Red Hat OpenStack Platform 16.2

RHOSP director Operator for OpenShift Container Platform

Deploying a Red Hat OpenStack Platform overcloud in an OpenShift Container Platform cluster

Last Updated: 2021-09-24

Red Hat OpenStack Platform 16.2 RHOSP director Operator for OpenShift Container Platform

Deploying a Red Hat OpenStack Platform overcloud in an OpenShift Container Platform cluster

OpenStack Team
rhos-docs@redhat.com

Legal Notice

Copyright © 2021 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at
http://creativecommons.org/licenses/by-sa/3.0/
. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Abstract

Learn how to install the RHOSP director Operator in your OpenShift Container Platform cluster and use the Operator to deploy an RHOSP overcloud. This feature is available in this release as a Technology Preview, and therefore is not fully supported by Red Hat. It should only be used for testing, and should not be deployed in a production environment.

Table of Contents

MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see our CTO Chris Wright's message .

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

We appreciate your input on our documentation. Tell us how we can make it better.

Using the Direct Documentation Feedback (DDF) function

Use the **Add Feedback** DDF function for direct comments on specific sentences, paragraphs, or code blocks.

1. View the documentation in the *Multi-page HTML* format.
2. Ensure that you see the **Feedback** button in the upper right corner of the document.
3. Highlight the part of text that you want to comment on.
4. Click **Add Feedback**.
5. Complete the **Add Feedback** field with your comments.
6. Optional: Add your email address so that the documentation team can contact you for clarification on your issue.
7. Click **Submit**.

TECHNOLOGY PREVIEW

This feature is available in this release as a *Technology Preview*, and therefore is not fully supported by Red Hat. It should only be used for testing, and should not be deployed in a production environment. For more information about Technology Preview features, see Scope of Coverage Details.

CHAPTER 1. INTRODUCTION TO THE RED HAT OPENSTACK PLATFORM DIRECTOR OPERATOR

OpenShift Container Platform (OCP) uses a modular system of operators to extend the functions of your OCP cluster. The Red Hat OpenStack Platform (RHOSP) director Operator adds the ability to install and run a RHOSP cloud within OCP. This operator manages a set of Custom Resource Definitions (CRDs) for managing and deploying the infrastructure and configuration of RHOSP nodes. The basic architecture of an operator-deployed RHOSP cloud includes the following features:

Virtualized control plane

The director Operator creates a set of virtual machines in OpenShift Virtualization to act as Controller nodes.

Bare metal machine provisioning

The director Operator uses OCP bare metal machine management to provision Compute nodes in an operator-deployed RHOSP cloud.

Networking

The director Operator configures the underlying networks for RHOSP services.

Heat and Ansible-based configuration

The director Operator stores custom Heat configuration in OCP and uses the **config-download** functionality in director to convert the configuration into Ansible playbooks. If you change the stored heat configuration, the director Operator automatically regenerates the Ansible playbooks.

CLI client

The director Operator creates a pod for users to run RHOSP CLI commands and interact with their RHOSP cloud.

Additional resources

- Operators on Red Hat OpenShift

1.1. PREREQUISITES FOR THE DIRECTOR OPERATOR

Before you install the Red Hat OpenStack Platform (RHOSP) director Operator, you must complete the following prerequisite tasks.

- Install an Openshift Container Platform (OCP) 4.6 or later cluster that contains a **baremetal** cluster operator that has been enabled and a **provisioning** network.

NOTE

OCP clusters that you install with the installer-provisioned infrastructure (IPI) or assisted installation (AI) use the **baremetal** platform type and have the **baremetal** cluster Operator enabled. OCP clusters that you install with user-provisioned infrastructure (UPI) use the **none** platform type and might have the **baremetal** cluster Operator disabled.

To check if the **baremetal** cluster Operator is enabled, navigate to **Administration > Cluster Settings > ClusterOperators > baremetal**, scroll to the **Conditions** section, and view the **Disabled** status.

To check the platform type of the OCP cluster, navigate to **Administration > Global Configuration > Infrastructure**, switch to **YAML** view, scroll to the **Conditions** section, and view the **status.platformStatus** value.

- Install the following prerequisite Operators from OperatorHub on your OCP cluster:
  - OpenShift Virtualization 2.6 or later
  - SR-IOV Network Operator
- Configure a remote Git repository for the director Operator to store the generated configuration for your overcloud.
- Create the following persistent volumes to fulfil the following persistent volume claims that the director Operator creates:
  - 4G for **openstackclient-cloud-admin**
  - 1G for **openstackclient-hosts**
  - 50G for the base image that the director Operator clones for each Controller virtual machine
  - A minimum of 50G for each Controller virtual machine

Additional resources

- "Adding Operators to a cluster"

1.2. INSTALLING THE DIRECTOR OPERATOR

To install the director Operator, you must create a namespace for the Operator and create the following three resources within the namespace:

- A CatalogSource, which identifies the index image to use for the director Operator catalog.
- A Subscription, which tracks changes in the director Operator catalog.
- An OperatorGroup, which defines the Operator group for the director Operator and restricts the director Operator to a target namespace.

Prerequisites

- Ensure your OpenShift Container Platform cluster is operational.

- Install the following prerequisite Operators from OperatorHub:
  - OpenShift Virtualization 2.6 or later
  - SR-IOV Network Operator
- Ensure that you have installed the **oc** command line tool on your workstation.

Procedure

1. Create the **openstack** namespace:

```
$ oc new-project openstack
```

2. Create a file named **osp-director-operator.yaml** and include the following YAML content that configures the three resources to install the director Operator:

```
apiVersion: operators.coreos.com/v1alpha1
kind: CatalogSource
metadata:
  name: osp-director-operator-index
  namespace: openstack
spec:
  sourceType: grpc
  image: quay.io/openstack-k8s-operators/osp-director-operator-index:1.0.0-1
---
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: "osp-director-operator-group"
  namespace: openstack
spec:
  targetNamespaces:
  - openstack
---
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: osp-director-operator-subscription
  namespace: openstack
spec:
  config:
    env:
    - name: WATCH_NAMESPACE
      value: openstack,openshift-machine-api,openshift-sriov-network-operator
  source: osp-director-operator-index
  sourceNamespace: openstack
  name: osp-director-operator
  startingCSV: osp-director-operator.v0.0.1
  channel: alpha
```

3. Create the three new resources within the **openstack** namespace:

```
$ oc create -f osp-director-operator.yaml
```

Verification

1. Confirm that you have successfully installed the director Operator:

```
$ oc get operators
NAME                                 AGE
osp-director-operator.openstack      5m
```

Additional resources

- "Installing from OperatorHub using the CLI"

1.3. CUSTOM RESOURCE DEFINITIONS FOR THE DIRECTOR OPERATOR

The director Operator includes a set of custom resource definitions (CRDs) that you can use to manage overcloud resources. There are two types of CRDs: hardware provisioning and software configuration.

Hardware Provisioning CRDs

openstackbaremetalsets

Creates sets of bare metal hosts for a specific overcloud role, such as Compute and Ceph Storage.

openstackcontrolplanes

Creates the overcloud control plane and manages associated **openstackvmsets**.

openstackipsets

Defines a set of IP addresses for a network and role. OpenShift Container Platform (OCP) uses this CRD to manage IP addresses.

openstacknets

Creates networks to assign IP addresses to the **openstackvmset** and **openstackbaremetalset** resources.

openstackprovisionservers

Serves custom images for bare metal provisioning.

openstackvmsets

Creates sets of virtual machines for a specific overcloud role, such as Controller, Database, and Networker.

Software Configuration CRDs

openstackplaybookgenerators

Creates Ansible playbooks for deployment and regenerates the playbooks when you scale the overcloud or make changes to custom ConfigMaps.

openstackclients

Creates a pod for you to run deployment and management commands.

Viewing the director Operator CRDs

- View a list of these CRDs with the **oc get crd** command:

```
$ oc get crd | grep "^openstack"
```

- View the definition for a specific CRD with the **oc describe crd** command:

```
$ oc describe crd openstackbaremetalset
Name:         openstackbaremetalsets.osp-director.openstack.org
Namespace:
Labels:       operators.coreos.com/osp-director-operator.openstack=
Annotations:  cert-manager.io/inject-ca-from:
$(CERTIFICATE_NAMESPACE)/$(CERTIFICATE_NAME)
              controller-gen.kubebuilder.io/version: v0.3.0
API Version:  apiextensions.k8s.io/v1
Kind:         CustomResourceDefinition
...
```

CRD naming conventions

Each CRD contains multiple names in the **spec.names** section. Use these names depending on the context of your actions:

- Use **kind** when you create and interact with resource manifests:

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackBaremetalSet
....
```

  The **kind** name in the resource manifest correlates to the **kind** name in the respective CRD.

- Use **plural** when you interact with multiple resources:

```
$ oc get openstackbaremetalsets
```

- Use **singular** when you interact with a single resource:

```
$ oc describe openstackbaremetalset/compute
```

- Use **shortName** for any CLI interactions:

```
$ oc get osbmset
```

Additional resources

- "Managing resources from custom resource definitions"

1.4. WORKFLOW FOR OVERCLOUD DEPLOYMENT WITH THE DIRECTOR OPERATOR

After you have installed the Red Hat OpenStack Platform director Operator, you can use the resources specific to the director Operator to provision your overcloud infrastructure, generate your overcloud configuration, and create an overcloud.

The following workflow outlines the general process for creating an overcloud:

1. Create the overcloud networks, including the control plane and any isolated networks.

2. Create ConfigMaps to store any custom heat templates and environment files for your overcloud.

3. Create a control plane, which includes three virtual machines for Controller nodes and a pod to perform client operations.

4. Create bare metal Compute nodes.

5. Create a generator to render Ansible playbooks for overcloud configuration.

6. Apply the Ansible playbook configuration to your overcloud nodes.

PART I. GENERAL DEPLOYMENT OPERATIONS

CHAPTER 2. PREPARING FOR OVERCLOUD DEPLOYMENT WITH THE DIRECTOR OPERATOR

Before you can deploy an overcloud with the director Operator, you must create a data volume for the base operating system and add authentication details for your remote git repository. You can also set the root password for your nodes. If you do not set a root password, you can still log into nodes with the SSH keys defined in the **osp-controlplane-ssh-keys** Secret.

2.1. CREATING A DATA VOLUME FOR THE BASE OPERATING SYSTEM

You must create a data volume with the OpenShift Container Platform (OCP) cluster to store the base operating system image for your Controller virtual machines.

Prerequisites

- Download a Red Hat Enterprise Linux 8 QCOW2 image to your workstation.You can download this image from the Product Download section of the Red Hat Customer Portal.
- Install the **virtctl** client tool on your workstation. You can install this tool on a Red Hat Enterprise Linux workstation using the following commands:

```
$ sudo subscription-manager repos --enable=cnv-2.6-for-rhel-8-x86_64-rpms
$ sudo dnf install -y kubevirt-virtctl
```

- Install the **virt-customize** client tool on your workstation. You can install this tool on a Red Hat Enterprise Linux workstation using the following command:

```
$ dnf install -y libguestfs-tools-c
```

Procedure

1. The default QCOW2 image that you have downloaded from access.redhat.com does not use biosdev predictable network interface names. Modify the image with **virt-customize** to use biosdev predictable network interface names:

```
$ sudo virt-customize -a <local path to image> --run-command 'sed -i -e "s/^\
(kernelopts=.*\)net.ifnames=0 \(.*\)/\1\2/" /boot/grub2/grubenv'
$ sudo virt-customize -a <local path to image> --run-command 'sed -i -e "s/^\
(GRUB_CMDLINE_LINUX=.*\)net.ifnames=0 \(.*\)/\1\2/" /etc/default/grub'
```

2. Upload the image to OpenShift Virtualization with **virtctl**:

```
$ virtctl image-upload dv openstack-base-img -n openstack --size=50Gi --image-path=<local
path to image> --storage-class <storage class> --insecure
```

   For the **--storage-class** option, choose a storage class from your cluster. View a list of storage classes with the following command:

```
$ oc get storageclass
```

3. When you create the OpenStackControlPlane resource and individual OpenStackVmSet resources, set the **baseImageVolumeName** parameter to the data volume name:

```
...
spec:
  ...
  baseImageVolumeName: openstack-base-img
...
```

Additional resources

- "Uploading local disk images by using the virtctl tool"

2.2. ADDING AUTHENTICATION DETAILS FOR YOUR REMOTE GIT REPOSITORY

The director Operator stores rendered Ansible playbooks to a remote Git repository and uses this repository to track changes to the overcloud configuration. You can use any Git repository that supports SSH authentication. You must provide details for the Git repository as an OpenShift Secret resource named **git-secret**.

Prerequisites

- Ensure your OpenShift Container Platform cluster is operational and you have installed the director Operator correctly.
- Ensure that you have installed the **oc** command line tool on your workstation.
- Prepare a remote Git repository for the director Operator to store the generated configuration for your overcloud.
- Prepare an SSH key pair. Upload the public key to the Git repository and keep the private key available to add to the **git-secret** Secret resource.

Procedure

- Create the Secret resource:

```
$ oc create secret generic git-secret -n openstack --from-file=git_ssh_identity=
<path_to_private_SSH_key> --from-literal=git_url=<git_server_URL>
```

The **git-secret** Secret resource contains two key-value pairs:

git_ssh_identity

The private key to access the Git repository. The **--from-file** option stores the content of the private SSH key file.

git_url

The SSH URL of the git repository to store the configuration. The **--from-literal** option stores the URL that you enter for this key.

Verification

- View the Secret resource:

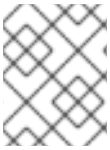
```
$ oc get secret/git-secret -n openstack
```

Additional resources

- "Providing sensitive data to pods"

2.3. SETTING THE ROOT PASSWORD FOR NODES

To access the **root** user with a password on each node, you can set a **root** password in a Secret resource named **userpassword**.



NOTE

Setting the root password for nodes is optional. If you do not set a **root** password, you can still log into nodes with the SSH keys defined in the **osp-controlplane-ssh-keys** Secret.

Prerequisites

- Ensure your OpenShift Container Platform cluster is operational and you have installed the director Operator correctly.
- Ensure that you have installed the **oc** command line tool on your workstation.

Procedure

1. Convert your chosen password to a base64 value:

```
$ echo -n "p@ssw0rd!" | base64
cEBzc3cwcmQh
```

NOTE

The **-n** option removes the trailing newline from the echo output.

2. Create a file named **openstack-userpassword.yaml** on your workstation. Include the following resource specification for the Secret in the file:

```
apiVersion: v1
kind: Secret
metadata:
  name: userpassword
  namespace: openstack
data:
  NodeRootPassword: "cEBzc3cwcmQh"
```

   Set the **NodeRootPassword** parameter to your base64 encoded password.

3. Create the **userpassword** Secret:

```
$ oc create -f openstack-userpassword.yaml -n openstack
```

Additional resources

- "Providing sensitive data to pods"

CHAPTER 3. CREATING NETWORKS WITH THE DIRECTOR OPERATOR

Use the OpenStackNet resource to create networks and bridges on OpenShift Virtualization worker nodes to connect your virtual machines to these networks. You must create one control plane network for your overcloud and additional networks to implement network isolation for your composable networks.

3.1. UNDERSTANDING VIRTUAL MACHINE BRIDGING WITH OPENSTACKNET

When you create virtual machines with the OpenStackVMSet resource, you must connect these virtual machines to the relevant Red Hat OpenStack Platform (RHOSP) networks. The OpenStackNet resource includes a **nodeNetworkConfigurationPolicy** option, which passes network interface data to the NodeNetworkConfigurationPolicy resource in OpenShift Virtualization. The NodeNetworkConfigurationPolicy resource uses the **nmstate** API to configure the end state of the network configuration on each OCP worker node. Through this method, you can create a bridge on OCP nodes to connect your Controller virtual machines to RHOSP networks.

For example, if you create a control plane network and set the **nodeNetworkConfigurationPolicy** option to create a Linux bridge and connect the bridge to a NIC on each worker, the NodeNetworkConfigurationPolicy resource configures each OCP worker node to match this desired end state:

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackNet
metadata:
  name: ctlplane
spec:
  ...
  attachConfiguration:
    nodeSelector:
      node-role.kubernetes.io/worker: ""
    nodeNetworkConfigurationPolicy:
      ...
      desiredState:
        interfaces:
        - bridge:
            options:
              stp:
                enabled: false
            port:
            - name: enp6s0
          description: Linux bridge with enp6s0 as a port
          name: br-osp
          state: up
          type: linux-bridge
```

After you apply this configuration, each worker contains a new bridge named **br-osp**, which is connected to the **enp6s0** NIC on each host. All RHOSP Controller virtual machines can connect to the **br-osp** bridge for control plane network traffic.

If you later create an Internal API network through VLAN 20, you can set the **nodeNetworkConfigurationPolicy** option to modify the networking configuration on each OCP worker node and connect the VLAN to the existing **br-osp** bridge:

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackNet
metadata:
  name: internalapi
spec:
  ...
  vlan: 20
  attachConfiguration:
    nodeNetworkConfigurationPolicy:
      nodeSelector:
        node-role.kubernetes.io/worker: ""
      desiredState:
        interfaces:
        - bridge:
            options:
              stp:
                enabled: false
            port:
            - name: enp6s0
          description: Linux bridge with enp6s0 as a port
          name: br-osp
          state: up
          type: linux-bridge
```

The **br-osp** already exists and is connected to the **enp6s0** NIC on each host, so no change occurs to the bridge itself. However, OpenStackNet associates VLAN 20 to this network, which means RHOSP Controller virtual machines can connect to the VLAN 20 on the **br-osp** bridge for Internal API network traffic.

When you create virtual machines with the OpenStackVMSet resource, the virtual machines use multiple Virtio devices connected to each network. OpenShift Virtualization sorts the network names in alphabetical order except for the **default** network, which is always the first interface.

For example, if you create the default RHOSP networks with OpenStackNet, the interface configuration for Controller virtual machines resembles the following example:

```
interfaces:
  - masquerade: {}
    model: virtio
    name: default
  - bridge: {}
    model: virtio
    name: ctlplane
  - bridge: {}
    model: virtio
    name: external
  - bridge: {}
    model: virtio
    name: internalapi
  - bridge: {}
    model: virtio
    name: storage
```

```
  - bridge: {}
    model: virtio
    name: storagemgmt
  - bridge: {}
    model: virtio
    name: tenant
```

This configuration results in the following network-to-interface mapping for Controller nodes:

Table 3.1. Default network-to-interface mapping

Network	Interface
default	nic1
ctlplane	nic2
external	nic3
internalapi	nic4
storage	nic5
storagemgmt	nic6
tenant	nic7

When you create heat templates for your Controller NICs, ensure the templates contain the respective NIC configuration for each network:

```
...
resources:
  OsNetConfigImpl:
    type: OS::Heat::SoftwareConfig
    properties:
      group: script
      config:
        str_replace:
          template:
            get_file: /usr/share/openstack-tripleo-heat-templates/network/scripts/run-os-net-config.sh
          params:
            $network_config:
              network_config:

              ## NIC2 - Control Plane ##
              - type: interface
                name: nic2
                ...

              ## NIC3 - External ##
              - type: ovs_bridge
                name: bridge_name
```

```
    ...
    members:
    - type: interface
      name: nic3

  ## NIC4 - Internal API ##
  - type: interface
    name: nic4
    ...

  ## NIC5 - Storage ##
  - type: interface
    name: nic5
    ...

  ## NIC6 - StorageMgmt ##
  - type: interface
    name: nic6
    ...

  ## NIC7 - Tenant ##
  - type: ovs_bridge
    name: br-isolated
    ...
    members:
    - type: interface
      name: nic7
      ...
```

Additional resources

- "Updating node network configuration"

3.2. CREATING AN OVERCLOUD CONTROL PLANE NETWORK WITH OPENSTACKNET

You must create one control plane network for your overcloud. In addition to IP address assignment, the OpenStackNet resource includes information to define the network configuration policy that OpenShift Virtualization uses to attach any virtual machines to the network.

Prerequisites

- Ensure your OpenShift Container Platform cluster is operational and you have installed the director Operator correctly.
- Ensure that you have installed the **oc** command line tool on your workstation.

Procedure

1. Create a file named **ctlplane-network.yaml** on your workstation. Include the resource specification for the control plane network, which is named **ctlplane**. For example, the specification for a control plane that uses a Linux bridge connected to the **enp6s0** Ethernet device on each worker node is as follows:

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackNet
metadata:
  name: ctlplane
spec:
  cidr: 192.168.25.0/24
  allocationStart: 192.168.25.100
  allocationEnd: 192.168.25.250
  gateway: 192.168.25.1
  attachConfiguration:
    nodeNetworkConfigurationPolicy:
      nodeSelector:
        node-role.kubernetes.io/worker: ""
      desiredState:
        interfaces:
        - bridge:
            options:
              stp:
                enabled: false
            port:
            - name: enp6s0
          description: Linux bridge with enp6s0 as a port
          name: br-osp
          state: up
          type: linux-bridge
```

Set the following values in the resource specification:

metadata.name

Set to the name of the control plane network, which is **ctlplane**.

spec

Set the network configuration for the control plane network. For descriptions of the values you can use in this section, view the specification schema in the custom resource definition for the **openstacknet** CRD:

```
$ oc describe crd openstacknet
```

Save the file when you have finished configuring the network specification.

2. Create the control plane network:

```
$ oc create -f ctlplane-network.yaml -n openstack
```

Verification

- View the resource for the control plane network:

```
$ oc get openstacknet/ctlplane
```

3.3. CREATING VLAN NETWORKS FOR NETWORK ISOLATION WITH OPENSTACKNET

You must create additional networks to implement network isolation for your composable networks. To accomplish this network isolation, you can place your composable networks networks on individual VLAN networks. In addition to IP address assignment, the OpenStackNet resource includes information to define the network configuration policy that OpenShift Virtualization uses to attach any virtual machines to VLAN networks.

To use the default Red Hat OpenStack Platform networks, you must create an OpenStackNet resource for each network.

Table 3.2. Default Red Hat OpenStack Platform networks

Network	VLAN	CIDR	Allocation
External	10	10.0.0.0/24	10.0.0.4 - 10.0.0.250
InternalApi	20	172.16.2.0/24	172.16.2.4 - 172.16.2.250
Storage	30	172.16.1.0/24	172.16.1.4 - 172.16.1.250
StorageMgmt	40	172.16.3.0/24	172.16.3.4 - 172.16.3.250
Tenant	50	172.16.0.0/24	172.16.0.4 - 172.16.0.250

IMPORTANT

To use different networking details for each network, you must create a custom **network_data.yaml** file.

Prerequisites

- Ensure your OpenShift Container Platform cluster is operational and you have installed the director Operator correctly.
- Ensure that you have installed the **oc** command line tool on your workstation.

Procedure

1. Create a file for your new network. For example, for the internal API network, create a file named **internalapi-network.yaml** on your workstation. Include the resource specification for the VLAN network. For example, the specification for an internal API network that manages VLAN-tagged traffic over a Linux bridge connected to the **enp6s0** Ethernet device on each worker node is as follows:

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackNet
metadata:
  name: internalapi
spec:
  cidr: 172.16.2.0/24
  vlan: 20
  allocationStart: 172.16.2.4
  allocationEnd: 172.16.2.250
  attachConfiguration:
```

```
    nodeNetworkConfigurationPolicy:
      nodeSelector:
        node-role.kubernetes.io/worker: ""
      desiredState:
        interfaces:
        - bridge:
            options:
              stp:
                enabled: false
            port:
            - name: enp6s0
          description: Linux bridge with enp7s0 as a port
          name: br-osp
          state: up
          type: linux-bridge
```

When you use a VLAN for network isolation with **linux-bridge** the following happens:

- The director Operator creates a Node Network Configuration Policy for the bridge interface specified in the resource, which uses **nmstate** to configure the bridge on worker nodes.
- The director Operator creates a Network Attach Definition for each network, which defines the Multus CNI plugin configuration. When you specify the VLAN ID on the Network Attach Definition, the Multus CNI plugin enables **vlan-filtering** on the bridge.
- The director Operator attaches a dedicated interface for each network on a virtual machine. This means the network template for the **OSVMSet** is a multi-NIC network template.

Set the following values in the resource specification:

metadata.name

Set to the name of the control plane network, which is **ctlplane**.

spec

Set the network configuration for the control plane network. For descriptions of the values you can use in this section, view the specification schema in the custom resource definition for the **openstacknet** CRD:

```
$ oc describe crd openstacknet
```

Save the file when you have finished configuring the network specification.

2. Create the internal API network:

```
$ oc create -f internalapi-network.yaml -n openstack
```

Verification

- View the resource for the internal API network:

```
$ oc get openstacknet/internalapi -n openstack
```

CHAPTER 4. ADDING HEAT TEMPLATES AND ENVIRONMENT FILES WITH THE DIRECTOR OPERATOR

If you want to customize your overcloud or enable certain features, you must create heat templates and environment files for the customizations you want. You can include your custom templates and environment files with your deployment to configure your overcloud. In the context of the director Operator, you store these files in ConfigMap resources before running an overcloud deployment.

4.1. UNDERSTANDING CUSTOM TEMPLATE USAGE WITH THE DIRECTOR OPERATOR

The director Operator converts a core set of templates into Ansible playbooks that you apply to provisioned nodes when you are ready to configure the Red Hat OpenStack Platform software on each node.

To add your own custom templates to the overcloud deployment, you must archive the template files into a tarball file and include the binary contents of the tarball file in an OpenShift ConfigMap named **tripleo-tarball-config**. This tarball file can contain complex directory structures to extend the core set of templates. The director Operator extracts the files and directories from the tarball file into the same directory as the core set of heat templates. If any of your custom templates have the same name as a template in the core collection, the custom template overrides the core template.

For example, your tarball file might contain the following YAML files:

```
$ tar -tf custom-config.tar.gz
custom-template.yaml
net-config-static-bridge.yaml
net-config-static.yaml
network_data.yaml
```

The **custom-template.yaml** file is a new custom template that does not override any existing templates. However, the **net-config-static-bridge.yaml** and **net-config-static.yaml** files override the default heat templates for preprovisioned node network configuration and the **network_data.yaml** file overrides the default composable network configuration.

Additional resources

- "heat templates"

4.2. ADDING CUSTOM TEMPLATES TO THE OVERCLOUD CONFIGURATION

Archive your custom templates into a tarball file so that you can include these templates as a part of your overcloud deployment.

Prerequisites

- Ensure your OpenShift Container Platform cluster is operational and you have installed the director Operator correctly.
- Ensure that you have installed the **oc** command line tool on your workstation.
- Create the custom templates that you want to apply to provisioned nodes.

Procedure

1. Navigate to the location of your custom templates:

```
$ cd ~/custom_templates
```

2. Archive the templates into a tarball:

```
$ tar -cvzf custom-config.tar.gz *.yaml
```

3. Create the **tripleo-tarball-config** ConfigMap and use the tarball as data:

```
$ oc create configmap tripleo-tarball-config --from-file=custom-config.tar.gz -n openstack
```

Verification

- View the ConfigMap:

```
$ oc get configmap/tripleo-tarball-config -n openstack
```

Additional resources

- "Creating and using config maps"
- "heat templates"

4.3. UNDERSTANDING CUSTOM ENVIRONMENT FILE USAGE WITH THE DIRECTOR OPERATOR

To enable features or set parameters in the overcloud, you must include environment files with your deployment runs. The director Operator uses a ConfigMap named **heat-env-config** to store and retrieve environment files. Use the following syntax for the data in the **heat-env-config** ConfigMap:

```
...
data:
  <environment_file_name>: |+
    <environment_file_contents>
```

For example, your **heat-env-config** ConfigMap might contain two environment files:

```
...
data:
  network_environment.yaml: |+
    resource_registry:
      OS::TripleO::Controller::Net::SoftwareConfig: net-config-static-bridge.yaml
      OS::TripleO::Compute::Net::SoftwareConfig: net-config-static-bridge-compute.yaml
  cloud_name.yaml: |+
    parameter_defaults:
      CloudDomain: ocp4.example.com
      CloudName: overcloud.ocp4.example.com
      CloudNameInternal: overcloud.internalapi.ocp4.example.com
```

```
CloudNameStorage: overcloud.storage.ocp4.example.com
CloudNameStorageManagement: overcloud.storagemgmt.ocp4.example.com
CloudNameCtlplane: overcloud.ctlplane.ocp4.example.com
```

- The first environment file is named **network_environment.yaml** and contains a **resource_registry** section to map network interface configuration to the appropriate heat templates.
- The second environment file is named **cloud_name.yaml** and contains a **parameter_defaults** section to set parameters relating to overcloud host names.
- When the director Operator deploys the overcloud, the Operator includes both files from the **heat-env-config** ConfigMap with the deployment.

Additional resources

- "Environment files"

4.4. ADDING CUSTOM ENVIRONMENT FILES TO THE OVERCLOUD CONFIGURATION

Upload a set of custom environment files from a directory to a ConfigMap that you can include as a part of your overcloud deployment.

Prerequisites

- Ensure your OpenShift Container Platform cluster is operational and you have installed the director Operator correctly.
- Ensure that you have installed the **oc** command line tool on your workstation.
- Create custom environment files for your overcloud deployment.

Procedure

1. Create the **heat-env-config** ConfigMap and use the directory that contains the environment files as data:

```
$ oc create configmap -n openstack heat-env-config --from-file=~/custom_environment_files/
--dry-run=client -o yaml | oc apply -f -
```

Verification

- View the ConfigMap:

```
$ oc get configmap/heat-env-config -n openstack
```

Additional resources

- "Creating and using config maps"
- "Environment files"

CHAPTER 5. CREATING OVERCLOUD NODES WITH THE DIRECTOR OPERATOR

A Red Hat OpenStack Platform overcloud consists of multiple nodes, such as Controller nodes to provide control plane services and Compute nodes to provide computing resources. For a functional overcloud with high availability, you must have 3 Controller nodes and at least one Compute node. You can create Controller nodes with the OpenStackControlPlane resource and Compute nodes with OpenStackBaremetalSet resource.

5.1. CREATING A CONTROL PLANE WITH OPENSTACKCONTROLPLANE

The overcloud control plane contains the main Red Hat OpenStack Platform services that manage overcloud functionality. The control plane usually consists of 3 Controller nodes and can scale to other control plane-based composable roles. When you use composable roles, each service must run on exactly 3 additional dedicated nodes and the total number of nodes in the control plane must be odd to maintain Pacemaker quorum.

The OpenStackControlPlane custom resource creates control plane-based nodes as virtual machines within OpenShift Virtualization.

Prerequisites

- Ensure your OpenShift Container Platform cluster is operational and you have installed the director Operator correctly.
- Ensure that you have installed the **oc** command line tool on your workstation.
- Use the OpenStackNet resource to create a control plane network and any additional isolated networks.

Procedure

1. Create a file named **openstack-controller.yaml** on your workstation. Include the resource specification for the Controller nodes. For example, the specification for a control plane that consists of 3 Controller nodes is as follows:

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackControlPlane
metadata:
  name: overcloud
  namespace: openstack
spec:
  openStackClientImageURL: registry.redhat.io/rhosp-beta/openstack-tripleoclient:16.2
  openStackClientNetworks:
        - ctlplane
        - external
        - internalapi
  openStackClientStorageClass: host-nfs-storageclass
  passwordSecret: userpassword
  gitSecret: git-secret
  virtualMachineRoles:
    controller:
      roleName: Controller
```

```
    roleCount: 3
    networks:
      - ctlplane
      - internalapi
      - external
      - tenant
      - storage
      - storagemgmt
    cores: 6
    memory: 12
    diskSize: 50
    baseImageVolumeName: openstack-base-img
    storageClass: host-nfs-storageclass
```

Set the following values in the resource specification:

metadata.name

Set to the name of the overcloud control plane, which is **overcloud**.

metadata.namespace

Set to the director Operator namespace, which is **openstack**.

spec

Set the configuration for the control plane. For descriptions of the values you can use in this section, view the specification schema in the custom resource definition for the **openstackcontrolplane** CRD:

```
$ oc describe crd openstackcontrolplane
```

Save the file when you have finished configuring the control plane specification.

2. Create the control plane:

```
$ oc create -f openstack-controller.yaml -n openstack
```

Wait until OCP creates the resources related to OpenStackControlPlane resource.

As a part of the OpenStackControlPlane resource, the director Operator also creates an OpenStackClient pod that you can access through a remote shell and run RHOSP commands.

Verification

1. View the resource for the control plane:

```
$ oc get openstackcontrolplane/overcloud -n openstack
```

2. View the OpenStackVMSet resources to verify the creation of the control plane virtual machine set:

```
$ oc get openstackvmsets -n openstack
```

3. View the virtual machine resources to verify the creation of the control plane virtual machines in OpenShift Virtualization:

```
$ oc get virtualmachines
```

4. Test access to the **openstackclient** remote shell:

```
$ oc rsh openstackclient -n openstack
```

5.2. CREATING COMPUTE NODES WITH OPENSTACKBAREMETALSET

Compute nodes provide computing resources to your Red Hat OpenStack Platform environment. You must have at least one Compute node in your overcloud and you can scale the number of Compute nodes after deployment.

The OpenStackBaremetalSet custom resource creates Compute nodes from bare metal machines that OpenShift Container Platform manages.

Prerequisites

- Ensure your OpenShift Container Platform cluster is operational and you have installed the director Operator correctly.
- Ensure that you have installed the **oc** command line tool on your workstation.
- Use the OpenStackNet resource to create a control plane network and any additional isolated networks.

Procedure

1. Create a file named **openstack-compute.yaml** on your workstation. Include the resource specification for the Compute nodes. For example, the specification for 1 Compute node is as follows:

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackBaremetalSet
metadata:
  name: compute
  namespace: openstack
spec:
  count: 1
  baseImageUrl: http://host/images/rhel-image-8.4.x86_64.qcow2
  deploymentSSHSecret: osp-controlplane-ssh-keys
  ctlplaneInterface: enp2s0
  networks:
    - ctlplane
    - internalapi
    - tenant
    - storage
  roleName: Compute
  passwordSecret: userpassword
```

Set the following values in the resource specification:

metadata.name

Set to the name of the Compute node bare metal set, which is **overcloud**.

metadata.namespace

Set to the director Operator namespace, which is **openstack**.

spec

Set the configuration for the Compute nodes. For descriptions of the values you can use in this section, view the specification schema in the custom resource definition for the **openstackbaremetalset** CRD:

```
$ oc describe crd openstackbaremetalset
```

Save the file when you have finished configuring the Compute node specification.

2. Create the Compute nodes:

```
$ oc create -f openstack-compute.yaml -n openstack
```

Verification

1. View the resource for the Compute nodes:

```
$ oc get openstackbaremetalset/compute -n openstack
```

2. View the bare metal machines that OpenShift manages to verify the creation of the Compute nodes:

```
$ oc get baremetalhosts -n openshift-machine-api
```

CHAPTER 6. CONFIGURING OVERCLOUD SOFTWARE WITH THE DIRECTOR OPERATOR

You can configure your overcloud after you have provisioned virtual and bare metal nodes for your overcloud. You must create an OpenStackPlaybookGenerator resource to generate your Ansible playbooks, register your nodes to either the Red Hat Customer Portal or Red Hat Satellite, and then run the **tripleo-deploy.sh** script to apply the configuration to your nodes

6.1. CREATING ANSIBLE PLAYBOOKS FOR OVERCLOUD CONFIGURATION WITH OPENSTACKPLAYBOOKGENERATOR

After you provision the overcloud infrastructure, you must create a set of Ansible playbooks to configure the Red Hat OpenStack Platform (RHOSP) software on the overcloud nodes. You create these playbooks with the OpenStackPlaybookGenerator resource, which uses the **config-download** feature in RHOSP director to convert heat configuration to playbooks.

Prerequisites

- Ensure your OpenShift Container Platform cluster is operational and you have installed the director Operator correctly.
- Ensure that you have installed the **oc** command line tool on your workstation.
- Configure a **git-secret** Secret that contains authentication details for your remote Git repository.
- Configure a **tripleo-tarball-config** ConfigMap that contains your custom heat templates.
- Configure a **heat-env-config** ConfigMap that contains your custom environment files.

Procedure

1. Create a file named **openstack-playbook-generator.yaml** on your workstation. Include the resource specification to generate the Ansible playbooks. For example, the specification to generate the playbooks is as follows:

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackPlaybookGenerator
metadata:
  name: default
  namespace: openstack
spec:
  imageURL: registry.redhat.io/rhosp-beta/openstack-tripleoclient:16.2
  gitSecret: git-secret
  heatEnvConfigMap: heat-env-config
  tarballConfigMap: tripleo-tarball-config
```

   Set the following values in the resource specification:

   metadata.name

   Set to the name of the Compute node bare metal set, which is **default**.

   metadata.namespace

   Set to the director Operator namespace, which is **openstack**.

spec.gitSecret

Set to the ConfigMap that contains the Git authentication credentials, which is **git-secret**.

spec.tarballConfigMap

Set to the ConfigMap that contains the tarball with your custom heat templates, which is **tripleo-tarball-config**.

spec.heatEnvConfigMap

Set to the ConfigMap that contains your custom environment files, which is **heat-env-config**.

For more descriptions of the values you can use in the **spec** section, view the specification schema in the custom resource definition for the **openstackplaybookgenerator** CRD:

```
$ oc describe crd openstackplaybookgenerator
```

Save the file when you have finished configuring the Ansible playbook generator specification.

2. Create the Ansible playbook generator:

```
$ oc create -f openstack-playbook-generator.yaml -n openstack
```

Verification

- View the resource for the playbook generator:

```
$ oc get openstackplaybookgenerator/default -n openstack
```

6.2. EPHEMERAL HEAT CONTAINER IMAGE SOURCE PARAMETERS

To create an ephemeral heat service, The OpenStackPlaybookGenerator resource requires four specific container images from registry.redhat.io:

- **openstack-heat-api**
- **openstack-heat-engine**
- **openstack-mariadb**
- **openstack-rabbitmq**

You can change the source location of these images with the **spec.ephemeralHeatSettings** parameter. For example, if you host these images or a Red Hat Satellite Server, you can change the **spec.ephemeralHeatSettings** parameter and sub-parameters to use the Red Hat Satellite Server as the source for these images.

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackPlaybookGenerator
metadata:
  name: default
  namespace: openstack
spec:
  imageURL: registry.redhat.io/rhosp-beta/openstack-tripleoclient:16.2
  gitSecret: git-secret
```

```
  heatEnvConfigMap: heat-env-config
  tarballConfigMap: tripleo-tarball-config
  ephemeralHeatSettings:
    heatAPIImageURL: <heat_api_image_location>
    heatEngineImageURL: <heat_engine_image_location>
    mariadbImageURL: <mariadb_image_location>
    rabbitImageURL: <rabbitmq_image-location>
```

Set the following values in the resource specification:

spec.ephemeralHeatSettings.heatAPIImageURL

Image location for the heat API.

spec.ephemeralHeatSettings.heatEngineImageURL

Image location for the heat engine.

spec.ephemeralHeatSettings.mariadbImageURL

Image location for MariaDB.

spec.ephemeralHeatSettings.rabbitImageURL

Image location for RabbitMQ.

6.3. PLAYBOOK GENERATION INTERACTIVE MODE PARAMETER

To debug playbook generation operations, you can set the OpenStackPlaybookGenerator resource to use interactive mode.

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackPlaybookGenerator
metadata:
  name: default
  namespace: openstack
spec:
  imageURL: registry.redhat.io/rhosp-beta/openstack-tripleoclient:16.2
  gitSecret: git-secret
  heatEnvConfigMap: heat-env-config
  tarballConfigMap: tripleo-tarball-config
  interactive: true
```

In this mode, the OpenStackPlaybookGenerator resource creates the environment to start rendering the playbooks but does not automatically render the playbooks.

6.4. USING THE TRIPLEO-DEPLOY.SH SCRIPT

As a part of the OpenStackControlPlane resource, the director Operator creates an OpenStackClient pod that you access through a remote shell and run Red Hat OpenStack Platform commands. This pod also contains a script named **tripleo-deploy.sh** which you use to update your configuration and apply the latest configuration to the overcloud nodes.

You can run the **tripleo-deploy.sh** script with the following options:

-d

Show the Git diff comparing the newest version of the rendered playbooks to the previous accepted version.

-a

Accept the newest available version of the rendered playbooks and tag them as **latest**.

-p

Apply the Ansible playbooks against the overcloud nodes.

6.5. REGISTERING THE OPERATING SYSTEM OF YOUR OVERCLOUD

Before the director Operator configures the overcloud software on nodes, you must register the operating system of all nodes to either the Red Hat Customer Portal or Red Hat Satellite Server, and enable repositories for your nodes. To register your nodes, you can use the **redhat_subscription** Ansible module with the inventory file that the OpentackPlaybookGenerator resource creates.

Prerequisites

- Ensure your OpenShift Container Platform cluster is operational and you have installed the director Operator correctly.
- Ensure that you have installed the **oc** command line tool on your workstation.
- Use the OpenStackControlPlane resource to create a control plane.
- Use the OpenStackBareMetalSet resource to create bare metal Compute nodes.
- Use the OpentackPlaybookGenerator to create the Ansible playbook configuration for your overcloud.

Procedure

1. Access the remote shell for **openstackclient**:

```
$ oc rsh openstackclient -n openstack
```

2. Change to the **cloud-admin** home directory:

```
$ cd /home/cloud-admin
```

3. Optional: Check the diff for the overcloud Ansible playbooks:

```
$ ./tripleo-deploy.sh -d
```

4. Accept the newest version of the rendered Ansible playbooks and tag them as **latest**:

```
$ ./tripleo-deploy.sh -a
```

The newest version of the playbooks includes an inventory file for the overcloud configuration. You can find this inventory file on the OpenstackClient pod at **/home/cloud-admin/playbooks/tripleo-ansible/inventory.yaml**.

5. Create a playbook that uses the **redhat_subscription** modules to register your nodes. For example, the following playbook registers Controller nodes:

```
---
- name: Register Controller nodes
```

```
hosts: Controller
become: yes
vars:
  repos:
    - rhel-8-for-x86_64-baseos-eus-rpms
    - rhel-8-for-x86_64-appstream-eus-rpms
    - rhel-8-for-x86_64-highavailability-eus-rpms
    - ansible-2.9-for-rhel-8-x86_64-rpms
    - advanced-virt-for-rhel-8-x86_64-rpms
    - openstack-16.2-for-rhel-8-x86_64-rpms
    - fast-datapath-for-rhel-8-x86_64-rpms
tasks:
  - name: Register system
    redhat_subscription:
      username: myusername
      password: p@55w0rd!
      org_id: 1234567
      release: 8.4
      pool_ids: 1a85f9223e3d5e43013e3d6e8ff506fd
  - name: Disable all repos
    command: "subscription-manager repos --disable *"
  - name: Enable Controller node repos
    command: "subscription-manager repos --enable {{ item }}"
    with_items: "{{ repos }}"
```

This play contains the following three tasks:

- Register the node.
- Disable any auto-enabled repositories.
- Enable only the repositories relevant to the Controller node. The repositories are listed with the **repos** variable.

6. Register the overcloud nodes to required repositories:

```
ansible-playpook -i /home/cloud-admin/playbooks/tripleo-ansible/inventory.yaml ./rhsm.yaml
```

Additional resources

- "redhat_subscription – Manage registration and subscriptions to RHSM using the subscription-manager command"
- "Running Ansible-based registration manually"

6.6. APPLYING OVERCLOUD CONFIGURATION WITH THE DIRECTOR OPERATOR

You can configure the overcloud with director Operator only after you have created your control plane, provisioned your bare metal Compute nodes, and generated the Ansible playbooks to configure software on each node. When you create an OpenStackControlPlane resource, the director Operator creates an OpenStackClient pod that you access through a remote shell and run the **tripleo-deploy.sh** script to configure the overcloud.

Prerequisites

- Ensure your OpenShift Container Platform cluster is operational and you have installed the director Operator correctly.
- Ensure that you have installed the **oc** command line tool on your workstation.
- Use the OpenStackControlPlane resource to create a control plane.
- Use the OpenStackBareMetalSet resource to create bare metal Compute nodes.
- Use the OpentackPlaybookGenerator to create the Ansible playbook configuration for your overcloud.

Procedure

1. Access the remote shell for **openstackclient**:

```
$ oc rsh openstackclient -n openstack
```

2. Change to the **cloud-admin** home directory:

```
$ cd /home/cloud-admin
```

3. Optional: Check the diff for the overcloud Ansible playbooks:

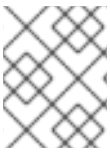
```
$ ./tripleo-deploy.sh -d
```

4. Accept the newest version of the rendered Ansible playbooks and tag them as **latest**:

```
$ ./tripleo-deploy.sh -a
```

5. Apply the Ansible playbooks against the overcloud nodes:

```
$ ./tripleo-deploy.sh -p
```



NOTE

You can view a log of the Ansible configuration at **~/ansible.log** within the **openstackclient** pod.

PART II. DEPLOYMENT SCENARIOS

CHAPTER 7. DIRECTOR OPERATOR DEPLOYMENT SCENARIO: OVERCLOUD WITH HYPER-CONVERGED INFRASTRUCTURE (HCI)

You can use the director Operator to deploy an overcloud with Hyper-Converged Infrastructure (HCI). This scenario installs both Compute and Ceph Storage OSD services on the same nodes.

Prerequisites

- Use the **quay.io/openstack-k8s-operators/rhosp16-openstack-tripleoclient:16.2_20210521.1** image or later for the **openStackClientImageURL** in the OpenStackClient resource.
- Your Compute HCI nodes require extra disks to use as OSDs.

7.1. CREATING A DATA VOLUME FOR THE BASE OPERATING SYSTEM

You must create a data volume with the OpenShift Container Platform (OCP) cluster to store the base operating system image for your Controller virtual machines.

Prerequisites

- Download a Red Hat Enterprise Linux 8 QCOW2 image to your workstation.You can download this image from the Product Download section of the Red Hat Customer Portal.
- Install the **virtctl** client tool on your workstation. You can install this tool on a Red Hat Enterprise Linux workstation using the following commands:

```
$ sudo subscription-manager repos --enable=cnv-2.6-for-rhel-8-x86_64-rpms
$ sudo dnf install -y kubevirt-virtctl
```

- Install the **virt-customize** client tool on your workstation. You can install this tool on a Red Hat Enterprise Linux workstation using the following command:

```
$ dnf install -y libguestfs-tools-c
```

Procedure

1. The default QCOW2 image that you have downloaded from access.redhat.com does not use biosdev predictable network interface names. Modify the image with **virt-customize** to use biosdev predictable network interface names:

```
$ sudo virt-customize -a <local path to image> --run-command 'sed -i -e "s/^\
(kernelopts=.*\)net.ifnames=0 \(.*\)/\1\2/" /boot/grub2/grubenv'
$ sudo virt-customize -a <local path to image> --run-command 'sed -i -e "s/^\
(GRUB_CMDLINE_LINUX=.*\)net.ifnames=0 \(.*\)/\1\2/" /etc/default/grub'
```

2. Upload the image to OpenShift Virtualization with **virtctl**:

```
$ virtctl image-upload dv openstack-base-img -n openstack --size=50Gi --image-path=<local
path to image> --storage-class <storage class> --insecure
```

For the **--storage-class** option, choose a storage class from your cluster. View a list of storage classes with the following command:

```
$ oc get storageclass
```

3. When you create the OpenStackControlPlane resource and individual OpenStackVmSet resources, set the **baseImageVolumeName** parameter to the data volume name:

```
...
spec:
  ...
  baseImageVolumeName: openstack-base-img
...
```

Additional resources

- "Uploading local disk images by using the virtctl tool"

7.2. ADDING AUTHENTICATION DETAILS FOR YOUR REMOTE GIT REPOSITORY

The director Operator stores rendered Ansible playbooks to a remote Git repository and uses this repository to track changes to the overcloud configuration. You can use any Git repository that supports SSH authentication. You must provide details for the Git repository as an OpenShift Secret resource named **git-secret**.

Prerequisites

- Ensure your OpenShift Container Platform cluster is operational and you have installed the director Operator correctly.
- Ensure that you have installed the **oc** command line tool on your workstation.
- Prepare a remote Git repository for the director Operator to store the generated configuration for your overcloud.
- Prepare an SSH key pair. Upload the public key to the Git repository and keep the private key available to add to the **git-secret** Secret resource.

Procedure

- Create the Secret resource:

```
$ oc create secret generic git-secret -n openstack --from-file=git_ssh_identity=
<path_to_private_SSH_key> --from-literal=git_url=<git_server_URL>
```

The **git-secret** Secret resource contains two key-value pairs:

git_ssh_identity

The private key to access the Git repository. The **--from-file** option stores the content of the private SSH key file.

git_url

The SSH URL of the git repository to store the configuration. The **--from-literal** option stores the URL that you enter for this key.

Verification

- View the Secret resource:

```
$ oc get secret/git-secret -n openstack
```

Additional resources

- "Providing sensitive data to pods"

7.3. SETTING THE ROOT PASSWORD FOR NODES

To access the **root** user with a password on each node, you can set a **root** password in a Secret resource named **userpassword**.

NOTE

Setting the root password for nodes is optional. If you do not set a **root** password, you can still log into nodes with the SSH keys defined in the **osp-controlplane-ssh-keys** Secret.

Prerequisites

- Ensure your OpenShift Container Platform cluster is operational and you have installed the director Operator correctly.
- Ensure that you have installed the **oc** command line tool on your workstation.

Procedure

1. Convert your chosen password to a base64 value:

```
$ echo -n "p@ssw0rd!" | base64
cEBzc3cwcmQh
```

NOTE

The **-n** option removes the trailing newline from the echo output.

2. Create a file named **openstack-userpassword.yaml** on your workstation. Include the following resource specification for the Secret in the file:

```
apiVersion: v1
kind: Secret
metadata:
  name: userpassword
  namespace: openstack
data:
  NodeRootPassword: "cEBzc3cwcmQh"
```

Set the **NodeRootPassword** parameter to your base64 encoded password.

3. Create the **userpassword** Secret:

```
$ oc create -f openstack-userpassword.yaml -n openstack
```

Additional resources

- "Providing sensitive data to pods"

7.4. CREATING AN OVERCLOUD CONTROL PLANE NETWORK WITH OPENSTACKNET

You must create one control plane network for your overcloud. In addition to IP address assignment, the OpenStackNet resource includes information to define the network configuration policy that OpenShift Virtualization uses to attach any virtual machines to the network.

Prerequisites

- Ensure your OpenShift Container Platform cluster is operational and you have installed the director Operator correctly.
- Ensure that you have installed the **oc** command line tool on your workstation.

Procedure

1. Create a file named **ctlplane-network.yaml** on your workstation. Include the resource specification for the control plane network, which is named **ctlplane**. For example, the specification for a control plane that uses a Linux bridge connected to the **enp6s0** Ethernet device on each worker node is as follows:

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackNet
metadata:
  name: ctlplane
spec:
  cidr: 192.168.25.0/24
  allocationStart: 192.168.25.100
  allocationEnd: 192.168.25.250
  gateway: 192.168.25.1
  attachConfiguration:
    nodeNetworkConfigurationPolicy:
      nodeSelector:
        node-role.kubernetes.io/worker: ""
      desiredState:
        interfaces:
        - bridge:
            options:
              stp:
                enabled: false
            port:
            - name: enp6s0
          description: Linux bridge with enp6s0 as a port
```

```
      name: br-osp
      state: up
      type: linux-bridge
```

Set the following values in the resource specification:

metadata.name

Set to the name of the control plane network, which is **ctlplane**.

spec

Set the network configuration for the control plane network. For descriptions of the values you can use in this section, view the specification schema in the custom resource definition for the **openstacknet** CRD:

```
$ oc describe crd openstacknet
```

Save the file when you have finished configuring the network specification.

2. Create the control plane network:

```
$ oc create -f ctlplane-network.yaml -n openstack
```

Verification

- View the resource for the control plane network:

```
$ oc get openstacknet/ctlplane
```

7.5. CREATING VLAN NETWORKS FOR NETWORK ISOLATION WITH OPENSTACKNET

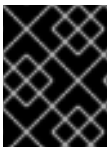
You must create additional networks to implement network isolation for your composable networks. To accomplish this network isolation, you can place your composable networks networks on individual VLAN networks. In addition to IP address assignment, the OpenStackNet resource includes information to define the network configuration policy that OpenShift Virtualization uses to attach any virtual machines to VLAN networks.

To use the default Red Hat OpenStack Platform networks, you must create an OpenStackNet resource for each network.

Table 7.1. Default Red Hat OpenStack Platform networks

Network	VLAN	CIDR	Allocation
External	10	10.0.0.0/24	10.0.0.4 - 10.0.0.250
InternalApi	20	172.16.2.0/24	172.16.2.4 - 172.16.2.250
Storage	30	172.16.1.0/24	172.16.1.4 - 172.16.1.250
StorageMgmt	40	172.16.3.0/24	172.16.3.4 - 172.16.3.250

Network	VLAN	CIDR	Allocation
Tenant	50	172.16.0.0/24	172.16.0.4 - 172.16.0.250



IMPORTANT

To use different networking details for each network, you must create a custom **network_data.yaml** file.

Prerequisites

- Ensure your OpenShift Container Platform cluster is operational and you have installed the director Operator correctly.
- Ensure that you have installed the **oc** command line tool on your workstation.

Procedure

1. Create a file for your new network. For example, for the internal API network, create a file named **internalapi-network.yaml** on your workstation. Include the resource specification for the VLAN network. For example, the specification for an internal API network that manages VLAN-tagged traffic over a Linux bridge connected to the **enp6s0** Ethernet device on each worker node is as follows:

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackNet
metadata:
  name: internalapi
spec:
  cidr: 172.16.2.0/24
  vlan: 20
  allocationStart: 172.16.2.4
  allocationEnd: 172.16.2.250
  attachConfiguration:
    nodeNetworkConfigurationPolicy:
      nodeSelector:
        node-role.kubernetes.io/worker: ""
      desiredState:
        interfaces:
        - bridge:
            options:
              stp:
                enabled: false
            port:
            - name: enp6s0
          description: Linux bridge with enp7s0 as a port
          name: br-osp
          state: up
          type: linux-bridge
```

When you use a VLAN for network isolation with **linux-bridge** the following happens:

- The director Operator creates a Node Network Configuration Policy for the bridge interface specified in the resource, which uses **nmstate** to configure the bridge on worker nodes.
- The director Operator creates a Network Attach Definition for each network, which defines the Multus CNI plugin configuration. When you specify the VLAN ID on the Network Attach Definition, the Multus CNI plugin enables **vlan-filtering** on the bridge.
- The director Operator attaches a dedicated interface for each network on a virtual machine. This means the network template for the **OSVMSet** is a multi-NIC network template.

Set the following values in the resource specification:

metadata.name

Set to the name of the control plane network, which is **ctlplane**.

spec

Set the network configuration for the control plane network. For descriptions of the values you can use in this section, view the specification schema in the custom resource definition for the **openstacknet** CRD:

```
$ oc describe crd openstacknet
```

Save the file when you have finished configuring the network specification.

2. Create the internal API network:

```
$ oc create -f internalapi-network.yaml -n openstack
```

Verification

- View the resource for the internal API network:

```
$ oc get openstacknet/internalapi -n openstack
```

7.6. CREATING A CONTROL PLANE WITH OPENSTACKCONTROLPLANE

The overcloud control plane contains the main Red Hat OpenStack Platform services that manage overcloud functionality. The control plane usually consists of 3 Controller nodes and can scale to other control plane-based composable roles. When you use composable roles, each service must run on exactly 3 additional dedicated nodes and the total number of nodes in the control plane must be odd to maintain Pacemaker quorum.

The OpenStackControlPlane custom resource creates control plane-based nodes as virtual machines within OpenShift Virtualization.

Prerequisites

- Ensure your OpenShift Container Platform cluster is operational and you have installed the director Operator correctly.
- Ensure that you have installed the **oc** command line tool on your workstation.

- Use the OpenStackNet resource to create a control plane network and any additional isolated networks.

Procedure

1. Create a file named **openstack-controller.yaml** on your workstation. Include the resource specification for the Controller nodes. For example, the specification for a control plane that consists of 3 Controller nodes is as follows:

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackControlPlane
metadata:
  name: overcloud
  namespace: openstack
spec:
  openStackClientImageURL: registry.redhat.io/rhosp-beta/openstack-tripleoclient:16.2
  openStackClientNetworks:
        - ctlplane
        - external
        - internalapi
  openStackClientStorageClass: host-nfs-storageclass
  passwordSecret: userpassword
  gitSecret: git-secret
  virtualMachineRoles:
    controller:
      roleName: Controller
      roleCount: 3
      networks:
        - ctlplane
        - internalapi
        - external
        - tenant
        - storage
        - storagemgmt
      cores: 6
      memory: 12
      diskSize: 50
      baseImageVolumeName: openstack-base-img
      storageClass: host-nfs-storageclass
```

Set the following values in the resource specification:

metadata.name

Set to the name of the overcloud control plane, which is **overcloud**.

metadata.namespace

Set to the director Operator namespace, which is **openstack**.

spec

Set the configuration for the control plane. For descriptions of the values you can use in this section, view the specification schema in the custom resource definition for the **openstackcontrolplane** CRD:

```
$ oc describe crd openstackcontrolplane
```

Save the file when you have finished configuring the control plane specification.

2. Create the control plane:

```
$ oc create -f openstack-controller.yaml -n openstack
```

Wait until OCP creates the resources related to OpenStackControlPlane resource.

As a part of the OpenStackControlPlane resource, the director Operator also creates an OpenStackClient pod that you can access through a remote shell and run RHOSP commands.

Verification

1. View the resource for the control plane:

```
$ oc get openstackcontrolplane/overcloud -n openstack
```

2. View the OpenStackVMSet resources to verify the creation of the control plane virtual machine set:

```
$ oc get openstackvmsets -n openstack
```

3. View the virtual machine resources to verify the creation of the control plane virtual machines in OpenShift Virtualization:

```
$ oc get virtualmachines
```

4. Test access to the **openstackclient** remote shell:

```
$ oc rsh openstackclient -n openstack
```

7.7. CREATING DIRECTORIES FOR TEMPLATES AND ENVIRONMENT FILES

Create directories on your workstation to store your custom templates and environment files, which you upload to ConfigMaps in OpenShift Container Platform (OCP).

Procedure

1. Create a directory for your custom templates:

```
$ mkdir custom_templates
```

2. Create a directory for your custom environment files:

```
$ mkdir custom_environment_files
```

7.8. CUSTOM NIC HEAT TEMPLATE FOR CONTROLLER NODES

The following example is a heat template that contains NIC configuration for the Controller virtual machines.

```
heat_template_version: rocky
description: >
  Software Config to drive os-net-config to configure multiple interfaces for the Controller role.
parameters:
  ControlPlaneIp:
    default: ''
    description: IP address/subnet on the ctlplane network
    type: string
  ControlPlaneSubnetCidr:
    default: ''
    description: >
      The subnet CIDR of the control plane network. (The parameter is
      automatically resolved from the ctlplane subnet's cidr attribute.)
    type: string
  ControlPlaneDefaultRoute:
    default: ''
    description: The default route of the control plane network. (The parameter
      is automatically resolved from the ctlplane subnet's gateway_ip attribute.)
    type: string
  ControlPlaneStaticRoutes:
    default: []
    description: >
      Routes for the ctlplane network traffic.
      JSON route e.g. [{'destination':'10.0.0.0/16', 'nexthop':'10.0.0.1'}]
      Unless the default is changed, the parameter is automatically resolved
      from the subnet host_routes attribute.
    type: json
  ControlPlaneMtu:
    default: 1500
    description: The maximum transmission unit (MTU) size(in bytes) that is
      guaranteed to pass through the data path of the segments in the network.
      (The parameter is automatically resolved from the ctlplane network's mtu attribute.)
    type: number
  StorageIpSubnet:
    default: ''
    description: IP address/subnet on the storage network
    type: string
  StorageNetworkVlanID:
    default: 30
    description: Vlan ID for the storage network traffic.
    type: number
  StorageMtu:
    default: 1500
    description: The maximum transmission unit (MTU) size(in bytes) that is
      guaranteed to pass through the data path of the segments in the
      Storage network.
    type: number
  StorageInterfaceRoutes:
    default: []
    description: >
      Routes for the storage network traffic.
      JSON route e.g. [{'destination':'10.0.0.0/16', 'nexthop':'10.0.0.1'}]
      Unless the default is changed, the parameter is automatically resolved
      from the subnet host_routes attribute.
    type: json
  StorageMgmtIpSubnet:
```

```
  default: ''
  description: IP address/subnet on the storage_mgmt network
  type: string
StorageMgmtNetworkVlanID:
  default: 40
  description: Vlan ID for the storage_mgmt network traffic.
  type: number
StorageMgmtMtu:
  default: 1500
  description: The maximum transmission unit (MTU) size(in bytes) that is
    guaranteed to pass through the data path of the segments in the
    StorageMgmt network.
  type: number
StorageMgmtInterfaceRoutes:
  default: []
  description: >
    Routes for the storage_mgmt network traffic.
    JSON route e.g. [{'destination':'10.0.0.0/16', 'nexthop':'10.0.0.1'}]
    Unless the default is changed, the parameter is automatically resolved
    from the subnet host_routes attribute.
  type: json
InternalApiIpSubnet:
  default: ''
  description: IP address/subnet on the internal_api network
  type: string
InternalApiNetworkVlanID:
  default: 20
  description: Vlan ID for the internal_api network traffic.
  type: number
InternalApiMtu:
  default: 1500
  description: The maximum transmission unit (MTU) size(in bytes) that is
    guaranteed to pass through the data path of the segments in the
    InternalApi network.
  type: number
InternalApiInterfaceRoutes:
  default: []
  description: >in your `custom_templates` directory.
    Routes for the internal_api network traffic.
    JSON route e.g. [{'destination':'10.0.0.0/16', 'nexthop':'10.0.0.1'}]
    Unless the default is changed, the parameter is automatically resolved
    from the subnet host_routes attribute.
  type: json
TenantIpSubnet:
  default: ''
  description: IP address/subnet on the tenant network
  type: string
TenantNetworkVlanID:
  default: 50
  description: Vlan ID for the tenant network traffic.
  type: number
TenantMtu:
  default: 1500
  description: The maximum transmission unit (MTU) size(in bytes) that is
    guaranteed to pass through the data path of the segments in the
    Tenant network.
```

```
    type: number
  TenantInterfaceRoutes:
    default: []
    description: >
      Routes for the tenant network traffic.
      JSON route e.g. [{'destination':'10.0.0.0/16', 'nexthop':'10.0.0.1'}]
      Unless the default is changed, the parameter is automatically resolved
      from the subnet host_routes attribute.
    type: json
  ExternalIpSubnet:
    default: ''
    description: IP address/subnet on the external network
    type: string
  ExternalNetworkVlanID:
    default: 10
    description: Vlan ID for the external network traffic.
    type: number
  ExternalMtu:
    default: 1500
    description: The maximum transmission unit (MTU) size(in bytes) that is
      guaranteed to pass through the data path of the segments in the
      External network.
    type: number
  ExternalInterfaceDefaultRoute:
    default: ''
    description: default route for the external network
    type: string
  ExternalInterfaceRoutes:
    default: []
    description: >
      Routes for the external network traffic.
      JSON route e.g. [{'destination':'10.0.0.0/16', 'nexthop':'10.0.0.1'}]
      Unless the default is changed, the parameter is automatically resolved
      from the subnet host_routes attribute.
    type: json
  DnsServers: # Override this via parameter_defaults
    default: []
    description: >
      DNS servers to use for the Overcloud (2 max for some implementations).
      If not set the nameservers configured in the ctlplane subnet's
      dns_nameservers attribute will be used.
    type: comma_delimited_list
  DnsSearchDomains: # Override this via parameter_defaults
    default: []
    description: A list of DNS search domains to be added (in order) to resolv.conf.
    type: comma_delimited_list
resources:
  OsNetConfigImpl:
    type: OS::Heat::SoftwareConfig
    properties:
      group: script
      config:
        str_replace:
          template:
            get_file: /usr/share/openstack-tripleo-heat-templates/network/scripts/run-os-net-config.sh
          params:
```

```
$network_config:
  network_config:

  ## NIC2 - Control Plane ##
  - type: interface
    name: nic2
    mtu:
      get_param: ControlPlaneMtu
    use_dhcp: false
    dns_servers:
      get_param: DnsServers
    domain:
      get_param: DnsSearchDomains
    addresses:
    - ip_netmask:
        list_join:
        - /
        - - get_param: ControlPlaneIp
          - get_param: ControlPlaneSubnetCidr
    routes:
      list_concat_unique:
        - get_param: ControlPlaneStaticRoutes

  ## NIC3 - External ##
  - type: ovs_bridge
    name: bridge_name
    mtu:
      get_param: ExternalMtu
    dns_servers:
      get_param: DnsServers
    use_dhcp: false
    addresses:
    - ip_netmask:
        get_param: ExternalIpSubnet
    routes:
      list_concat_unique:
        - get_param: ExternalInterfaceRoutes
        - - default: true
            next_hop:
              get_param: ExternalInterfaceDefaultRoute
    members:
    - type: interface
      name: nic3
      mtu:
        get_param: ExternalMtu
      use_dhcp: false
      primary: true

  ## NIC4 - Internal API ##
  - type: interface
    name: nic4
    mtu:
      get_param: InternalApiMtu
    use_dhcp: false
    addresses:
    - ip_netmask:
```

```
            get_param: InternalApiIpSubnet
          routes:
            list_concat_unique:
              - get_param: InternalApiInterfaceRoutes

        ## NIC5 - Storage ##
        - type: interface
          name: nic5
          mtu:
            get_param: StorageMtu
          use_dhcp: false
          addresses:
          - ip_netmask:
              get_param: StorageIpSubnet
          routes:
            list_concat_unique:
              - get_param: StorageInterfaceRoutes

        ## NIC6 - StorageMgmt ##
        - type: interface
          name: nic6
          mtu:
            get_param: StorageMgmtMtu
          use_dhcp: false
          addresses:
          - ip_netmask:
              get_param: StorageMgmtIpSubnet
          routes:
            list_concat_unique:
              - get_param: StorageMgmtInterfaceRoutes

        ## NIC7 - Tenant ##
        - type: ovs_bridge
          name: br-isolated
          mtu:
            get_param: TenantMtu
          dns_servers:
            get_param: DnsServers
          use_dhcp: false
          addresses:
          - ip_netmask:
              get_param: TenantIpSubnet
          routes:
            list_concat_unique:
              - get_param: TenantInterfaceRoutes
          members:
          - type: interface
            name: nic7
            mtu:
              get_param: TenantMtu
            use_dhcp: false
            primary: true

outputs:
  OS::stack_id:
```

```
    description: The OsNetConfigImpl resource.
    value:
      get_resource: OsNetConfigImpl
```

This configuration maps the the networks to the following bridges and interfaces:

Networks	Bridge	interface
Control Plane		**nic2**
External	**br-ex**	**nic3**
Internalapi		**nic4**
Storage		**nic5**
StorageMgmt		**nic6**
Tenant	**br-isolated**	**nic7**

To use this template in your deployment, copy the contents of the example to **net-config-multi-nic-controller.yaml** in your **custom_templates** directory on your workstation.

7.9. CUSTOM NIC HEAT TEMPLATE FOR HCI COMPUTE NODES

The following example is a heat template that contains NIC configuration for the HCI Compute bare metal nodes.

```
heat_template_version: rocky
description: >
  Software Config to drive os-net-config to configure VLANs for the Compute role.
parameters:
  ControlPlaneIp:
    default: ''
    description: IP address/subnet on the ctlplane network
    type: string
  ControlPlaneSubnetCidr:
    default: ''
    description: >
      The subnet CIDR of the control plane network. (The parameter is
      automatically resolved from the ctlplane subnet's cidr attribute.)
    type: string
  ControlPlaneDefaultRoute:
    default: ''
    description: The default route of the control plane network. (The parameter
      is automatically resolved from the ctlplane subnet's gateway_ip attribute.)
    type: string
  ControlPlaneStaticRoutes:
    default: []
    description: >
      Routes for the ctlplane network traffic.
```

```
    JSON route e.g. [{'destination':'10.0.0.0/16', 'nexthop':'10.0.0.1'}]
    Unless the default is changed, the parameter is automatically resolved
    from the subnet host_routes attribute.
  type: json
ControlPlaneMtu:
  default: 1500
  description: The maximum transmission unit (MTU) size(in bytes) that is
    guaranteed to pass through the data path of the segments in the network.
    (The parameter is automatically resolved from the ctlplane network's mtu attribute.)
  type: number
StorageIpSubnet:
  default: ''
  description: IP address/subnet on the storage network
  type: string
StorageNetworkVlanID:
  default: 30
  description: Vlan ID for the storage network traffic.
  type: number
StorageMtu:
  default: 1500
  description: The maximum transmission unit (MTU) size(in bytes) that is
    guaranteed to pass through the data path of the segments in the
    Storage network.
  type: number
StorageInterfaceRoutes:
  default: []
  description: >
    Routes for the storage network traffic.
    JSON route e.g. [{'destination':'10.0.0.0/16', 'nexthop':'10.0.0.1'}]
    Unless the default is changed, the parameter is automatically resolved
    from the subnet host_routes attribute.
  type: json
StorageMgmtIpSubnet:
  default: ''
  description: IP address/subnet on the storage_mgmt network
  type: string
StorageMgmtNetworkVlanID:
  default: 40
  description: Vlan ID for the storage_mgmt network traffic.
  type: number
StorageMgmtMtu:
  default: 1500
  description: The maximum transmission unit (MTU) size(in bytes) that is
    guaranteed to pass through the data path of the segments in the
    StorageMgmt network.
  type: number
StorageMgmtInterfaceRoutes:
  default: []
  description: >
    Routes for the storage_mgmt network traffic.
    JSON route e.g. [{'destination':'10.0.0.0/16', 'nexthop':'10.0.0.1'}]
    Unless the default is changed, the parameter is automatically resolved
    from the subnet host_routes attribute.
  type: json
InternalApiIpSubnet:
  default: ''
```

```
  description: IP address/subnet on the internal_api network
  type: string
InternalApiNetworkVlanID:
  default: 20
  description: Vlan ID for the internal_api network traffic.
  type: number
InternalApiMtu:
  default: 1500
  description: The maximum transmission unit (MTU) size(in bytes) that is
    guaranteed to pass through the data path of the segments in the
    InternalApi network.
  type: number
InternalApiInterfaceRoutes:
  default: []
  description: >
    Routes for the internal_api network traffic.
    JSON route e.g. [{'destination':'10.0.0.0/16', 'nexthop':'10.0.0.1'}]
    Unless the default is changed, the parameter is automatically resolved
    from the subnet host_routes attribute.
  type: json
TenantIpSubnet:
  default: ''
  description: IP address/subnet on the tenant network
  type: string
TenantNetworkVlanID:
  default: 50
  description: Vlan ID for the tenant network traffic.
  type: number
TenantMtu:
  default: 1500
  description: The maximum transmission unit (MTU) size(in bytes) that is
    guaranteed to pass through the data path of the segments in the
    Tenant network.
  type: number
TenantInterfaceRoutes:
  default: []
  description: >
    Routes for the tenant network traffic.
    JSON route e.g. [{'destination':'10.0.0.0/16', 'nexthop':'10.0.0.1'}]
    Unless the default is changed, the parameter is automatically resolved
    from the subnet host_routes attribute.
  type: json
ExternalMtu:
  default: 1500
  description: The maximum transmission unit (MTU) size(in bytes) that is
    guaranteed to pass through the data path of the segments in the
    External network.
  type: number
DnsServers: # Override this via parameter_defaults
  default: []
  description: >
    DNS servers to use for the Overcloud (2 max for some implementations).
    If not set the nameservers configured in the ctlplane subnet's
    dns_nameservers attribute will be used.
  type: comma_delimited_list
DnsSearchDomains: # Override this via parameter_defaults
```

```
    default: []
    description: A list of DNS search domains to be added (in order) to resolv.conf.
    type: comma_delimited_list

resources:

  MinViableMtu:
    # This resource resolves the minimum viable MTU for interfaces, bonds and
    # bridges that carry multiple VLANs. Each VLAN may have different MTU. The
    # bridge, bond or interface must have an MTU to allow the VLAN with the
    # largest MTU.
    type: OS::Heat::Value
    properties:
      type: number
      value:
        yaql:
          expression: $.data.max()
          data:
            - {get_param: ControlPlaneMtu}
            - {get_param: StorageMtu}
            - {get_param: StorageMgmtMtu}
            - {get_param: InternalApiMtu}
            - {get_param: TenantMtu}
            - {get_param: ExternalMtu}

  OsNetConfigImpl:
    type: OS::Heat::SoftwareConfig
    properties:
      group: script
      config:
        str_replace:
          template:
            get_file: /usr/share/openstack-tripleo-heat-templates/network/scripts/run-os-net-config.sh
          params:
            $network_config:
              network_config:
              - type: ovs_bridge
                name: br-isolated
                mtu:
                  get_attr: [MinViableMtu, value]
                use_dhcp: false
                dns_servers:
                  get_param: DnsServers
                domain:
                  get_param: DnsSearchDomains
                addresses:
                - ip_netmask:
                    list_join:
                    - /
                    - - get_param: ControlPlaneIp
                      - get_param: ControlPlaneSubnetCidr
                routes:
                  list_concat_unique:
                    - get_param: ControlPlaneStaticRoutes
                    - - default: true
                        next_hop:
```

```
            get_param: ControlPlaneDefaultRoute
      members:
      - type: interface
        name: nic4
        mtu:
          get_attr: [MinViableMtu, value]
        # force the MAC address of the bridge to this interface
        primary: true
      - type: vlan
        mtu:
          get_param: StorageMtu
        vlan_id:
          get_param: StorageNetworkVlanID
        addresses:
        - ip_netmask:
            get_param: StorageIpSubnet
        routes:
          list_concat_unique:
            - get_param: StorageInterfaceRoutes
      - type: vlan
        mtu:
          get_param: StorageMgmtMtu
        vlan_id:
          get_param: StorageMgmtNetworkVlanID
        addresses:
        - ip_netmask:
            get_param: StorageMgmtIpSubnet
        routes:
          list_concat_unique:
            - get_param: StorageMgmtInterfaceRoutes
      - type: vlan
        mtu:
          get_param: InternalApiMtu
        vlan_id:
          get_param: InternalApiNetworkVlanID
        addresses:
        - ip_netmask:
            get_param: InternalApiIpSubnet
        routes:
          list_concat_unique:
            - get_param: InternalApiInterfaceRoutes
      - type: vlan
        mtu:
          get_param: TenantMtu
        vlan_id:
          get_param: TenantNetworkVlanID
        addresses:
        - ip_netmask:
            get_param: TenantIpSubnet
        routes:
          list_concat_unique:
            - get_param: TenantInterfaceRoutes
    - type: ovs_bridge
      # This will default to br-ex, anything else requires specific
      # brige mapping entries for it to be used.
      name: bridge_name
```

```
            mtu:
              get_param: ExternalMtu
            use_dhcp: false
            members:
            - type: interface
              name: nic3
              mtu:
                get_param: ExternalMtu
              use_dhcp: false
              primary: true
outputs:
  OS::stack_id:
    description: The OsNetConfigImpl resource.
    value:
      get_resource: OsNetConfigImpl
```

This configuration maps the the networks to the following bridges and interfaces:

Networks	Bridge	interface
Control Plane, Storage, StorageMgmt, Internal API, Tenant	**br-isolated**	**nic4**
External	**br-ex**	**nic3**

NOTE

You can modify this configuration to suit the NIC configuration of your bare metal nodes.

To use this template in your deployment, copy the contents of the example to **net-config-two-nic-vlan-computehci.yaml** in your **custom_templates** directory on your workstation.

7.10. CREATING A ROLES_DATA.YAML FILE WITH THE COMPUTE HCI ROLE FOR THE DIRECTOR OPERATOR

To include configuration for the Compute HCI role in your overcloud, you must include the Compute HCI role in the **roles_data.yaml** file that you include with your overcloud deployment.

NOTE

Ensure you use **roles_data.yaml** as the file name.

Prerequisites

- Ensure your OpenShift Container Platform cluster is operational and you have installed the director Operator correctly.
- Ensure that you have installed the **oc** command line tool on your workstation.
- Use the OpenStackControlPlane resource to create a control plane.

Procedure

1. Access the remote shell for **openstackclient**:

```
$ oc rsh openstackclient -n openstack
```

2. Unset the **OS_CLOUD** environment variable:

```
$ unset OS_CLOUD
```

3. Change to the **cloud-admin** directory:

```
$ cd /home/cloud-admin/
```

4. Generate a new **roles_data.yaml** file with the **Controller** and **ComputeHCI** roles:

```
$ openstack overcloud roles generate Controller ComputeHCI > roles_data.yaml
```

5. Exit the **openstackclient** pod:

```
$ exit
```

6. Copy the custom **roles_data.yaml** file from the **openstackclient** pod to your custom templates directory:

```
$ oc cp openstackclient:/home/cloud-admin/roles_data.yaml
custom_templates/roles_data.yaml -n openstack
```

Additional resources

- "Creating a roles_data file"

7.11. ADDING CUSTOM TEMPLATES TO THE OVERCLOUD CONFIGURATION

Archive your custom templates into a tarball file so that you can include these templates as a part of your overcloud deployment.

Prerequisites

- Ensure your OpenShift Container Platform cluster is operational and you have installed the director Operator correctly.
- Ensure that you have installed the **oc** command line tool on your workstation.
- Create the custom templates that you want to apply to provisioned nodes.

Procedure

1. Navigate to the location of your custom templates:

```
$ cd ~/custom_templates
```

2. Archive the templates into a tarball:

```
$ tar -cvzf custom-config.tar.gz *.yaml
```

3. Create the **tripleo-tarball-config** ConfigMap and use the tarball as data:

```
$ oc create configmap tripleo-tarball-config --from-file=custom-config.tar.gz -n openstack
```

Verification

- View the ConfigMap:

```
$ oc get configmap/tripleo-tarball-config -n openstack
```

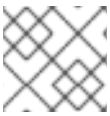
Additional resources

- "Creating and using config maps"
- "heat templates"

7.12. CUSTOM ENVIRONMENT FILE FOR CONFIGURING HCI NETWORKING IN THE DIRECTOR OPERATOR

The following example is an environment file that map the network software configuration resources to the NIC templates for your overcloud.

```
resource_registry:
  OS::TripleO::Controller::Net::SoftwareConfig: net-config-multi-nic-controller.yaml
  OS::TripleO::ComputeHCI::Net::SoftwareConfig: net-config-two-nic-vlan-computehci.yaml
```



NOTE

Add any additional network configuration in a **parameter_defaults** section.

To use this template in your deployment, copy the contents of the example to **network-environment.yaml** in your **custom_environment_files** directory on your workstation.

Additional resources

- "Custom network environment file"
- "Network environment parameters

7.13. CUSTOM ENVIRONMENT FILE FOR CONFIGURING HYPER-CONVERGED INFRASTRUCTURE (HCI) STORAGE IN THE DIRECTOR OPERATOR

The following example is an environment file that contains Ceph Storage configuration for the Compute HCI nodes.

```
resource_registry:
  OS::TripleO::Services::CephMgr: deployment/ceph-ansible/ceph-mgr.yaml
  OS::TripleO::Services::CephMon: deployment/ceph-ansible/ceph-mon.yaml
  OS::TripleO::Services::CephOSD: deployment/ceph-ansible/ceph-osd.yaml
  OS::TripleO::Services::CephClient: deployment/ceph-ansible/ceph-client.yaml

parameter_defaults:
  # needed for now because of the repo used to create tripleo-deploy image
  CephAnsibleRepo: "rhelosp-ceph-4-tools"
  CephAnsiblePlaybookVerbosity: 3
  CinderEnableIscsiBackend: false
  CinderEnableRbdBackend: true
  CinderBackupBackend: ceph
  CinderEnableNfsBackend: false
  NovaEnableRbdBackend: true
  GlanceBackend: rbd
  CinderRbdPoolName: "volumes"
  NovaRbdPoolName: "vms"
  GlanceRbdPoolName: "images"
  CephPoolDefaultPgNum: 32
  CephPoolDefaultSize: 2
  CephAnsibleDisksConfig:
    devices:
      - '/dev/sdb'
      - '/dev/sdc'
      - '/dev/sdd'
    osd_scenario: lvm
    osd_objectstore: bluestore
  CephAnsibleExtraConfig:
    is_hci: true
  CephConfigOverrides:
    rgw_swift_enforce_content_length: true
    rgw_swift_versioning_enabled: true
```

This configuration maps the OSD nodes to the **sdb**, **sdc**, and **sdd** devices and enables HCI with the **is_hci** option.

NOTE

You can modify this configuration to suit the storage configuration of your bare metal nodes. Use the "Ceph Placement Groups (PGs) per Pool Calculator" to determine the value for the **CephPoolDefaultPgNum** parameter.

To use this template in your deployment, copy the contents of the example to **compute-hci.yaml** in your **custom_environment_files** directory on your workstation.

Additional resources

- "Configuring resource isolation on hyperconverged nodes"

7.14. ADDING CUSTOM ENVIRONMENT FILES TO THE OVERCLOUD CONFIGURATION

Upload a set of custom environment files from a directory to a ConfigMap that you can include as a part of your overcloud deployment.

Prerequisites

- Ensure your OpenShift Container Platform cluster is operational and you have installed the director Operator correctly.
- Ensure that you have installed the **oc** command line tool on your workstation.
- Create custom environment files for your overcloud deployment.

Procedure

1. Create the **heat-env-config** ConfigMap and use the directory that contains the environment files as data:

```
$ oc create configmap -n openstack heat-env-config --from-file=~/custom_environment_files/
--dry-run=client -o yaml | oc apply -f -
```

Verification

- View the ConfigMap:

```
$ oc get configmap/heat-env-config -n openstack
```

Additional resources

- "Creating and using config maps"
- "Environment files"

7.15. CREATING HCI COMPUTE NODES WITH OPENSTACKBAREMETALSET

Compute nodes provide computing resources to your Red Hat OpenStack Platform environment. You must have at least one Compute node in your overcloud and you can scale the number of Compute nodes after deployment.

The OpenStackBaremetalSet custom resource creates Compute nodes from bare metal machines that OpenShift Container Platform manages.

Prerequisites

- Ensure your OpenShift Container Platform cluster is operational and you have installed the director Operator correctly.
- Ensure that you have installed the **oc** command line tool on your workstation.
- Use the OpenStackNet resource to create a control plane network and any additional isolated networks.

Procedure

1. Create a file named **openstack-hcicompute.yaml** on your workstation. Include the resource specification for the Compute nodes. For example, the specification for 1 Compute node is as follows:

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackBaremetalSet
metadata:
  name: computehci
  namespace: openstack
spec:
  replicas: 3
  baseImageUrl: http://host/images/rhel-image-8.4.x86_64.qcow2
  deploymentSSHSecret: osp-controlplane-ssh-keys
  ctlplaneInterface: enp8s0
  networks:
    - ctlplane
    - internalapi
    - tenant
    - storage
    - storagemgmt
  roleName: ComputeHCI
  passwordSecret: userpassword
```

Set the following values in the resource specification:

metadata.name

Set to the name of the Compute node bare metal set, which is **overcloud**.

metadata.namespace

Set to the director Operator namespace, which is **openstack**.

spec

Set the configuration for the Compute nodes. For descriptions of the values you can use in this section, view the specification schema in the custom resource definition for the **openstackbaremetalset** CRD:

```
$ oc describe crd openstackbaremetalset
```

Save the file when you have finished configuring the Compute node specification.

2. Create the Compute nodes:

```
$ oc create -f openstack-hcicompute.yaml -n openstack
```

Verification

1. View the resource for the Compute HCI nodes:

```
$ oc get openstackbaremetalset/computehci -n openstack
```

2. View the bare metal machines that OpenShift manages to verify the creation of the Compute HCI nodes:

```
$ oc get baremetalhosts -n openshift-machine-api
```

-

7.16. CREATING ANSIBLE PLAYBOOKS FOR OVERCLOUD CONFIGURATION WITH OPENSTACKPLAYBOOKGENERATOR

After you provision the overcloud infrastructure, you must create a set of Ansible playbooks to configure the Red Hat OpenStack Platform (RHOSP) software on the overcloud nodes. You create these playbooks with the OpenStackPlaybookGenerator resource, which uses the **config-download** feature in RHOSP director to convert heat configuration to playbooks.

Prerequisites

- Ensure your OpenShift Container Platform cluster is operational and you have installed the director Operator correctly.
- Ensure that you have installed the **oc** command line tool on your workstation.
- Configure a **git-secret** Secret that contains authentication details for your remote Git repository.
- Configure a **tripleo-tarball-config** ConfigMap that contains your custom heat templates.
- Configure a **heat-env-config** ConfigMap that contains your custom environment files.

Procedure

1. Create a file named **openstack-playbook-generator.yaml** on your workstation. Include the resource specification to generate the Ansible playbooks. For example, the specification to generate the playbooks is as follows:

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackPlaybookGenerator
metadata:
  name: default
  namespace: openstack
spec:
  imageURL: registry.redhat.io/rhosp-beta/openstack-tripleoclient:16.2
  gitSecret: git-secret
  heatEnvConfigMap: heat-env-config
  tarballConfigMap: tripleo-tarball-config
```

   Set the following values in the resource specification:

   metadata.name

   Set to the name of the Compute node bare metal set, which is **default**.

   metadata.namespace

   Set to the director Operator namespace, which is **openstack**.

   spec.gitSecret

   Set to the ConfigMap that contains the Git authentication credentials, which is **git-secret**.

   spec.tarballConfigMap

   Set to the ConfigMap that contains the tarball with your custom heat templates, which is **tripleo-tarball-config**.

   spec.heatEnvConfigMap

Set to the ConfigMap that contains your custom environment files, which is **heat-env-config**.

For more descriptions of the values you can use in the **spec** section, view the specification schema in the custom resource definition for the **openstackplaybookgenerator** CRD:

```
$ oc describe crd openstackplaybookgenerator
```

Save the file when you have finished configuring the Ansible playbook generator specification.

2. Create the Ansible playbook generator:

```
$ oc create -f openstack-playbook-generator.yaml -n openstack
```

Verification

- View the resource for the playbook generator:

```
$ oc get openstackplaybookgenerator/default -n openstack
```

7.17. REGISTERING THE OPERATING SYSTEM OF YOUR OVERCLOUD

Before the director Operator configures the overcloud software on nodes, you must register the operating system of all nodes to either the Red Hat Customer Portal or Red Hat Satellite Server, and enable repositories for your nodes. To register your nodes, you can use the **redhat_subscription** Ansible module with the inventory file that the OpentackPlaybookGenerator resource creates.

Prerequisites

- Ensure your OpenShift Container Platform cluster is operational and you have installed the director Operator correctly.
- Ensure that you have installed the **oc** command line tool on your workstation.
- Use the OpenStackControlPlane resource to create a control plane.
- Use the OpenStackBareMetalSet resource to create bare metal Compute nodes.
- Use the OpentackPlaybookGenerator to create the Ansible playbook configuration for your overcloud.

Procedure

1. Access the remote shell for **openstackclient**:

```
$ oc rsh openstackclient -n openstack
```

2. Change to the **cloud-admin** home directory:

```
$ cd /home/cloud-admin
```

3. Optional: Check the diff for the overcloud Ansible playbooks:

```
$ ./tripleo-deploy.sh -d
```

4. Accept the newest version of the rendered Ansible playbooks and tag them as **latest**:

```
$ ./tripleo-deploy.sh -a
```

The newest version of the playbooks includes an inventory file for the overcloud configuration. You can find this inventory file on the OpenstackClient pod at **/home/cloud-admin/playbooks/tripleo-ansible/inventory.yaml**.

5. Create a playbook that uses the **redhat_subscription** modules to register your nodes. For example, the following playbook registers Controller nodes:

```
---
- name: Register Controller nodes
  hosts: Controller
  become: yes
  vars:
    repos:
      - rhel-8-for-x86_64-baseos-eus-rpms
      - rhel-8-for-x86_64-appstream-eus-rpms
      - rhel-8-for-x86_64-highavailability-eus-rpms
      - ansible-2.9-for-rhel-8-x86_64-rpms
      - advanced-virt-for-rhel-8-x86_64-rpms
      - openstack-16.2-for-rhel-8-x86_64-rpms
      - fast-datapath-for-rhel-8-x86_64-rpms
  tasks:
    - name: Register system
      redhat_subscription:
        username: myusername
        password: p@55w0rd!
        org_id: 1234567
        release: 8.4
        pool_ids: 1a85f9223e3d5e43013e3d6e8ff506fd
    - name: Disable all repos
      command: "subscription-manager repos --disable *"
    - name: Enable Controller node repos
      command: "subscription-manager repos --enable {{ item }}"
      with_items: "{{ repos }}"
```

This play contains the following three tasks:

- Register the node.
- Disable any auto-enabled repositories.
- Enable only the repositories relevant to the Controller node. The repositories are listed with the **repos** variable.

6. Register the overcloud nodes to required repositories:

```
ansible-playpook -i /home/cloud-admin/playbooks/tripleo-ansible/inventory.yaml ./rhsm.yaml
```

Additional resources

- "redhat_subscription – Manage registration and subscriptions to RHSM using the subscription-manager command"
- "Running Ansible-based registration manually"

7.18. APPLYING OVERCLOUD CONFIGURATION WITH THE DIRECTOR OPERATOR

You can configure the overcloud with director Operator only after you have created your control plane, provisioned your bare metal Compute nodes, and generated the Ansible playbooks to configure software on each node. When you create an OpenStackControlPlane resource, the director Operator creates an OpenStackClient pod that you access through a remote shell and run the **tripleo-deploy.sh** script to configure the overcloud.

Prerequisites

- Ensure your OpenShift Container Platform cluster is operational and you have installed the director Operator correctly.
- Ensure that you have installed the **oc** command line tool on your workstation.
- Use the OpenStackControlPlane resource to create a control plane.
- Use the OpenStackBareMetalSet resource to create bare metal Compute nodes.
- Use the OpentackPlaybookGenerator to create the Ansible playbook configuration for your overcloud.

Procedure

1. Access the remote shell for **openstackclient**:

```
$ oc rsh openstackclient -n openstack
```

2. Change to the **cloud-admin** home directory:

```
$ cd /home/cloud-admin
```

3. Optional: Check the diff for the overcloud Ansible playbooks:

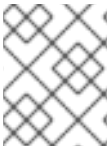
```
$ ./tripleo-deploy.sh -d
```

4. Accept the newest version of the rendered Ansible playbooks and tag them as **latest**:

```
$ ./tripleo-deploy.sh -a
```

5. Apply the Ansible playbooks against the overcloud nodes:

```
$ ./tripleo-deploy.sh -p
```



NOTE

You can view a log of the Ansible configuration at ~/**ansible.log** within the **openstackclient** pod.

CHAPTER 8. DIRECTOR OPERATOR DEPLOYMENT SCENARIO: OVERCLOUD WITH EXTERNAL CEPH STORAGE

You can use the director Operator to deploy an overcloud that connects to an external Red Hat Ceph Storage cluster.

Prerequisites

- An external Red Hat Ceph Storage cluster.

8.1. CREATING A DATA VOLUME FOR THE BASE OPERATING SYSTEM

You must create a data volume with the OpenShift Container Platform (OCP) cluster to store the base operating system image for your Controller virtual machines.

Prerequisites

- Download a Red Hat Enterprise Linux 8 QCOW2 image to your workstation.You can download this image from the Product Download section of the Red Hat Customer Portal.
- Install the **virtctl** client tool on your workstation. You can install this tool on a Red Hat Enterprise Linux workstation using the following commands:

```
$ sudo subscription-manager repos --enable=cnv-2.6-for-rhel-8-x86_64-rpms
$ sudo dnf install -y kubevirt-virtctl
```

- Install the **virt-customize** client tool on your workstation. You can install this tool on a Red Hat Enterprise Linux workstation using the following command:

```
$ dnf install -y libguestfs-tools-c
```

Procedure

1. The default QCOW2 image that you have downloaded from access.redhat.com does not use biosdev predictable network interface names. Modify the image with **virt-customize** to use biosdev predictable network interface names:

```
$ sudo virt-customize -a <local path to image> --run-command 'sed -i -e "s/^\
(kernelopts=.*\)net.ifnames=0 \(.*\)/\1\2/" /boot/grub2/grubenv'
$ sudo virt-customize -a <local path to image> --run-command 'sed -i -e "s/^\
(GRUB_CMDLINE_LINUX=.*\)net.ifnames=0 \(.*\)/\1\2/" /etc/default/grub'
```

2. Upload the image to OpenShift Virtualization with **virtctl**:

```
$ virtctl image-upload dv openstack-base-img -n openstack --size=50Gi --image-path=<local
path to image> --storage-class <storage class> --insecure
```

For the **--storage-class** option, choose a storage class from your cluster. View a list of storage classes with the following command:

```
$ oc get storageclass
```

3. When you create the OpenStackControlPlane resource and individual OpenStackVmSet resources, set the **baseImageVolumeName** parameter to the data volume name:

```
...
spec:
  ...
  baseImageVolumeName: openstack-base-img
...
```

Additional resources

- "Uploading local disk images by using the virtctl tool"

8.2. ADDING AUTHENTICATION DETAILS FOR YOUR REMOTE GIT REPOSITORY

The director Operator stores rendered Ansible playbooks to a remote Git repository and uses this repository to track changes to the overcloud configuration. You can use any Git repository that supports SSH authentication. You must provide details for the Git repository as an OpenShift Secret resource named **git-secret**.

Prerequisites

- Ensure your OpenShift Container Platform cluster is operational and you have installed the director Operator correctly.
- Ensure that you have installed the **oc** command line tool on your workstation.
- Prepare a remote Git repository for the director Operator to store the generated configuration for your overcloud.
- Prepare an SSH key pair. Upload the public key to the Git repository and keep the private key available to add to the **git-secret** Secret resource.

Procedure

- Create the Secret resource:

```
$ oc create secret generic git-secret -n openstack --from-file=git_ssh_identity=
<path_to_private_SSH_key> --from-literal=git_url=<git_server_URL>
```

The **git-secret** Secret resource contains two key-value pairs:

git_ssh_identity

The private key to access the Git repository. The **--from-file** option stores the content of the private SSH key file.

git_url

The SSH URL of the git repository to store the configuration. The **--from-literal** option stores the URL that you enter for this key.

Verification

- View the Secret resource:

```
$ oc get secret/git-secret -n openstack
```

Additional resources

- "Providing sensitive data to pods"

8.3. SETTING THE ROOT PASSWORD FOR NODES

To access the **root** user with a password on each node, you can set a **root** password in a Secret resource named **userpassword**.

NOTE

Setting the root password for nodes is optional. If you do not set a **root** password, you can still log into nodes with the SSH keys defined in the **osp-controlplane-ssh-keys** Secret.

Prerequisites

- Ensure your OpenShift Container Platform cluster is operational and you have installed the director Operator correctly.
- Ensure that you have installed the **oc** command line tool on your workstation.

Procedure

1. Convert your chosen password to a base64 value:

```
$ echo -n "p@ssw0rd!" | base64
cEBzc3cwcmQh
```

NOTE

The **-n** option removes the trailing newline from the echo output.

2. Create a file named **openstack-userpassword.yaml** on your workstation. Include the following resource specification for the Secret in the file:

```
apiVersion: v1
kind: Secret
metadata:
  name: userpassword
  namespace: openstack
data:
  NodeRootPassword: "cEBzc3cwcmQh"
```

   Set the **NodeRootPassword** parameter to your base64 encoded password.

3. Create the **userpassword** Secret:

```
$ oc create -f openstack-userpassword.yaml -n openstack
```

Additional resources

- "Providing sensitive data to pods"

8.4. CREATING AN OVERCLOUD CONTROL PLANE NETWORK WITH OPENSTACKNET

You must create one control plane network for your overcloud. In addition to IP address assignment, the OpenStackNet resource includes information to define the network configuration policy that OpenShift Virtualization uses to attach any virtual machines to the network.

Prerequisites

- Ensure your OpenShift Container Platform cluster is operational and you have installed the director Operator correctly.
- Ensure that you have installed the **oc** command line tool on your workstation.

Procedure

1. Create a file named **ctlplane-network.yaml** on your workstation. Include the resource specification for the control plane network, which is named **ctlplane**. For example, the specification for a control plane that uses a Linux bridge connected to the **enp6s0** Ethernet device on each worker node is as follows:

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackNet
metadata:
  name: ctlplane
spec:
  cidr: 192.168.25.0/24
  allocationStart: 192.168.25.100
  allocationEnd: 192.168.25.250
  gateway: 192.168.25.1
  attachConfiguration:
    nodeNetworkConfigurationPolicy:
      nodeSelector:
        node-role.kubernetes.io/worker: ""
      desiredState:
        interfaces:
        - bridge:
            options:
              stp:
                enabled: false
            port:
            - name: enp6s0
          description: Linux bridge with enp6s0 as a port
          name: br-osp
          state: up
          type: linux-bridge
```

Set the following values in the resource specification:

metadata.name

Set to the name of the control plane network, which is **ctlplane**.

spec

Set the network configuration for the control plane network. For descriptions of the values you can use in this section, view the specification schema in the custom resource definition for the **openstacknet** CRD:

```
$ oc describe crd openstacknet
```

Save the file when you have finished configuring the network specification.

2. Create the control plane network:

```
$ oc create -f ctlplane-network.yaml -n openstack
```

Verification

- View the resource for the control plane network:

```
$ oc get openstacknet/ctlplane
```

8.5. CREATING VLAN NETWORKS FOR NETWORK ISOLATION WITH OPENSTACKNET

You must create additional networks to implement network isolation for your composable networks. To accomplish this network isolation, you can place your composable networks networks on individual VLAN networks. In addition to IP address assignment, the OpenStackNet resource includes information to define the network configuration policy that OpenShift Virtualization uses to attach any virtual machines to VLAN networks.

To use the default Red Hat OpenStack Platform networks, you must create an OpenStackNet resource for each network.

Table 8.1. Default Red Hat OpenStack Platform networks

Network	VLAN	CIDR	Allocation
External	10	10.0.0.0/24	10.0.0.4 - 10.0.0.250
InternalApi	20	172.16.2.0/24	172.16.2.4 - 172.16.2.250
Storage	30	172.16.1.0/24	172.16.1.4 - 172.16.1.250
StorageMgmt	40	172.16.3.0/24	172.16.3.4 - 172.16.3.250
Tenant	50	172.16.0.0/24	172.16.0.4 - 172.16.0.250

IMPORTANT

To use different networking details for each network, you must create a custom **network_data.yaml** file.

Prerequisites

- Ensure your OpenShift Container Platform cluster is operational and you have installed the director Operator correctly.
- Ensure that you have installed the **oc** command line tool on your workstation.

Procedure

1. Create a file for your new network. For example, for the internal API network, create a file named **internalapi-network.yaml** on your workstation. Include the resource specification for the VLAN network. For example, the specification for an internal API network that manages VLAN-tagged traffic over a Linux bridge connected to the **enp6s0** Ethernet device on each worker node is as follows:

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackNet
metadata:
  name: internalapi
spec:
  cidr: 172.16.2.0/24
  vlan: 20
  allocationStart: 172.16.2.4
  allocationEnd: 172.16.2.250
  attachConfiguration:
    nodeNetworkConfigurationPolicy:
      nodeSelector:
        node-role.kubernetes.io/worker: ""
      desiredState:
        interfaces:
        - bridge:
            options:
              stp:
                enabled: false
            port:
            - name: enp6s0
          description: Linux bridge with enp7s0 as a port
          name: br-osp
          state: up
          type: linux-bridge
```

When you use a VLAN for network isolation with **linux-bridge** the following happens:

- The director Operator creates a Node Network Configuration Policy for the bridge interface specified in the resource, which uses **nmstate** to configure the bridge on worker nodes.
- The director Operator creates a Network Attach Definition for each network, which defines the Multus CNI plugin configuration. When you specify the VLAN ID on the Network Attach Definition, the Multus CNI plugin enables **vlan-filtering** on the bridge.
- The director Operator attaches a dedicated interface for each network on a virtual machine. This means the network template for the **OSVMSet** is a multi-NIC network template.

Set the following values in the resource specification:

metadata.name

Set to the name of the control plane network, which is **ctlplane**.

spec

Set the network configuration for the control plane network. For descriptions of the values you can use in this section, view the specification schema in the custom resource definition for the **openstacknet** CRD:

```
$ oc describe crd openstacknet
```

Save the file when you have finished configuring the network specification.

2. Create the internal API network:

```
$ oc create -f internalapi-network.yaml -n openstack
```

Verification

- View the resource for the internal API network:

```
$ oc get openstacknet/internalapi -n openstack
```

8.6. CREATING A CONTROL PLANE WITH OPENSTACKCONTROLPLANE

The overcloud control plane contains the main Red Hat OpenStack Platform services that manage overcloud functionality. The control plane usually consists of 3 Controller nodes and can scale to other control plane-based composable roles. When you use composable roles, each service must run on exactly 3 additional dedicated nodes and the total number of nodes in the control plane must be odd to maintain Pacemaker quorum.

The OpenStackControlPlane custom resource creates control plane-based nodes as virtual machines within OpenShift Virtualization.

Prerequisites

- Ensure your OpenShift Container Platform cluster is operational and you have installed the director Operator correctly.
- Ensure that you have installed the **oc** command line tool on your workstation.
- Use the OpenStackNet resource to create a control plane network and any additional isolated networks.

Procedure

1. Create a file named **openstack-controller.yaml** on your workstation. Include the resource specification for the Controller nodes. For example, the specification for a control plane that consists of 3 Controller nodes is as follows:

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackControlPlane
metadata:
  name: overcloud
  namespace: openstack
```

```
spec:
  openStackClientImageURL: registry.redhat.io/rhosp-beta/openstack-tripleoclient:16.2
  openStackClientNetworks:
        - ctlplane
        - external
        - internalapi
  openStackClientStorageClass: host-nfs-storageclass
  passwordSecret: userpassword
  gitSecret: git-secret
  virtualMachineRoles:
    controller:
      roleName: Controller
      roleCount: 3
      networks:
        - ctlplane
        - internalapi
        - external
        - tenant
        - storage
        - storagemgmt
      cores: 6
      memory: 12
      diskSize: 50
      baseImageVolumeName: openstack-base-img
      storageClass: host-nfs-storageclass
```

Set the following values in the resource specification:

metadata.name

Set to the name of the overcloud control plane, which is **overcloud**.

metadata.namespace

Set to the director Operator namespace, which is **openstack**.

spec

Set the configuration for the control plane. For descriptions of the values you can use in this section, view the specification schema in the custom resource definition for the **openstackcontrolplane** CRD:

```
$ oc describe crd openstackcontrolplane
```

Save the file when you have finished configuring the control plane specification.

2. Create the control plane:

```
$ oc create -f openstack-controller.yaml -n openstack
```

Wait until OCP creates the resources related to OpenStackControlPlane resource.

As a part of the OpenStackControlPlane resource, the director Operator also creates an OpenStackClient pod that you can access through a remote shell and run RHOSP commands.

Verification

1. View the resource for the control plane:

```
$ oc get openstackcontrolplane/overcloud -n openstack
```

2. View the OpenStackVMSet resources to verify the creation of the control plane virtual machine set:

```
$ oc get openstackvmsets -n openstack
```

3. View the virtual machine resources to verify the creation of the control plane virtual machines in OpenShift Virtualization:

```
$ oc get virtualmachines
```

4. Test access to the **openstackclient** remote shell:

```
$ oc rsh openstackclient -n openstack
```

8.7. CREATING DIRECTORIES FOR TEMPLATES AND ENVIRONMENT FILES

Create directories on your workstation to store your custom templates and environment files, which you upload to ConfigMaps in OpenShift Container Platform (OCP).

Procedure

1. Create a directory for your custom templates:

```
$ mkdir custom_templates
```

2. Create a directory for your custom environment files:

```
$ mkdir custom_environment_files
```

8.8. CUSTOM NIC HEAT TEMPLATE FOR CONTROLLER NODES

The following example is a heat template that contains NIC configuration for the Controller virtual machines.

```
heat_template_version: rocky
description: >
  Software Config to drive os-net-config to configure multiple interfaces for the Controller role.
parameters:
  ControlPlaneIp:
    default: ''
    description: IP address/subnet on the ctlplane network
    type: string
  ControlPlaneSubnetCidr:
    default: ''
    description: >
      The subnet CIDR of the control plane network. (The parameter is
      automatically resolved from the ctlplane subnet's cidr attribute.)
    type: string
```

```
  ControlPlaneDefaultRoute:
    default: ''
    description: The default route of the control plane network. (The parameter
      is automatically resolved from the ctlplane subnet's gateway_ip attribute.)
    type: string
  ControlPlaneStaticRoutes:
    default: []
    description: >
      Routes for the ctlplane network traffic.
      JSON route e.g. [{'destination':'10.0.0.0/16', 'nexthop':'10.0.0.1'}]
      Unless the default is changed, the parameter is automatically resolved
      from the subnet host_routes attribute.
    type: json
  ControlPlaneMtu:
    default: 1500
    description: The maximum transmission unit (MTU) size(in bytes) that is
      guaranteed to pass through the data path of the segments in the network.
      (The parameter is automatically resolved from the ctlplane network's mtu attribute.)
    type: number
  StorageIpSubnet:
    default: ''
    description: IP address/subnet on the storage network
    type: string
  StorageNetworkVlanID:
    default: 30
    description: Vlan ID for the storage network traffic.
    type: number
  StorageMtu:
    default: 1500
    description: The maximum transmission unit (MTU) size(in bytes) that is
      guaranteed to pass through the data path of the segments in the
      Storage network.
    type: number
  StorageInterfaceRoutes:
    default: []
    description: >
      Routes for the storage network traffic.
      JSON route e.g. [{'destination':'10.0.0.0/16', 'nexthop':'10.0.0.1'}]
      Unless the default is changed, the parameter is automatically resolved
      from the subnet host_routes attribute.
    type: json
  StorageMgmtIpSubnet:
    default: ''
    description: IP address/subnet on the storage_mgmt network
    type: string
  StorageMgmtNetworkVlanID:
    default: 40
    description: Vlan ID for the storage_mgmt network traffic.
    type: number
  StorageMgmtMtu:
    default: 1500
    description: The maximum transmission unit (MTU) size(in bytes) that is
      guaranteed to pass through the data path of the segments in the
      StorageMgmt network.
    type: number
  StorageMgmtInterfaceRoutes:
```

```
  default: []
  description: >
    Routes for the storage_mgmt network traffic.
    JSON route e.g. [{'destination':'10.0.0.0/16', 'nexthop':'10.0.0.1'}]
    Unless the default is changed, the parameter is automatically resolved
    from the subnet host_routes attribute.
  type: json
InternalApiIpSubnet:
  default: ''
  description: IP address/subnet on the internal_api network
  type: string
InternalApiNetworkVlanID:
  default: 20
  description: Vlan ID for the internal_api network traffic.
  type: number
InternalApiMtu:
  default: 1500
  description: The maximum transmission unit (MTU) size(in bytes) that is
    guaranteed to pass through the data path of the segments in the
    InternalApi network.
  type: number
InternalApiInterfaceRoutes:
  default: []
  description: >in your `custom_templates` directory.
    Routes for the internal_api network traffic.
    JSON route e.g. [{'destination':'10.0.0.0/16', 'nexthop':'10.0.0.1'}]
    Unless the default is changed, the parameter is automatically resolved
    from the subnet host_routes attribute.
  type: json
TenantIpSubnet:
  default: ''
  description: IP address/subnet on the tenant network
  type: string
TenantNetworkVlanID:
  default: 50
  description: Vlan ID for the tenant network traffic.
  type: number
TenantMtu:
  default: 1500
  description: The maximum transmission unit (MTU) size(in bytes) that is
    guaranteed to pass through the data path of the segments in the
    Tenant network.
  type: number
TenantInterfaceRoutes:
  default: []
  description: >
    Routes for the tenant network traffic.
    JSON route e.g. [{'destination':'10.0.0.0/16', 'nexthop':'10.0.0.1'}]
    Unless the default is changed, the parameter is automatically resolved
    from the subnet host_routes attribute.
  type: json
ExternalIpSubnet:
  default: ''
  description: IP address/subnet on the external network
  type: string
ExternalNetworkVlanID:
```

```
    default: 10
    description: Vlan ID for the external network traffic.
    type: number
  ExternalMtu:
    default: 1500
    description: The maximum transmission unit (MTU) size(in bytes) that is
      guaranteed to pass through the data path of the segments in the
      External network.
    type: number
  ExternalInterfaceDefaultRoute:
    default: ''
    description: default route for the external network
    type: string
  ExternalInterfaceRoutes:
    default: []
    description: >
      Routes for the external network traffic.
      JSON route e.g. [{'destination':'10.0.0.0/16', 'nexthop':'10.0.0.1'}]
      Unless the default is changed, the parameter is automatically resolved
      from the subnet host_routes attribute.
    type: json
  DnsServers: # Override this via parameter_defaults
    default: []
    description: >
      DNS servers to use for the Overcloud (2 max for some implementations).
      If not set the nameservers configured in the ctlplane subnet's
      dns_nameservers attribute will be used.
    type: comma_delimited_list
  DnsSearchDomains: # Override this via parameter_defaults
    default: []
    description: A list of DNS search domains to be added (in order) to resolv.conf.
    type: comma_delimited_list
resources:
  OsNetConfigImpl:
    type: OS::Heat::SoftwareConfig
    properties:
      group: script
      config:
        str_replace:
          template:
            get_file: /usr/share/openstack-tripleo-heat-templates/network/scripts/run-os-net-config.sh
          params:
            $network_config:
              network_config:

              ## NIC2 - Control Plane ##
              - type: interface
                name: nic2
                mtu:
                  get_param: ControlPlaneMtu
                use_dhcp: false
                dns_servers:
                  get_param: DnsServers
                domain:
                  get_param: DnsSearchDomains
                addresses:
```

```
    - ip_netmask:
        list_join:
        - /
        - - get_param: ControlPlaneIp
          - get_param: ControlPlaneSubnetCidr
    routes:
      list_concat_unique:
        - get_param: ControlPlaneStaticRoutes

  ## NIC3 - External ##
  - type: ovs_bridge
    name: bridge_name
    mtu:
      get_param: ExternalMtu
    dns_servers:
      get_param: DnsServers
    use_dhcp: false
    addresses:
    - ip_netmask:
        get_param: ExternalIpSubnet
    routes:
      list_concat_unique:
        - get_param: ExternalInterfaceRoutes
        - - default: true
            next_hop:
              get_param: ExternalInterfaceDefaultRoute
    members:
    - type: interface
      name: nic3
      mtu:
        get_param: ExternalMtu
      use_dhcp: false
      primary: true

  ## NIC4 - Internal API ##
  - type: interface
    name: nic4
    mtu:
      get_param: InternalApiMtu
    use_dhcp: false
    addresses:
    - ip_netmask:
        get_param: InternalApiIpSubnet
    routes:
      list_concat_unique:
        - get_param: InternalApiInterfaceRoutes

  ## NIC5 - Storage ##
  - type: interface
    name: nic5
    mtu:
      get_param: StorageMtu
    use_dhcp: false
    addresses:
    - ip_netmask:
        get_param: StorageIpSubnet
```

```
            routes:
              list_concat_unique:
                - get_param: StorageInterfaceRoutes

          ## NIC6 - StorageMgmt ##
          - type: interface
            name: nic6
            mtu:
              get_param: StorageMgmtMtu
            use_dhcp: false
            addresses:
            - ip_netmask:
                get_param: StorageMgmtIpSubnet
            routes:
              list_concat_unique:
                - get_param: StorageMgmtInterfaceRoutes

          ## NIC7 - Tenant ##
          - type: ovs_bridge
            name: br-isolated
            mtu:
              get_param: TenantMtu
            dns_servers:
              get_param: DnsServers
            use_dhcp: false
            addresses:
            - ip_netmask:
                get_param: TenantIpSubnet
            routes:
              list_concat_unique:
                - get_param: TenantInterfaceRoutes
            members:
            - type: interface
              name: nic7
              mtu:
                get_param: TenantMtu
              use_dhcp: false
              primary: true

outputs:
  OS::stack_id:
    description: The OsNetConfigImpl resource.
    value:
      get_resource: OsNetConfigImpl
```

This configuration maps the the networks to the following bridges and interfaces:

Networks	Bridge	interface
Control Plane		**nic2**
External	**br-ex**	**nic3**
Internalapi		**nic4**

Networks	Bridge	interface
Storage		**nic5**
StorageMgmt		**nic6**
Tenant	**br-isolated**	**nic7**

To use this template in your deployment, copy the contents of the example to **net-config-multi-nic-controller.yaml** in your **custom_templates** directory on your workstation.

8.9. CUSTOM NIC HEAT TEMPLATE FOR COMPUTE NODES

The following example is a heat template that contains NIC configuration for the Compute bare metal nodes.

```
heat_template_version: rocky
description: >
  Software Config to drive os-net-config to configure VLANs for the Compute role.
parameters:
  ControlPlaneIp:
    default: ''
    description: IP address/subnet on the ctlplane network
    type: string
  ControlPlaneSubnetCidr:
    default: ''
    description: >
      The subnet CIDR of the control plane network. (The parameter is
      automatically resolved from the ctlplane subnet's cidr attribute.)
    type: string
  ControlPlaneDefaultRoute:
    default: ''
    description: The default route of the control plane network. (The parameter
      is automatically resolved from the ctlplane subnet's gateway_ip attribute.)
    type: string
  ControlPlaneStaticRoutes:
    default: []
    description: >
      Routes for the ctlplane network traffic.
      JSON route e.g. [{'destination':'10.0.0.0/16', 'nexthop':'10.0.0.1'}]
      Unless the default is changed, the parameter is automatically resolved
      from the subnet host_routes attribute.
    type: json
  ControlPlaneMtu:
    default: 1500
    description: The maximum transmission unit (MTU) size(in bytes) that is
      guaranteed to pass through the data path of the segments in the network.
      (The parameter is automatically resolved from the ctlplane network's mtu attribute.)
    type: number
  StorageIpSubnet:
    default: ''
    description: IP address/subnet on the storage network
```

```
    type: string
  StorageNetworkVlanID:
    default: 30
    description: Vlan ID for the storage network traffic.
    type: number
  StorageMtu:
    default: 1500
    description: The maximum transmission unit (MTU) size(in bytes) that is
      guaranteed to pass through the data path of the segments in the
      Storage network.
    type: number
  StorageInterfaceRoutes:
    default: []
    description: >
      Routes for the storage network traffic.
      JSON route e.g. [{'destination':'10.0.0.0/16', 'nexthop':'10.0.0.1'}]
      Unless the default is changed, the parameter is automatically resolved
      from the subnet host_routes attribute.
    type: json
  InternalApiIpSubnet:
    default: ''
    description: IP address/subnet on the internal_api network
    type: string
  InternalApiNetworkVlanID:
    default: 20
    description: Vlan ID for the internal_api network traffic.
    type: number
  InternalApiMtu:
    default: 1500
    description: The maximum transmission unit (MTU) size(in bytes) that is
      guaranteed to pass through the data path of the segments in the
      InternalApi network.
    type: number
  InternalApiInterfaceRoutes:
    default: []
    description: >
      Routes for the internal_api network traffic.
      JSON route e.g. [{'destination':'10.0.0.0/16', 'nexthop':'10.0.0.1'}]
      Unless the default is changed, the parameter is automatically resolved
      from the subnet host_routes attribute.
    type: json
  TenantIpSubnet:
    default: ''
    description: IP address/subnet on the tenant network
    type: string
  TenantNetworkVlanID:
    default: 50
    description: Vlan ID for the tenant network traffic.
    type: number
  TenantMtu:
    default: 1500
    description: The maximum transmission unit (MTU) size(in bytes) that is
      guaranteed to pass through the data path of the segments in the
      Tenant network.
    type: number
  TenantInterfaceRoutes:
```

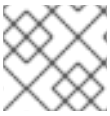
```
    default: []
    description: >
      Routes for the tenant network traffic.
      JSON route e.g. [{'destination':'10.0.0.0/16', 'nexthop':'10.0.0.1'}]
      Unless the default is changed, the parameter is automatically resolved
      from the subnet host_routes attribute.
    type: json
  ExternalMtu:
    default: 1500
    description: The maximum transmission unit (MTU) size(in bytes) that is
      guaranteed to pass through the data path of the segments in the
      External network.
    type: number
  DnsServers: # Override this via parameter_defaults
    default: []
    description: >
      DNS servers to use for the Overcloud (2 max for some implementations).
      If not set the nameservers configured in the ctlplane subnet's
      dns_nameservers attribute will be used.
    type: comma_delimited_list
  DnsSearchDomains: # Override this via parameter_defaults
    default: []
    description: A list of DNS search domains to be added (in order) to resolv.conf.
    type: comma_delimited_list

resources:

  MinViableMtu:
    # This resource resolves the minimum viable MTU for interfaces, bonds and
    # bridges that carry multiple VLANs. Each VLAN may have different MTU. The
    # bridge, bond or interface must have an MTU to allow the VLAN with the
    # largest MTU.
    type: OS::Heat::Value
    properties:
      type: number
      value:
        yaql:
          expression: $.data.max()
          data:
            - {get_param: ControlPlaneMtu}
            - {get_param: StorageMtu}
            - {get_param: InternalApiMtu}
            - {get_param: TenantMtu}
            - {get_param: ExternalMtu}

  OsNetConfigImpl:
    type: OS::Heat::SoftwareConfig
    properties:
      group: script
      config:
        str_replace:
          template:
            get_file: /usr/share/openstack-tripleo-heat-templates/network/scripts/run-os-net-config.sh
          params:
            $network_config:
              network_config:
```

```
- type: ovs_bridge
  name: br-isolated
  mtu:
    get_attr: [MinViableMtu, value]
  use_dhcp: false
  dns_servers:
    get_param: DnsServers
  domain:
    get_param: DnsSearchDomains
  addresses:
  - ip_netmask:
      list_join:
      - /
      - - get_param: ControlPlaneIp
        - get_param: ControlPlaneSubnetCidr
  routes:
    list_concat_unique:
      - get_param: ControlPlaneStaticRoutes
      - - default: true
          next_hop:
            get_param: ControlPlaneDefaultRoute
  members:
  - type: interface
    name: nic4
    mtu:
      get_attr: [MinViableMtu, value]
    # force the MAC address of the bridge to this interface
    primary: true
  - type: vlan
    mtu:
      get_param: StorageMtu
    vlan_id:
      get_param: StorageNetworkVlanID
    addresses:
    - ip_netmask:
        get_param: StorageIpSubnet
    routes:
      list_concat_unique:
        - get_param: StorageInterfaceRoutes
  - type: vlan
    mtu:
      get_param: InternalApiMtu
    vlan_id:
      get_param: InternalApiNetworkVlanID
    addresses:
    - ip_netmask:
        get_param: InternalApiIpSubnet
    routes:
      list_concat_unique:
        - get_param: InternalApiInterfaceRoutes
  - type: vlan
    mtu:
      get_param: TenantMtu
    vlan_id:
      get_param: TenantNetworkVlanID
    addresses:
```

```
              - ip_netmask:
                  get_param: TenantIpSubnet
                routes:
                  list_concat_unique:
                    - get_param: TenantInterfaceRoutes
            - type: ovs_bridge
              # This will default to br-ex, anything else requires specific
              # brige mapping entries for it to be used.
              name: bridge_name
              mtu:
                get_param: ExternalMtu
              use_dhcp: false
              members:
              - type: interface
                name: nic3
                mtu:
                  get_param: ExternalMtu
                use_dhcp: false
                primary: true
outputs:
  OS::stack_id:
    description: The OsNetConfigImpl resource.
    value:
      get_resource: OsNetConfigImpl
```

This configuration maps the the networks to the following bridges and interfaces:

Networks	Bridge	interface
Control Plane, Storage, Internal API, Tenant	**br-isolated**	**nic4**
External	**br-ex**	**nic3**



NOTE

You can modify this configuration to suit the NIC configuration of your bare metal nodes.

To use this template in your deployment, copy the contents of the example to **net-config-two-nic-vlan-compute.yaml** in your **custom_templates** directory on your workstation.

8.10. ADDING CUSTOM TEMPLATES TO THE OVERCLOUD CONFIGURATION

Archive your custom templates into a tarball file so that you can include these templates as a part of your overcloud deployment.

Prerequisites

- Ensure your OpenShift Container Platform cluster is operational and you have installed the director Operator correctly.

- Ensure that you have installed the **oc** command line tool on your workstation.
- Create the custom templates that you want to apply to provisioned nodes.

Procedure

1. Navigate to the location of your custom templates:

```
$ cd ~/custom_templates
```

2. Archive the templates into a tarball:

```
$ tar -cvzf custom-config.tar.gz *.yaml
```

3. Create the **tripleo-tarball-config** ConfigMap and use the tarball as data:

```
$ oc create configmap tripleo-tarball-config --from-file=custom-config.tar.gz -n openstack
```

Verification

- View the ConfigMap:

```
$ oc get configmap/tripleo-tarball-config -n openstack
```

Additional resources

- "Creating and using config maps"
- "heat templates"

8.11. CUSTOM ENVIRONMENT FILE FOR CONFIGURING NETWORKING IN THE DIRECTOR OPERATOR

The following example is an environment file that map the network software configuration resources to the NIC templates for your overcloud.

```
resource_registry:
  OS::TripleO::Controller::Net::SoftwareConfig: net-config-multi-nic-controller.yaml
  OS::TripleO::Compute::Net::SoftwareConfig: net-config-two-nic-vlan-compute.yaml
```

NOTE

Add any additional network configuration in a **parameter_defaults** section.

To use this template in your deployment, copy the contents of the example to **network-environment.yaml** in your **custom_environment_files** directory on your workstation.

Additional resources

- "Custom network environment file"

- "Network environment parameters

8.12. CUSTOM ENVIRONMENT FILE FOR CONFIGURING EXTERNAL CEPH STORAGE USAGE IN THE DIRECTOR OPERATOR

The following example is an environment file that contains configuration to integrate with an external Ceph Storage cluster.

```
resource_registry:
  OS::TripleO::Services::CephExternal: deployment/ceph-ansible/ceph-external.yaml

parameter_defaults:
  # needed for now because of the repo used to create tripleo-deploy image
  CephAnsibleRepo: "rhelosp-ceph-4-tools"
  CephAnsiblePlaybookVerbosity: 3

  NovaEnableRbdBackend: true
  GlanceBackend: rbd
  CinderEnableRbdBackend: true
  CinderBackupBackend: ceph
  CinderEnableIscsiBackend: false

  # Change the following values for your external ceph cluster
  NovaRbdPoolName: vms
  CinderRbdPoolName: volumes
  CinderBackupRbdPoolName: backups
  GlanceRbdPoolName: images
  CephClientUserName: openstack
  CephClientKey: AQDLOh1VgEp6FRAAFzT7Zw+Y9V6JJExQAsRnRQ==
  CephClusterFSID: 4b5c8c0a-ff60-454b-a1b4-9747aa737d19
  CephExternalMonHost: 172.16.1.7, 172.16.1.8, 172.16.1.9
```

This configuration contains parameters to enable the **CephExternal** and **CephClient** services on your nodes, set the pools for different RHOSP services, and sets the following parameters to integrate with your external Ceph Storage cluster:

CephClientKey

The Ceph client key of your external Ceph Storage cluster.

CephClusterFSID

The file system ID of your external Ceph Storage cluster.

CephExternalMonHost

A comma-delimited list of the IPs of all MON hosts in your external Ceph Storage cluster.

NOTE

You can modify this configuration to suit your storage configuration.

To use this template in your deployment, copy the contents of the example to **ceph-ansible-external.yaml** in your **custom_environment_files** directory on your workstation.

Additional resources

- "Integrate with an existing Ceph Storage cluster"

8.13. ADDING CUSTOM ENVIRONMENT FILES TO THE OVERCLOUD CONFIGURATION

Upload a set of custom environment files from a directory to a ConfigMap that you can include as a part of your overcloud deployment.

Prerequisites

- Ensure your OpenShift Container Platform cluster is operational and you have installed the director Operator correctly.
- Ensure that you have installed the **oc** command line tool on your workstation.
- Create custom environment files for your overcloud deployment.

Procedure

1. Create the **heat-env-config** ConfigMap and use the directory that contains the environment files as data:

```
$ oc create configmap -n openstack heat-env-config --from-file=~/custom_environment_files/
--dry-run=client -o yaml | oc apply -f -
```

Verification

- View the ConfigMap:

```
$ oc get configmap/heat-env-config -n openstack
```

Additional resources

- "Creating and using config maps"
- "Environment files"

8.14. CREATING COMPUTE NODES WITH OPENSTACKBAREMETALSET

Compute nodes provide computing resources to your Red Hat OpenStack Platform environment. You must have at least one Compute node in your overcloud and you can scale the number of Compute nodes after deployment.

The OpenStackBaremetalSet custom resource creates Compute nodes from bare metal machines that OpenShift Container Platform manages.

Prerequisites

- Ensure your OpenShift Container Platform cluster is operational and you have installed the director Operator correctly.
- Ensure that you have installed the **oc** command line tool on your workstation.

- Use the OpenStackNet resource to create a control plane network and any additional isolated networks.

Procedure

1. Create a file named **openstack-compute.yaml** on your workstation. Include the resource specification for the Compute nodes. For example, the specification for 1 Compute node is as follows:

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackBaremetalSet
metadata:
  name: compute
  namespace: openstack
spec:
  count: 1
  baseImageUrl: http://host/images/rhel-image-8.4.x86_64.qcow2
  deploymentSSHSecret: osp-controlplane-ssh-keys
  ctlplaneInterface: enp2s0
  networks:
    - ctlplane
    - internalapi
    - tenant
    - storage
  roleName: Compute
  passwordSecret: userpassword
```

   Set the following values in the resource specification:

   metadata.name

   Set to the name of the Compute node bare metal set, which is **overcloud**.

   metadata.namespace

   Set to the director Operator namespace, which is **openstack**.

   spec

   Set the configuration for the Compute nodes. For descriptions of the values you can use in this section, view the specification schema in the custom resource definition for the **openstackbaremetalset** CRD:

```
$ oc describe crd openstackbaremetalset
```

   Save the file when you have finished configuring the Compute node specification.

2. Create the Compute nodes:

```
$ oc create -f openstack-compute.yaml -n openstack
```

Verification

1. View the resource for the Compute nodes:

```
$ oc get openstackbaremetalset/compute -n openstack
```

2. View the bare metal machines that OpenShift manages to verify the creation of the Compute nodes:

```
$ oc get baremetalhosts -n openshift-machine-api
```

8.15. CREATING ANSIBLE PLAYBOOKS FOR OVERCLOUD CONFIGURATION WITH OPENSTACKPLAYBOOKGENERATOR

After you provision the overcloud infrastructure, you must create a set of Ansible playbooks to configure the Red Hat OpenStack Platform (RHOSP) software on the overcloud nodes. You create these playbooks with the OpenStackPlaybookGenerator resource, which uses the **config-download** feature in RHOSP director to convert heat configuration to playbooks.

Prerequisites

- Ensure your OpenShift Container Platform cluster is operational and you have installed the director Operator correctly.
- Ensure that you have installed the **oc** command line tool on your workstation.
- Configure a **git-secret** Secret that contains authentication details for your remote Git repository.
- Configure a **tripleo-tarball-config** ConfigMap that contains your custom heat templates.
- Configure a **heat-env-config** ConfigMap that contains your custom environment files.

Procedure

1. Create a file named **openstack-playbook-generator.yaml** on your workstation. Include the resource specification to generate the Ansible playbooks. For example, the specification to generate the playbooks is as follows:

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackPlaybookGenerator
metadata:
  name: default
  namespace: openstack
spec:
  imageURL: registry.redhat.io/rhosp-beta/openstack-tripleoclient:16.2
  gitSecret: git-secret
  heatEnvConfigMap: heat-env-config
  tarballConfigMap: tripleo-tarball-config
```

Set the following values in the resource specification:

metadata.name

Set to the name of the Compute node bare metal set, which is **default**.

metadata.namespace

Set to the director Operator namespace, which is **openstack**.

spec.gitSecret

Set to the ConfigMap that contains the Git authentication credentials, which is **git-secret**.

spec.tarballConfigMap

Set to the ConfigMap that contains the tarball with your custom heat templates, which is **tripleo-tarball-config**.

spec.heatEnvConfigMap

Set to the ConfigMap that contains your custom environment files, which is **heat-env-config**.

For more descriptions of the values you can use in the **spec** section, view the specification schema in the custom resource definition for the **openstackplaybookgenerator** CRD:

```
$ oc describe crd openstackplaybookgenerator
```

Save the file when you have finished configuring the Ansible playbook generator specification.

2. Create the Ansible playbook generator:

```
$ oc create -f openstack-playbook-generator.yaml -n openstack
```

Verification

- View the resource for the playbook generator:

```
$ oc get openstackplaybookgenerator/default -n openstack
```

8.16. REGISTERING THE OPERATING SYSTEM OF YOUR OVERCLOUD

Before the director Operator configures the overcloud software on nodes, you must register the operating system of all nodes to either the Red Hat Customer Portal or Red Hat Satellite Server, and enable repositories for your nodes. To register your nodes, you can use the **redhat_subscription** Ansible module with the inventory file that the OpentackPlaybookGenerator resource creates.

Prerequisites

- Ensure your OpenShift Container Platform cluster is operational and you have installed the director Operator correctly.
- Ensure that you have installed the **oc** command line tool on your workstation.
- Use the OpenStackControlPlane resource to create a control plane.
- Use the OpenStackBareMetalSet resource to create bare metal Compute nodes.
- Use the OpentackPlaybookGenerator to create the Ansible playbook configuration for your overcloud.

Procedure

1. Access the remote shell for **openstackclient**:

```
$ oc rsh openstackclient -n openstack
```

2. Change to the **cloud-admin** home directory:

```
-
```

```
$ cd /home/cloud-admin
```

3. Optional: Check the diff for the overcloud Ansible playbooks:

```
$ ./tripleo-deploy.sh -d
```

4. Accept the newest version of the rendered Ansible playbooks and tag them as **latest**:

```
$ ./tripleo-deploy.sh -a
```

The newest version of the playbooks includes an inventory file for the overcloud configuration. You can find this inventory file on the OpenstackClient pod at **/home/cloud-admin/playbooks/tripleo-ansible/inventory.yaml**.

5. Create a playbook that uses the **redhat_subscription** modules to register your nodes. For example, the following playbook registers Controller nodes:

```
---
- name: Register Controller nodes
  hosts: Controller
  become: yes
  vars:
    repos:
      - rhel-8-for-x86_64-baseos-eus-rpms
      - rhel-8-for-x86_64-appstream-eus-rpms
      - rhel-8-for-x86_64-highavailability-eus-rpms
      - ansible-2.9-for-rhel-8-x86_64-rpms
      - advanced-virt-for-rhel-8-x86_64-rpms
      - openstack-16.2-for-rhel-8-x86_64-rpms
      - fast-datapath-for-rhel-8-x86_64-rpms
  tasks:
    - name: Register system
      redhat_subscription:
        username: myusername
        password: p@55w0rd!
        org_id: 1234567
        release: 8.4
        pool_ids: 1a85f9223e3d5e43013e3d6e8ff506fd
    - name: Disable all repos
      command: "subscription-manager repos --disable *"
    - name: Enable Controller node repos
      command: "subscription-manager repos --enable {{ item }}"
      with_items: "{{ repos }}"
```

This play contains the following three tasks:

- Register the node.
- Disable any auto-enabled repositories.
- Enable only the repositories relevant to the Controller node. The repositories are listed with the **repos** variable.

6. Register the overcloud nodes to required repositories:

```
ansible-playpook -i /home/cloud-admin/playbooks/tripleo-ansible/inventory.yaml ./rhsm.yaml
```

Additional resources

- "redhat_subscription – Manage registration and subscriptions to RHSM using the subscription-manager command"
- "Running Ansible-based registration manually"

8.17. APPLYING OVERCLOUD CONFIGURATION WITH THE DIRECTOR OPERATOR

You can configure the overcloud with director Operator only after you have created your control plane, provisioned your bare metal Compute nodes, and generated the Ansible playbooks to configure software on each node. When you create an OpenStackControlPlane resource, the director Operator creates an OpenStackClient pod that you access through a remote shell and run the **tripleo-deploy.sh** script to configure the overcloud.

Prerequisites

- Ensure your OpenShift Container Platform cluster is operational and you have installed the director Operator correctly.
- Ensure that you have installed the **oc** command line tool on your workstation.
- Use the OpenStackControlPlane resource to create a control plane.
- Use the OpenStackBareMetalSet resource to create bare metal Compute nodes.
- Use the OpentackPlaybookGenerator to create the Ansible playbook configuration for your overcloud.

Procedure

1. Access the remote shell for **openstackclient**:

```
$ oc rsh openstackclient -n openstack
```

2. Change to the **cloud-admin** home directory:

```
$ cd /home/cloud-admin
```

3. Optional: Check the diff for the overcloud Ansible playbooks:

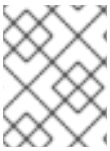
```
$ ./tripleo-deploy.sh -d
```

4. Accept the newest version of the rendered Ansible playbooks and tag them as **latest**:

```
$ ./tripleo-deploy.sh -a
```

5. Apply the Ansible playbooks against the overcloud nodes:

```
$ ./tripleo-deploy.sh -p
```



NOTE

You can view a log of the Ansible configuration at **~/ansible.log** within the **openstackclient** pod.

PART III. POST-DEPLOYMENT OPERATIONS

CHAPTER 9. ACCESSING AN OVERCLOUD DEPLOYED WITH THE DIRECTOR OPERATOR

After you deploy the overcloud with the director Operator, you can access it and run commands with the **openstack** client tool. The main access point for the overcloud is through the OpenStackClient pod that the director Operator deploys as a part of the OpenStackControlPlane resource that you create.

9.1. ACCESSING THE OPENSTACKCLIENT POD

The OpenStackClient pod is the main access point to run commands against the overcloud. This pod contains the client tools and authentication details that you require to perform actions on your overcloud. To access the pod from your workstation, you must use the **oc** command on your workstation to connect to the remote shell for the pod.

NOTE

When you access an overcloud that you deploy without the director Operator, you usually run the **source ~/overcloudrc** command to set environment variables to access the overcloud. You do not require this step with an overcloud that you deploy with the director Operator.

Prerequisites

- Ensure your OpenShift Container Platform cluster is operational and you have installed the director Operator correctly.
- Deploy and configure an overcloud that runs in your OCP cluster.
- Ensure that you have installed the **oc** command line tool on your workstation.

Procedure

1. Access the remote shell for **openstackclient**:

```
$ oc rsh openstackclient -n openstack
```

2. Change to the **cloud-admin** home directory:

```
$ cd /home/cloud-admin
```

3. Run your **openstack** commands. For example, you can create a **default** network with the following command:

```
$ openstack network create default
```

Additional resources

- "Creating basic overcloud flavors"
- "Creating a default tenant network"
- "Creating a default floating IP network"

- "Creating a default provider network"

9.2. ACCESSING THE OVERCLOUD DASHBOARD

Access the dashboard of an overcloud that you deploy with the director Operator with the same method as a standard overcloud. Access the IP address of the overcloud host name or public VIP, which you usually set with the **PublicVirtualFixedIPs** heat parameter, with a web browser and log in to the overcloud dashboard with your username and password.

Prerequisites

- Ensure your OpenShift Container Platform cluster is operational and you have installed the director Operator correctly.
- Deploy and configure an overcloud that runs in your OCP cluster.
- Ensure that you have installed the **oc** command line tool on your workstation.

Procedure

1. Optional: To login as the **admin** user, obtain the admin password from the **AdminPassword** parameter in the **tripleo-passwords** secret:

```
$ oc get secret tripleo-passwords -o jsonpath='{.data.tripleo-overcloud-passwords\.yaml}' |
base64 -d
```

2. Open your web browser.
3. Enter the host name or public VIP of the overcloud dashboard in the URL field.
4. Log in to the dashboard with your chosen username and password.

CHAPTER 10. SCALING COMPUTE NODES WITH DIRECTOR OPERATOR

In you require more or fewer compute resources for your overcloud, you can scale the number of Compute nodes according to your requirements.

10.1. ADDING COMPUTE NODES TO YOUR OVERCLOUD WITH THE DIRECTOR OPERATOR

To add more Compute nodes to your overcloud, you must increase the node count for the **compute** OpenStackBaremetalSet resource. When you increase the node count, the OpenStackPlaybookGenerator resource regenerates a new set of Ansible playbooks. Run the **tripleo-deploy.sh** script to reapply the new Ansible configuration to your overcloud

Prerequisites

- Ensure your OpenShift Container Platform cluster is operational and you have installed the director Operator correctly.
- Deploy and configure an overcloud that runs in your OCP cluster.
- Ensure that you have installed the **oc** command line tool on your workstation.
- Check that you have enough hosts in a ready state in the **openshift-machine-api** namespace. Run the **oc get baremetalhosts -n openshift-machine-api** command to check the hosts available. For more information on managing your bare metal hosts, see "Managing bare metal hosts"

Procedure

1. Modify the YAML configuration for the **compute** OpenStackBaremetalSet and increase **count** parameter for the resource:

```
$ oc patch osbms compute --type=merge --patch '{"spec":{"count":3}}' -n openstack
```

2. The OpenStackBaremetalSet resource automatically provisions new nodes with the Red Hat Enterprise Linux base operating system. Wait until the provisioning process completes. Check the nodes periodically to determine the readiness of the nodes:

```
$ oc get baremetalhosts -n openshift-machine-api
```

3. The OpenStackPlaybookGenerator resource automatically generates new Ansible playbooks for configuration. Wait until the regeneration process completes. Check the OpenStackPlaybookGenerator resource periodically to determine the readiness of the playbooks:

```
$ oc describe openstackplaybookgenerator/default -n openstack
```

4. Access the remote shell for **openstackclient**:

```
$ oc rsh openstackclient -n openstack
```

5. Change to the **cloud-admin** home directory:

```
$ cd /home/cloud-admin
```

6. Optional: Check the diff for the overcloud Ansible playbooks:

```
$ ./tripleo-deploy.sh -d
```

7. Accept the newest version of the rendered Ansible playbooks and tag them as **latest**:

```
$ ./tripleo-deploy.sh -a
```

8. Apply the Ansible playbooks against the overcloud nodes:

```
$ ./tripleo-deploy.sh -p
```

Additional resources

- "Managing bare metal hosts"

10.2. REMOVING COMPUTE NODES FROM YOUR OVERCLOUD WITH THE DIRECTOR OPERATOR

To remove a Compute node from your overcloud, you must disable the Compute node, mark it for deletion, and decrease the node count for the **compute** OpenStackBaremetalSet resource.

Prerequisites

- Ensure your OpenShift Container Platform cluster is operational and you have installed the director Operator correctly.
- Deploy and configure an overcloud that runs in your OCP cluster.
- Ensure that you have installed the **oc** command line tool on your workstation.

Procedure

1. Access the remote shell for **openstackclient**:

```
$ oc rsh openstackclient -n openstack
```

2. Identify the Compute node that you want to remove and disable the Compute service on the node to prevent the node from scheduling new instances:

```
$ openstack compute service list
$ openstack compute service set <hostname> nova-compute --disable
```

3. Exit from **openstackclient**:

```
$ exit
```

4. Annotate the BareMetalHost resource that corresponds to the node that you want to remove with the **osp-director.openstack.org/delete-host=true** annotation:

```
$ oc annotate -n openshift-machine-api bmh/openshift-worker-3 osp-
director.openstack.org/delete-host=true --overwrite
```

The **annotatedForDeletion** status changes in the OpenStackBaremetalSet resource:

```
$ oc get osbms compute -o json -n openstack | jq .status
{
  "baremetalHosts": {
    "compute-0": {
      "annotatedForDeletion": true,
      "ctlplaneIP": "192.168.25.105/24",
      "hostRef": "openshift-worker-3",
      "hostname": "compute-0",
      "networkDataSecretName": "compute-cloudinit-networkdata-openshift-worker-3",
      "provisioningState": "provisioned",
      "userDataSecretName": "compute-cloudinit-userdata-openshift-worker-3"
    },
    "compute-1": {
      "annotatedForDeletion": false,
      "ctlplaneIP": "192.168.25.106/24",
      "hostRef": "openshift-worker-4",
      "hostname": "compute-1",
      "networkDataSecretName": "compute-cloudinit-networkdata-openshift-worker-4",
      "provisioningState": "provisioned",
      "userDataSecretName": "compute-cloudinit-userdata-openshift-worker-4"
    }
  },
  "provisioningStatus": {
    "readyCount": 2,
    "reason": "All requested BaremetalHosts have been provisioned",
    "state": "provisioned"
  }
}
```

5. Modify the YAML configuration for the **compute** OpenStackBaremetalSet resource and decrease **count** parameter for the resource:

```
oc patch osbms compute --type=merge --patch '{"spec":{"count":1}}' -n openstack
```

When you reduce the resource count of the OpenStackBaremetalSet resource, you trigger the corresponding controller to handle the resource deletion, which causes the following actions:

- The director Operator deletes the corresponding OpenStackIPSet for the node
- The director Operator flags the IP reservation entry in the OpenStackNet resource as deleted

```
oc get osnet ctlplane -o json -n openstack | jq .status.roleReservations.compute
{
  "addToPredictableIPs": true,
  "reservations": [
    {
```

```
      "deleted": true,
      "hostname": "compute-0",
      "ip": "192.168.25.105",
      "vip": false
    },
    {
      "deleted": false,
      "hostname": "compute-1",
      "ip": "192.168.25.106",
      "vip": false
    }
  ]
}
```

The following consequences occur as a result of the node deletion and IP reservation changes:

- The IP is not free for another role to use.
- If you scale the overcloud with a new node in the same role, the node reuses the hostnames starting with lowest ID suffix and corresponding IP reservation.
- If you delete the OpenStackBaremetalSet resource, you will delete all IP reservations for the corresponding role, which means other roles can use these IP addresses.

6. Access the remote shell for **openstackclient**:

```
$ oc rsh openstackclient -n openstack
```

7. Remove the Compute service entries from the overcloud:

```
$ openstack compute service list
$ openstack compute service delete <service-id>
```

8. Check the Compute network agents entries in the overcloud and remove them if they exist:

```
$ openstack network agent list
$ for AGENT in $(openstack network agent list --host <scaled-down-node> -c ID -f value) ;
do openstack network agent delete $AGENT ; done
```

9. Exit from **openstackclient**:

```
$ exit
```