Red Hat OpenStack Platform 16.2

RHOSP director Operator for OpenShift Container Platform

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

Last Updated: 2024-04-15

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

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

OpenStack Team
rhos-docs@redhat.com

Legal Notice

Abstract

Learn how to install the RHOSP director Operator in your Red Hat OpenShift Container Platform cluster, and use director Operator to deploy an RHOSP overcloud. Support for Red Hat OpenStack Platform director Operator will only be granted if your architecture is approved by Red Hat Services or by a Technical Account Manager. Please contact Red Hat before deploying this feature.

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.

Providing documentation feedback in Jira

Use the Create Issue form to provide feedback on the documentation. The Jira issue will be created in the Red Hat OpenStack Platform Jira project, where you can track the progress of your feedback.

1. Ensure that you are logged in to Jira. If you do not have a Jira account, create an account to submit feedback.
2. Click the following link to open a the **Create Issue** page: Create Issue
3. Complete the **Summary** and **Description** fields. In the **Description** field, include the documentation URL, chapter or section number, and a detailed description of the issue. Do not modify any other fields in the form.
4. Click **Create**.

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.

NOTE

Support for Red Hat OpenStack Platform director Operator will only be granted if your architecture is approved by Red Hat Services or by a Technical Account Manager. Please contact Red Hat before deploying this feature.

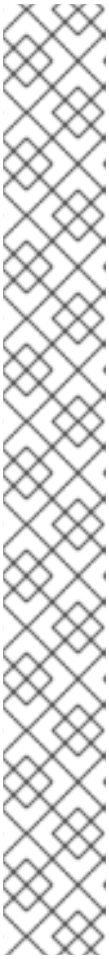
Additional resources

- 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 LTS version (OCP) 4.10 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.

If the cluster is of type AI or IPI, it uses **metal3**, a Kubernetes API for the management of baremetal hosts. It maintains an inventory of available hosts as instances of the BareMetalHost custom resource definition (CRD). The bare metal operator knows how to:

- Inspect the host's hardware details and report them to the corresponding BareMetalHost. This includes information about CPUs, RAM, disks, and NICs.
- Provision hosts with a specific image.
- Clean a host's disk contents before or after provisioning.

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 Operators from OperatorHub on your OCP cluster:
  - OpenShift Virtualization Operator
  - SR-IOV Network Operator
  - For OCP 4.11+ clusters: Kubernetes NMState Operator
- For OCP 4.11+ clusters: Create an NMState instance to finish installing all the NMState CRDs:

```
cat <<EOF | oc apply -f -
apiVersion: nmstate.io/v1
kind: NMState
metadata:
  name: nmstate
  namespace: openshift-nmstate
EOF
```

- 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**

- 500G for the base image that the director Operator clones for each Controller virtual machine
- A minimum of 50G for each Controller virtual machine. For more information see, Controller node requirements

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 4.10
  - SR-IOV Network Operator 4.10
- Ensure that you have installed the **oc** command line tool on your workstation.

Procedure

1. Create the **openstack** namespace:

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

2. Obtain the latest **osp-director-operator-bundle** image from https://catalog.redhat.com/software/containers/search.
3. Download the Operator Package Manager (**opm**) tool from https://console.redhat.com/openshift/downloads.
4. Use the **opm** tool to create an index image:

```
$ BUNDLE_IMG="registry.redhat.io/rhosp-rhel8/osp-director-operator-bundle@sha256:c19099ac3340d364307a43e0ae2be949a588fefe8fcb17663049342e7587f055"
$ INDEX_IMG="quay.io/<account>/osp-director-operator-index:x.y.z-a"
$ opm index add --bundles ${BUNDLE_IMG} --tag ${INDEX_IMG} -u podman --pull-tool podman
```

5. Push the index image to your registry:

```
$ podman push ${INDEX_IMG}
```

6. 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/<account>/osp-director-operator-index:x.y.z-a (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
```

(1) For information about how to apply the Quay authentication so that the Operator deployment can pull the image, see Accessing images for Operators from private registries.

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

```
$ oc apply -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

openstacknetattachment (internal)

Manages NodeNetworkConfigurationPolicy and NodeSriovConfigurationPolicy used to attach networks to virtual machines

openstacknetconfig

High level CRD to specify openstacknetattachments and openstacknets to describe the full network configuration. The set of reserved IP/MAC addresses per node are reflected in the status.

openstackbaremetalset

Create sets of baremetal hosts for a specific TripleO role (Compute, Storage, etc.)

openstackcontrolplane

A CRD used to create the OpenStack control plane and manage associated openstackvmsets

openstacknet (internal)

Create networks which are used to assign IPs to the vmset and baremetalset resources below

openstackipset (internal)

Contains a set of IPs for a given network and role. Used internally to manage IP addresses.

openstackprovisionservers

Used to serve custom images for baremetal provisioning with Metal3

openstackvmset

Create sets of VMs using OpenShift Virtualization for a specific TripleO role (Controller, Database, NetworkController, etc.)

Software Configuration CRDs

openstackconfiggenerator

Automatically generate Ansible playbooks for deployment when you scale up or make changes to custom ConfigMaps for deployment

openstackconfigversion

Represents a set of executable Ansible playbooks

openstackdeploy

Executes a set of Ansible playbooks (openstackconfigversion)

openstackclient

Creates a pod used to run TripleO deployment 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. FEATURES NOT SUPPORTED BY DIRECTOR OPERATOR

Fiber Channel back end

Block Storage (cinder) image-to-volume is not supported for back ends that use Fiber Channel. Red Hat OpenShift Virtualization does not support N_Port ID Virtualization (NPIV). Therefore, Block Storage drivers that need to map LUNs from a storage back end to the controllers, where **cinder-volume** runs by default, do not work. You must create a dedicated role for **cinder-volume** and use

the role to create physical nodes instead of including it on the virtualized controllers. For more information, see Composable services and custom roles in the *Customizing your Red Hat OpenStack Platform deployment* guide.

Role-based Ansible playbooks

Director Operator (OSPdO) does not support running Ansible playbooks to configure role-based node attributes after the bare-metal nodes are provisioned. This means that you cannot use the **role_growvols_args** extra Ansible variable to configure whole disk partitions for the Object Storage service (swift). Role-based Ansible playbook configuration only applies to bare-metal nodes that are provisioned by using a node definition file.

Migration of workloads from Red Hat Virtualization to OSPdO

You cannot migrate workloads from a Red Hat Virtualization environment to an OSPdO environment.

Using a VLAN for the control plane network

TripleO does not support using a VLAN for the control plane (**ctlplane**) network.

Multiple Compute cells

You cannot add additional Compute cells to an OSPdO environment.

BGP for the control plane

BGP is not supported for the control plane in an OSPdO environment.

PCI passthrough and attaching hardware devices to Controller VMs

You cannot attach SRIOV devices and FC SAN Storage to Controller VMs.

1.5. LIMITATIONS WITH A DIRECTOR OPERATOR DEPLOYMENT

A director Operator (OSPdO) environment has the following support limitations:

- Single-stack IPv6 is not supported. Only IPv4 is supported on the **ctlplane** network.
- You cannot create VLAN provider networks without dedicated networker nodes, because the NMState Operator cannot attach a VLAN trunk to the OSPdO Controller VMs. Therefore, to create VLAN provider networks, you must create dedicated Networker nodes on bare metal. For more information, see https://github.com/openstack/tripleo-heat-templates/blob/stable/wallaby/roles/Networker.yaml.
- You cannot removed the provisioning network.
- You cannot use a proxy for SSH connections to communicate with the Git repository.
- You cannot use HTTP or HTTPS to connect to the Git repository.

1.6. RECOMMENDATIONS FOR A DIRECTOR OPERATOR DEPLOYMENT

Storage class

For back end performance, use low latency SSD/NVMe-backed storage to create the RWX/RWO storage class required by the Controller virtual machines (VMs), the client pod, and images.

1.7. 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 using the **openstacknetconfig** CRD, 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 an **openstackconfiggenerator** to render Ansible playbooks for overcloud configuration.
6. Apply the Ansible playbook configuration to your overcloud nodes using **openstackdeploy**.

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 8.4 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-4.10-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' --truncate
/etc/machine-id
```

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

```
$ virtctl image-upload dv <datavolume_name> -n openstack \
 --size=<size> --image-path=<local_path_to_image> \
 --storage-class <storage_class> --access-mode <access_mode> --insecure
```

   - Replace **<datavolume_name>** with the name of the data volume, for example, **openstack-base-img**.
   - Replace **<size>** with the size of the data volume required for your environment, for example, **500Gi**. The minimum size is 500GB.

- Replace **<storage_class>** with the required storage class from your cluster. Use the following command to retrieve the available storage classes:

```
$ oc get storageclass
```

- Replace **<access_mode>** with the access mode for the PVC. The default value is **ReadWriteOnce**.

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

1. 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

1. 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:

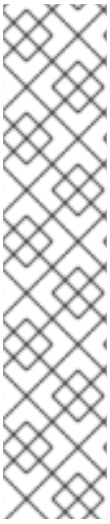
```
apiVersion: v1
kind: Secret
metadata:
  name: userpassword
```

```
  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
```



NOTE

Enter the **userpassword** Secret in **passwordSecret** when you create **OpenStackControlPlane** or **OpenStackBaremetalSet**:

```
apiVersion: osp-director.openstack.org/v1beta2
kind: OpenStackControlPlane
metadata:
  name: overcloud
  namespace: openstack
spec:
  passwordSecret: <userpassword>
```

- Replace **<userpassword>** with your **userpassword** Secret.

Additional resources

- "Providing sensitive data to pods"

CHAPTER 3. CREATING NETWORKS WITH THE DIRECTOR OPERATOR

Use the OpenStackNetConfig 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 OpenStackNetConfig resource includes an **attachConfigurations** option which is a hash of **nodeNetworkConfigurationPolicy**. Each specified **attachConfiguration** in the OpenStackNetConfig creates an OpenStackNet Attachment, which passes network interface data to the NodeNetworkConfigurationPolicy resource in OpenShift. The NodeNetworkConfigurationPolicy resource uses the **nmstate** API to configure the end state of the network configuration on each OCP worker node. Each network, configured in the OpenStackNetConfig, references one of the **attachConfigurations**. Inside the virtual machines, there is one interface per network. Through this method, you can create required bridges on OCP worker nodes and connect your Controller virtual machines to RHOSP networks.

For example, if you create a **br-osp attachConfiguration** 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: OpenStackNetConfig
metadata:
  name: openstacknetconfig
spec:
  attachConfigurations:
    br-osp:
      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
            mtu: 1500
...
  networks:
  - name: Control
```

```
    nameLower: ctlplane
    subnets:
    - name: ctlplane
      ipv4:
        allocationEnd: 192.168.25.250
        allocationStart: 192.168.25.100
        cidr: 192.168.25.0/24
        gateway: 192.168.25.1
      attachConfiguration: br-osp
```

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

If you specify an Internal API network through VLAN 20, you can set the **attachConfiguration** 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: OpenStackNetConfig
metadata:
  name: openstacknetconfig
spec:
  attachConfigurations:
    br-osp:
…
  networks:
…
 - isControlPlane: false
    mtu: 1500
    name: InternalApi
    nameLower: internal_api
    subnets:
    - attachConfiguration: br-osp
      ipv4:
        allocationEnd: 172.17.0.250
        allocationStart: 172.17.0.10
        cidr: 172.17.0.0/24
        gateway: 172.17.0.1
        routes:
        - destination: 172.17.1.0/24
          nexthop: 172.17.0.1
        - destination: 172.17.2.0/24
          nexthop: 172.17.0.1
      name: internal_api
      vlan: 20
```

The **br-osp** already exists and is connected to the **enp6s0** NIC on each host, so no change occurs to the bridge itself. However, the InternalAPI 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.

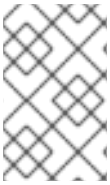
For example, if you create the default RHOSP networks with OpenStackNetConfig, the interface configuration for Controller virtual machines resembles the following example:

```
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



NOTE

The role NIC template used by OpenStackVMSet is auto generated. It can be overwritten by adding a nic-template.role.j2 file to your tarball file. Include the binary contents of the tarball file in an OpenShift ConfigMap names **tripleo-tarball-config**.

Additional resources

- "Updating node network configuration"

3.2. CREATING AN OVERCLOUD CONTROL PLANE NETWORK WITH OPENSTACKNETCONFIG

You must define at least one control plane network for your overcloud in OpenStackNetConfig. In addition to IP address assignment, the network definition includes the mapping information for OpenStackNetAttachment. OpenShift Virtualization uses this information 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 **osnetconfig.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: OpenStackNetConfig
metadata:
  name: openstacknetconfig
spec:
  attachConfigurations:
    br-osp:
      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
            mtu: 1500
  # optional DnsServers list
  dnsServers:
  - 192.168.25.1
  # optional DnsSearchDomains list
  dnsSearchDomains:
  - osptest.test.metalkube.org
```

```
  - some.other.domain
  # DomainName of the OSP environment
  domainName: osptest.test.metalkube.org
  networks:
  - name: Control
    nameLower: ctlplane
    subnets:
    - name: ctlplane
      ipv4:
        allocationEnd: 172.22.0.250
        allocationStart: 172.22.0.100
        cidr: 172.22.0.0/24
        gateway: 172.22.0.1
      attachConfiguration: br-osp
  # optional: configure static mapping for the networks per nodes. If there is none, a random
gets created
    reservations:
      controller-0:
        ipReservations:
          ctlplane: 172.22.0.120
      compute-0:
        ipReservations:
          ctlplane: 172.22.0.140
```

Set the following values in the networks specification:

name

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

nameLower

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

subnets

Set the subnet specifications.

subnets.name

Set the name of the control plane subnet, which is ctlplane.

subnets.attachConfiguration

Set the reference to which of the attach configuration should be used.

subnets.ipv4

Details of the ipv4 subnet with allocationStart, allocationEnd, cidr, gateway and optional list of routes (with destination and nexthop)

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

```
$ oc describe crd openstacknetconfig
```

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

2. Create the control plane network:

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

Verification

1. View the resource for the control plane network:

```
$ oc get openstacknetconfig/openstacknetconfig
```

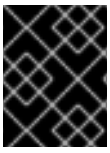
3.3. CREATING VLAN NETWORKS FOR NETWORK ISOLATION WITH OPENSTACKNETCONFIG

You must create additional networks to implement network isolation for your composable networks. To accomplish this network isolation, you can place your composable networks on individual VLAN networks. In addition to IP address assignment, the OpenStackNetConfig 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 OpenStackNetConfig resource which defines each network.

Table 3.2. Default Red Hat OpenStack Platform networks

Network	VLAN	CIDR	Allocation
External	10	10.0.0.0/24	10.0.0.10 - 10.0.0.250
InternalApi	20	172.17.0.0/24	172.17.0.10 - 172.17.0.250
Storage	30	172.18.0.0/24	172.18.0.10 - 172.18.0.250
StorageMgmt	40	172.19.0.0/24	172.19.0.10 - 172.19..250
Tenant	50	172.20.0.0/24	172.20.0.10 - 172.20.0.250



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 network configuration. Include the resource specification for the VLAN network. For example, the specification for internal API, storage, storage mgmt, tenant, and external network that manages VLAN-tagged traffic over Linux bridges **br-ex** and **br-osp** connected to the **enp6s0** and **enp7s0** Ethernet device on each worker node is as follows:

```
kind: OpenStackNetConfig
metadata:
  name: openstacknetconfig
spec:
  attachConfigurations:
    br-osp:
      nodeNetworkConfigurationPolicy:
        nodeSelector:
          node-role.kubernetes.io/worker: ""
        desiredState:
          interfaces:
          - bridge:
              options:
                stp:
                  enabled: false
              port:
              - name: enp7s0
            description: Linux bridge with enp7s0 as a port
            name: br-osp
            state: up
            type: linux-bridge
            mtu: 1500
    br-ex:
      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-ex
            state: up
            type: linux-bridge
            mtu: 1500
  # optional DnsServers list
  dnsServers:
  - 172.22.0.1
  # optional DnsSearchDomains list
  dnsSearchDomains:
  - osptest.test.metalkube.org
  - some.other.domain
  # DomainName of the OSP environment
  domainName: osptest.test.metalkube.org
  networks:
  - name: Control
    nameLower: ctlplane
    subnets:
    - name: ctlplane
      ipv4:
        allocationEnd: 172.22.0.250
        allocationStart: 172.22.0.10
```

```
      cidr: 172.22.0.0/24
      gateway: 172.22.0.1
    attachConfiguration: br-osp
- name: InternalApi
  nameLower: internal_api
  mtu: 1350
  subnets:
  - name: internal_api
    attachConfiguration: br-osp
    vlan: 20
    ipv4:
      allocationEnd: 172.17.0.250
      allocationStart: 172.17.0.10
      cidr: 172.17.0.0/24
- name: External
  nameLower: external
  subnets:
  - name: external
    ipv4:
      allocationEnd: 10.0.0.250
      allocationStart: 10.0.0.10
      cidr: 10.0.0.0/24
      gateway: 10.0.0.1
    attachConfiguration: br-ex
- name: Storage
  nameLower: storage
  mtu: 1500
  subnets:
  - name: storage
    ipv4:
      allocationEnd: 172.18.0.250
      allocationStart: 172.18.0.10
      cidr: 172.18.0.0/24
    vlan: 30
    attachConfiguration: br-osp
- name: StorageMgmt
  nameLower: storage_mgmt
  mtu: 1500
  subnets:
  - name: storage_mgmt
    ipv4:
      allocationEnd: 172.19.0.250
      allocationStart: 172.19.0.10
      cidr: 172.19.0.0/24
    vlan: 40
    attachConfiguration: br-osp
- name: Tenant
  nameLower: tenant
  vip: False
  mtu: 1500
  subnets:
  - name: tenant
    ipv4:
      allocationEnd: 172.20.0.250
      allocationStart: 172.20.0.10
```

```
      cidr: 172.20.0.0/24
    vlan: 50
    attachConfiguration: br-osp
```

When you use 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 that the network template for the **OpenStackVMSet** is a multi-NIC network template.

Set the following values in the resource specification:

metadata.name

Set to the name of the OpenStackNetConfig.

spec

Set the network configuration for attaching the networks and the network specifics. For descriptions of the values you can use in this section, view the specification schema in the custom resource definition for the **openstacknetconfig** CRD:

```
$ oc describe crd openstacknetconfig
```

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

2. Create the network configuration:

```
$ oc apply -f openstacknetconfig.yaml -n openstack
```

Verification

1. View the OpenStackNetConfig API and created child resources:

```
$ oc get openstacknetconfig/openstacknetconfig -n openstack
$ oc get openstacknetattachment -n openstack
$ oc get openstacknet -n openstack
```

If you see errors, check the underlying **network-attach-definition** and node network configuration policies:

```
$ oc get network-attachment-definitions -n openstack
$ oc get nncp
```

3.4. CONFIGURING JUMBO FRAMES WITH OPENSTACKNETCONFIG

To use Jumbo Frames for a bridge, you can create a configuration for the device to configure the correct MTU:

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackNetConfig
metadata:
  name: openstacknetconfig
spec:
  attachConfigurations:
    br-osp:
      nodeNetworkConfigurationPolicy:
        nodeSelector:
          node-role.kubernetes.io/worker: ""
        desiredState:
          interfaces:
          - bridge:
              options:
                stp:
                  enabled: false
              port:
              - name: enp7s0
            description: Linux bridge with enp7s0 as a port
            name: br-osp
            state: up
            type: linux-bridge
            mtu: 9000
          - name: enp7s0
            description: Configuring enp7s0 on workers
            type: ethernet
            state: up
            mtu: 9000
```

3.5. STATIC IP RESERVATION WITH OPENSTACKNETCONFIG

You can use the **OpenStackNetConfig** specification reservations parameter to reserve a static IP address per host and network. The reservations provided there are populated down to the` OpenStackNet` specifications reservations and have precedence over any auto generated IPs. The following example shows an overcloud with 3 Controllers and 2 Compute nodes, all nodes have static reservations except **controller-2** and **compute-1**:

```
spec:
  …
  reservations:
    compute-0:
      ipReservations:
        ctlplane: 172.22.0.140
        internal_api: 172.17.0.40
        storage: 172.18.0.40
        tenant: 172.20.0.40
      macReservations: {}
    controller-0:
      ipReservations:
        ctlplane: 172.22.0.120
        external: 10.0.0.20
        internal_api: 172.17.0.20
        storage: 172.18.0.20
        storage_mgmt: 172.19.0.20
        tenant: 172.20.0.20
```

```
    macReservations: {}
  controller-1:
    ipReservations:
      ctlplane: 172.22.0.130
      external: 10.0.0.30
      internal_api: 172.17.0.30
      storage: 172.18.0.30
      storage_mgmt: 172.19.0.30
      tenant: 172.20.0.30
    macReservations: {}
  controlplane:
    ipReservations:
      ctlplane: 172.22.0.110
      external: 10.0.0.10
      internal_api: 172.17.0.10
      storage: 172.18.0.10
      storage_mgmt: 172.19.0.10
    macReservations: {}
  openstackclient-0:
    ipReservations:
      ctlplane: 172.22.0.251
      external: 10.0.0.251
      internal_api: 172.17.0.251
    macReservations: {}
```

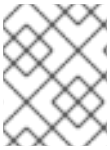
CHAPTER 4. ADDING HEAT TEMPLATES AND ENVIRONMENT FILES WITH THE DIRECTOR OPERATOR

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 heat templates and custom roles file into 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.



NOTE

All references in the environment files must be relative to the TripleO Heat Templates where the tarball is extracted.

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
roles_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 **roles_data.yaml** file overrides the default roles configuration.

Additional resources

- "Understanding 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

1. View the ConfigMap:

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

Additional resources

- Creating and using config maps
- "Understanding 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::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

1. 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.

By default, there is no automatic discovery and acting on OpenShift worker nodes where the virtual machines are hosted. For more information on how to auto discover issues on OpenShift worker nodes, see Deploying machine health checks .

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 OpenStackNetConfig 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/v1beta2
kind: OpenStackControlPlane
metadata:
  name: overcloud
  namespace: openstack
spec:
  openStackClientNetworks:
        - ctlplane
        - internal_api
        - external
  openStackClientStorageClass: host-nfs-storageclass
  passwordSecret: userpassword
  virtualMachineRoles:
```

```
    Controller:
      roleName: Controller
      roleCount: 3
      networks:
        - ctlplane
        - internal_api
        - external
        - tenant
        - storage
        - storage_mgmt
      cores: 12
      memory: 64
      rootDisk:
        diskSize: 500
        baseImageVolumeName: openstack-base-img
        # storageClass must support RWX to be able to live migrate VMs
        storageClass: host-nfs-storageclass
        storageAccessMode:  ReadWriteMany
        # When using OpenShift Virtualization with OpenShift Container Platform Container
Storage,
        # specify RBD block mode persistent volume claims (PVCs) when creating virtual
machine disks.
        # With virtual machine disks, RBD block mode volumes are more efficient and provide
better
        # performance than Ceph FS or RBD filesystem-mode PVCs.
        # To specify RBD block mode PVCs, use the 'ocs-storagecluster-ceph-rbd' storage
class and
        # VolumeMode: Block.
        storageVolumeMode: Filesystem
      # optional configure additional discs to be attached to the VMs,
      # need to be configured manually inside the VMs where to be used.
      additionalDisks:
        - name: datadisk
          diskSize: 500
          storageClass: host-nfs-storageclass
          storageAccessMode:  ReadWriteMany
          storageVolumeMode: Filesystem
  openStackRelease: "16.2"
```

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 -n openstack openstackclient
```

5.2. CREATING A PROVISIONING SERVER WITH OPENSTACKPROVISIONSERVER (OPTIONAL)

Provisioning servers provide a specific Red Hat Enterprise Linux (RHEL) QCOW2 image for provisioning Compute nodes for the Red Hat OpenStack Platform (RHOSP). An **OpenStackProvisionServer** is automatically created for any **OpenStackBaremetalSets** you create. However, you can decide to create the **OpenStackProvisionServer** manually and later provide the name to any future **OpenStackBaremetalSets**.

The **OpenStackProvisionServer** creates an Apache server on the OpenShift Container Platform provisioning network for a specific RHEL QCOW2 image.

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 **openstack-provision.yaml** on your workstation. Include the resource specification for the Provisioning server. For example, the specification for a Provisioning server using a specific RHEL 8.4 QCOW2 images:

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackProvisionServer
metadata:
  name: openstack-provision-server
  namespace: openstack
spec:
  baseImageUrl: http://<source_host>/rhel-guest-image-8.4-992.x86_64.qcow2
  port: 8080
```

Set the following values in the resource specification:

metadata.name

Set a name to identify the OpenStackProvisionServer.

metadata.namespace

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

spec

baseImageURL

Set the initial source of the RHEL QCOW2 image for the Provisioing server. The image is download from this remote source when the server is created.

port

By default, set to 8080. You can change it for a specific port configuration.

For further descriptions of the values you can use in this section, view the specification schema in the custom resource definition for the **OpenStackProvisionServer** CRD:

```
$ oc describe crd openstackprovisionserver
```

Save the file when you have finished configuring the Provisioning server specification.

2. Create the Provisioning Server:

```
$ oc create -f openstack-provision-server.yaml -n openstack
```

Verification

1. View the resource for the Provisioning server:

```
$ oc get openstackprovisionserver/openstack-provision-server -n openstack
```

5.3. 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 OpenStackNetConfig 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
  # If you manually created an OpenStackProvisionServer, you can use it here,
  # otherwise the director Operator will create one for you (with `baseImageUrl` as the image
that it server)
  # to use with this OpenStackBaremetalSet
  # provisionServerName: openstack-provision-server
  ctlplaneInterface: enp2s0
  networks:
    - ctlplane
    - internal_api
    - 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 **OpenStackConfigGenerator** resource to generate your Ansible playbooks, register your nodes to either the Red Hat Customer Portal or Red Hat Satellite, and then create an **OpenStackDeploy** resource to apply the configuration to your nodes

6.1. CREATING ANSIBLE PLAYBOOKS FOR OVERCLOUD CONFIGURATION WITH OPENSTACKCONFIGGENERATOR

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 OpenStackConfigGenerator 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.
- OpenStackControlPlane and OpenStackBaremetalSets created as required.
- 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-config-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: OpenStackConfigGenerator
metadata:
  name: default
  namespace: openstack
spec:
  enableFencing: true
  gitSecret: git-secret
  imageURL: registry.redhat.io/rhosp-rhel8/openstack-tripleoclient:16.2
  heatEnvConfigMap: heat-env-config
  # List of heat environment files to include from tripleo-heat-templates/environments
  heatEnvs:
  - ssl/tls-endpoints-public-dns.yaml
  - ssl/enable-tls.yaml
  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, by default **default**.

metadata.namespace

Set to the director Operator namespace, by default **openstack**.

spec.enableFencing

Enable the automatic creation of required heat environment files to enable fencing.

NOTE

Production OSP environments must have fencing enabled. Virtual machines running pacemaker require the **fence-agents-kubevirt** package.

spec.gitSecret

Set to the ConfigMap that contains the Git authentication credentials, by default **git-secret**.

spec.heatEnvs

A list of default tripleo environment files used to generate the playbooks.

spec.heatEnvConfigMap

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

spec.tarballConfigMap

Set to the ConfigMap that contains the tarball with your custom heat templates, by default **tripleo-tarball-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 **openstackconfiggenerator** CRD:

```
$ oc describe crd openstackconfiggenerator
```

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

2. Create the Ansible config generator:

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

Verification

1. View the resource for the config generator:

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

6.2. EPHEMERAL HEAT CONTAINER IMAGE SOURCE PARAMETERS

To create an ephemeral heat service, The OpenStackConfigGenerator 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: OpenStackConfigGenerator
metadata:
  name: default
  namespace: openstack
spec:
  …
  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. CONFIG GENERATION INTERACTIVE MODE

To debug config generation operations, you can set the OpenStackConfigGenerator resource to use interactive mode.

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackConfigGenerator
metadata:
  name: default
  namespace: openstack
spec:
  …
  interactive: true
```

In this mode, the OpenStackConfigGenerator resource creates the environment to start rendering the playbooks but does not automatically render the playbooks. When the OpenStackConfigGenerator pod, with the prefix **generate-config** starts, you can **rsh** into the pod and inspect files and playbook

rendering:

```
$ oc rsh $(oc get pod -o name -l job-name=generate-config-default)
$ ls -la /home/cloud-admin/
...
config 1
config-custom 2
config-passwords 3
create-playbooks.sh 4
process-heat-environment.py 5
tht-tars 6
```

1. Directory storing auto rendered files by the director operator
2. Directory storing environment files provided via heatEnvConfigMap
3. Directory storing the overcloud service passwords created by the director operator
4. Script to render the ansible playbooks
5. Internal script used by **create-playbooks**, to replicate the undocumented heat client merging of map parameters
6. Directory storing the tarball from the **tarballConfigMap**

6.4. USING THE HEAT ENVIRONMENT FROM TRIPLEO-HEAT-TEMPLATES/ENVIRONMENTS

TripleO is delivered with heat environment files for different deployment scenarios, for example, TLS for public endpoints. Heat environment files can be included into the playbook generation using the **heatEnvs** parameter list.

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackConfigGenerator
metadata:
  name: default
  namespace: openstack
spec:
  …
  heatEnvs:
  - ssl/tls-endpoints-public-dns.yaml
  - ssl/enable-tls.yaml
```

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.

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 (RHOSP) commands. This pod also contains an ansible inventory script named **/home/cloud-admin/ctlplane-ansible-**

inventory.

To register your nodes, you can use the **redhat_subscription** Ansible module with the inventory script from the OpentackClient pod.

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.

Procedure

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

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

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

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

3. 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
      - 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.

4. Register the overcloud nodes to required repositories:

```
ansible-playbook -i /home/cloud-admin/ctlplane-ansible-inventory ./rhsm.yaml
```

Additional resources

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

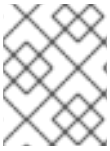
6.6. OBTAIN THE LATEST OPENSTACKCONFIGVERSION

Different versions of Ansible playbooks are stored in the git repository. For each version an OpenStackConfigVersion object exists which references the **hash/digest** of git.

Procedure

1. Select the **hash/digest** of the latest OpenStackConfigVersion:

```
$ oc get -n openstack --sort-by {.metadata.creationTimestamp} osconfigversions -o json
```



NOTE

OpenStackConfigVersion objects also have a **git diff** attribute that can be used to easily compare the changes between Ansible playbook versions.

6.7. 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 OpenStackDeploy resource, the director Operator creates a job that runs the ansible playbooks 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 OpentackConfigGenerator to create the Ansible playbook configuration for your overcloud.
- Use the OpeenstackConfigVersion to select the hash/digest of the ansible playbooks which should be used to configure the overcloud.

Procedure

1. Create a file named **openstack-deployment.yaml** on your workstation. Include the resource specification to the Ansible playbooks. For example:

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackDeploy
metadata:
  name: default
spec:
  configVersion: n5fch96h548h75hf4hbdhb8hfdh676h57bh96h5c5h59hf4h88h…
  configGenerator: default
```

   Set the following values in the resource specification:

   metadata.name

   Set the name of the Compute node baremetal set, by default **default**.

   metadata.namespace

   Set to the diretor Operator namespace, by default **openstack**.

   spec.configVersion

   The config version/git hash of the playbooks to deploy.

   spec.configGenerator

   The name of the configGenerator.

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

```
$ oc describe crd openstackdeploy
```

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

2. Create the OpenStackDeploy resource:

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

   As the deployment runs it creates a Kubernetes job to execute the Ansible playbooks. You can tail the logs of the job to watch the Ansible playbooks running:

```
$ oc logs -f jobs/deploy-openstack-default
```

   Additionally, you can manually access the executed Ansible playbooks by logging into the **openstackclient** pod. In the **/home/cloud-admin/work/directory** you can find the ansible playbooks and the **ansible.log** file for the current deployment.

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

- 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 8.4 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-4.10-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' --truncate
/etc/machine-id
```

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

```
$ virtctl image-upload dv <datavolume_name> -n openstack \
 --size=<size> --image-path=<local_path_to_image> \
 --storage-class <storage_class> --access-mode <access_mode> --insecure
```

   - Replace **<datavolume_name>** with the name of the data volume, for example, **openstack-base-img**.

- Replace **<size>** with the size of the data volume required for your environment, for example, **500Gi**. The minimum size is 500GB.
- Replace **<storage_class>** with the required storage class from your cluster. Use the following command to retrieve the available storage classes:

```
$ oc get storageclass
```

- Replace **<access_mode>** with the access mode for the PVC. The default value is **ReadWriteOnce**.

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

1. 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

1. View the Secret resource:

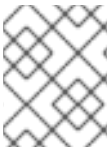
```
$ oc get secret/git-secret -n openstack
```

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.

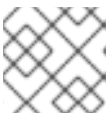
Prerequisites

- 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:

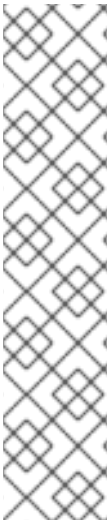
```
apiVersion: v1
kind: Secret
metadata:
```

```
  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
```



NOTE

Enter the **userpassword** Secret in **passwordSecret** when you create **OpenStackControlPlane** or **OpenStackBaremetalSet**:

```
apiVersion: osp-director.openstack.org/v1beta2
kind: OpenStackControlPlane
metadata:
  name: overcloud
  namespace: openstack
spec:
  passwordSecret: <userpassword>
```

- Replace **<userpassword>** with your **userpassword** Secret.

Additional resources

- "Providing sensitive data to pods"

7.4. CREATING AN OVERCLOUD CONTROL PLANE NETWORK WITH OPENSTACKNETCONFIG

You must define at least one control plane network for your overcloud in OpenStackNetConfig. In addition to IP address assignment, the network definition includes the mapping information for OpenStackNetAttachment. OpenShift Virtualization uses this information 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 **osnetconfig.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: OpenStackNetConfig
metadata:
  name: openstacknetconfig
spec:
  attachConfigurations:
    br-osp:
      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
            mtu: 1500
  # optional DnsServers list
  dnsServers:
  - 192.168.25.1
  # optional DnsSearchDomains list
  dnsSearchDomains:
  - osptest.test.metalkube.org
  - some.other.domain
  # DomainName of the OSP environment
  domainName: osptest.test.metalkube.org
  networks:
  - name: Control
    nameLower: ctlplane
    subnets:
    - name: ctlplane
      ipv4:
        allocationEnd: 172.22.0.250
        allocationStart: 172.22.0.100
        cidr: 172.22.0.0/24
        gateway: 172.22.0.1
      attachConfiguration: br-osp
  # optional: configure static mapping for the networks per nodes. If there is none, a random
gets created
  reservations:
    controller-0:
      ipReservations:
        ctlplane: 172.22.0.120
    compute-0:
      ipReservations:
        ctlplane: 172.22.0.140
```

Set the following values in the networks specification:

name

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

nameLower

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

subnets

Set the subnet specifications.

subnets.name

Set the name of the control plane subnet, which is ctlplane.

subnets.attachConfiguration

Set the reference to which of the attach configuration should be used.

subnets.ipv4

Details of the ipv4 subnet with allocationStart, allocationEnd, cidr, gateway and optional list of routes (with destination and nexthop)

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

```
$ oc describe crd openstacknetconfig
```

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

2. Create the control plane network:

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

Verification

1. View the resource for the control plane network:

```
$ oc get openstacknetconfig/openstacknetconfig
```

7.5. CREATING VLAN NETWORKS FOR NETWORK ISOLATION WITH OPENSTACKNETCONFIG

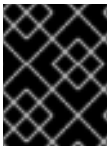
You must create additional networks to implement network isolation for your composable networks. To accomplish this network isolation, you can place your composable networks on individual VLAN networks. In addition to IP address assignment, the OpenStackNetConfig resource includes information to define 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 OpenStackNetConfig resource which defines each network.

Table 7.1. Default Red Hat OpenStack Platform networks

Network	VLAN	CIDR	Allocation
External	10	10.0.0.0/24	10.0.0.10 - 10.0.0.250
InternalApi	20	172.17.0.0/24	172.17.0.10 - 172.17.0.250

Network	VLAN	CIDR	Allocation
Storage	30	172.18.0.0/24	172.18.0.10 - 172.18.0.250
StorageMgmt	40	172.19.0.0/24	172.19.0.10 - 172.19..250
Tenant	50	172.20.0.0/24	172.20.0.10 - 172.20.0.250



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 network configuration. Include the resource specification for the VLAN network. For example, the specification for internal API, storage, storage mgmt, tenant, and external network that manages VLAN-tagged traffic over Linux bridges **br-ex** and **br-osp** connected to the **enp6s0** and **enp7s0** Ethernet device on each worker node is as follows:

```
kind: OpenStackNetConfig
metadata:
  name: openstacknetconfig
spec:
  attachConfigurations:
    br-osp:
      nodeNetworkConfigurationPolicy:
        nodeSelector:
          node-role.kubernetes.io/worker: ""
        desiredState:
          interfaces:
          - bridge:
              options:
                stp:
                  enabled: false
              port:
              - name: enp7s0
            description: Linux bridge with enp7s0 as a port
            name: br-osp
            state: up
            type: linux-bridge
            mtu: 1500
    br-ex:
      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-ex
          state: up
          type: linux-bridge
          mtu: 1500
# optional DnsServers list
dnsServers:
- 172.22.0.1
# optional DnsSearchDomains list
dnsSearchDomains:
- osptest.test.metalkube.org
- some.other.domain
# DomainName of the OSP environment
domainName: osptest.test.metalkube.org
networks:
- name: Control
  nameLower: ctlplane
  subnets:
  - name: ctlplane
    ipv4:
      allocationEnd: 172.22.0.250
      allocationStart: 172.22.0.10
      cidr: 172.22.0.0/24
      gateway: 172.22.0.1
    attachConfiguration: br-osp
- name: InternalApi
  nameLower: internal_api
  mtu: 1350
  subnets:
  - name: internal_api
    attachConfiguration: br-osp
    vlan: 20
    ipv4:
      allocationEnd: 172.17.0.250
      allocationStart: 172.17.0.10
      cidr: 172.17.0.0/24
- name: External
  nameLower: external
  subnets:
  - name: external
    ipv4:
      allocationEnd: 10.0.0.250
      allocationStart: 10.0.0.10
      cidr: 10.0.0.0/24
      gateway: 10.0.0.1
    attachConfiguration: br-ex
- name: Storage
```

```
    nameLower: storage
    mtu: 1500
    subnets:
    - name: storage
      ipv4:
        allocationEnd: 172.18.0.250
        allocationStart: 172.18.0.10
        cidr: 172.18.0.0/24
      vlan: 30
      attachConfiguration: br-osp
  - name: StorageMgmt
    nameLower: storage_mgmt
    mtu: 1500
    subnets:
    - name: storage_mgmt
      ipv4:
        allocationEnd: 172.19.0.250
        allocationStart: 172.19.0.10
        cidr: 172.19.0.0/24
      vlan: 40
      attachConfiguration: br-osp
  - name: Tenant
    nameLower: tenant
    vip: False
    mtu: 1500
    subnets:
    - name: tenant
      ipv4:
        allocationEnd: 172.20.0.250
        allocationStart: 172.20.0.10
        cidr: 172.20.0.0/24
      vlan: 50
      attachConfiguration: br-osp
```

When you use 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 that the network template for the **OpenStackVMSet** is a multi-NIC network template.

Set the following values in the resource specification:

metadata.name

Set to the name of the OpenStackNetConfig.

spec

Set the network configuration for attaching the networks and the network specifics. For descriptions of the values you can use in this section, view the specification schema in the custom resource definition for the **openstacknetconfig** CRD:

```
-
```

```
$ oc describe crd openstacknetconfig
```

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

2. Create the network configuration:

```
$ oc apply -f openstacknetconfig.yaml -n openstack
```

Verification

1. View the OpenStackNetConfig API and created child resources:

```
$ oc get openstacknetconfig/openstacknetconfig -n openstack
$ oc get openstacknetattachment -n openstack
$ oc get openstacknet -n openstack
```

If you see errors, check the underlying **network-attach-definition** and node network configuration policies:

```
$ oc get network-attachment-definitions -n openstack
$ oc get nncp
```

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 OpenStackNetConfig 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/v1beta2
kind: OpenStackControlPlane
```

```
metadata:
  name: overcloud
  namespace: openstack
spec:
  openStackClientNetworks:
        - ctlplane
        - internal_api
        - external
  openStackClientStorageClass: host-nfs-storageclass
  passwordSecret: userpassword
  virtualMachineRoles:
    Controller:
      roleName: Controller
      roleCount: 3
      networks:
        - ctlplane
        - internal_api
        - external
        - tenant
        - storage
        - storage_mgmt
      cores: 12
      memory: 64
      rootDisk:
        diskSize: 500
        baseImageVolumeName: openstack-base-img
        # storageClass must support RWX to be able to live migrate VMs
        storageClass: host-nfs-storageclass
        storageAccessMode:  ReadWriteMany
        # When using OpenShift Virtualization with OpenShift Container Platform Container
Storage,
        # specify RBD block mode persistent volume claims (PVCs) when creating virtual
machine disks.
        # With virtual machine disks, RBD block mode volumes are more efficient and provide
better
        # performance than Ceph FS or RBD filesystem-mode PVCs.
        # To specify RBD block mode PVCs, use the 'ocs-storagecluster-ceph-rbd' storage
class and
        # VolumeMode: Block.
        storageVolumeMode: Filesystem
      # optional configure additional discs to be attached to the VMs,
      # need to be configured manually inside the VMs where to be used.
      additionalDisks:
        - name: datadisk
          diskSize: 500
          storageClass: host-nfs-storageclass
          storageAccessMode:  ReadWriteMany
          storageVolumeMode: Filesystem
  openStackRelease: "16.2"
```

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 -n openstack openstackclient
```

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 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: interface
              name: nic4
              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
            - type: vlan
              mtu:
                get_param: StorageMtu
              device: nic4
              vlan_id:
                get_param: StorageNetworkVlanID
              addresses:
              - ip_netmask:
                  get_param: StorageIpSubnet
              routes:
                list_concat_unique:
                  - get_param: StorageInterfaceRoutes
            - type: vlan
              mtu:
                get_param: InternalApiMtu
              device: nic4
              vlan_id:
                get_param: InternalApiNetworkVlanID
              addresses:
              - ip_netmask:
                  get_param: InternalApiIpSubnet
              routes:
```

```
            list_concat_unique:
              - get_param: InternalApiInterfaceRoutes

          - type: ovs_bridge
            # This will default to br-ex, anything else  requires specific
            # bridge 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
            - type: vlan
              mtu:
                get_param: TenantMtu
              vlan_id:
                get_param: TenantNetworkVlanID
              addresses:
              - ip_netmask:
                  get_param: TenantIpSubnet
              routes:
                list_concat_unique:
                  - get_param: TenantInterfaceRoutes
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	N/A	**nic4**
External, Tenant	**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.

7.9. 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 -n openstack openstackclient
```

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"

7.10. 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

1. View the ConfigMap:

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

Additional resources

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

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

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

```
resource_registry:
  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

7.12. 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"

7.13. 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

1. View the ConfigMap:

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

Additional resources

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

7.14. 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 OpenStackNetConfig 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:
  count: 3
  baseImageUrl: http://host/images/rhel-image-8.4.x86_64.qcow2
  deploymentSSHSecret: osp-controlplane-ssh-keys
  ctlplaneInterface: enp8s0
  networks:
    - ctlplane
    - internal_api
    - tenant
    - storage
    - storage_mgmt
  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
```

7.15. CREATING ANSIBLE PLAYBOOKS FOR OVERCLOUD CONFIGURATION WITH OPENSTACKCONFIGGENERATOR

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 OpenStackConfigGenerator 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.
- OpenStackControlPlane and OpenStackBaremetalSets created as required.
- 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-config-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: OpenStackConfigGenerator
metadata:
  name: default
  namespace: openstack
spec:
```

```
enableFencing: true
gitSecret: git-secret
imageURL: registry.redhat.io/rhosp-rhel8/openstack-tripleoclient:16.2
heatEnvConfigMap: heat-env-config
# List of heat environment files to include from tripleo-heat-templates/environments
heatEnvs:
- ssl/tls-endpoints-public-dns.yaml
- ssl/enable-tls.yaml
tarballConfigMap: tripleo-tarball-config
```

Set the following values in the resource specification:

metadata.name

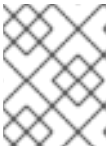
Set to the name of the Compute node bare metal set, by default **default**.

metadata.namespace

Set to the director Operator namespace, by default **openstack**.

spec.enableFencing

Enable the automatic creation of required heat environment files to enable fencing.



NOTE

Production OSP environments must have fencing enabled. Virtual machines running pacemaker require the **fence-agents-kubevirt** package.

spec.gitSecret

Set to the ConfigMap that contains the Git authentication credentials, by default **git-secret**.

spec.heatEnvs

A list of default tripleo environment files used to generate the playbooks.

spec.heatEnvConfigMap

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

spec.tarballConfigMap

Set to the ConfigMap that contains the tarball with your custom heat templates, by default **tripleo-tarball-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 **openstackconfiggenerator** CRD:

```
$ oc describe crd openstackconfiggenerator
```

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

2. Create the Ansible config generator:

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

Verification

1. View the resource for the config generator:

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

7.16. 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.

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 (RHOSP) commands. This pod also contains an ansible inventory script named **/home/cloud-admin/ctlplane-ansible-inventory**.

To register your nodes, you can use the **redhat_subscription** Ansible module with the inventory script from the OpentackClient pod.

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.

Procedure

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

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

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

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

3. 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
      - 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.

4. Register the overcloud nodes to required repositories:

```
ansible-playbook -i /home/cloud-admin/ctlplane-ansible-inventory ./rhsm.yaml
```

Additional resources

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

7.17. 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 OpenStackDeploy resource, the director Operator creates a job that runs the ansible playbooks 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 OpentackConfigGenerator to create the Ansible playbook configuration for your overcloud.

- Use the OpeenstackConfigVersion to select the hash/digest of the ansible playbooks which should be used to configure the overcloud.

Procedure

1. Create a file named **openstack-deployment.yaml** on your workstation. Include the resource specification to the Ansible playbooks. For example:

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackDeploy
metadata:
  name: default
spec:
  configVersion: n5fch96h548h75hf4hbdhb8hfdh676h57bh96h5c5h59hf4h88h…
  configGenerator: default
```

   Set the following values in the resource specification:

   metadata.name

   Set the name of the Compute node baremetal set, by default **default**.

   metadata.namespace

   Set to the diretor Operator namespace, by default **openstack**.

   spec.configVersion

   The config version/git hash of the playbooks to deploy.

   spec.configGenerator

   The name of the configGenerator.

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

```
$ oc describe crd openstackdeploy
```

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

2. Create the OpenStackDeploy resource:

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

   As the deployment runs it creates a Kubernetes job to execute the Ansible playbooks. You can tail the logs of the job to watch the Ansible playbooks running:

```
$ oc logs -f jobs/deploy-openstack-default
```

   Additionally, you can manually access the executed Ansible playbooks by logging into the **openstackclient** pod. In the **/home/cloud-admin/work/directory** you can find the ansible playbooks and the **ansible.log** file for the current deployment.

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 8.4 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-4.10-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' --truncate
/etc/machine-id
```

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

```
$ virtctl image-upload dv <datavolume_name> -n openstack \
 --size=<size> --image-path=<local_path_to_image> \
 --storage-class <storage_class> --access-mode <access_mode> --insecure
```

   - Replace **<datavolume_name>** with the name of the data volume, for example, **openstack-base-img**.

- Replace **<size>** with the size of the data volume required for your environment, for example, **500Gi**. The minimum size is 500GB.
- Replace **<storage_class>** with the required storage class from your cluster. Use the following command to retrieve the available storage classes:

```
$ oc get storageclass
```

- Replace **<access_mode>** with the access mode for the PVC. The default value is **ReadWriteOnce**.

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

1. 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

1. View the Secret resource:

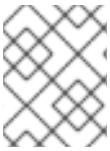
```
$ oc get secret/git-secret -n openstack
```

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:

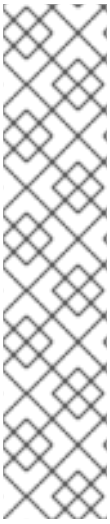
```
apiVersion: v1
kind: Secret
metadata:
```

```
  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
```



NOTE

Enter the **userpassword** Secret in **passwordSecret** when you create **OpenStackControlPlane** or **OpenStackBaremetalSet**:

```
apiVersion: osp-director.openstack.org/v1beta2
kind: OpenStackControlPlane
metadata:
  name: overcloud
  namespace: openstack
spec:
  passwordSecret: <userpassword>
```

- Replace **<userpassword>** with your **userpassword** Secret.

Additional resources

- "Providing sensitive data to pods"

8.4. CREATING AN OVERCLOUD CONTROL PLANE NETWORK WITH OPENSTACKNETCONFIG

You must define at least one control plane network for your overcloud in OpenStackNetConfig. In addition to IP address assignment, the network definition includes the mapping information for OpenStackNetAttachment. OpenShift Virtualization uses this information 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 **osnetconfig.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: OpenStackNetConfig
metadata:
  name: openstacknetconfig
spec:
  attachConfigurations:
    br-osp:
      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
            mtu: 1500
  # optional DnsServers list
  dnsServers:
  - 192.168.25.1
  # optional DnsSearchDomains list
  dnsSearchDomains:
  - osptest.test.metalkube.org
  - some.other.domain
  # DomainName of the OSP environment
  domainName: osptest.test.metalkube.org
  networks:
  - name: Control
    nameLower: ctlplane
    subnets:
    - name: ctlplane
      ipv4:
        allocationEnd: 172.22.0.250
        allocationStart: 172.22.0.100
        cidr: 172.22.0.0/24
        gateway: 172.22.0.1
      attachConfiguration: br-osp
  # optional: configure static mapping for the networks per nodes. If there is none, a random
gets created
  reservations:
    controller-0:
      ipReservations:
        ctlplane: 172.22.0.120
    compute-0:
      ipReservations:
        ctlplane: 172.22.0.140
```

Set the following values in the networks specification:

name

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

nameLower

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

subnets

Set the subnet specifications.

subnets.name

Set the name of the control plane subnet, which is ctlplane.

subnets.attachConfiguration

Set the reference to which of the attach configuration should be used.

subnets.ipv4

Details of the ipv4 subnet with allocationStart, allocationEnd, cidr, gateway and optional list of routes (with destination and nexthop)

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

```
$ oc describe crd openstacknetconfig
```

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

2. Create the control plane network:

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

Verification

1. View the resource for the control plane network:

```
$ oc get openstacknetconfig/openstacknetconfig
```

8.5. CREATING VLAN NETWORKS FOR NETWORK ISOLATION WITH OPENSTACKNETCONFIG

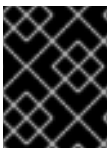
You must create additional networks to implement network isolation for your composable networks. To accomplish this network isolation, you can place your composable networks on individual VLAN networks. In addition to IP address assignment, the OpenStackNetConfig resource includes information to define 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 OpenStackNetConfig resource which defines each network.

Table 8.1. Default Red Hat OpenStack Platform networks

Network	VLAN	CIDR	Allocation
External	10	10.0.0.0/24	10.0.0.10 - 10.0.0.250
InternalApi	20	172.17.0.0/24	172.17.0.10 - 172.17.0.250

Network	VLAN	CIDR	Allocation
Storage	30	172.18.0.0/24	172.18.0.10 - 172.18.0.250
StorageMgmt	40	172.19.0.0/24	172.19.0.10 - 172.19..250
Tenant	50	172.20.0.0/24	172.20.0.10 - 172.20.0.250



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 network configuration. Include the resource specification for the VLAN network. For example, the specification for internal API, storage, storage mgmt, tenant, and external network that manages VLAN-tagged traffic over Linux bridges **br-ex** and **br-osp** connected to the **enp6s0** and **enp7s0** Ethernet device on each worker node is as follows:

```
kind: OpenStackNetConfig
metadata:
  name: openstacknetconfig
spec:
  attachConfigurations:
    br-osp:
      nodeNetworkConfigurationPolicy:
        nodeSelector:
          node-role.kubernetes.io/worker: ""
        desiredState:
          interfaces:
          - bridge:
              options:
                stp:
                  enabled: false
              port:
              - name: enp7s0
            description: Linux bridge with enp7s0 as a port
            name: br-osp
            state: up
            type: linux-bridge
            mtu: 1500
    br-ex:
      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-ex
          state: up
          type: linux-bridge
          mtu: 1500
# optional DnsServers list
dnsServers:
- 172.22.0.1
# optional DnsSearchDomains list
dnsSearchDomains:
- osptest.test.metalkube.org
- some.other.domain
# DomainName of the OSP environment
domainName: osptest.test.metalkube.org
networks:
- name: Control
  nameLower: ctlplane
  subnets:
  - name: ctlplane
    ipv4:
      allocationEnd: 172.22.0.250
      allocationStart: 172.22.0.10
      cidr: 172.22.0.0/24
      gateway: 172.22.0.1
    attachConfiguration: br-osp
- name: InternalApi
  nameLower: internal_api
  mtu: 1350
  subnets:
  - name: internal_api
    attachConfiguration: br-osp
    vlan: 20
    ipv4:
      allocationEnd: 172.17.0.250
      allocationStart: 172.17.0.10
      cidr: 172.17.0.0/24
- name: External
  nameLower: external
  subnets:
  - name: external
    ipv4:
      allocationEnd: 10.0.0.250
      allocationStart: 10.0.0.10
      cidr: 10.0.0.0/24
      gateway: 10.0.0.1
    attachConfiguration: br-ex
- name: Storage
```

```
    nameLower: storage
    mtu: 1500
    subnets:
    - name: storage
      ipv4:
        allocationEnd: 172.18.0.250
        allocationStart: 172.18.0.10
        cidr: 172.18.0.0/24
      vlan: 30
      attachConfiguration: br-osp
  - name: StorageMgmt
    nameLower: storage_mgmt
    mtu: 1500
    subnets:
    - name: storage_mgmt
      ipv4:
        allocationEnd: 172.19.0.250
        allocationStart: 172.19.0.10
        cidr: 172.19.0.0/24
      vlan: 40
      attachConfiguration: br-osp
  - name: Tenant
    nameLower: tenant
    vip: False
    mtu: 1500
    subnets:
    - name: tenant
      ipv4:
        allocationEnd: 172.20.0.250
        allocationStart: 172.20.0.10
        cidr: 172.20.0.0/24
      vlan: 50
      attachConfiguration: br-osp
```

When you use 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 that the network template for the **OpenStackVMSet** is a multi-NIC network template.

Set the following values in the resource specification:

metadata.name

Set to the name of the OpenStackNetConfig.

spec

Set the network configuration for attaching the networks and the network specifics. For descriptions of the values you can use in this section, view the specification schema in the custom resource definition for the **openstacknetconfig** CRD:

```
-
```

```
$ oc describe crd openstacknetconfig
```

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

2. Create the network configuration:

```
$ oc apply -f openstacknetconfig.yaml -n openstack
```

Verification

1. View the OpenStackNetConfig API and created child resources:

```
$ oc get openstacknetconfig/openstacknetconfig -n openstack
$ oc get openstacknetattachment -n openstack
$ oc get openstacknet -n openstack
```

If you see errors, check the underlying **network-attach-definition** and node network configuration policies:

```
$ oc get network-attachment-definitions -n openstack
$ oc get nncp
```

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 OpenStackNetConfig 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/v1beta2
kind: OpenStackControlPlane
```

```
metadata:
  name: overcloud
  namespace: openstack
spec:
  openStackClientNetworks:
        - ctlplane
        - internal_api
        - external
  openStackClientStorageClass: host-nfs-storageclass
  passwordSecret: userpassword
  virtualMachineRoles:
    Controller:
      roleName: Controller
      roleCount: 3
      networks:
        - ctlplane
        - internal_api
        - external
        - tenant
        - storage
        - storage_mgmt
      cores: 12
      memory: 64
      rootDisk:
        diskSize: 500
        baseImageVolumeName: openstack-base-img
        # storageClass must support RWX to be able to live migrate VMs
        storageClass: host-nfs-storageclass
        storageAccessMode:  ReadWriteMany
        # When using OpenShift Virtualization with OpenShift Container Platform Container
Storage,
        # specify RBD block mode persistent volume claims (PVCs) when creating virtual
machine disks.
        # With virtual machine disks, RBD block mode volumes are more efficient and provide
better
        # performance than Ceph FS or RBD filesystem-mode PVCs.
        # To specify RBD block mode PVCs, use the 'ocs-storagecluster-ceph-rbd' storage
class and
        # VolumeMode: Block.
        storageVolumeMode: Filesystem
      # optional configure additional discs to be attached to the VMs,
      # need to be configured manually inside the VMs where to be used.
      additionalDisks:
        - name: datadisk
          diskSize: 500
          storageClass: host-nfs-storageclass
          storageAccessMode:  ReadWriteMany
          storageVolumeMode: Filesystem
  openStackRelease: "16.2"
```

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 -n openstack openstackclient
```

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 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: interface
              name: nic4
              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
      - type: vlan
        mtu:
          get_param: StorageMtu
        device: nic4
        vlan_id:
          get_param: StorageNetworkVlanID
        addresses:
        - ip_netmask:
            get_param: StorageIpSubnet
        routes:
          list_concat_unique:
            - get_param: StorageInterfaceRoutes
      - type: vlan
        mtu:
          get_param: InternalApiMtu
        device: nic4
        vlan_id:
          get_param: InternalApiNetworkVlanID
        addresses:
        - ip_netmask:
            get_param: InternalApiIpSubnet
        routes:
          list_concat_unique:
            - get_param: InternalApiInterfaceRoutes

      - type: ovs_bridge
        # This will default to br-ex, anything else requires specific
        # bridge 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
        - type: vlan
          mtu:
            get_param: TenantMtu
          vlan_id:
            get_param: TenantNetworkVlanID
          addresses:
```

```
            - ip_netmask:
                get_param: TenantIpSubnet
              routes:
                list_concat_unique:
                  - get_param: TenantInterfaceRoutes
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	N/A	**nic4**
External, Tenant	**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.

8.9. 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

1. View the ConfigMap:

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

Additional resources

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

8.10. 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::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

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

To integrate with an external Ceph Storage cluster, include an environment file with parameters and values similar to those shown in the following example.

```
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

# Change the ContainerImageRegistryCredentials to a registry service account
# OR remove ContainerImageRegistryCredentials and set ContainerCephDaemonImage
# to a local registry not requiring authentication.
ContainerCephDaemonImage: registry.redhat.io/rhceph/rhceph-4-rhel8:latest
ContainerImageRegistryCredentials:
  registry.redhat.io:
    my_username: my_password
```

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

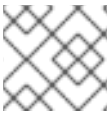
The Ceph client key of your external Ceph Storage cluster.

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.

A Ceph container is required by the **ceph-ansible** client role to configure the Ceph client. You must set the **ContainerCephDaemonImage** parameter. The Ceph container, hosted at **registry.redhat.io**, requires authentication. For more information on the **ContainerImageRegistryLogin** parameter see, Transitioning to Containerized Services

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
- Red Hat Container Registry Authentication

8.12. 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

1. View the ConfigMap:

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

Additional resources

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

8.13. 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 OpenStackNetConfig 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
  # If you manually created an OpenStackProvisionServer, you can use it here,
  # otherwise the director Operator will create one for you (with `baseImageUrl` as the image
that it server)
  # to use with this OpenStackBaremetalSet
  # provisionServerName: openstack-provision-server
  ctlplaneInterface: enp2s0
  networks:
    - ctlplane
    - internal_api
    - 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
```

8.14. CREATING ANSIBLE PLAYBOOKS FOR OVERCLOUD CONFIGURATION WITH OPENSTACKCONFIGGENERATOR

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 OpenStackConfigGenerator 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.
- OpenStackControlPlane and OpenStackBaremetalSets created as required.
- 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-config-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: OpenStackConfigGenerator
metadata:
  name: default
  namespace: openstack
spec:
  enableFencing: true
  gitSecret: git-secret
  imageURL: registry.redhat.io/rhosp-rhel8/openstack-tripleoclient:16.2
  heatEnvConfigMap: heat-env-config
  # List of heat environment files to include from tripleo-heat-templates/environments
  heatEnvs:
  - ssl/tls-endpoints-public-dns.yaml
  - ssl/enable-tls.yaml
  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, by default **default**.

metadata.namespace

Set to the director Operator namespace, by default **openstack**.

spec.enableFencing

Enable the automatic creation of required heat environment files to enable fencing.

NOTE

Production OSP environments must have fencing enabled. Virtual machines running pacemaker require the **fence-agents-kubevirt** package.

spec.gitSecret

Set to the ConfigMap that contains the Git authentication credentials, by default **git-secret**.

spec.heatEnvs

A list of default tripleo environment files used to generate the playbooks.

spec.heatEnvConfigMap

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

spec.tarballConfigMap

Set to the ConfigMap that contains the tarball with your custom heat templates, by default **tripleo-tarball-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 **openstackconfiggenerator** CRD:

```
$ oc describe crd openstackconfiggenerator
```

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

2. Create the Ansible config generator:

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

Verification

1. View the resource for the config generator:

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

8.15. 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.

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 (RHOSP) commands. This pod also contains an ansible inventory script named **/home/cloud-admin/ctlplane-ansible-inventory**.

To register your nodes, you can use the **redhat_subscription** Ansible module with the inventory script from the OpentackClient pod.

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.

Procedure

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

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

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

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

3. 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
      - 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.

4. Register the overcloud nodes to required repositories:

```
ansible-playbook -i /home/cloud-admin/ctlplane-ansible-inventory ./rhsm.yaml
```

Additional resources

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

8.16. 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 OpenStackDeploy resource, the director Operator creates a job that runs the ansible playbooks 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 OpentackConfigGenerator to create the Ansible playbook configuration for your overcloud.
- Use the OpeenstackConfigVersion to select the hash/digest of the ansible playbooks which should be used to configure the overcloud.

Procedure

1. Create a file named **openstack-deployment.yaml** on your workstation. Include the resource specification to the Ansible playbooks. For example:

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

```
kind: OpenStackDeploy
metadata:
  name: default
spec:
  configVersion: n5fch96h548h75hf4hbdhb8hfdh676h57bh96h5c5h59hf4h88h…
  configGenerator: default
```

Set the following values in the resource specification:

metadata.name

Set the name of the Compute node baremetal set, by default **default**.

metadata.namespace

Set to the diretor Operator namespace, by default **openstack**.

spec.configVersion

The config version/git hash of the playbooks to deploy.

spec.configGenerator

The name of the configGenerator.

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

```
$ oc describe crd openstackdeploy
```

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

2. Create the OpenStackDeploy resource:

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

As the deployment runs it creates a Kubernetes job to execute the Ansible playbooks. You can tail the logs of the job to watch the Ansible playbooks running:

```
$ oc logs -f jobs/deploy-openstack-default
```

Additionally, you can manually access the executed Ansible playbooks by logging into the **openstackclient** pod. In the **/home/cloud-admin/work/directory** you can find the ansible playbooks and the **ansible.log** file for the current deployment.

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 -n openstack openstackclient
```

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

If 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 a new node is provisioned, a new OpenStackConfigGenerator resource is created to generate a new set of Ansible playbooks. Use the OpenStackConfig Version to create or update the OpenStackDeploy object to reapply the 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
$ oc get openstackbaremetalset
```

3. Generate the Ansible Playbooks using OpenStackConfigGenerator, see Configuring overcloud software with the director Operator.

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.

NOTE

If you scale the overcloud with a new node in the same role, the node reuses the host names starting with the lowest ID suffix and corresponding IP reservation.

Prerequisites

- The workloads on the Compute nodes have been migrated to other Compute nodes. For more information, see Migrating virtual machine instances between Compute nodes .

Procedure

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

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

2. Identify the Compute node that you want to remove:

```
$ openstack compute service list
```

3. Disable the Compute service on the node to prevent the node from scheduling new instances:

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

4. Annotate the bare-metal node to prevent Metal^3 from starting the node:

```
$ oc annotate baremetalhost <node> baremetalhost.metal3.io/detached=true
$ oc logs --since=1h <metal3-pod> metal3-baremetal-operator | grep -i detach
$ oc get baremetalhost <node> -o json | jq .status.operationalStatus
"detached"
```

   - Replace **<node>** with the name of the **BareMetalHost** resource.
   - Replace **<metal3-pod>** with the name of your **metal3** pod.

5. Log in to the Compute node as the **root** user and shut down the bare-metal node:

```
[root@compute-0 ~]# shutdown -h now
```

   If the Compute node is not accessible, complete the following steps:

   a. Log in to a Controller node as the **root** user.

   b. If Instance HA is enabled, disable the STONITH device for the Compute node:

```
[root@controller-0 ~]# pcs stonith disable <stonith_resource_name>
```

      - Replace **<stonith_resource_name>** with the name of the STONITH resource that corresponds to the node. The resource name uses the the format **<resource_agent>-<host_mac>**. You can find the resource agent and the host MAC address in the

FencingConfig section of the **fencing.yaml** file.

c. Use IPMI to power off the bare-metal node. For more information, see your hardware vendor documentation.

6. Retrieve the **BareMetalHost** resource that corresponds to the node that you want to remove:

```
$ oc get openstackbaremetalset compute -o json | jq '.status.baremetalHosts | to_entries[] | "\
(.key) => \(.value | .hostRef)"'
"compute-0, openshift-worker-3"
"compute-1, openshift-worker-4"
```

7. To change the status of the **annotatedForDeletion** parameter to **true** in the **OpenStackBaremetalSet** resource, annotate the **BareMetalHost** resource with **osp-director.openstack.org/delete-host=true**:

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

8. Optional: Confirm that the **annotatedForDeletion** status has changed to **true** in the **OpenStackBaremetalSet** resource:

```
$ oc get openstackbaremetalset 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"
  }
}
```

9. Decrease the **count** parameter for the **compute OpenStackBaremetalSet** resource:

```
$ oc patch openstackbaremetalset 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:

- Director Operator deletes the corresponding IP reservations from **OpenStackIPSet** and **OpenStackNetConfig** for the node.
- Director Operator flags the IP reservation entry in the **OpenStackNet** resource as deleted:

```
$ oc get osnet ctlplane -o json -n openstack | jq .status.reservations
{
  "compute-0": {
    "deleted": true,
    "ip": "172.22.0.140"
  },
  "compute-1": {
    "deleted": false,
    "ip": "172.22.0.100"
  },
  "controller-0": {
    "deleted": false,
    "ip": "172.22.0.120"
  },
  "controlplane": {
    "deleted": false,
    "ip": "172.22.0.110"
  },
  "openstackclient-0": {
    "deleted": false,
    "ip": "172.22.0.251"
  }
```

10. Optional: To make the IP reservations of the deleted **OpenStackBaremetalSet** resource available for other roles to use, set the value of the **spec.preserveReservations** parameter to false in the **OpenStackNetConfig** object.
11. Access the remote shell for **openstackclient**:

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

12. Remove the Compute service entries from the overcloud:

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

13. 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
```

14. Exit from **openstackclient**:

```
$ exit
```

CHAPTER 11. UPDATING THE OVERCLOUD FOR DIRECTOR OPERATOR

After you update the **openstackclient** pod, update the overcloud by running the overcloud and container image preparation deployments, updating your nodes, and running the overcloud update converge deployment. During a minor update, the control plane API is available.

11.1. PREPARING DIRECTOR OPERATOR FOR A MINOR UPDATE

The Red Hat OpenStack Platform (RHOSP) minor update process workflow:

1. Prepare your environment for the RHOSP minor update.
2. Update the **openstackclient** pod image to the latest OpenStack 16.2.z version.
3. Update the overcloud to the latest OpenStack 16.2.z version.
4. Update all Red Hat Ceph Storage services.
5. Run the convergence deployment to refresh your overcloud stack.

11.1.1. Locking the environment to a Red Hat Enterprise Linux release

Red Hat OpenStack Platform (RHOSP) 16.2 is supported on Red Hat Enterprise Linux (RHEL) 8.4. Before you perform the update, lock the overcloud repositories to the RHEL 8.4 release to avoid upgrading the operating system to a newer minor release.

Procedure

1. Copy the **rhsm.yaml** file to **openstackclient**:

```
$ oc cp rhsm.yaml openstackclient:/home/cloud-admin/rhsm.yaml
```

2. Open a remote shell on the **openstackclient** pod:

```
$ oc rsh openstackclient
```

3. Open the **rhsm.yaml** file and check if your subscription management configuration includes the **rhsm_release** parameter. If the **rhsm_release** parameter is not present, add it and set it to **8.4**:

```
parameter_defaults:
  RhsmVars:
    …
    rhsm_username: "myusername"
    rhsm_password: "p@55w0rd!"
    rhsm_org_id: "1234567"
    rhsm_pool_ids: "1a85f9223e3d5e43013e3d6e8ff506fd"
    rhsm_method: "portal"
    rhsm_release: "8.4"
```

4. Save the overcloud subscription management environment file.

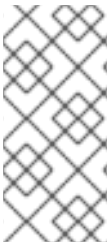
5. Create a playbook that contains a task to lock the operating system version to RHEL 8.4 on all nodes:

```
$ cat > ~/set_release.yaml <<'EOF'
- hosts: all
  gather_facts: false
  tasks:
    - name: set release to 8.4
      command: subscription-manager release --set=8.4
      become: true
EOF
```

6. Run the ansible playbook on the **openstackclient** pod:

```
$ ansible-playbook -i /home/cloud-admin/ctlplane-ansible-inventory /home/cloud-admin/set_release.yaml --limit Controller,Compute
```

Use the **--limit** option to apply the content to all RHOSP nodes. Do not run this playbook against Red Hat Ceph Storage nodes because you are probably using a different subscription for these nodes.



NOTE

To manually lock a node to a version, log in to the node and run the **subscription-manager release** command:

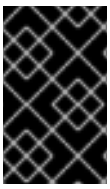
```
$ sudo subscription-manager release --set=8.4
```

11.1.2. Changing to Extended Update Support (EUS) repositories

Your Red Hat OpenStack Platform (RHOSP) subscription includes repositories for Red Hat Enterprise Linux (RHEL) 8.4 Extended Update Support (EUS). The EUS repositories include the latest security patches and bug fixes for RHEL 8.4. Switch to the following repositories before you perform an update.

Table 11.1. EUS repositories for RHEL 8.4

Standard repository	EUS repository
rhel-8-for-x86_64-baseos-rpms	rhel-8-for-x86_64-baseos-eus-rpms
rhel-8-for-x86_64-appstream-rpms	rhel-8-for-x86_64-appstream-eus-rpms
rhel-8-for-x86_64-highavailability-rpms	rhel-8-for-x86_64-highavailability-eus-rpms



IMPORTANT

You must use EUS repositories to retain compatibility with a specific version of Podman. Later versions of Podman are untested with RHOSP 16.2 and can cause unexpected results.

Prerequisites

- Copy the **rhsm.yaml** file for the **openstackclient** pod to the **/home/cloud-admin** directory.

Procedure

1. Open a remote shell on the **openstackclient** pod:

```
$ oc rsh openstackclient
```

2. Open the **rhsm.yaml** file and check the **rhsm_repos** parameter in your subscription management configuration. If this parameter does not include the EUS repositories, change the relevant repositories to the EUS versions:

```
parameter_defaults:
  RhsmVars:
    rhsm_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
      - openstack-16.2-for-rhel-8-x86_64-rpms
      - rhceph-4-tools-for-rhel-8-x86_64-rpms
      - fast-datapath-for-rhel-8-x86_64-rpms
```

3. Save the overcloud subscription management environment file.

4. Create a playbook that contains a task to set the repositories to **RHEL 8.4 EUS** on all nodes:

```
$ cat > ~/change_eus.yaml <<'EOF'
- hosts: all
  gather_facts: false
  tasks:
    - name: change to eus repos
      command: subscription-manager repos --disable=rhel-8-for-x86_64-baseos-rpms --
disable=rhel-8-for-x86_64-appstream-rpms --disable=rhel-8-for-x86_64-highavailability-rpms
--enable=rhel-8-for-x86_64-baseos-eus-rpms --enable=rhel-8-for-x86_64-appstream-eus-
rpms --enable=rhel-8-for-x86_64-highavailability-eus-rpms
      become: true
EOF
```

5. Run the **change_eus.yaml** playbook:

```
$ ansible-playbook -i /home/cloud-admin/ctlplane-ansible-inventory /home/cloud-
admin/change_eus.yaml --limit Controller,Compute
```

Use the **--limit** option to apply the content to all RHOSP nodes. Do not run this playbook against Red Hat Ceph Storage nodes because they use a different subscription.

11.1.3. Updating Red Hat Openstack Platform and Ansible repositories

Update your repositories to use Red Hat OpenStack Platform (RHOSP) 16.2 and Ansible 2.9 packages. For more information, see Overcloud repositories.

Prerequisites

- You have copied the **rhsm.yaml** file for the **openstackclient** pod to the **/home/cloud-admin** directory.

Procedure

1. Open a remote shell on the **openstackclient** pod:

```
$ oc rsh openstackclient
```

2. Open the **rhsm.yaml** file and check the **rhsm_repos** parameter in your subscription management configuration. If the **rhsm_repos** parameter is using the RHOSP 16.1 and Ansible 2.8 repositories, change the repository to the correct versions:

```
parameter_defaults:
  RhsmVars:
    rhsm_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
      - openstack-16.2-for-rhel-8-x86_64-rpms
      - fast-datapath-for-rhel-8-x86_64-rpms
```

3. Save the overcloud subscription management environment file.

4. Create a playbook that contains a task to set the repositories to **RHOSP {osp_curr_ver}** on all RHOSP nodes:

```
$ cat > ~/update_rhosp_repos.yaml <<'EOF'
- hosts: all
  gather_facts: false
  tasks:
    - name: change osp repos
      command: subscription-manager repos --disable=openstack-16.1-for-rhel-8-x86_64-rpms
--enable=openstack-16.2-for-rhel-8-x86_64-rpms --disable=ansible-2.8-for-rhel-8-x86_64-
rpms --enable=ansible-2.9-for-rhel-8-x86_64-rpms
      become: true
EOF
```

5. Run the **update_rhosp_repos.yaml** playbook:

```
$ ansible-playbook -i /home/cloud-admin/ctlplane-ansible-inventory /home/cloud-
admin/update_rhosp_repos.yaml --limit Controller,Compute
```

Use the **--limit** option to apply the content to all RHOSP nodes. Do not run this playbook against Red Hat Ceph Storage nodes because they use a different subscription.

6. Create a playbook that contains a task to set the repositories to **RHOSP {osp_curr_ver}** on all Red Hat Ceph Storage nodes:

```
$ cat > ~/update_ceph_repos.yaml <<'EOF'
- hosts: all
  gather_facts: false
  tasks:
```

```
    - name: change ceph repos
      command: subscription-manager repos --disable=openstack-16-deployment-tools-for-rhel-8-x86_64-rpms --enable=openstack-16.2-deployment-tools-for-rhel-8-x86_64-rpms --disable=ansible-2.8-for-rhel-8-x86_64-rpms --enable=ansible-2.9-for-rhel-8-x86_64-rpms
      become: true
EOF
```

7. Run the **update_ceph_repos.yaml** playbook:

```
$ ansible-playbook -i /home/cloud-admin/ctlplane-ansible-inventory /home/cloud-admin/update_ceph_repos.yaml --limit CephStorage
```

Use the **--limit** option to apply the content to Red Hat Ceph Storage nodes.

11.1.4. Setting the container-tools version

Set the **container-tools** module to version **2.0** to ensure you use the correct package versions on all nodes.

Procedure

1. Open a remote shell on the **openstackclient** pod:

```
$ oc rsh openstackclient
```

2. Create a playbook that contains a task to set the **container-tools** module to version **3.0** on all nodes:

```
$ cat > ~/container-tools.yaml <<'EOF'
- hosts: all
  gather_facts: false
  tasks:
    - name: disable default dnf module for container-tools
      command: dnf module reset container-tools
      become: true
    - name: set dnf module for container-tools:3.0
      command: dnf module enable -y container-tools:3.0
      become: true
    - name: disable dnf module for virt:8.2
      command: dnf module disable -y virt:8.2
      become: true
    - name: set dnf module for virt:rhel
      command: dnf module enable -y virt:rhel
      become: true
EOF
```

3. Run the **container-tools.yaml** playbook against all nodes:

```
$ ansible-playbook -i /home/cloud-admin/ctlplane-ansible-inventory ~/container-tools.yaml
```

11.1.5. Updating the container image preparation file

The container preparation file is the file that contains the **ContainerImagePrepare** parameter. You use this file to define the rules for obtaining container images for the overcloud.

Before you update your environment, check the file to ensure that you obtain the correct image versions.

Procedure

1. Edit the container preparation file. The default name for this file is usually **containers-prepare-parameter.yaml**.
2. Check the **tag** parameter is set to **16.2** for each rule set:

```
parameter_defaults:
  ContainerImagePrepare:
  - push_destination: true
    set:
      ...
      tag: '16.2'
    tag_from_label: '{version}-{release}'
```

NOTE

If you do not want to use a specific tag for the update, such as **16.2** or **16.2.2**, remove the **tag** key-value pair and specify **tag_from_label** only. This uses the installed Red Hat OpenStack Platform version to determine the value for the tag to use as part of the update process.

3. Save this file.

11.1.6. Disabling fencing in the overcloud

Before you update the overcloud, ensure that fencing is disabled.

If fencing is deployed in your environment during the Controller nodes update process, the overcloud might detect certain nodes as disabled and attempt fencing operations, which can cause unintended results.

If you have enabled fencing in the overcloud, you must temporarily disable fencing for the duration of the update to avoid any unintended results.

Procedure

1. Open a remote shell on the **openstackclient** pod:

```
$ oc rsh openstackclient
```

2. Log in to a Controller node and run the Pacemaker command to disable fencing:

```
ssh <controller-0.ctlplane> "sudo pcs property set stonith-enabled=false"
```

- Replace **<controller-0.ctlplane>** with the name of your Controller node.

3. In the **fencing.yaml** environment file, set the **EnableFencing** parameter to **false** to ensure that fencing stays disabled during the update process.

Additional Resources

- Fencing Controller nodes with STONITH

11.2. RUNNING THE OVERCLOUD UPDATE PREPARATION FOR DIRECTOR OPERATOR

To prepare the overcloud for the update process, generate an update prepare configuration, which creates updated ansible playbooks and prepares the nodes for the update.

Procedure

1. Modify the heat parameter ConfigMap called **fencing.yaml**, to disable fencing for the duration of the update:

```
parameter_defaults:
  EnableFencing: false
```

2. Create an OpenStackConfigGenerator resource called **osconfiggenerator-update-prepare.yaml**:

```
$ cat <<EOF > osconfiggenerator-update-prepare.yaml
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackConfigGenerator
metadata:
  name: "update"
  namespace: openstack
spec:
  gitSecret: git-secret
  heatEnvs:
    - lifecycle/update-prepare.yaml
  heatEnvConfigMap: heat-env-config-update
  tarballConfigMap: tripleo-tarball-config-update
EOF
```

3. Apply the configuration:

```
$ oc apply -f osconfiggenerator-update-prepare.yaml
```

4. Wait until the update preparation process completes.

11.3. RUNNING THE CONTAINER IMAGE PREPARATION FOR DIRECTOR OPERATOR

Before you can update the overcloud, you must prepare all container image configurations that are required for your environment.

To complete the container image preparation, you must run the overcloud deployment against tasks that have the **container_image_prepare** tag.

Procedure

1. Create an **osdeploy** job called **osdeploy-container-image-prepare.yaml**:

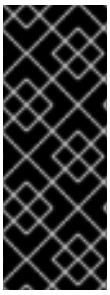
```
$ cat <<EOF > osdeploy-container-image-prepare.yaml
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackDeploy
metadata:
  name: container_image_prepare
spec:
  configVersion: <config_version>
  configGenerator: update
  mode: external-update
  advancedSettings:
    tags:
      - container_image_prepare
EOF
```

2. Apply the configuration:

```
$ oc apply -f osdeploy-container-image-prepare.yaml
```

11.4. OPTIONAL: UPDATING THE OVN-CONTROLLER CONTAINER ON ALL OVERCLOUD SERVERS

If you deployed your overcloud with the Modular Layer 2 Open Virtual Network mechanism driver (ML2/OVN), update the ovn-controller container to the latest RHOSP 16.2 version. The update occurs on every overcloud server that runs the ovn-controller container.



IMPORTANT

The following procedure updates the ovn-controller containers on servers that are assigned the Compute role before it updates the ovn-northd service on servers that are assigned the Controller role.
If you accidentally updated the ovn-northd service before following this procedure, you might not be able to reach your virtual machines or create new virtual machines or virtual networks. The following procedure restores connectivity.

Procedure

1. Create an **osdeploy** job called **osdeploy-ovn-update.yaml**:

```
$ cat <<EOF > osdeploy-ovn-update.yaml
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackDeploy
metadata:
  name: ovn-update
spec:
  configVersion: <config_version>
  configGenerator: update
  mode: update
  advancedSettings:
```

```
    tags:
      - ovn
EOF
```

2. Apply the configuration:

```
$ oc apply -f osdeploy-ovn-update.yaml
```

3. Wait until the ovn-controller container update completes.

11.5. UPDATING ALL CONTROLLER NODES ON DIRECTOR OPERATOR

Update all the Controller nodes to the latest Red Hat OpenStack Platform (RHOSP) 16.2 version.

IMPORTANT

Until BZ#1872404 is resolved, for nodes based on composable roles, you must update the **Database** role first, before you can update **Controller**, **Messaging**, **Compute**, **Ceph**, and other roles.

Procedure

1. Create an **osdeploy** job called **osdeploy-controller-update.yaml**:

```
$ cat <<EOF > osdeploy-controller-update.yaml
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackDeploy
metadata:
  name: controller-update
spec:
  configVersion: <config_version>
  configGenerator: update
  mode: update
  advancedSettings:
    limit: Controller
EOF
```

2. Apply the configuration:

```
$ oc apply -f osdeploy-controller-update.yaml
```

3. Wait until the Controller node update completes.

11.6. UPDATING ALL COMPUTE NODES ON DIRECTOR OPERATOR

Update all Compute nodes to the latest Red Hat OpenStack Platform (RHOSP) 16.2 version. To update Compute nodes, run a deployment with the **limit: Compute** option to restrict operations to the Compute nodes only.

Procedure

1. Create an **osdeploy** job called **osdeploy-compute-update.yaml**:

```
$ cat <<EOF > osdeploy-compute-update.yaml
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackDeploy
metadata:
  name: compute-update
spec:
  configVersion: <config_version>
  configGenerator: update
  mode: update
  advancedSettings:
    limit: Compute
EOF
```

2. Apply the configuration:

```
$ oc apply -f osdeploy-compute-update.yaml
```

3. Wait until the Compute node update completes.

11.7. UPDATING ALL HCI COMPUTE NODES ON DIRECTOR OPERATOR

Update the Hyperconverged Infrastructure (HCI) Compute nodes to the latest Red Hat OpenStack Platform (RHOSP) 16.2 version. To update the HCI Compute nodes, run a deployment with the **limit: ComputeHCI** option to restrict operations to only the HCI nodes. You must also run a deployment with the **mode: external-update** and **tags: ["ceph"]** options to perform an update to a containerized Red Hat Ceph Storage 4 cluster.

Procedure

1. Create an **osdeploy** job called **osdeploy-computehci-update.yaml**:

```
$ cat <<EOF > osdeploy-computehci-update.yaml
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackDeploy
metadata:
  name: computehci-update
spec:
  configVersion: <config_version>
  configGenerator: update
  mode: update
  advancedSettings:
    limit: ComputeHCI
EOF
```

2. Apply the configuration:

```
$ oc apply -f osdeploy-computehci-update.yaml
```

3. Wait until the ComputeHCI node update completes.

4. Create an **osdeploy** job called **osdeploy-ceph-update.yaml**:

```
$ cat <<EOF > osdeploy-ceph-update.yaml
```

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackDeploy
metadata:
  name: ceph-update
spec:
  configVersion: <config_version>
  configGenerator: update
  mode: external-update
  advancedSettings:
    tags:
      - ceph
EOF
```

5. Apply the configuration:

```
$ oc apply -f osdeploy-ceph-update.yaml
```

6. Wait until the Red Hat Ceph Storage node update completes.

11.8. UPDATING ALL RED HAT CEPH STORAGE NODES ON DIRECTOR OPERATOR

Update the Red Hat Ceph Storage nodes to the latest Red Hat OpenStack Platform (RHOSP) 16.2 version.

IMPORTANT

RHOSP 16.2 is supported on RHEL 8.4. However, hosts that are mapped to the CephStorage role update to the latest major RHEL release. For more information, see Red Hat Ceph Storage: Supported configurations .

Procedure

1. Create an **osdeploy** job called **osdeploy-cephstorage-update.yaml**:

```
$ cat <<EOF > osdeploy-cephstorage-update.yaml
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackDeploy
metadata:
  name: cephstorage-update
spec:
  configVersion: <config_version>
  configGenerator: update
  mode: update
  advancedSettings:
    limit: CephStorage
EOF
```

2. Apply the configuration:

```
$ oc apply -f osdeploy-cephstorage-update.yaml
```

3. Wait until the Red Hat Ceph Storage node update completes.

4. Create an **osdeploy** job called **osdeploy-ceph-update.yaml**:

```
$ cat <<EOF > osdeploy-ceph-update.yaml
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackDeploy
metadata:
  name: ceph-update
spec:
  configVersion: <config_version>
  configGenerator: update
  mode: external-update
  advancedSettings:
    tags:
      - ceph
EOF
```

5. Apply the configuration:

```
$ oc apply -f osdeploy-ceph-update.yaml
```

6. Wait until the Red Hat Ceph Storage node update completes.

11.9. PERFORMING ONLINE DATABASE UPDATES ON DIRECTOR OPERATOR

Some overcloud components require an online update or migration of their databases tables.

Online database updates apply to the following components:

- OpenStack Block Storage (cinder)
- OpenStack Compute (nova)

Procedure

1. Create an **osdeploy** job called **osdeploy-online-migration.yaml**:

```
$ cat <<EOF > osdeploy-online-migration.yaml
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackDeploy
metadata:
  name: online-migration
spec:
  configVersion: <config_version>
  configGenerator: update
  mode: external-update
  advancedSettings:
    tags:
      - online_upgrade
EOF
```

2. Apply the configuration:

```
$ oc apply -f osdeploy-online-migration.yaml
```

-

11.10. FINALIZING THE UPDATE

To finalize the update to the latest Red Hat OpenStack Platform 16.2 version, you must update the overcloud generated configuration. This ensures that the stack resource structure aligns with a regular deployment of OSP 16.2 and you can perform standard overcloud deployments in the future.

Procedure

1. Re-enable fencing in the **fencing.yaml** environment file:

```
parameter_defaults:
  EnableFencing: true
```

2. Regenerate the default configuration, ensuring that **lifecycle/update-prepare.yaml** is not included in the heatEnvs. For more information, see Configuring overcloud software with the director Operator.

3. Delete OpenStackConfigGenerator, ConfigVersion, and configuration deployment resources.

```
$ oc delete <type> <name>
```

   - Replace <type> with the type of resource to delete.
   - Replace <name> with the name of the resource to delete.

4. Wait until the update finalization completes.

CHAPTER 12. DEPLOYING TLS FOR PUBLIC ENDPOINTS USING DIRECTOR OPERATOR

Deploy the overcloud using TLS to create public endpoint IPs or DNS names for RHOSP Director Operator.

Prerequisites

- Your OpenShift Container Platform cluster is operational.
- You have installed director Operator correctly.
- You have installed the **oc** command line tool on your workstation.

12.1. TLS FOR PUBLIC ENDPOINT IP ADDRESSES

To reference public endpoint IP addresses, add certificates to the **openstackclient** pod.

Prerequisites

- Create the certificate authority, key, and certificate using the procedure: Enabling SSL/TLS on overcloud public endpoints.

Procedure

1. Create a **ConfigMap** to store the CA certificates. The **ConfigMap** is the interface used to add CA certificates to the **openstackclient** pod, using the OpenStackControlPlane object:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cacerts
  namespace: openstack
data:
 local_CA: |
    -----BEGIN CERTIFICATE-----
    …
   -----END CERTIFICATE-----
  another_CA: |
    -----BEGIN CERTIFICATE-----
    …
   -----END CERTIFICATE-----
```

2. Create the OpenStackControlPlane and reference the **ConfigMap**:

```
apiVersion: osp-director.openstack.org/v1beta2
kind: OpenStackControlPlane
metadata:
  name: <overcloud>
  namespace: openstack
spec:
  caConfigMap: cacerts
```

- Replace **<overcloud>** with the name of your stack.

3. In the **~/custom_environment_files** directory create a file called **tls-certs.yaml** containing the generated certificates for the deployment using **SSLCertificate**, **SSLIntermediateCertificate**, **SSLKey**, and **CAMap** parameters.

NOTE

For more information on creating a certificate file, see Enabling SSL/TLS.

4. Update the **heatEnvConfigMap** to add the **tls-certs.yaml** file:

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

5. Create an OpenStackConfigGenerator and add the required **heatEnvs** configuration files to configure TLS for public endpoint IPs:

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackConfigGenerator
…
spec:
  …
  heatEnvs:
    - ssl/tls-endpoints-public-ip.yaml
    - ssl/enable-tls.yaml
  …
  heatEnvConfigMap: heat-env-config
  tarballConfigMap: tripleo-tarball-config
```

6. The OpenStackConfigGenerator and a new OpenStackConfigVersion are created, run the Ansible playbooks against the overcloud using the OpenStackDeploy resource:

- Registering the operating system of your overcloud .
- Obtain the latest OpenStackConfig Version .
- Applying overcloud configuration with the director Operator .

Additional Resources

- Enabling SSL/TLS on overcloud public endpoints .

12.2. TLS FOR PUBLIC ENDPOINT DNS NAMES

To reference public endpoint DNS names add certificates to the **openstackclient** pod.

Prerequisites

- Create the certificate authority, key, and certificate following the procedure in, Enabling SSL/TLS on overcloud public endpoints.

Procedure

1. Create a **ConfigMap** to store the CA certificates. The **ConfigMap** is the interface used to add additional CA certificates to the **openstackclient** pod, using the OpenStackControlPlane object:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cacerts
  namespace: openstack
data:
 local_CA: |
    -----BEGIN CERTIFICATE-----
    …
   -----END CERTIFICATE-----
  another_CA: |
    -----BEGIN CERTIFICATE-----
    …
   -----END CERTIFICATE-----
```

2. Create the OpenStackControlPlane and reference the **ConfigMap**:

```
apiVersion: osp-director.openstack.org/v1beta2
kind: OpenStackControlPlane
metadata:
  name: <overcloud>
  namespace: openstack
spec:
  caConfigMap: cacerts
```

- Replace **<overcloud>** with the name of your stack.

3. In the **~/custom_environment_files** directory create a file called **tls-certs.yaml** containing the generated certificates for the deployment using **SSLCertificate**, **SSLIntermediateCertificate**, **SSLKey**, and **CAMap** parameters.

NOTE

For more information on creating a certificate file, see Enabling SSL/TLS.

4. Update the **heatEnvConfigMap** to add the **tls-certs.yaml** file:

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

5. Create an OpenStackConfigGenerator and add the required **heatEnvs** configuration files to configure TLS for public endpoint DNS names:

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackConfigGenerator
…
spec:
  …
  heatEnvs:
    - ssl/tls-endpoints-public-dns.yaml
```

```
  - ssl/enable-tls.yaml
...
heatEnvConfigMap: heat-env-config
tarballConfigMap: tripleo-tarball-config
```

6. The OpenStackConfigGenerator and a new OpenStackConfigVersion are created, run the Ansible playbooks against the overcloud using the OpenStackDeploy resource:
   - Registering the operating system of your overcloud .
   - Obtain the latest OpenStackConfig Version .
   - Applying overcloud configuration with the director Operator .

Additional Resources

- Enabling SSL/TLS on overcloud public endpoints.

CHAPTER 13. CHANGING SERVICE ACCOUNT PASSWORDS USING DIRECTOR OPERATOR

Red Hat OpenStack Platform (RHOSP) services and the databases that they use are authenticated by their Identity service (keystone) credentials. The Identity service generates these RHOSP passwords during the initial RHOSP deployment process. You might be required to periodically update passwords for threat mitigation or security compliance. You can use tools native to director Operator (OSPdO) to change many of the generated passwords after your RHOSP environment is deployed.

13.1. ROTATING OVERCLOUD SERVICE ACCOUNT PASSWORDS WITH DIRECTOR OPERATOR

You can rotate the overcloud service account passwords used with a director Operator (OSPdO) deployed Red Hat OpenStack Platform (RHOSP) environment.

Procedure

1. Create a backup of the current **tripleo-passwords** secret:

```
$ oc get secret tripleo-passwords -n openstack -o yaml > tripleo-passwords_backup.yaml
```

2. Create a plain text file named **tripleo-overcloud-passwords_preserve_list** to specify that the passwords for the following services should not be rotated:

```
parameter_defaults
BarbicanSimpleCryptoKek
KeystoneCredential0
KeystoneCredential1
KeystoneFernetKey0
KeystoneFernetKey1
KeystoneFernetKeys
CephClientKey
CephClusterFSID
CephManilaClientKey
CephRgwKey
HeatAuthEncryptionKey
MysqlClustercheckPassword
MysqlMariabackupPassword
PacemakerRemoteAuthkey
PcsdPassword
```

   You can add additional services to this list if there are other services for which you want to preserve the password.

3. Create a password parameter file, **tripleo-overcloud-passwords.yaml**, that lists the passwords that should not be modified:

```
$ oc get secret tripleo-passwords -n openstack \
-o jsonpath='{.data.tripleo-overcloud-passwords\.yaml}' \
| base64 -d | grep -f ./tripleo-overcloud-passwords_preserve_list > tripleo-overcloud-passwords.yaml
```

4. Validate that the **tripleo-overcloud-passwords.yaml** file contains the passwords that you do not want to rotate.

5. Update the **tripleo-password** secret:

```
$ oc create secret generic tripleo-passwords -n openstack \
--from-file=./tripleo-overcloud-passwords.yaml \
--dry-run=client -o yaml | oc apply -f -
```

6. Create Ansible playbooks to configure the overcloud with the OpenStackConfigGenerator CRD. For more information, see Creating Ansible playbooks for overcloud configuration with the OpenStackConfigGenerator CRD.

7. Apply the updated configuration. For more information, see Applying overcloud configuration with director Operator.

Verification

Compare the new **NovaPassword** in the secret to what is now installed on the Controller node.

1. Get the password from the updated secret:

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

Example output:

```
NovaPassword: hp4xpt7t2p79ktqjjnxpqwbp6
```

2. Retrieve the password for the Compute service (nova) running on the Controller nodes:

   a. Access the **openstackclient** remote shell:

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

   b. Ensure that you are in the home directory:

```
$ cd
```

   c. Retrieve the Compute service password:

```
$ ansible -i /home/cloud-admin/ctlplane-ansible-inventory Controller -b -a "grep ^connection /var/lib/config-data/puppet-generated/nova/etc/nova/nova.conf"
```

   Example output:

```
172.22.0.120 | CHANGED | rc=0 >>
connection=mysql+pymysql://nova_api:hp4xpt7t2p79ktqjjnxpqwbp6@172.17.0.10/nova_api?read_default_file=/etc/my.cnf.d/tripleo.cnf&read_default_group=tripleo
connection=mysql+pymysql://nova:hp4xpt7t2p79ktqjjnxpqwbp6@172.17.0.10/nova?read_default_file=/etc/my.cnf.d/tripleo.cnf&read_default_group=tripleo
```

CHAPTER 14. DEPLOYING NODES WITH SPINE-LEAF CONFIGURATION BY USING DIRECTOR OPERATOR

Deploy nodes with spine-leaf networking architecture to replicate an extensive network topology within your environment. Current restrictions allow only one provisioning network for **Metal3**.

14.1. CREATING OR UPDATING THE OPENSTACKNETCONFIG CUSTOM RESOURCE TO DEFINE ALL SUBNETS

Define your OpenStackNetConfig custom resource and specify the subnets for the overcloud networks. Director Operator then renders the configuration and creates, or updates, the network topology.

Prerequisites

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

Procedure

1. Create a configuration file called **openstacknetconfig.yaml**:

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackNetConfig
metadata:
  name: openstacknetconfig
spec:
  attachConfigurations:
    br-osp:
      nodeNetworkConfigurationPolicy:
        nodeSelector:
          node-role.kubernetes.io/worker: ""
        desiredState:
          interfaces:
          - bridge:
              options:
                stp:
                  enabled: false
              port:
              - name: enp7s0
            description: Linux bridge with enp7s0 as a port
            name: br-osp
            state: up
            type: linux-bridge
            mtu: 1500
    br-ex:
      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-ex
          state: up
          type: linux-bridge
          mtu: 1500
# optional DnsServers list
dnsServers:
- 192.168.25.1
# optional DnsSearchDomains list
dnsSearchDomains:
- osptest.test.metalkube.org
- some.other.domain
# DomainName of the OSP environment
domainName: osptest.test.metalkube.org
networks:
- name: Control
  nameLower: ctlplane
  subnets:
  - name: ctlplane
    ipv4:
      allocationEnd: 192.168.25.250
      allocationStart: 192.168.25.100
      cidr: 192.168.25.0/24
      gateway: 192.168.25.1
    attachConfiguration: br-osp
- name: InternalApi
  nameLower: internal_api
  mtu: 1350
  subnets:
  - name: internal_api
    ipv4:
      allocationEnd: 172.17.0.250
      allocationStart: 172.17.0.10
      cidr: 172.17.0.0/24
      routes:
      - destination: 172.17.1.0/24
        nexthop: 172.17.0.1
      - destination: 172.17.2.0/24
        nexthop: 172.17.0.1
    vlan: 20
    attachConfiguration: br-osp
  - name: internal_api_leaf1
    ipv4:
      allocationEnd: 172.17.1.250
      allocationStart: 172.17.1.10
      cidr: 172.17.1.0/24
      routes:
      - destination: 172.17.0.0/24
        nexthop: 172.17.1.1
      - destination: 172.17.2.0/24
        nexthop: 172.17.1.1
    vlan: 21
```

```
    attachConfiguration: br-osp
  - name: internal_api_leaf2
    ipv4:
      allocationEnd: 172.17.2.250
      allocationStart: 172.17.2.10
      cidr: 172.17.2.0/24
      routes:
      - destination: 172.17.1.0/24
        nexthop: 172.17.2.1
      - destination: 172.17.0.0/24
        nexthop: 172.17.2.1
    vlan: 22
    attachConfiguration: br-osp
- name: External
  nameLower: external
  subnets:
  - name: external
    ipv4:
      allocationEnd: 10.0.0.250
      allocationStart: 10.0.0.10
      cidr: 10.0.0.0/24
      gateway: 10.0.0.1
    attachConfiguration: br-ex
- name: Storage
  nameLower: storage
  mtu: 1350
  subnets:
  - name: storage
    ipv4:
      allocationEnd: 172.18.0.250
      allocationStart: 172.18.0.10
      cidr: 172.18.0.0/24
      routes:
      - destination: 172.18.1.0/24
        nexthop: 172.18.0.1
      - destination: 172.18.2.0/24
        nexthop: 172.18.0.1
    vlan: 30
    attachConfiguration: br-osp
  - name: storage_leaf1
    ipv4:
      allocationEnd: 172.18.1.250
      allocationStart: 172.18.1.10
      cidr: 172.18.1.0/24
      routes:
      - destination: 172.18.0.0/24
        nexthop: 172.18.1.1
      - destination: 172.18.2.0/24
        nexthop: 172.18.1.1
    vlan: 31
    attachConfiguration: br-osp
  - name: storage_leaf2
    ipv4:
      allocationEnd: 172.18.2.250
      allocationStart: 172.18.2.10
      cidr: 172.18.2.0/24
```

```
      routes:
      - destination: 172.18.0.0/24
        nexthop: 172.18.2.1
      - destination: 172.18.1.0/24
        nexthop: 172.18.2.1
    vlan: 32
    attachConfiguration: br-osp
- name: StorageMgmt
  nameLower: storage_mgmt
  mtu: 1350
  subnets:
  - name: storage_mgmt
    ipv4:
      allocationEnd: 172.19.0.250
      allocationStart: 172.19.0.10
      cidr: 172.19.0.0/24
      routes:
      - destination: 172.19.1.0/24
        nexthop: 172.19.0.1
      - destination: 172.19.2.0/24
        nexthop: 172.19.0.1
    vlan: 40
    attachConfiguration: br-osp
  - name: storage_mgmt_leaf1
    ipv4:
      allocationEnd: 172.19.1.250
      allocationStart: 172.19.1.10
      cidr: 172.19.1.0/24
      routes:
      - destination: 172.19.0.0/24
        nexthop: 172.19.1.1
      - destination: 172.19.2.0/24
        nexthop: 172.19.1.1
    vlan: 41
    attachConfiguration: br-osp
  - name: storage_mgmt_leaf2
    ipv4:
      allocationEnd: 172.19.2.250
      allocationStart: 172.19.2.10
      cidr: 172.19.2.0/24
      routes:
      - destination: 172.19.0.0/24
        nexthop: 172.19.2.1
      - destination: 172.19.1.0/24
        nexthop: 172.19.2.1
    vlan: 42
    attachConfiguration: br-osp
- name: Tenant
  nameLower: tenant
  vip: False
  mtu: 1350
  subnets:
  - name: tenant
    ipv4:
      allocationEnd: 172.20.0.250
      allocationStart: 172.20.0.10
```

```
    cidr: 172.20.0.0/24
    routes:
    - destination: 172.20.1.0/24
      nexthop: 172.20.0.1
    - destination: 172.20.2.0/24
      nexthop: 172.20.0.1
  vlan: 50
  attachConfiguration: br-osp
- name: tenant_leaf1
  ipv4:
    allocationEnd: 172.20.1.250
    allocationStart: 172.20.1.10
    cidr: 172.20.1.0/24
    routes:
    - destination: 172.20.0.0/24
      nexthop: 172.20.1.1
    - destination: 172.20.2.0/24
      nexthop: 172.20.1.1
  vlan: 51
  attachConfiguration: br-osp
- name: tenant_leaf2
  ipv4:
    allocationEnd: 172.20.2.250
    allocationStart: 172.20.2.10
    cidr: 172.20.2.0/24
    routes:
    - destination: 172.20.0.0/24
      nexthop: 172.20.2.1
    - destination: 172.20.1.0/24
      nexthop: 172.20.2.1
  vlan: 52
  attachConfiguration: br-osp
```

2. Create the internal API network:

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

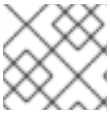
Verification

1. View the resources and child resources for OpenStackNetConfig:

```
$ oc get openstacknetconfig/openstacknetconfig -n openstack
$ oc get openstacknetattachment -n openstack
$ oc get openstacknet -n openstack
```

14.2. ADD ROLES FOR LEAF NETWORKS TO YOUR DEPLOYMENT

To add roles for the leaf networks to your deployment, update the **roles_data.yaml** configuration file and create the ConfigMap.



NOTE

You must use **roles_data.yaml** as the filename.

Prerequisites

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

Procedure

1. Update the **roles_data.yaml** file:

```
...
################################################################################
####
# Role: ComputeLeaf1                                           #
################################################################################
####
- name: ComputeLeaf1
  description: |
    Basic ComputeLeaf1 Node role
  # Create external Neutron bridge (unset if using ML2/OVS without DVR)
  tags:
    - external_bridge
  networks:
    InternalApi:
      subnet: internal_api_leaf1
    Tenant:
      subnet: tenant_leaf1
    Storage:
      subnet: storage_leaf1
  HostnameFormatDefault: '%stackname%-novacompute-leaf1-%index%'
...
################################################################################
####
# Role: ComputeLeaf2                                           #
################################################################################
####
- name: ComputeLeaf2
  description: |
    Basic ComputeLeaf1 Node role
  # Create external Neutron bridge (unset if using ML2/OVS without DVR)
  tags:
    - external_bridge
  networks:
    InternalApi:
      subnet: internal_api_leaf2
    Tenant:
      subnet: tenant_leaf2
    Storage:
      subnet: storage_leaf2
  HostnameFormatDefault: '%stackname%-novacompute-leaf2-%index%'
...
```

2. In the **~/custom_environment_files** directory, archive the templates into a tarball:

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

3. Create the **tripleo-tarball-config** ConfigMap:

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

14.3. CREATING NIC TEMPLATES FOR THE NEW ROLES

In Red Hat OpenStack Platform (RHOSP) 16.2, the tripleo NIC templates include the InterfaceRoutes parameter by default. You usually set up the routes parameter that you rendered in the **environments/network-environment.yaml** configuration file on the **host_routes** property of the Networking service (neutron) network. You then add it to the InterfaceRoutes parameter.

In director Operator the Networking service (neutron) is not present. To create new NIC templates for new roles, you must add the routes for a specific network to the NIC template and concatenate the lists.

14.3.1. Creating default network routes

Create the default network routes by adding the networking routes to the NIC template, and then concatenate the lists.

Procedure

1. Open the NIC template.
2. Add the network routes to the template, and then concatenate the lists:

```
parameters:
  ...
  {{ $net.Name }}Routes:
    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
  ...
              - type: interface
                ...
                routes:
                  list_concat_unique:
                    - get_param: {{ $net.Name }}Routes
                    - get_param: {{ $net.Name }}InterfaceRoutes
```

14.3.2. Subnet routes

Routes subnet information is auto rendered to the tripleo environment file **environments/network-environment.yaml** that is used by the Ansible playbooks. In the NIC templates use the **Routes_<subnet_name>** parameter to set the correct routing on the host, for example, **StorageRoutes_storage_leaf1**.

14.3.3. Modifying NIC templates for spine-leaf networking

To configure spine-leaf networking, modify the NIC templates for each role and re-create the ConfigMap.

Prerequisites

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

Procedure

1. Create NIC templates for each Compute role:

```
...
  StorageRoutes_storage_leaf1:
    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
...
  InternalApiRoutes_internal_api_leaf1:
    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
...
  TenantRoutes_tenant_leaf1:
    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
...
                    get_param: StorageIpSubnet
                routes:
                  list_concat_unique:
                    - get_param: StorageRoutes_storage_leaf1
              - type: vlan
...
                    get_param: InternalApiIpSubnet
                routes:
                  list_concat_unique:
                    - get_param: InternalApiRoutes_internal_api_leaf1
...
                    get_param: TenantIpSubnet
                routes:
                  list_concat_unique:
```

```
                - get_param: TenantRoutes_tenant_leaf1
            - type: ovs_bridge
...
```

2. In the ~/**custom_environment_files** directory, archive the templates into a tarball:

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

3. Create the **tripleo-tarball-config** ConfigMap:

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

14.3.4. Creating or updating an environment file to register the NIC templates

To create or update your environment file, add the NIC templates for the new nodes to the resource registry and re-create the ConfigMap.

Prerequisites

- Your OpenShift Container Platform cluster is operational and you have installed the director Operator correctly.
- You have installed the **oc** command line tool on your workstation.
- The **tripleo-tarball-config** ConfigMap was updated with the required **roles_data.yaml** and NIC template for the role.

Procedure

1. Add the NIC templates for the new nodes to an environment file in the resource_registry section:

```
resource_registry:
  OS::TripleO::Compute::Net::SoftwareConfig: net-config-two-nic-vlan-compute.yaml
  OS::TripleO::ComputeLeaf1::Net::SoftwareConfig: net-config-two-nic-vlan-
compute_leaf1.yaml
  OS::TripleO::ComputeLeaf2::Net::SoftwareConfig: net-config-two-nic-vlan-
compute_leaf2.yaml
```

2. In the ~/**custom_environment_files** directory archive the templates into a tarball:

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

3. Create the **tripleo-tarball-config** ConfigMap:

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

14.4. DEPLOYING THE OVERCLOUD WITH MULTIPLE ROUTED NETWORKS

To deploy the overcloud with multiple sets of routed networking, create the control plane and the compute nodes for spine-leaf networking, and then render the Ansible playbooks and apply them.

14.4.1. Creating the control plane

To create the control plane, specify the resources for the Controller nodes and director Operator will create the **openstackclient** pod for remote shell access.

Prerequisites

- Your OpenShift Container Platform cluster is operational and you have installed the director Operator correctly.
- You have installed the **oc** command line tool on your workstation.
- You have used the OpenStackNetConfig resource to create a control plane network and any additional network resources.

Procedure

1. Create a file named **openstack-controller.yaml** on your workstation. Include the resource specification for the Controller nodes. The following example shows a specification for a control plane that consists of three Controller nodes:

```
apiVersion: osp-director.openstack.org/v1beta2
kind: OpenStackControlPlane
metadata:
  name: overcloud
  namespace: openstack
spec:
  gitSecret: git-secret
  openStackClientImageURL: registry.redhat.io/rhosp-rhel8/openstack-tripleoclient:16.2
  openStackClientNetworks:
    - ctlplane
    - external
    - internal_api
    - internal_api_leaf1 # optionally the openstackclient can also be connected to subnets
  openStackClientStorageClass: host-nfs-storageclass
  passwordSecret: userpassword
  domainName: ostest.test.metalkube.org
  virtualMachineRoles:
    Controller:
      roleName: Controller
      roleCount: 1
      networks:
        - ctlplane
        - internal_api
        - external
        - tenant
        - storage
        - storage_mgmt
      cores: 6
      memory: 20
      rootDisk:
        diskSize: 500
        baseImageVolumeName: openstack-base-img
        storageClass: host-nfs-storageclass
```

```
      storageAccessMode: ReadWriteMany
      storageVolumeMode: Filesystem
  enableFencing: False
```

2. Create the control plane:

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

Wait until OCP creates the resources related to OpenStackControlPlane resource.

The director Operator also creates an **openstackclient** pod providing remote shell access to run Red Hat OpenStack Platform (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** pod remote shell:

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

14.4.2. Creating the compute nodes for the leafs

To create the Compute nodes from baremetal machines, include the resource specification in the OpenStackBaremetalSet custom resource.

Prerequisites

- Your OpenShift Container Platform cluster is operational and you have installed the director Operator correctly.
- You have installed the **oc** command line tool on your workstation.
- You have used the OpenStackNetConfig resource to create a control plane network and any additional network resources.

Procedure

1. Create a file named **openstack-computeleaf1.yaml** on your workstation. Include the resource specification for the Compute nodes. The following example shows a specification for one Compute leaf node:

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackBaremetalSet
metadata:
  name: computeleaf1
  namespace: openstack
spec:
  # How many nodes to provision
  count: 1
  # The image to install on the provisioned nodes
  baseImageUrl: http://host/images/rhel-image-8.4.x86_64.qcow2
  # The secret containing the SSH pub key to place on the provisioned nodes
  deploymentSSHSecret: osp-controlplane-ssh-keys
  # The interface on the nodes that will be assigned an IP from the mgmtCidr
  ctlplaneInterface: enp7s0
  # Networks to associate with this host
  networks:
        - ctlplane
        - internal_api_leaf1
        - external
        - tenant_leaf1
        - storage_leaf1
  roleName: ComputeLeaf1
  passwordSecret: userpassword
```

2. Create the Compute nodes:

```
$ oc create -f openstack-computeleaf1.yaml -n openstack
```

Verification

1. View the resource for the Compute node:

```
$ oc get openstackbaremetalset/computeleaf1 -n openstack
```

2. View the baremetal machines managed by OpenShift to verify the creation of the Compute node:

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

14.5. RENDER PLAYBOOKS AND APPLY THEM

You can now configure your overcloud. For more information, see Configuring overcloud software with the director Operator.

CHAPTER 15. BACKING UP AND RESTORING DIRECTOR OPERATOR

You use Red Hat OpenStack Platform (RHOSP) Director Operator to create and restore backups of the current CR, ConfigMap, and Secret configurations. The API consists of two CustomResourceDefinition (CRDs):

- OpenStackBackupRequest
- OpenStackBackup

15.1. DIRECTOR OPERATOR CUSTOMRESOURCEDEFINITION (CRD)

You use the OpenStackBackupRequest CRD to initiate the creation or restoration of a backup. You use the OpenStackBackup CRD to store the CR, ConfigMap and Secret data for a specific namespace. These CRDs provide you with the following features:

- You do not have to manually export and import multiple configurations, because the CRDs store the backup in a single OpenStackBackup CR.
- Director Operator does not backup any configuration that is in an incomplete or error state because it is aware of the state of all resources.
- Director operator knows which CRs, ConfigMaps and Secrets it needs to create a complete backup.

15.2. BACKING UP DIRECTOR OPERATOR

To create a backup, director Operator includes the contents of the current namespace and anything declared in the **additionalConfigMaps** and **additionalSecrets** lists. You can include manually created ConfigMaps and Secrets in the additional specifications.

Procedure

1. Create the OpenStackBackupRequest CRD and set the **mode** to **save** to request a backup:

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackBackupRequest
metadata:
  name: openstackbackupsave
  namespace: openstack
spec:
  mode: save
  additionalConfigMaps: []
  additionalSecrets: []
```

2. Optional: Use the **additionalConfigMaps** and **additionalSecrets** specifications to include ConfigMaps or Secrets that you created manually.

3. Monitor the status of the OpenStackBackupRequest:

```
$ oc get -n openstack osbackuprequest openstackbackupsave
NAME                 OPERATION   SOURCE   STATUS      COMPLETION TIMESTAMP
openstackbackupsave  save                 Quiescing
```

-

NOTE

The **Quiescing** state indicates that director Operator is waiting for the CRs to reach their finished state. The length of time for the backup to finish can vary depending on the number of CRs.

4. You can investigate the director Operator logs to check progress:

```
$ oc logs <operator_pod> -c manager -f
2022-01-11T18:26:15.180Z INFO controllers.OpenStackBackupRequest Quiesce for save for OpenStackBackupRequest openstackbackupsave is waiting for: [OpenStackBaremetalSet: compute, OpenStackControlPlane: overcloud, OpenStackVMSet: controller]
```

- Replace **<operator_pod>** with the name of the Operator pod.

Verification

1. View the OpenStackBackupRequest to confirm that the STATUS is **Saved**:

```
$ oc get -n openstack osbackuprequest
NAME                 OPERATION   SOURCE   STATUS   COMPLETION TIMESTAMP
openstackbackupsave  save                 Saved    2022-01-11T19:12:58Z
```

2. If the OpenStackBackupRequest enters the **Error** state, review the request contents to find the error:

```
$ oc get -n openstack openstackbackuprequest <request_name> -o yaml
```

- Replace **<request_name>** with the name of the backup request.

3. View the OpenStackBackup to confirm it exists:

```
$ oc get -n openstack osbackup

NAME                          AGE
openstackbackupsave-1641928378   6m7s
```

15.3. RESTORING DIRECTOR OPERATOR FROM A BACKUP

When you request director Operator to restore a backup, director Operator takes the contents of the **restoreSource** OpenStackBackup and attempts to apply them to all existing CR, ConfigMap, and Secret resources present within the namespace. Director operator overwrites any existing director Operator resources in the namespace, and creates new resources for those not found within the namespace.

Procedure

1. List the available backups:

```
$ oc get osbackup
```

NOTE

To inspect the details of a specific backup:

```
$ oc get backup <name> -o yaml
```

- Replace **<name>** with the name of the backup you want to inspect.

2. Create the OpenStackBackupRequest CRD and set the **mode** to **restore**. For example:

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackBackupRequest
metadata:
  name: openstackbackupsave
  namespace: openstack
spec:
  mode: <mode>
  restoreSource: <restore_source>
```

- Replace **<mode>** with one of the following options:
  - **restore** - to request a restore from an existing OpenStackBackup.
  - **cleanRestore** - to completely wipe the existing director Operator resources within the namespace before any restoration attempt.
- Replace **<restore_source>** with the ID of the OpenStackBackup to restore, for example, **openstackbackupsave-1641928378**.

3. Monitor the status of the OpenStackBackupRequest:

```
$ oc get -n openstack osbackuprequest openstackbackuprestore
NAME                  OPERATION  SOURCE                        STATUS     COMPLETION
TIMESTAMP
openstackbackuprestore   restore    openstackbackupsave-1641928378   Loading
```

After all resources are loaded, director Operator begins reconciling:

```
$ oc get -n openstack osbackuprequest openstackbackuprestore
NAME                  OPERATION  SOURCE                        STATUS      COMPLETION
TIMESTAMP
openstackbackuprestore   restore    openstackbackupsave-1641928378   Reconciling
```

NOTE

If the OpenStackBackupRequest enters the Error state, view the request contents to find the error:

```
$ oc get -n openstack openstackbackuprequest <name> -o yaml
```

Verification

1. View the OpenStackBackupRequest to confirm that the STATUS is **Restored**:

```
$ oc get -n openstack osbackuprequest
NAME                    OPERATION  SOURCE                        STATUS     COMPLETION
TIMESTAMP
openstackbackuprestore  restore    openstackbackupsave-1641928378  Restored  2022-
01-12T13:48:57Z
```

CHAPTER 16. CHANGE RESOURCES ON VIRTUAL MACHINES USING DIRECTOR OPERATOR

To change the CPU, RAM, and disk resources of a OpenStackVMSet use the OpenStackControlPlane.

16.1. CHANGE THE CPU OR RAM OF AN OPENSTACKVMSET

You can use the OpenStackControlPlane to change the CPU or RAM of an OpenStackVMSet.

Procedure

1. Change the number of Controller virtualMachineRole cores to 8:

```
$ oc patch -n openstack osctlplane overcloud --type='json' -p='[{"op": "add", "path":
"/spec/virtualMachineRoles/controller/cores", "value": 8 }]'
```

2. Change the Controller virtualMachineRole RAM size to 22GB:

```
$ oc patch -n openstack osctlplane overcloud --type='json' -p='[{"op": "add", "path":
"/spec/virtualMachineRoles/controller/memory", "value": 22 }]'
```

3. Validate the virtualMachineRole resource:

```
$ oc get osvmset
NAME        CORES   RAM   DESIRED   READY   STATUS        REASON
controller  8       22    1         1       Provisioned   All requested VirtualMachines have been
provisioned
```

4. From inside the virtual machine do a graceful shutdown. Shutdown each updated virtual machine one by one.

5. Power on the virtual machine:

```
$ `virtctl start <VM>` to power on the virtual machine.
```

   - Replace **<VM>** with the name of your virtual machine.

16.2. ADD ADDITIONAL DISKS TO AN OPENSTACKVMSET

You can use the OpenStackControlPlane to add additional disks to a virtual machine by editing the **additionalDisks** property.

Procedure

1. Add or update the **additionalDisks** parameter in the OpenStackControlPlane object:

```
spec:
  ...
  virtualMachineRoles:
    Controller:
      ...
      additionalDisks:
```

```
    - baseImageVolumeName: openstack-base-img
      dedicatedIOThread: false
      diskSize: 10
      name: "data-disk1"
      storageAccessMode: ReadWriteMany
      storageClass: host-nfs-storageclass
      storageVolumeMode: Filesystem
```

2. Apply the patch:

```
$ oc patch -n openstack osctlplane overcloud --patch-file controller_add_data_disk1.yaml
```

3. Validate the virtualMachineRole resource:

```
$ oc get osvmset controller -o json | jq .spec.additionalDisks
[
  {
    "baseImageVolumeName": "openstack-base-img",
    "dedicatedIOThread": false,
    "diskSize": 10,
    "name": "data-disk1",
    "storageAccessMode": "ReadWriteMany",
    "storageClass": "host-nfs-storageclass",
    "storageVolumeMode": "Filesystem"
  }
]
```

4. From inside the virtual machine do a graceful shutdown. Shutdown each updated virtual machine one by one.

5. Power on the virtual machine:

```
$ `virtctl start <VM>` to power on the virtual machine.
```

- Replace **<VM>** with the name of your virtual machine.

CHAPTER 17. AIRGAPPED ENVIRONMENT

An air-gapped environment ensures security by physically isolating it from other networks and systems. You can install director Operator in an air-gapped environment to ensure security and provides certain regulatory requirements.

17.1. CONFIGURING AN AIRGAPPED ENVIRONMENT

To configure an airgapped environment, you must have access to both **registry.redhat.io** and the registry for airgapped environment. For more information on how to access both registries, see Mirroring catalog contents to airgapped registries .

Prerequisites

- You have installed an Openshift Container Platform LTS release (OCP) 4.10 or later cluster, with an enabled baremetal cluster operator, and a provisioning network.
- You have installed the Kubevirt-Hyperconvered (OCP Virtualization Operator) and SR-IOV operator in the cluster.
- You have a disconnected registry adhering to docker v2 schema. For more information, see Mirroring images for a disconnected installation .
- You have access to a Satellite server or any other repository used to register the overcloud nodes and install packages.
- You have access to a local git repository to store deployment artifacts.
- You have installed the **oc** command line tool on your workstation.
- You have installed the **podman** and **skopeo** command line tools on your workstation.

Procedure

1. Create the openstack namespace:

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

2. Create the index image and push it to your registry:

```
$ podman login registry.redhat.io
$ podman login your.registry.local
$ BUNDLE_IMG="registry.redhat.io/rhosp-rhel8/osp-director-operator-
bundle@sha256:c19099ac3340d364307a43e0ae2be949a588fefe8fcb17663049342e7587f055
"
```

NOTE

You can get the latest bundle image from: Certified container images. Search for **osp-director-operator-bundle**.

3. Mirror the relevant images based on the operator index image:

```
$ oc adm catalog mirror ${INDEX_IMG} your.registry.local --insecure --index-filter-by-os='Linux/x86_64'
```

4. After mirroring is complete, a **manifests** directory is generated in your current directory called **manifests-osp-director-operator-index-<random_number>**. Apply the created ImageContentSourcePolicy to your cluster:

```
$ os apply -f manifests-osp-director-operator-index-<random_number>/imageContentSourcePolicy.yaml
```

- Replace <random_number> with the randomly generated number.

5. Create a file named **osp-director-operator.yaml** and include the following YAML content to configure the three resources required to install director Operator:

```
apiVersion: operators.coreos.com/v1alpha1
kind: CatalogSource
metadata:
  name: osp-director-operator-index
  namespace: openstack
spec:
  sourceType: grpc
  image: your.registry.local/osp-director-operator-index:1.3.x-y
---
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
```

6. Create the new resources in the openstack namespace:

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

7. Copy the required overcloud images to the respository:

```
$ for i in $(podman search --limit 1000 "registry.redhat.io/rhosp-rhel8/openstack" --format="{{
.Name }}" | awk '{print $1 ":" "16.2.4"}' | awk -F "/" '{print $2 "/" $3}'); do skopeo copy --all
docker://registry.redhat.io/$i docker://your.registry.local/$i;done
```

NOTE

You can refer to Preparing a Satellite server for container images if Red Hat Satellite is used as the local registry.

8. You can now proceed with Preparing the overcloud deployment with the director Operator .

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 .
- Mirroring Operator catalogs for use with disconnected clusters .
- Mirroring catalog contents to airgapped registries .
- Preparing a Satellite server for container images .
- Obtaining container images from private registries .