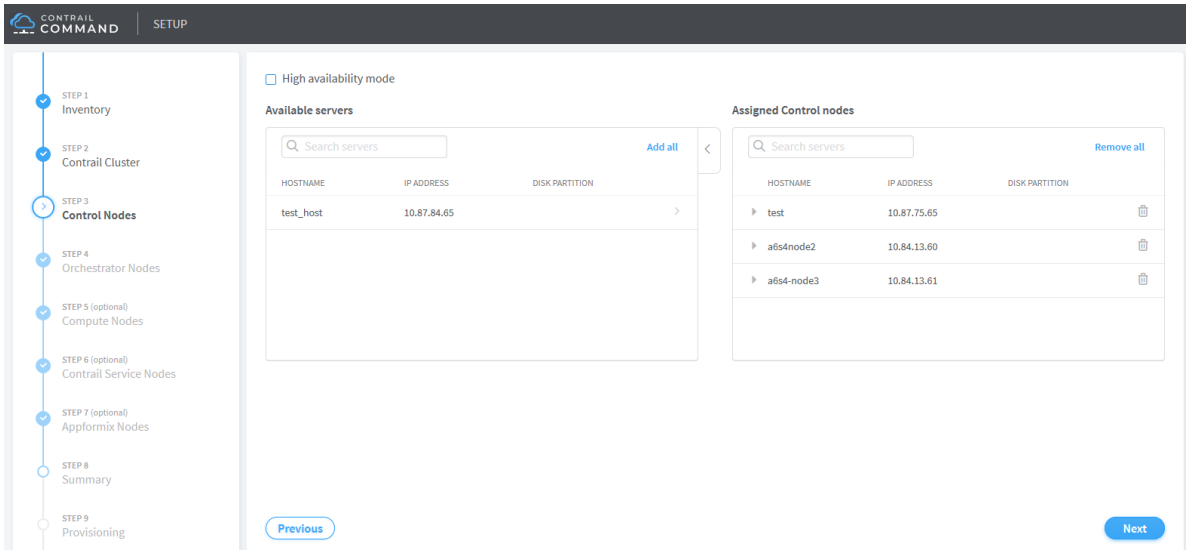
Figure 3: Contrail Cluster



Click **Next**. The **Control Nodes** page appears.

3. Select the Contrail control nodes.

Figure 4: Control Nodes



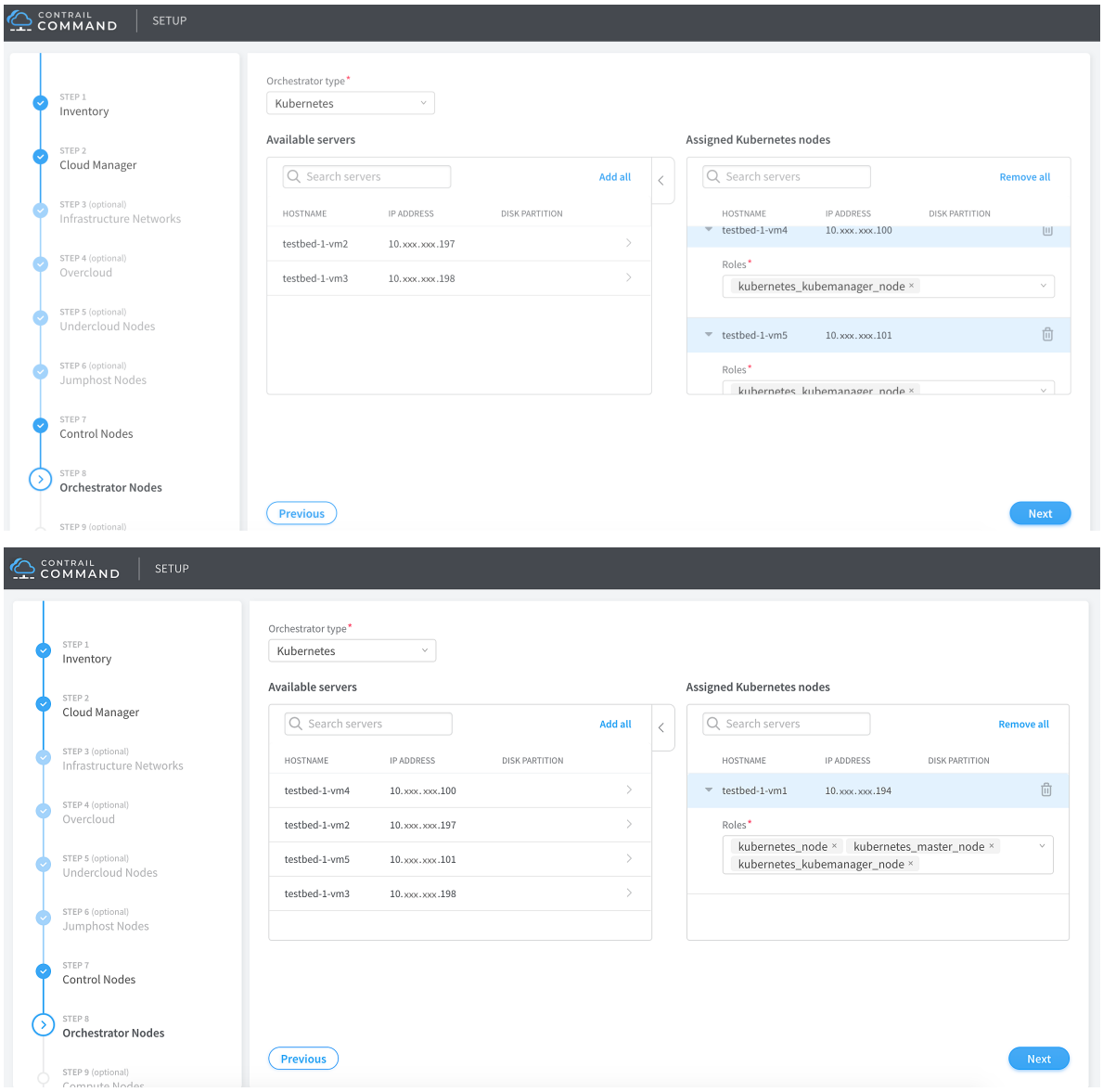
Click **Next**. The **Orchestrator Nodes** page appears.

4. Select the Kubernetes orchestration type.

   Select the Kubernetes nodes from the list of available servers.

Select the Kubernetes nodes from the list of available servers and assign corresponding roles to the servers. By default , the Kubernetes nodes are assigned the kubernetes_master_node, kubernetes_kubemanager_node, and kubernetes_node roles.

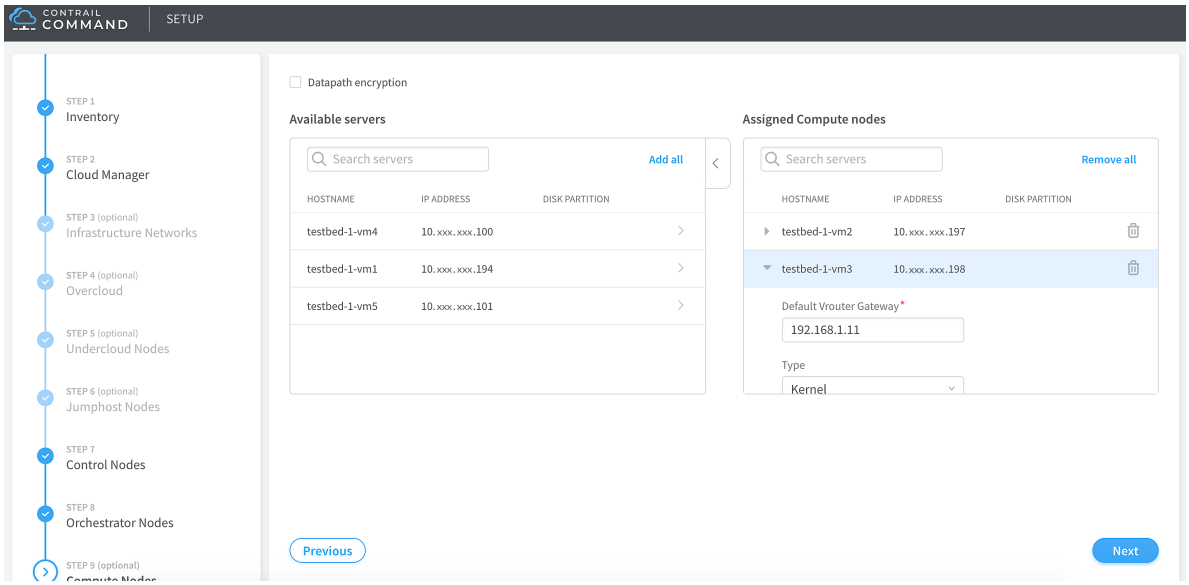
Figure 5: Orchestrator Nodes



Click **Next**. The **Compute Nodes** page appears.

5. Select the compute node associated with the kunernetes_node role from the list of available servers, .

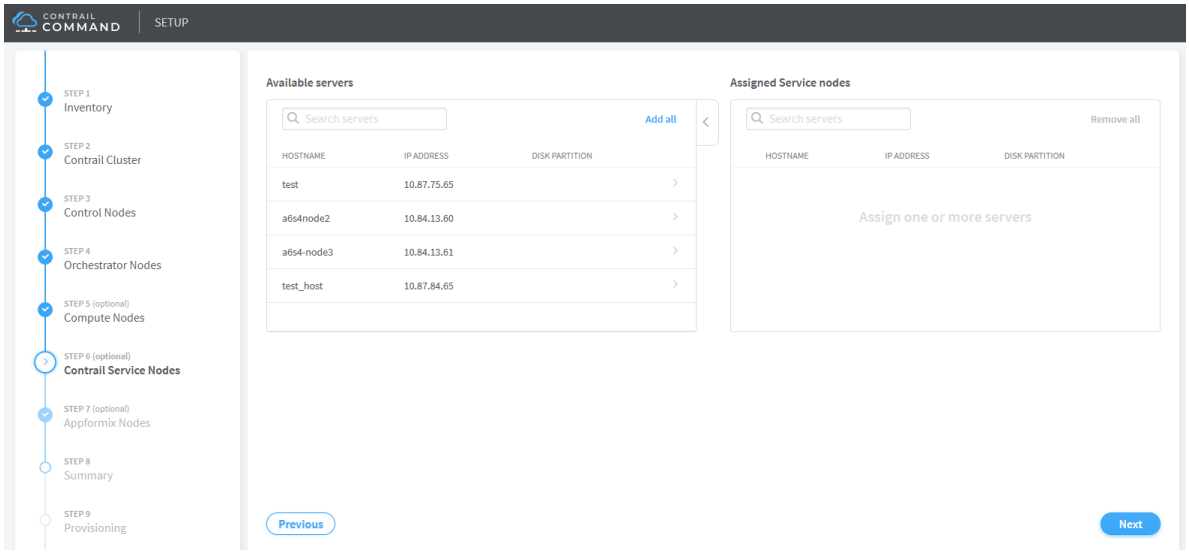
Figure 6: Compute Nodes



Click **Next**. The **Contrail Service Nodes** page appears.

6. (Optional) Select the Contrail service nodes from the list of available servers.

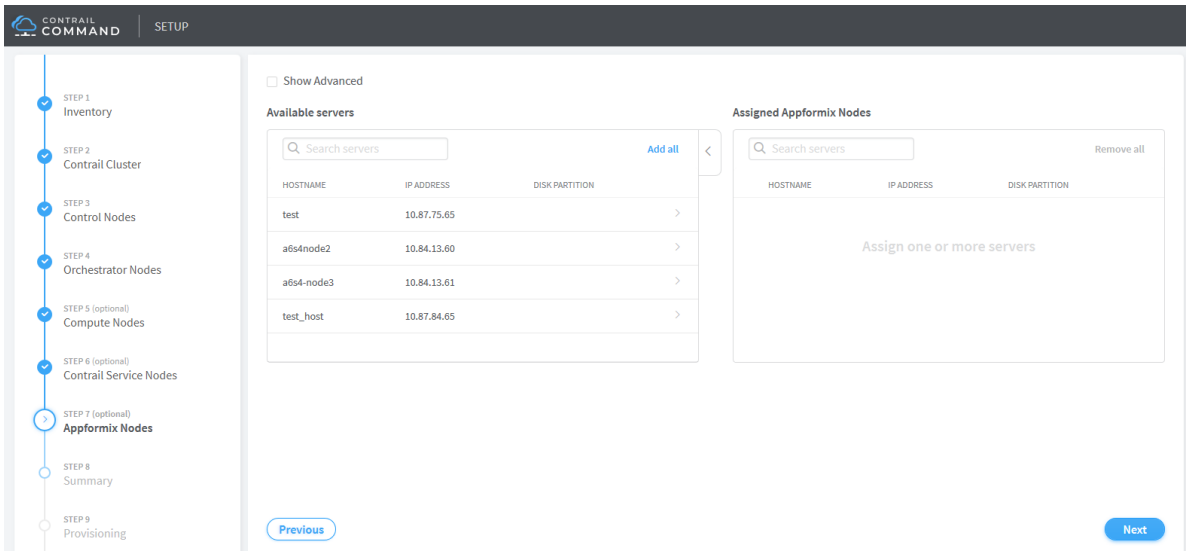
Figure 7: Contrail Service Nodes



Click **Next**. The **Appformix Nodes** page appears.

7. (Optional) Select the Contrail Insights nodes from the list of available nodes.

Figure 8: Contrail Insights Nodes



Click **Next**. The **Summary** page appears.

8. The summary page displays the cluster details as well as the node details. Verify the summary of your cluster configuration and click **Provision**.

Figure 9: Summary - Cluster Overview

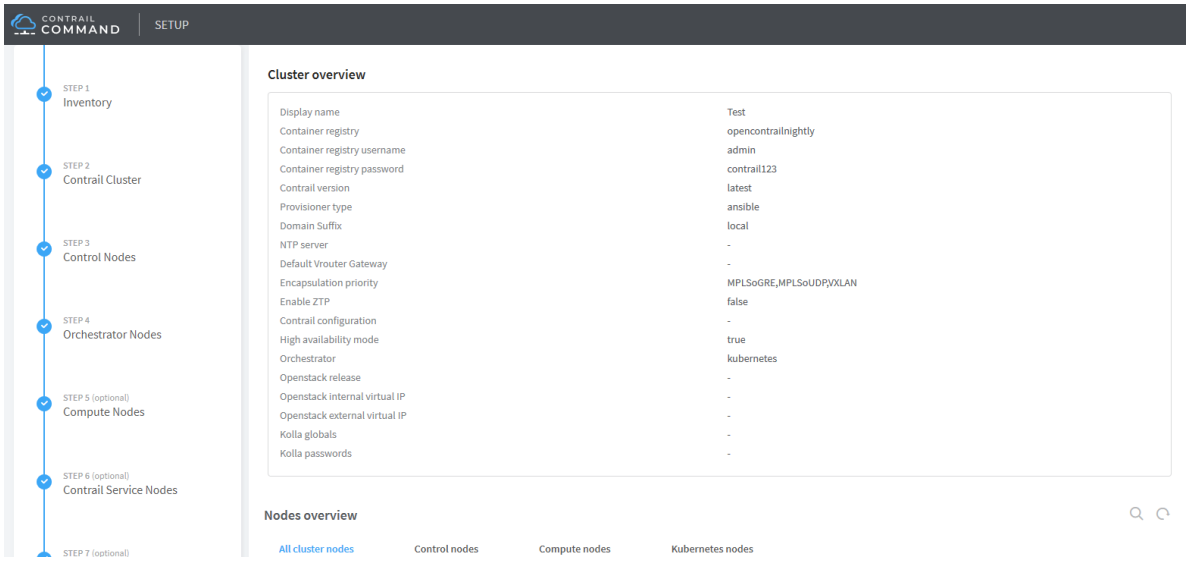
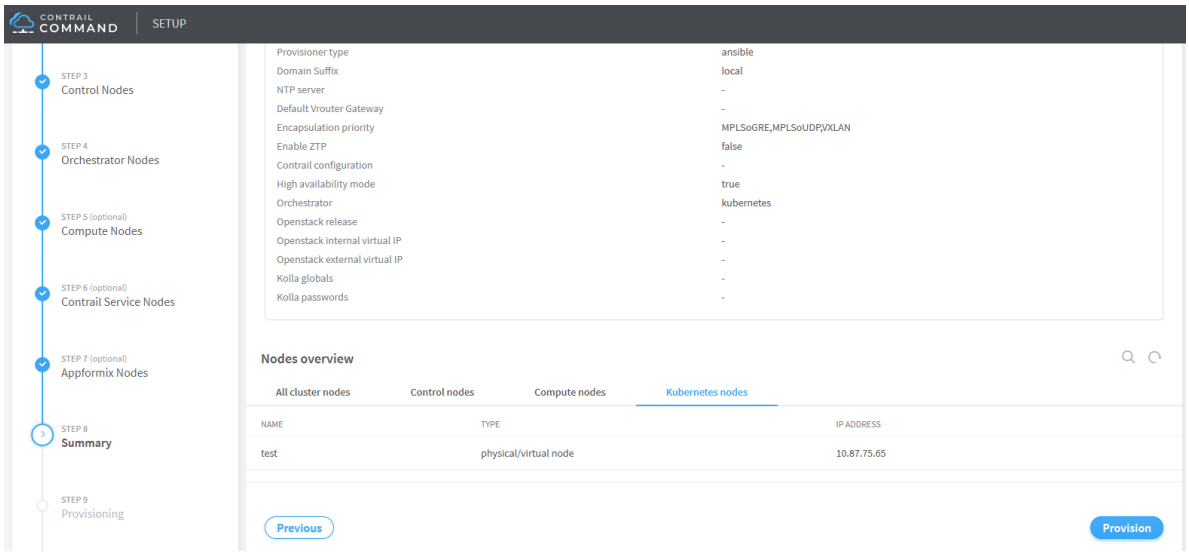


Figure 10: Summary - Nodes Overview



Sample command_servers.yml File

RELATED DOCUMENTATION

Installing Contrail Command

Installing a Contrail Cluster using Contrail Command and instances.yml

Importing Contrail Cluster Data using Contrail Command

Verifying Configuration for CNI for Kubernetes

IN THIS SECTION

NOTE: This topic covers Contrail Networking in Kubernetes-orchestrated environments that are using Contrail Networking Release 21-based releases.

Starting in Release 22.1, Contrail Networking evolved into Cloud-Native Contrail Networking. Cloud-Native Contrail Networking offers significant enhancements to optimize networking performance in Kubernetes-orchestrated environments. We recommend using Cloud-Native Contrail for networking in most Kubernetes-orchestrated environments.

For general information about Cloud-Native Contrail, see the Cloud-Native Contrail Networking Techlibrary homepage.

Use the verification steps in this topic to view and verify your configuration of Contrail Container Network Interface (CNI) for Kubernetes.

View Pod Name and IP Address

Use the following command to view the IP address allocated to a pod.

```
[root@device ~]# kubectl get pods --all-namespaces -o wide
NAMESPACE         NAME               READY     STATUS    RESTARTS   AGE
IP                         NODE
default               client-1            1/1           Running       0
19d       10.47.25.247   k8s-minion-1-3
default               client-2            1/1          Running       0
19d       10.47.25.246   k8s-minion-1-1
default               client-x             1/1           Running       0
19d       10.84.21.272   k8s-minion-1-1
```

Verify Reachability of Pods

Perform the following steps to verify if the pods are reachable to each other.

1. Determine the IP address and name of the pod.

```
[root@device ~]# kubectl get pods --all-namespaces -o wide
NAME                         READY     STATUS    RESTARTS   AGE       IP              NODE
example1-36xpr   1/1       Running   0          43s       10.47.25.251   b3s37
example2-pldp1   1/1       Running   0          39s       10.47.25.250   b3s37
```

2. Ping the destination pod from the source pod to verify if the pod is reachable.

```
root@device ~]# kubectl exec -it example1-36xpr  ping 10.47.25.250
PING 10.47.25.250 (10.47.25.250): 56 data bytes
64 bytes from 10.47.25.250: icmp_seq=0 ttl=63 time=1.510 ms
64 bytes from 10.47.25.250: icmp_seq=1 ttl=63 time=0.094 ms
```

Verify If Isolated Namespace-Pods Are Not Reachable

Perform the following steps to verify if pods in isolated namespaces cannot be reached by pods in non-isolated namespaces.

1. Determine the IP address and name of a pod in an isolated namespace.

```
[root@device ~]# kubectl get pod -n test-isolated-ns -o wide
NAME                         READY     STATUS    RESTARTS   AGE       IP              NODE
example3-bvqx5   1/1       Running   0          1h        10.47.25.249   b3s37
```

2. Determine the IP address of a pod in a non-solated namespace.

```
[root@device ~]# kubectl get pods
NAME                         READY     STATUS    RESTARTS   AGE
example1-36xpr   1/1       Running   0          15h
example2-pldp1   1/1       Running   0          15h
```

3. Ping the IP address of the pod in the isolated namespace from the pod in the non-isolated namespace.

```
[root@device ~]# kubectl exec -it example1-36xpr ping 10.47.25.249
        --- 10.47.255.249 ping statistics ---
 2 packets transmitted, 0 packets received, 100% packet loss
```

Verify If Non-Isolated Namespace-Pods Are Reachable

Perform the following steps to verify if pods in non-isolated namespaces can be reached by pods in isolated namespaces.

1. Determine the IP address of a pod in a non-isolated namespace.

```
[root@device ~]# kubectl get pods -o wide
NAME                        READY     STATUS    RESTARTS   AGE       IP              NODE
example1-36xpr   1/1       Running   0          15h       10.47.25.251   b3s37
example2-pldp1   1/1       Running   0          15h       10.47.25.250   b3s37
```

2. Determine the IP address and name of a pod in an isolated namespace.

```
[root@device ~]# kubectl get pod -n test-isolated-ns -o wide
NAME                        READY     STATUS    RESTARTS   AGE       IP              NODE
example3-bvqx5   1/1       Running   0          1h        10.47.25.249   b3s37
```

3. Ping the IP address of the pod in the non-isolated namespace from a pod in the isolated namespace.

```
[root@device ~]# kubectl exec -it example3-bvqx5 -n test-isolated-ns ping 10.47.25.251
PING 10.47.25.251 (10.47.25.251): 56 data bytes
64 bytes from 10.47.25.251: icmp_seq=0 ttl=63 time=1.467 ms
64 bytes from 10.47.25.251: icmp_seq=1 ttl=63 time=0.137 ms
^C--- 10.47.25.251 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max/stddev = 0.137/0.802/1.467/0.665 ms
```

Verify If a Namespace is Isolated

Namespace annotations are used to turn on isolation in a Kubernetes namespace. In isolated Kubernetes namespaces, the namespace metadata is annotated with the `opencontrail.org/isolation : true` annotation.

Use the following command to view annotations on a namespace.

```
[root@a7s16 ~]#
kubectl describe namespace test-isolated-ns
Name:       test-isolated-ns
Labels:     <none>
Annotations:    opencontrail.org/isolation : true     Namespace is isolated
Status:     Active
```

RELATED DOCUMENTATION

Contrail Integration with Kubernetes | 2

Implementation of Kubernetes Network Policy with Contrail Firewall Policy

IN THIS SECTION

NOTE: This topic covers Contrail Networking in Kubernetes-orchestrated environments that are using Contrail Networking Release 21-based releases.

Starting in Release 22.1, Contrail Networking evolved into Cloud-Native Contrail Networking. Cloud-Native Contrail Networking offers significant enhancements to optimize networking performance in Kubernetes-orchestrated environments. We recommend using Cloud-Native Contrail for networking in most Kubernetes-orchestrated environments.

For general information about Cloud-Native Contrail, see the Cloud-Native Contrail Networking Techlibrary homepage.

Contrail Networking—starting in Contrail Networking Release 5.0—supports implementing Kubernetes 1.9.2 network policy in Contrail using the Contrail firewall security policy framework. While Kubernetes network policy can be implemented using other security objects in Contrail like security groups and Contrail network policies, the support of tags by Contrail firewall security policy aids in the simplification and abstraction of Kubernetes workloads.

Contrail firewall security policy allows decoupling of routing from security policies and provides multi-dimension segmentation and policy portability while significantly enhancing user visibility and analytics functions. Contrail firewall security policy uses tags to achieve multi-dimension traffic segmentation among various entities, and with security features. Tags are key-value pairs associated with different entities in the deployment. Tags can be pre-defined or custom defined. Kubernetes network policy is a specification of how groups of Kubernetes workloads, which are hereafter referred to as pods, are allowed to communicate with each other and other network endpoints. Network policy resources use labels to select pods and define rules which specify what traffic is allowed to the selected pods.

Kubernetes Network Policy Characteristics

Kubernetes network policies have the following characteristics:

- A network policy is pod specific and applies to a pod or a group of pods. If a specified network policy applies to a pod, the traffic to the pod is dictated by rules of the network policy.
- If a network policy is not applied to a pod then the pod accepts traffic from all sources.
- A network policy can define traffic rules for a pod at the ingress, egress, or both directions. By default, a network policy is applied to the ingress direction, if no direction is explicitly specified.
- When a network policy is applied to a pod, the policy must have explicit rules to specify an allowlist of permitted traffic in the ingress and egress directions. All traffic that does not match the allowlist rules are denied and dropped.

- Multiple network policies can be applied on any pod. Traffic matching any one of the network policies must be permitted.
- A network policy acts on connections rather than individual packets. For example, if traffic from pod A to pod B is allowed by the configured policy, then the return packets for that connection from pod B to pod A are also allowed, even if the policy in place does not allow pod B to initiate a connection to pod A.
- **Ingress Policy**: An ingress rule consists of the identity of the source and the protocol:port type of traffic from the source that is allowed to be forwarded to a pod.

  The identity of the source can be of the following types:

  - Classless Interdomain Routing (CIDR) block—If the source IP address is from the CIDR block and the traffic matches the protocol:port, then traffic is forwarded to the pod.
  - Kubernetes namespaces—Namespace selectors identify namespaces, whose pods can send the defined protocol:port traffic to the ingress pod.
  - Pods—Pod selectors identify the pods in the namespace corresponding to the network policy, that can send matching protocol:port traffic to the ingress pods.
- **Egress Policy**: This specifies an allowlist CIDR to which a particular protocol:port type of traffic is permitted from the pods targeted by the network policy

  The identity of the destination can be of the following types:

  - CIDR block—If the destination IP address is from the CIDR block and the traffic matches the protocol:port, then traffic is forwarded to the destination.
  - Kubernetes namespaces—Namespace selectors identify namespaces, whose pods can send the defined protocol:port traffic to the egress pod.
  - Pods—Pod selectors identify the pods in the namespace corresponding to the network policy, that can receive matching protocol:port traffic from the egress pods.

Representing Kubernetes Network Policy as Contrail Firewall Security Policy

Kubernetes and Contrail firewall policy are different in terms of the semantics in which network policy is specified in each. The key to efficient implementation of a Kubernetes network policy through Contrail firewall policy is in mapping the corresponding configuration constructs between these two entities.

The constructs are mapped as displayed in Table 3 on page 231:

Table 3: Kubernetes Network Policy and Contrail Firewall Policy Mapping

Kubernetes Network Policy Constructs	Contrail Firewall Policy Constructs
Label	Custom Tag (one for each label)
Namespace	Custom Tag (one for each namespace)
Network Policy	Firewall Policy (one firewall policy per Network Policy)
Rule	Firewall Rule (one firewall rule per network policy rule)
CIDR Rules	Address Group
Cluster	Default Application Policy Set

NOTE: The project in which Contrail firewall policy constructs are created is the one that houses the Kubernetes cluster. For example, the Contrail firewall policy constructs are created in the global scope, if the Kubernetes cluster is a standalone cluster and the Contrail firewall policy constructs are created in the project scope, if the Kubernetes cluster is a nested cluster.

Resolving Kubernetes Network Policy Labels

The representation of pods in Contrail firewall policy is exactly the same as in the corresponding Kubernetes network policy. Contrail firewall policy deals with labels or tags in Contrail terminology. Contrail does not expand labels to IP addresses.

For example, in the `default` namespace, if network `policy-podSelector` specifies: `role=db`, then the corresponding firewall rule specifies the pods as `(role=db && namespace=default)`. No other translations to pod IP address or otherwise are done.

If the same network-policy also has `namespaceSelector` as `namespace=myproject`, then the corresponding firewall rule represents that namespace as `(namespace=myproject)`. No other translations or rules representing pods in “myproject“ namespace is done.

Similarly, each CIDR is represented by one rule. In essence, the Kubernetes network policy is translated 1:1 to Contrail firewall policy. There is only one additional firewall rule created for each Kubernetes network policy. The purpose of that rule is to implement the implicit deny requirements of the network policy and no other rule is created.

Contrail Firewall Policy Naming Convention

Contrail firewall security policies and rules are named as follows:

- A Contrail firewall security policy created for a Kubernetes network policy is named in the following format:

```
< Namespace-name >-< Network Policy Name >
```

  For example, a network policy "world" in namespace "Hello" is named:

```
Hello-world
```

- Contrail firewall rules created for a Kubernetes network policy are named in the following format:

```
< Namespace-name >-<PolicyType>-< Network Policy Name >-<Index of from/to blocks>-<selector type>-<rule-index>-<svc/port index>
```

  For example:

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: world
  namespace: hello
spec:
  podSelector:
    matchLabels:
      role: db
  policyTypes:
  - Ingress
  ingress:
  - from:
    - podSelector:
        matchLabels:
          role: frontend
```

A rule corresponding to this policy is named:

```
hello-ingress-world-0-podSelector-0-0
```

Implementation of Kubernetes Network Policy

The contrail-kube-manager daemon binds Kubernetes and Contrail together. This daemon connects to the API server of Kubernetes clusters and coverts Kubernetes events, including network policy events, into appropriate Contrail objects. With respect to a Kubernetes network policy, contrail-kube-manager performs the following actions:

- Creates a Contrail tag for each Kubernetes label
- Creates a firewall policy for each Kubernetes network policy
- Creates an Application Policy Set (APS) to represent the cluster. All firewall policies created in that cluster are attached to this application policy set.
- Modifications to existing Kubernetes network policies result in the corresponding firewall policies being updated.

Example Network Policy Configurations

The following examples illustrate various sample network policies and the corresponding firewall security policies created.

Example 1 - Conditional egress and ingress traffic

The following policy specifies a sample network policy with specific conditions for ingress and egress traffic to and from all pods in a namespace:

Sample Kubernetes network policy

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: test-network-policy
  namespace: default
```

```
spec:
  podSelector:
    matchLabels:
      role: db
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - from:
    - ipBlock:
        cidr: 17x.xx.0.0/16
        except:
        - 17x.xx.1.0/24
    - namespaceSelector:
        matchLabels:
          project: myproject
    - podSelector:
        matchLabels:
          role: frontend
    ports:
    - protocol: TCP
      port: 6379
  egress:
  - to:
    - ipBlock:
        cidr: 10.0.0.0/24
    ports:
    - protocol: TCP
      port: 5978
```

Sample Contrail firewall security policy

The test-network-policy defined in Kubernetes results in the following objects being created in Contrail.

Tags—The following tags are created, if they do not exist. In a regular workflow, these tags must have been created by the time the namespace and pods were created.

Key	Value
role	db

(Continued)

Key	Value
namespace	default

Address Groups

The following address groups are created:

Name	Prefix
17x.xx.1.0/24	17x.xx.1.0/24
17x.xx.0.0/16	17x.xx.0.0/16
10.0.0.0/24	10.0.0.0/24

Firewall Rules

The following firewall rules are created:

Rule Name	Action	Services	Endpoint1	Dir	Endpoint2	Match Tags
default-ingress-test-network-policy-0-ipBlock-0-17x.xx.1.0/24-0	deny	tcp:6379	Address Group: 17x.xx.1.0/24	>	role=db && namespace=default	
default-ingress-test-network-policy-0-ipBlock-0-cidr-17x.xx.0.0/16-0	pass	tcp:6379	Address Group: 17x.xx.0.0/16	>	role=db && namespace=default	
default-ingress-test-network-policy-0-namespaceSelector-1-0	pass	tcp:6379	project=myproject	>	role=db && namespace=default	

(Continued)

Rule Name	Action	Services	Endpoint1	Dir	Endpoint2	Match Tags
default-ingress-test-network-policy-0-podSelector-2-0	pass	tcp:6379	namespace=default && role=frontend	>	role=db && namespace=default	
default-egress-test-network-policy-ipBlock-0-cidr-10.0.0.0/24-0	pass	tcp:5978	role=db && namespace=default	>	Address Group: 10.0.0.0/24	

Firewall Policy

The following firewall security policy is created with the following rules.

Name	Rules
default-test-network-policy	• default-ingress-test-network-policy-0-ipBlock-0-17x.xx.1.0/24-0 • default-ingress-test-network-policy-0-ipBlock-0-cidr-17x.xx.0.0/16-0 • default-ingress-test-network-policy-0-namespaceSelector-1-0 • default-ingress-test-network-policy-0-podSelector-2-0 • default-egress-test-network-policy-ipBlock-0-cidr-10.0.0.0/24-0

Example 2 - Allow all Ingress Traffic

The following policy explicitly allows all traffic for all pods in a namespace:

Sample Kubernetes network policy

```
apiVersion: networking.k8s.io/v1
    kind: NetworkPolicy
    metadata:
      name: allow-all-ingress
    spec:
      podSelector:
```

```
    ingress:
    - {}
```

Sample Contrail firewall security policy

Tags—The following tags are created, if they do not exist. In a regular workflow, these tags are created before the namespace and pods are created.

Key	Value
namespace	default

Address Groups - None

Firewall Rules

The following firewall rule is created:

Rule Name	Action	Services	Endpoint1	Dir	Endpoint2	Match Tags
default-ingress-allow-all-ingress-0-allow-all-0	pass	any	any	>	namespace=default	

Firewall Policy

The following firewall policy are created:

Name	Rules
default-allow-all-ingress	default-ingress-allow-all-ingress-0-allow-all-0

Example 3 - Deny all ingress traffic

The following policy explicitly denies all ingress traffic to all pods in a namespace:

Sample Kubernetes network policy

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
```

```
  name: deny-ingress
spec:
  podSelector:
  policyTypes:
  - Ingress
```

Sample Contrail firewall security policy

Tags—The following tags are created, if they do not exist. In a regular workflow, these tags are created before the namespace and pods are created.

Key	Value
namespace	default

Address Groups - None

Firewall Rules - None

NOTE: The implicit behavior of any network policy is to deny traffic not matching explicit allow flows. However in this policy, there are no explicit allow rules. Hence, no firewall rules are created for this policy.

Firewall Policy

The following firewall policy is created:

Name	Rules
default-deny-ingress	

Example 4 - Allow all egress traffic

The following policy explicitly allows all egress traffic from all pods in a namespace:

Sample Kubernetes network policy

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
```

```
metadata:
  name: allow-all-egress
spec:
  podSelector:
  egress:
  - {}
```

Sample Contrail firewall security policy

Tags—The following tag is created, if they do not exist. In a regular workflow, these tags are created before the namespace and pods are created.

Key	Value
namespace	default

Address Groups - None

Firewall Rules

The following firewall rule is created:

Rule Name	Action	Services	Endpoint1	Dir	Endpoint2	Match Tags
default-egress-allow-all-egress-allow-all-0	pass	any	namespace=default	>	any	

Firewall Policy

The following firewall policy is created:

Name	Rules
default-allow-all-egress	default-egress-allow-all-egress-allow-all-0

Example 5 - Default deny all egress traffic

The following policy explicitly denies all egress traffic from all pods in a namespace:

Sample Kubernetes network policy

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: deny-all-egress
spec:
  podSelector: {}
  policyTypes:
  - Egress
```

Sample Contrail firewall security policy

Tags—The following tag is created, if they do not exist. In a regular workflow, these tags are created before the namespace and pods are created.

Key	Value
namespace	default

Address Groups - None

Firewall Rules - None

NOTE: The implicit behavior of any network policy with egress policy type is to deny egress traffic not matching explicit egress allow flows. In this policy, there are no explicit egress allow rules. Hence, no firewall rules are created for this policy.

Firewall Policy

The following firewall policy is created:

Name	Rules
default-deny-all-egress	

Example 6 - Default deny all ingress and egress traffic

The following policy explicitly denies all ingress and egress traffic to and from all pods in that namespace:

Sample Kubernetes network policy

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: deny-all-ingress-egress
spec:
  podSelector:
  policyTypes:
  - Ingress
  - Egress
```

Sample Contrail firewall security policy

Tags—The following tags is created, if they do not exist. In a regular workflow, these tags are created before the namespace and pods are created.

Key	Value
namespace	default

Address Groups - None

Firewall Rules - None

NOTE: The implicit behavior of any network policy with ingress/egress policy type is to deny corresponding traffic not matching explicit allow flows. In this policy, there are no explicit allow rules. Hence, no firewall rules are created for this policy.

Firewall Policy

The following firewall policy is created:

Name	Rules
default-deny-all-ingress-egress	

Cluster-wide Policy Action Enforcement

The specification and the syntax of network policies allow for maximum flexibility and varied combinations. However, you must exercise caution while configuring the network policies.

Consider a case where two network policies are created:

- Policy 1: Pod A can send to Pod B.
- Policy 2: Pod B can only receive from Pod C.

From a networking flow perspective, there is an inherent contradiction between the above policies. Policy 1 states that a flow from Pod A to Pod B is allowed. Policy 2 implies that flow from Pod A to Pod B is not allowed. From a networking perspective, Contrail prioritizes flow behavior as more critical. In the event of inherent contradiction in network policies, Contrail will honor the flow perspective. One of the core aspects of this notion is that if a policy matches a flow, the action is honored cluster-wide.

For instance, if a flow matches a policy at the source, the flow will match the same policy in the destination as well. Therefore, the flow behavior in a Contrail-managed Kubernetes cluster is as shown below:

- Flow from Pod A to Pod B is allowed (due to Policy 1)
- Flow from Pod C to Pod B is allowed (due to Policy 2)
- Any other flow to Pod B is disallowed (due to Policy 2)

Example Network Policy Action Enforcement Scenarios

Consider the following examples of network policy action enforcement:

- Allow all egress traffic and deny all ingress traffic

  Setup: Namespace NS1 has two pods, Pod A and Pod B.

  Policy: A network policy applied on namespace NS1 states:

  - Rule 1. Allow all egress traffic from all pods in NS1.

- Rule 2. Deny all ingress traffic to all pods in NS1.

Behavior:

- Pod A can send traffic to Pod B (due to rule 1)
- Pod B can send traffic to Pod A (due to rule 1)
- PodX from a different namespace cannot send traffic to Pod A or Pod B (due to rule 2)

- Allow all ingress traffic and deny all egress traffic

  Setup: Namespace NS1 has two pods, Pod A and Pod B.

  Policy: A network policy applied on namespace NS1 states:

  - Rule 1. Allow all ingress traffic to all pods in NS1
  - Rule 2. Deny all egress traffic from all pods in NS1.

  Behavior:

  - Pod A can send traffic to Pod B (due to rule 1)
  - Pod B can send traffic to Pod A (due to rule 1)
  - Pod A and Pod B cannot send traffic to pods in any other namespace.

- Egress CIDR rule

  Setup: Namespace NS1 has two pods, Pod A and Pod B.

  Policy: A network policy applied on namespace NS1 states:

  - Policy 1: Allow Pod A to send traffic to CIDR of Pod B.
  - Policy 2: Deny all ingress traffic to all pods in NS1.

  Behavior:

  - Pod A can send traffic to Pod B (due to Policy 1)
  - All other traffic to Pod A and Pod B is dropped (due to policy 2)

RELATED DOCUMENTATION

Kubernetes Updates | 253

How to Enable Keystone Authentication in a Juju Cluster within a Kubernetes Environment

IN THIS SECTION

NOTE: This topic covers Contrail Networking in Kubernetes-orchestrated environments that are using Contrail Networking Release 21-based releases.

Starting in Release 22.1, Contrail Networking evolved into Cloud-Native Contrail Networking. Cloud-Native Contrail Networking offers significant enhancements to optimize networking performance in Kubernetes-orchestrated environments. We recommend using Cloud-Native Contrail for networking in most Kubernetes-orchestrated environments.

For general information about Cloud-Native Contrail, see the Cloud-Native Contrail Networking Techlibrary homepage.

Starting in Contrail Networking Release 2011, Kubernetes can use the Keystone authentication service in Openstack for authentication in environments that contain cloud networks using Openstack and Kubernetes orchestrators when the Kubernetes environment is using Juju. This capability is available when the cloud networks are both using Contrail Networking and when the Kubernetes cluster was created in an environment using Juju.

This document discusses how to enable keystone authentication in Kubernetes environments and contains the following sections:

Overview: Keystone Authentication in Kubernetes Environments with a Juju Cluster

A cloud environment that includes Contrail clusters in Kubernetes-orchestrated environments and OpenStack-orchestrated environments can simplify authentication processes by having a single authentication service in place of each orchestrator authenticating separately. The ability for a

Kubernetes-orchestrated environment to authenticate using the Keystone service from Openstack can provide this capability when the Kubernetes environment is using Juju.

Kubernetes is able to authenticate users using Keystone when the contrail-controller charm in Juju has relations with both an Openstack orchestrator and the Kubernetes orchestrator. The contrail-controller charm—when the Keystone service in Kubernetes is enabled—passes the credentials from Keystone to the contrail-kubernetes-master charm. The contrail-kubernetes-master charm then passes the Keystone parameters to kubemanager.

Both orchestrators use their native authentication processes by default. The ability for Kubernetes to use Keystone authentication in an environment using Juju was introduced in Contrail Networking Release 2011 and must be user-enabled.

How to Enable Keystone Authentication in a Kubernetes Environment

To enable Keystone authentication for Kubernetes:

1. In Juju running in the Kubernetes cluster, add a relation between the kubernetes-master and Keystone and configure the Kubernetes master to use Keystone authorization:

   ```
   juju add-relation kubernetes-master keystone
   juju config kubernetes-master authorization-mode="Node,RBAC" enable-keystone-
   authorization=true
   ```

2. Ensure that IP Fabric Forwarding for the pod network in the default kube-system project is disabled and that SNAT is enabled. SNAT enablement is required to reach the Keystone service from the keystone-auth pod in Kubernetes.

   You can disable IP Fabric Forwarding and enable SNAT from the kubectl CLI or from the Tungsten Fabric GUI.

   - *Kubectl*:

     Navigate to **kubectl edit ns default** and add the following configuration:

     ```
     metadata:
       annotations:
         opencontrail.org/ip_fabric_snat: "true"
     ```

   - *Tungsten Fabric Graphical User Interface*

Change the appropriate settings in the **Configure > Networking > Networks > default-domain > k8s-kube-system** workflow.

3. In Juju, apply the policy.json configuration:

```
juju config kubernetes-master keystone-policy="$(cat policy.json)"
```

The JSON configuration varies by environment and the JSON configuration option descriptions are beyond the scope of this document.

A sample JSON configuration file is provided for reference:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: k8s-auth-policy
  namespace: kube-system
  labels:
    k8s-app: k8s-keystone-auth
data:
  policies: |
    [
      {
       "resource": {
          "verbs": ["get", "list", "watch"],
          "resources": ["*"],
          "version": "*",
          "namespace": "*"
        },
        "match": [
          {
            "type": "role",
            "values": ["*"]
          },
          {
            "type": "project",
            "values": ["k8s"]
          }
        ]
      },
      {
       "resource": {
          "verbs": ["*"],
```

```
          "resources": ["*"],
          "version": "*",
          "namespace": "myproject"
        },
        "match": [
          {
            "type": "role",
            "values": ["*"]
          },
          {
            "type": "project",
            "values": ["k8s-myproject"]
          }
        ]
      }
    ]
```

4. Install client tools on the jumphost or an another node outside of the cluster.

```
sudo snap install kubectl --classic
sudo snap install client-keystone-auth --edge
```

5. In Kubernetes, configure the Keystone context and set credentials:

```
kubectl config set-context keystone --user=keystone-user
kubectl config use-context keystone
kubectl config set-credentials keystone-user --exec-command=/snap/bin/client-keystone-auth
kubectl config set-credentials keystone-user --exec-api-version=client.authentication.k8s.io/
v1beta1
```

6. Apply the required settings to the environment:

```
export OS_IDENTITY_API_VERSION=3
export OS_USER_DOMAIN_NAME=admin_domain
export OS_USERNAME=admin
export OS_PROJECT_DOMAIN_NAME=admin_domain
export OS_PROJECT_NAME=admin
export OS_DOMAIN_NAME=admin_domain
export OS_PASSWORD=password
export OS_AUTH_URL=http://192.168.30.78:5000/v3
```

If preferred, you can also perform this step from stackrc.

7. From kubectl, use the configuration to create a namespace from keystone authentication.

```
root@noden18:[~]$ kubectl -v=5 --insecure-skip-tls-verify=true -s https://192.168.30.29:6443
get pods --all-namespaces
NAMESPACE     NAME                                READY   STATUS    RESTARTS   AGE
default       cirros                              1/1     Running   0          30h
kube-system   coredns-6b59b8bd9f-2nb4x            1/1     Running   3          33h
kube-system   k8s-keystone-auth-db47ff559-sh59p   1/1     Running   0          33h
kube-system   k8s-keystone-auth-db47ff559-vrfwd   1/1     Running   0          33h
```

Multiple Network Interfaces for Containers

NOTE: This topic covers Contrail Networking in Kubernetes-orchestrated environments that are using Contrail Networking Release 21-based releases.

Starting in Release 22.1, Contrail Networking evolved into Cloud-Native Contrail Networking. Cloud-Native Contrail Networking offers significant enhancements to optimize networking performance in Kubernetes-orchestrated environments. We recommend using Cloud-Native Contrail for networking in most Kubernetes-orchestrated environments.

For general information about Cloud-Native Contrail, see the Cloud-Native Contrail Networking Techlibrary homepage.

Starting in Release 4.0, Contrail provides networking support for containers using Kubernetes Orchestration. You can allocate a network interface to every container that you create using standard Container Networking Interface (CNI plugin). For more information on Contrail Containers Networking, see "Contrail Integration with Kubernetes" on page 2.

Starting in Contrail Release 5.1, you can allocate multiple network interfaces (multi-net) to a container to enable the container to connect to multiple networks. You can specify the networks the container can connect to. A network interface is either a physical interface or a virtual interface and is connected to the Linux network namespace. A network namespace is the network stack in the Linux kernel. More than one container can share the same network namespace.

The following limitations and caveats apply when you create multi-net interfaces:

- You cannot add or remove sidecar networks while the pod is still running.

- The administrator is responsible for removing corresponding Contrail pods before deleting the network attachment definition from the Kubernetes API server.
- Contrail creates a default `cluster-wide-network` in addition to custom networks.
- Contrail CNI plugin is not a delegating plugin. It does not support specifications for delegating plugins that are provided in the Kubernetes Network Custom Resource Definition De Facto Standard Version 1. For more information, view **[v1] Kubernetes Network Custom Resource Definition De-facto Standard.md** from the https://github.com/K8sNetworkPlumbingWG/multi-net-spec page.

Contrail multi-net support is based on the Kubernetes multi-net model. Kubernetes multi-net model has a specific design and construct, and can be extended to non-kubernetes models like Contrail multi-net. Contrail multi-net model does not require changes to the Kubernetes API and Kubernetes CNI driver. Contrail multi-net model, as in the case of Kubernetes multi-net model, does not change the existing cluster-wide network behavior.

Creating Multi-Net Interfaces

Follow these steps to create multi-net interfaces.

1. Create Network Object Model.

   You create the network object model if the cluster does not support the model.

   The object model of the container orchestration platform represents the network and attaches the network to a container. If the model does not support network objects by default, you can use extensions to represent the network.

   Creating Network Object Model by using Kubernetes NetworkAttachmentDefinition CRD object

```
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
  # name must match the spec fields below, and be in the form: <plural>.<group>
  name: network-attachment-definitions.k8s.cni.cncf.io
spec:
  # group name to use for REST API: /apis/<group>/<version>
  group: k8s.cni.cncf.io
  # version name to use for REST API: /apis/<group>/<version>
  version: v1
  # either Namespaced or Cluster
  scope: Namespaced
  names:
    # plural name to be used in the URL: /apis/<group>/<version>/<plural>
    plural: network-attachment-definitions
    # singular name to be used as an alias on the CLI and for display
```

```
    singular: network-attachment-definition
    # kind is normally the CamelCased singular type. Your resource manifests use this.
    kind: NetworkAttachmentDefinition
    # shortNames allow shorter string to match your resource on the CLI
    shortNames:
    - net-attach-def
  validation:
    openAPIV3Schema:
      properties:
        spec:
          properties:
            config:
              type: string
```

Kubernetes uses custom extensions to represent networks in its object model. CustomResourceDefinition(CRD) feature of Kubernetes helps support custom extensions.

NOTE: A CRD is created automatically when you install Contrail. Networks specified by CRD are sidecars that are not recognized by Kubernetes. The interaction of additional pod network attachments with Kubernetes API and its objects, such as services, endpoints, proxies, etc. are not specified. Kubernetes does not recognize the association of these objects to any pod.

2. Create networks.

   You create networks in the cluster:

   - Through the API server.

```
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  annotations:
    opencontrail.org/cidr: "<ip address>/24"
    opencontrail.org/ip_fabric_forwarding: "false"
    opencontrail.org/ip_fabric_snat: "false"
  name: right-network
  namespace: default
spec:
  config: '{ "cniVersion": "0.3.0", "type": "contrail-k8s-cni" }'
```

   Create a **right-network.yaml** file.

- By mapping to an existing network created from the Contrail Web user interface or from the Contrail Command user interface.

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: extns-network
  annotations:
    "opencontrail.org/network" : '{"domain":"default-domain", "project": "k8s-extns",
"name":"k8s-extns-pod-network"}'
spec:
  config: '{
    "cniVersion": "0.3.1",
    "type": "contrail-k8s-cni"
}'
```

Command to create the network:

```
kubectl apply -f right-network.yaml
```

3. Assign networks to pods.

You assign the networks that you created in Step 2 to pods. Each pod also has a default network assigned to it. Therefore, each pod will have the following networks assigned:

- default network (assigned by Kubernetes)

> **NOTE**: Contrail internally creates a default network called `cluster-wide-network`. This interface is the default interface for the pod

- network that you created in Step 2

Assigning networks to pods by using *k8s-semantics*:

Option 1

```
apiVersion: v1
kind: Pod
metadata:
  name: multiNetworkPod
  annotations:
```

```
    k8s.v1.cni.cncf.io/networks: '[
      { "name": "network-a" },
      { "name": "network-b" }
    ]'
spec:
  containers:
  - image: busybox
    command:
      - sleep
      - "3600"
    imagePullPolicy: IfNotPresent
    name: busybox
    stdin: true
    tty: true
  restartPolicy: Always
```

Option 2

```
apiVersion: v1
kind: Pod
metadata:
  name: ubuntu-pod-3
  annotations:
    k8s.v1.cni.cncf.io/networks: left-network,blue-network,right-network,extns/data-network
spec:
  containers:
  - name: ubuntuapp
    image: ubuntu-upstart
    securityContext:
      capabilities:
        add:
        - NET_ADMIN
```

RELATED DOCUMENTATION

Contrail Integration with Kubernetes | 2

Kubernetes Updates

IN THIS SECTION

NOTE: This topic covers Contrail Networking in Kubernetes-orchestrated environments that are using Contrail Networking Release 21-based releases.

Starting in Release 22.1, Contrail Networking evolved into Cloud-Native Contrail Networking. Cloud-Native Contrail Networking offers significant enhancements to optimize networking performance in Kubernetes-orchestrated environments. We recommend using Cloud-Native Contrail for networking in most Kubernetes-orchestrated environments.

For general information about Cloud-Native Contrail, see the Cloud-Native Contrail Networking Techlibrary homepage.

This topic describes updates to Kubernetes and supported features in Contrail.

TLS Bootstrapping of Kubernetes Nodes

Contrail supports TLS Bootstrapping of Kubernetes Nodes starting in Contrail Networking Release 5.1. TLS bootstrapping streamlines Kubernetes' ability to add and remove nodes from the Contrail cluster.

Priority Based Multitenancy

Contrail supports priority on the various resource quotas through the ResourceQuotaScopeSelector feature starting in Contrail Networking Release 5.1.

Improved Autoscaling

Contrail Networking supports improved pod autoscaling by creating and deleting pods based on the load starting in Contrail Networking Release 5.1.

Reachability to Kubernetes Pods Using the IP Fabric Forwarding Feature

A Kubernetes pod is a group of one or more containers (such as Docker containers), the shared storage for those containers, and options on how to run the containers. Since pods are in the overlay network, they cannot be reached directly from the underlay without a gateway or vRouter. Starting in Contrail Networking Release 5.0, the IP fabric forwarding (`ip-fabric-forwarding`) feature enables virtual networks to be created as part of the underlay network and eliminates the need for encapsulation and decapsulation of data. The ip-fabric-forwarding feature is only applicable for pod networks. If `ip-fabric-forwarding` is enabled, pod-networks are associated to ip-fabric-ipam instead of pod-ipam which is also a flat subnet.

The ip-fabric-forwarding feature is enabled and disabled in the global and namespace levels. By default, `ip-fabric-forwarding` is disabled in the global level. To enable it in global level, you must set "`ip_fabric_forwarding`" to "`true`" in the "`[KUBERNETES]`" section of the **/etc/contrail/contrail-kubernetes.conf** file. To enable or disable the feature in namespace level, you must set "`ip_fabric_forwarding`" to "`true`" or "`false`" respectively in namespace annotation. For example, "`opencontrail.org/ip_fabric_forwarding": "true`". Once the feature is enabled, it cannot be disabled.

For more information, see Gateway-less Forwarding.

Service Isolation Through Virtual Networks

In namespace isolation mode, services in one namespace are not accessible from other namespaces, unless security groups or network policies are explicitly defined to allow access. If any Kubernetes service is implemented by pods in an isolated namespace, those services are reachable only to pods in the same namespace through the Kubernetes `service-ip`.

The Kubernetes service-ip is allocated from the cluster network despite being in an isolated namespace. So, by default, service from one namespace can reach services from another namespace. However, security groups in isolated namespaces prevent reachability from external namespace and also prevent reachability from outside of the cluster. In order to enable access by external namespaces, the security group must be edited to allow access to all namespaces which defeats the purpose of isolation.

Contrail Networking—starting in Contrail Networking Release 5.0—enables service or ingress reachability from external clusters in isolated namespaces. Two virtual networks are created in isolated namespaces. One network is dedicated to pods and one is dedicated to services. Contrail network-policy is created between the pod network and the service network for reachability between pods and services. Service uses the same service-ipam which is a flat-subnet like pod-ipam. It is applicable for default namespace as well.

Contrail ip-fabric-snat Feature

With the Contrail ip-fabric-snat feature, pods that are in the overlay can reach the Internet without floating IPs or a logical-router. The ip-fabric-snat feature uses compute node IP for creating a source NAT to reach the required services and is applicable only to pod networks. The kube-manager reserves ports 56000 through 57023 for TCP and 57024 through 58047 for UDP to create a source NAT in global-config during the initialization.

The ip-fabric-snat feature can be enabled or disabled in the global or namespace levels. By default, the feature is disabled in the global level. To enable the ip-fabric-snat feature in the global level, you must set "ip-fabric-snat" to "true" in the "[KUBERNETES]" section in the **/etc/contrail/contrail-kubernetes.conf** file. To enable or disable it in the namespace level, you must set "ip_fabric_snat" to "true" or "false" respectively in namespace annotation. For example, "opencontrail.org/ip_fabric_snat": "true". The ip_fabric_snat feature can be at enabled and disabled any time. To enable or disable the ip_fabric_snat feature in the default-pod-network, default namespace must be used. If the ip_fabric_forwarding is enabled, ip_fabric_snat is ignored.

For more information, see Distributed SNAT.

Third-Party Ingress Controllers

Multiple ingress controllers can co-exist in Contrail. If "kubernetes.io/ingress.class" is absent or is "opencontrail" in the annotations of the Kubernetes ingress resource, the kube-manager creates a HAProxy loadbalancer. Otherwise it is ignored and the respective ingress controller handles the ingress resource. Since Contrail ensures the reachability between pods and services, any ingress controller can reach the endpoints or pods directly or through services.

Custom Network Support for Ingress Resources

Contrail supports custom networks in namespace level for pods. Starting with Contrail Release 5.0, custom networks are supported for ingress resources as well.

Kubernetes Probes and Kubernetes Service Node-Port

The Kubelet needs reachability to pods for liveness and readiness probes. Contrail network policy is created between the IP fabric network and pod network to provide reachability between node and pods. Whenever the pod network is created, the network policy is attached to the pod network to provide reachability between node and pods. So, any process in the node can reach the pods.

Kubernetes Service Node-Port is based on node reachability to pods. Since Contrail provides connectivity between node and pods through Contrail the network policy, Node Port is supported.

Kubernetes Network-Policy Support

Contrail Networking supports the following Kubernetes release 1.12 network policy features:

- Egress support for network policy—Each `NetworkPolicy` includes a `policyTypes` list which can include either `Ingress`, `Egress`, or both. The `policyTypes` field indicates whether or not the given policy applies to ingress traffic to selected pod, egress traffic from the selected pod, or both. Contrail Networking—starting in Contrail Networking Release 5.1—supports the `podSelector&namespaceSelector` egress specification. Contrail Networking—starting in Contrail Networking Release 5.0—supports `podSelector`, `namespaceSelector`, and `egress CIDR` egress specifications.
- Classless Interdomain Routing (CIDR) selector support for egress and ingress network policies
- Contrail-ansible-deployer provisioning—Contrail-ansible-deployer is updated to support Kubernetes 1.12.

Contrail Networking supports Kubernetes release 1.9.2 and enables implementing Kubernetes network policy in Contrail using the Contrail firewall security policy framework. While Kubernetes network policy can be implemented using other security objects in Contrail like security groups and Contrail network policies, the support of tags by Contrail firewall security policy aids in the simplification and abstraction of workloads.

For more information, see "Implementation of Kubernetes Network Policy with Contrail Firewall Policy" on page 228.

RELATED DOCUMENTATION

Implementation of Kubernetes Network Policy with Contrail Firewall Policy | 228

Contrail Integration with Kubernetes | 2