Red Hat OpenStack Platform 17.1

Deploying an overcloud in a Red Hat OpenShift Container Platform cluster with director Operator

Using director Operator to deploy and manage a Red Hat OpenStack Platform overcloud in a Red Hat OpenShift Container Platform

Last Updated: 2023-09-21

Red Hat OpenStack Platform 17.1 Deploying an overcloud in a Red Hat OpenShift Container Platform cluster with director Operator

Using director Operator to deploy and manage a Red Hat OpenStack Platform overcloud in a Red Hat OpenShift Container Platform

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 Network Platform Sustaining Subscription (NPSS). 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.

Using the Direct Documentation Feedback (DDF) function

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

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

CHAPTER 1. CREATING AND DEPLOYING A RHOSP OVERCLOUD WITH DIRECTOR OPERATOR

Red Hat OpenShift Container Platform (RHOCP) uses a modular system of Operators to extend the functions of your RHOCP cluster. Red Hat OpenStack Platform (RHOSP) director Operator (OSPdO) adds the ability to install and run a RHOSP cloud within RHOCP. OSPdO manages a set of Custom Resource Definitions (CRDs) that deploy and manage the infrastructure and configuration of RHOSP nodes. The basic architecture of an OSPdO-deployed RHOSP cloud includes the following features:

Virtualized control plane

The Controller nodes are virtual machines (VMs) that OSPdO creates in Red Hat OpenShift Virtualization.

Bare-metal machine provisioning

OSPdO uses RHOCP bare-metal machine management to provision the Compute nodes for the RHOSP cloud.

Networking

OSPdO configures the underlying networks for RHOSP services.

Heat and Ansible-based configuration

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

CLI client

OSPdO creates an **openstackclient** pod for users to run RHOSP CLI commands and interact with their RHOSP cloud.

You can use the resources specific to OSPdO to provision your overcloud infrastructure, generate your overcloud configuration, and create an overcloud. To create a RHOSP overcloud with OSPdO, you must complete the following tasks:

1. Install OSPdO on an operational RHOCP cluster.
2. Create a RHOCP cluster data volume for the base operating system and add authentication details for your remote Git repository.
3. Create the overcloud networks using the **OpenStackNetConfig** CRD, including the control plane and any isolated networks.
4. Create **ConfigMaps** to store any custom heat templates and environment files for your overcloud.
5. Create a control plane, which includes three virtual machines for Controller nodes and a pod to perform client operations.
6. Create bare-metal Compute nodes.
7. Create an **openstackconfiggenerator** resource to render Ansible playbooks for overcloud configuration.
8. Apply the Ansible playbook configuration to your overcloud nodes by using **openstackdeploy**.

1.1. CUSTOM RESOURCE DEFINITIONS FOR DIRECTOR OPERATOR

The Red Hat OpenStack Platform (RHOSP) director Operator (OSPdO) includes a set of custom resource definitions (CRDs) that you can use to manage overcloud resources.

Use the following command to view a complete list of the OSPdO CRDs:

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

Use the following command to view the definition for a specific CRD:

```
$ 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
...
```

There are two types of CRD: hardware provisioning and software configuration.

Hardware Provisioning CRDs

openstacknetattachment (internal)

Used by the OSPdO to manage the **NodeNetworkConfigurationPolicy** and **NodeSriovConfigurationPolicy** CRDs, which are used to attach networks to virtual machines (VMs).

openstacknetconfig

Use to specify **openstacknetattachment** and **openstacknet** CRDs that describe the full network configuration. The set of reserved IP and MAC addresses for each node are reflected in the status.

openstackbaremetalset

Use to create sets of bare-metal hosts for specific RHOSP roles, such as "Compute" and "Storage".

openstackcontrolplane

Use to create the RHOSP control plane and manage associated **openstackvmset** CRs.

openstacknet (internal)

Use to create networks that are used to assign IPs to the **openstackvmset** and **openstackbaremetalset** CRs.

openstackipset (internal)

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

openstackprovisionservers

Use to serve custom images for provisioning bare-metal nodes with Metal3.

openstackvmset

Use to create sets of OpenShift Virtualization VMs for a specific RHOSP role, such as "Controller", "Database", or "NetworkController".

Software Configuration CRDs

openstackconfiggenerator

Use to automatically generate Ansible playbooks for deployment when you scale up or make changes to custom **ConfigMaps** for deployment.

openstackconfigversion

Use to represent a set of executable Ansible playbooks.

openstackdeploy

Use to execute the set of Ansible playbooks defined in the **openstackconfigversion** CR.

openstackclient

Creates a pod used to run RHOSP deployment commands.

Additional resources

- Managing resources from custom resource definitions

1.2. CRD NAMING CONVENTIONS

Each custom resource definition (CRD) can have multiple names defined with the **spec.names** parameter. Which name you use depends on the context of the action you perform:

- 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
```

1.3. 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 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.

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.

1.4. ADDITIONAL RESOURCES

- Operators
- Adding Operators to a cluster

CHAPTER 2. INSTALLING AND PREPARING DIRECTOR OPERATOR

You install Red Hat OpenStack Platform (RHOSP) director Operator (OSPdO) on an existing operational Red Hat Openshift Container Platform (RHOCP) cluster. You perform the OSPdO installation tasks and all overcloud creation tasks on a workstation that has access to the RHOCP cluster. After you have installed OSPdO, 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. PREREQUISITES

- An operational Red Hat Openshift Container Platform (RHOCP) cluster, version 4.12 or later. The cluster must contain a **provisioning** network, and the following Operators:
  - A **baremetal** cluster Operator. The **baremetal** cluster Operator must be enabled. For more information on **baremetal** cluster Operators, see Bare-metal cluster Operators.
  - OpenShift Virtualization Operator. For more information on installing the OpenShift Virtualization Operator, see Installing OpenShift Virtualization using the web console .
  - SR-IOV Network Operator.
  - Kubernetes NMState Operator. You must also 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
    ```

    For more information on installing the Kubernetes NMState Operator, see Installing the Kubernetes NMState Operator.
- The **oc** command line tool is installed on your workstation.
- A remote Git repository for OSPdO to store the generated configuration for your overcloud.
- An SSH key pair is generated for the Git repository and the public key is uploaded to the Git repository.
- The following persistent volumes to fulfill the persistent volume claims that OSPdO creates:
  - 4G for **openstackclient-cloud-admin**.
  - 1G for **openstackclient-hosts**.
  - 50G for the base image that OSPdO clones for each Controller virtual machine.
  - A minimum of 50G for each Controller virtual machine. For more information, see Controller node requirements

2.2. BARE-METAL CLUSTER OPERATORS

Red Hat Openshift Container Platform (RHOCP) 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. RHOCP 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 bare-metal hosts. It maintains an inventory of available hosts as instances of the **BareMetalHost** custom resource definition (CRD). You can use the bare-metal Operator to perform the following tasks:

- Inspect the host's hardware details and report them to the corresponding **BareMetalHost** CR. 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 RHOCP cluster, navigate to **Administration > Global Configuration > Infrastructure**, switch to **YAML** view, scroll to the **Conditions** section, and view the **status.platformStatus** value.

2.3. INSTALLING DIRECTOR OPERATOR

To install director Operator (OSPdO), you must create the **openstack** project (**namespace**) for OSPdO and create the following custom resources (CRs) within the project:

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

Procedure

1. Create the OSPdO project:

```
$ 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-rhel9/osp-director-operator-bundle:1.3.1"
$ 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 an environment file to configure the **CatalogSource**, **OperatorGroup**, and **Subscription** CRs required to install OSPdO, for example, **osp-director-operator.yaml**.

7. To configure the **CatalogSource** CR, add the following configuration to **osp-director-operator.yaml**:

```
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
```

   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.

8. To configure the **OperatorGroup** CR, add the following configuration to **osp-director-operator.yaml**:

```
---
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: "osp-director-operator-group"
  namespace: openstack
spec:
  targetNamespaces:
  - openstack
```

9. To configure the **Subscription** CR, add the following configuration to **osp-director-operator.yaml**:

```
---
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
```

10. Create the new **CatalogSource**, **OperatorGroup**, and **Subscription** CRs within the **openstack** namespace:

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

11. Confirm that you have installed OSPdO, **osp-director-operator.openstack**, by listing the installed operators:

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

Next steps

- Creating a data volume for the base operating system

2.4. CREATING A DATA VOLUME FOR THE BASE OPERATING SYSTEM

You must create a data volume with the Red Hat OpenShift Container Platform (RHOCP) cluster to store the base operating system image for your Controller virtual machines (VMs). You use the **baseImageVolumeName** parameter to specify this data volume when you create the **OpenStackControlPlane** and **OpenStackVmSet** custom resources.

Prerequisites

- The **virtctl** client tool is installed on your workstation. To install this tool on a Red Hat Enterprise Linux (RHEL) workstation, use the following commands:

```
$ sudo subscription-manager repos --enable=cnv-4.12-for-rhel-8-x86_64-rpms
$ sudo dnf install -y kubevirt-virtctl
```

- The **virt-customize** client tool is installed on your workstation. To install this tool on a RHEL workstation, use the following command:

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

Procedure

1. Download a RHEL 9.2 QCOW2 image from the Product Download section of the Red Hat Customer Portal to your workstation.

2. Optional: Add a custom CA certificate:

```
$ sudo -s
$ export LIBGUESTFS_BACKEND=direct
$ virt-copy-in -a <local_path_to_image> <ca_certificate>.pem /etc/pki/ca-trust/source/anchors/
```

   You might want to add a custom CA certificate to secure LDAP communication for the Identity service, or to communicate with any non-RHOSP system.

3. Create a script to customize the image to assign predictable network interface names:

```
#!/bin/bash
set -eux
```

```
if [ -e /etc/kernel/cmdline ]; then
  echo 'Updating /etc/kernel/cmdline'
  sed -i -e "s/^\(.*\)net\.ifnames=0\s*\(.*\)/\1\2/" /etc/kernel/cmdline
fi

source /etc/default/grub
if grep -q "net.ifnames=0" <<< "$GRUB_CMDLINE_LINUX"; then
  echo 'Updating /etc/default/grub'
  sed -i -e "s/^\(GRUB_CMDLINE_LINUX=.*\)net\.ifnames=0\s*\(.*\)/\1\2/" /etc/default/grub
fi
if [ "$GRUB_ENABLE_BLSCFG" == "true" ]; then
  echo 'Fixing BLS entries'
  find /boot/loader/entries -type f -exec sed -i -e "s/^\(.*\)net\.ifnames=0\s*\(.*\)/\1\2/" {} \;
fi
# Always do this, on RHEL8 with BLS we still need it as the BLS entry uses $kernelopts from
grubenv
echo 'Running grub2-mkconfig'
grub2-mkconfig -o /etc/grub2.cfg
grub2-mkconfig -o /etc/grub2-efi.cfg
rm -f /etc/sysconfig/network-scripts/ifcfg-ens* /etc/sysconfig/network-scripts/ifcfg-eth*
update-ca-trust extract
```

4. Run the image customization script:

```
$ sudo -s
$ export LIBGUESTFS_BACKEND=direct
$ chmod 755 customize_image.sh
$ virt-customize -a <local_path_to_image> --run customize_image.sh --truncate
/etc/machine-id
```

5. Use **virtctl** to upload the image to OpenShift Virtualization:

```
$ virtctl image-upload dv <datavolume_name> -n openstack \
 --size=<size> --image-path=<local_path_to_image> \
 --storage-class <storage_class> --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, **50Gi**. The minimum size is 50GB.
- Replace **<storage_class>** with the required storage class from your cluster. Use the following command to retrieve the available storage classes:

```
$ oc get storageclass
```

Additional resources

- Uploading local disk images by using the virtctl tool

Next steps

- Adding authentication details for your remote Git repository

2.5. ADDING AUTHENTICATION DETAILS FOR YOUR REMOTE GIT REPOSITORY

Director Operator (OSPdO) 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 a Red Hat OpenShift Platform (RHOCP) Secret resource named **git-secret**.

Prerequisites

- The private key of the SSH key pair for your OSPdO Git repository.

Procedure

1. Create the **git-secret** Secret resource:

```
$ oc create secret generic <secret_name> -n <namespace> \
 --from-file=git_ssh_identity=<path_to_private_SSH_key> \
 --from-literal=git_url=<git_server_URL>
```

   - Replace **<secret_name>** with the name of the secret, in this case, **git-secret**.
   - Replace **<namespace>** with the name of the namespace to create the secret in, for example, **openstack**.
   - Replace **<path_to_private_SSH_key>** with the path to the private key to access the Git repository.
   - Replace **<git_server_URL>** with the SSH URL of the git repository that stores the OSPdO configuration, for example, **ssh://<user>@<server>:2202/repo.git**.

2. Verify that the Secret resource is created:

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

Additional resources

- Providing sensitive data to pods

Next steps

- Creating networks with director Operator

2.6. 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**. 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.

NOTE

If you set the root password, you must use the **passwordSecret** parameter to specify the name of this **Secret** resource when you create **OpenStackControlPlane** and **OpenStackBaremetalSet** custom resources. The examples in this guide use the **Secret** resource name **userpassword**.

Procedure

1. Convert your chosen password to a base64 value:

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

IMPORTANT

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: <secret_name>
  namespace: openstack
data:
  NodeRootPassword: "<password>"
```

- Replace **<secret_name>** with the name of this Secret resource, for example, **userpassword**.
- Replace **<password>** with your base64 encoded password.

3. Create the **userpassword** Secret:

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

Additional resources

- Providing sensitive data to pods

Next steps

- Creating networks with director Operator

CHAPTER 3. CREATING NETWORKS WITH DIRECTOR OPERATOR

To create networks and bridges on OpenShift Virtualization worker nodes and connect your virtual machines (VMs) to these networks, you define your **OpenStackNetConfig** custom resource (CR) and specify all the subnets for the overcloud networks. You must create one control plane network for your overcloud. You can also optionally create additional networks to implement network isolation for your composable networks.

3.1. CREATING AN OVERCLOUD NETWORK WITH THE OPENSTACKNETCONFIG CRD

You must use the **OpenStackNetConfig** CRD to define at least one control plane network for your overcloud. You can also optionally define VLAN networks to create network isolation for composable networks such as **InternalAPI**, **Storage**, and **External**. Each network definition must include the IP address assignment, and the mapping information for the **OpenStackNetAttachment** CRD. OpenShift Virtualization uses the network definition to attach any virtual machines (VMs) to the control plane and VLAN networks.

TIP

For descriptions of the values you can use to configure your **OpenStackNetConfig**, view the **OpenStackNetConfig** CRD specification schema:

$ oc describe crd openstacknetconfig

Procedure

1. Create a file named **openstacknetconfig.yaml** on your workstation.
2. Add the following configuration to **openstacknetconfig.yaml** to create the **OpenStackNetConfig** custom resource (CR):

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackNetConfig
metadata:
  name: openstacknetconfig
```

3. Configure network attachment definitions for the bridges you require for your network. For example, add the following configuration to **openstacknetconfig.yaml** to create the RHOSP bridge network attachment definition **br-osp**, and set the **nodeNetworkConfigurationPolicy** option to create a Linux bridge:

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackNetConfig
metadata:
  name: openstacknetconfig
spec:
  attachConfigurations:
    br-osp: 1
      nodeNetworkConfigurationPolicy:
        nodeSelector:
          node-role.kubernetes.io/worker: ""
```

```
      desiredState:
        interfaces:
        - bridge:
            options:
              stp:
                enabled: false
            port:
            - name: enp6s0 (2)
          description: Linux bridge with enp6s0 as a port
          name: br-osp (3)
          state: up
          type: linux-bridge
          mtu: 1500 (4)
  # 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
```

1. The network attachment definition for **br-osp**.
2. The NIC / Ethernet device to attach to on each host.
3. The interface name.
4. The maximum amount of data that can be transferred in a single network packet.

4. Optional: To use Jumbo Frames for a bridge, configure the bridge interface to use Jumbo Frames and update the value of **mtu** for the bridge:

```
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: 9000 (1)
      - name: enp6s0 (2)
        description: Configuring enp6s0 on workers
        type: ethernet
        state: up
        mtu: 9000
...
```

(1) Update the maximum amount of data that can be transferred in a single network packet with Jumbo Frames.

(2) Configure the bridge interface to use Jumbo Frames.

5. Define each overcloud network. The following example creates a control plane network and an isolated network for **InternalAPI** traffic:

```
spec:
  ...
  networks:
  - name: Control (1)
    nameLower: ctlplane (2)
    subnets: (3)
    - name: ctlplane (4)
      ipv4: (5)
        allocationEnd: 172.22.0.250
        allocationStart: 172.22.0.100
        cidr: 172.22.0.0/24
        gateway: 172.22.0.1
      attachConfiguration: br-osp (6)
  - name: InternalApi (7)
    nameLower: internal_api
    mtu: 1350
    subnets:
    - name: internal_api
      attachConfiguration: br-osp
      vlan: 20 (8)
      ipv4:
        allocationEnd: 172.17.0.250
        allocationStart: 172.17.0.10
        cidr: 172.17.0.0/24
...
```

(1) The name of the network, for example, **Control**.

(2) The lowercase version of the network name, for example, **ctlplane**.

(3) The subnet specifications.

(4) The name of the subnet, for example, **ctlplane**.

(5) Details of the IPv4 subnet with **allocationStart**, **allocationEnd**, **cidr**, **gateway**, and an optional list of routes with **destination** and **nexthop**.

6 The network attachment definition to connect the network to. In this example, the RHOSP bridge, **br-osp**, is connected to a NIC on each worker.

7 The network definition for a composable network. To use the default RHOSP networks, you must create an **OpenStackNetConfig** resource for each network. For information on the default RHOSP networks, see Default Red Hat OpenStack Platform networks . To use different networks, you must create a custom **network_data.yaml** file. For information on creating a custom **network_data.yaml** file, see Configuring overcloud networking.

8 The network VLAN. For information on the default RHOSP networks, see Default Red Hat OpenStack Platform networks. For more information on virtual machine bridging with the **OpenStackNetConfig** CRD, see Understanding virtual machine bridging with the OpenStackNetConfig CRD.

6. Optional: Reserve static IP addresses for networks on specific nodes:

```
spec:
  ...
  reservations:
    controller-0:
      ipReservations:
        ctlplane: 172.22.0.120
    compute-0:
      ipReservations:
        ctlplane: 172.22.0.140
        internal_api: 172.17.0.40
        storage: 172.18.0.40
        tenant: 172.20.0.40
```

NOTE

Reservations have precedence over any autogenerated IP addresses.

7. Save the **openstacknetconfig.yaml** definition file.
8. Create the overcloud network:

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

9. To verify that the overcloud network is created, view the resources for the overcloud network:

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

10. View the **OpenStackNetConfig** API and 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
```

Next steps

- Customizing the overcloud with director Operator

3.1.1. 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
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

3.2. UNDERSTANDING VIRTUAL MACHINE BRIDGING WITH THE OPENSTACKNETCONFIG CRD

When you create virtual machines (VMs) with the **OpenStackVMSet** CRD, you must connect these VMs to the relevant Red Hat OpenStack Platform (RHOSP) networks. You can use the **OpenStackNetConfig** CRD to create the required bridges on the Red Hat OpenShift Container Platform (RHOCP) worker nodes and connect your Controller VMs to your RHOSP overcloud networks. RHOSP requires dedicated NICs to deploy.

The **OpenStackNetConfig** CRD includes an **attachConfigurations** option, which is a hash of **nodeNetworkConfigurationPolicy**. Each specified **attachConfiguration** in an **OpenStackNetConfig** custom resource (CR) creates a **NetworkAttachmentDefinition** object, which passes network interface data to the **NodeNetworkConfigurationPolicy** resource in the RHOCP cluster. The **NodeNetworkConfigurationPolicy** resource uses the **nmstate** API to configure the end state of the network configuration on each RHOCP worker node. The **NetworkAttachmentDefinition** object for each network defines the Multus CNI plugin configuration. When you specify the VLAN ID for the **NetworkAttachmentDefinition** object, the Multus CNI plugin enables **vlan-filtering** on the bridge. Each network configured in the **OpenStackNetConfig** CR references one of the **attachConfigurations**. Inside the VMs, there is one interface for each network.

The following example creates a **br-osp attachConfiguration**, and configures the **nodeNetworkConfigurationPolicy** option to create a Linux bridge and connect the bridge to a NIC on each worker. When you apply this configuration, the **NodeNetworkConfigurationPolicy** object configures each RHOCP worker node to match the required end state: each worker contains a new bridge named **br-osp**, which is connected to the **enp6s0** NIC on each host. All RHOSP Controller VMs can connect to the **br-osp** bridge for control plane network traffic.

```
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
```

If you specify an Internal API network through VLAN 20, you can set the **attachConfiguration** option to modify the networking configuration on each RHOCP 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 VMs can connect to the VLAN 20 on the **br-osp** bridge for Internal API network traffic.

When you create VMs with the **OpenStackVMSet** CRD, the VMs 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 following interface configuration is generated for Controller VMs:

```
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

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

NOTE

The role NIC template used by **OpenStackVMSet** is auto generated. You can overwrite the default configuration in a custom NIC template file, **<role>-nic-template.j2**, for example, **controller-nic-template.j2**. You must add your custom NIC file to the tarball file that contains your overcloud configuration, which is implemented by using an OpenShift ConfigMap object. For more information, see Chapter 4. Customizing the overcloud with director Operator.

Additional resources

- Updating node network configuration

3.3. EXAMPLE OPENSTACKNETCONFIG CUSTOM RESOURCE FILE

The following example **OpenStackNetConfig** custom resource (CR) file defines an overcloud network which includes a control plane network and isolated VLAN networks for a default RHOSP deployment. The example also reserves static IP addresses for networks on specific nodes.

```
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
    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
      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: 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
reservations:
  compute-0:
    ipReservations:
      ctlplane: 172.22.0.140
```

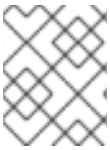
```
    internal_api: 172.17.0.40
    storage: 172.18.0.40
    tenant: 172.20.0.40
  macReservations: {}
controller-0:
  ipReservations:
    ctlplane: 172.22.0.120
    external: 10.0.0.20
    internal_api: 172.17.0.20
    storage: 172.18.0.20
    storage_mgmt: 172.19.0.20
    tenant: 172.20.0.20
  macReservations: {}
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. CUSTOMIZING THE OVERCLOUD WITH DIRECTOR OPERATOR

You can customize your overcloud or enable certain features by creating heat templates and environment files that you include with your overcloud deployment. With a director Operator (OSPdO) overcloud deployment, you store these files in **ConfigMap** objects before running the overcloud deployment.

4.1. ADDING CUSTOM TEMPLATES TO THE OVERCLOUD CONFIGURATION

Director Operator (OSPdO) converts a core set of overcloud heat templates into Ansible playbooks that you apply to provisioned nodes when you are ready to configure the Red Hat OpenStack Platform (RHOSP) 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** object named **tripleo-tarball-config**. This tarball file can contain complex directory structures to extend the core set of templates. OSPdO 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.

Prerequisites

- The custom overcloud 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 gzipped tarball:

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

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

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

4. Verify that the **ConfigMap** CR is created:

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

Additional resources

- Creating and using config maps

- Understanding heat templates

Next steps

- Adding custom environment files to the overcloud configuration

4.2. ADDING CUSTOM ENVIRONMENT FILES TO THE OVERCLOUD CONFIGURATION

To enable features or set parameters in the overcloud, you must include environment files with your overcloud deployment. Director Operator (OSPdO) uses a **ConfigMap** object named **heat-env-config** to store and retrieve environment files. The **ConfigMap** object stores the environment files in the following format:

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

For example, the following **ConfigMap** contains two environment files:

```
...
data:
  network_environment.yaml: |+
    parameter_defaults:
      ComputeNetworkConfigTemplate: 'multiple_nics_vlans_dvr.j2'
  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
```

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

Prerequisites

- The custom environment files for your overcloud deployment.

Procedure

1. Create the **heat-env-config ConfigMap** object:

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

   - Replace **<dir_custom_environment_files>** with the directory that contains the environment files you want to use in your overcloud deployment. The **ConfigMap** object stores these as individual **data** entries.

2. Verify that the **heat-env-config ConfigMap** object contains all the required environment files:

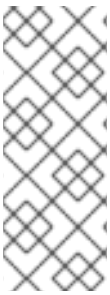
```
$ oc get configmap/<configmap_name> -n openstack
```

4.3. ADDITIONAL RESOURCES

- Understanding heat templates
- Environment files
- Creating and using config maps

CHAPTER 5. CREATING OVERCLOUD NODES WITH DIRECTOR OPERATOR

A Red Hat OpenStack Platform (RHOSP) 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** Custom Resource Definition (CRD) and Compute nodes with the **OpenStackBaremetalSet** CRD.



NOTE

Red Hat OpenShift Container Platform (RHOCP) does not autodiscover issues on RHOCP worker nodes, or perform autorecovery of worker nodes that host RHOSP Controller VMs if the worker node fails or has an issue. You must enable health checks on your RHOCP cluster to automatically relocate Controller VM pods when a host worker node fails. For information on how to autodiscover issues on RHOCP worker nodes, see Deploying machine health checks .

5.1. CREATING A CONTROL PLANE WITH THE OPENSTACKCONTROLPLANE CRD

The Red Hat OpenStack Platform (RHOSP) control plane contains the RHOSP services that manage the overcloud. The default control plane consists of 3 Controller nodes. You can use composable roles to manage services on dedicated controller virtual machines (VMs). For more information on composable roles, see Composable services and custom roles .

Define an **OpenStackControlPlane** custom resource (CR) to create the Controller nodes as OpenShift Virtualization virtual machines (VMs).

TIP

For descriptions of the values you can use to configure your **OpenStackControlPlane** CR, view the **OpenStackControlPlane** CRD specification schema:

```
$ oc describe crd openstackcontrolplane
```

Prerequisites

- You have used the **OpenStackNetConfig** CR 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. The following example defines a specification for a control plane that consists of 3 Controller nodes:

```
apiVersion: osp-director.openstack.org/v1beta2
kind: OpenStackControlPlane
metadata:
  name: overcloud (1)
  namespace: openstack (2)
```

```
spec: 3
  gitSecret: git-secret
  openStackClientNetworks:
        - ctlplane
        - internal_api
        - external
  openStackClientStorageClass: host-nfs-storageclass
  passwordSecret: userpassword 4
  virtualMachineRoles:
    Controller:
      roleName: Controller
      roleCount: 3
      networks:
        - ctlplane
        - internal_api
        - external
        - tenant
        - storage
        - storage_mgmt
      cores: 12
      memory: 64
      rootDisk:
        diskSize: 100
        baseImageVolumeName: openstack-base-img 5
        storageClass: host-nfs-storageclass 6
        storageAccessMode:  ReadWriteMany
        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: "17.1"
```

1. The name of the overcloud control plane, for example, **overcloud**.
2. The OSPdO namespace, for example, **openstack**.
3. The configuration for the control plane.
4. Optional: The **Secret** resource that provides root access on each node to users with the password.
5. The name of the data volume that stores the base operating system image for your Controller VMs. For more information on creating the data volume, see Creating a data volume for the base operating system.
6. For information on configuring Red Hat OpenShift Container Platform (RHOCP) storage, see Dynamic provisioning.

2. Save the **openstack-controller.yaml** file.

3. Create the control plane:

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

4. Wait until RHOCP creates the resources related to **OpenStackControlPlane** CR. OSPdO also creates an **OpenStackClient** pod that you can access through a remote shell to 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 VM set:

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

3. View the VMs to verify the creation of the control plane OpenShift Virtualization VMs:

```
$ oc get virtualmachines
```

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

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

5.2. CREATING COMPUTE NODES WITH THE OPENSTACKBAREMETALSET CRD

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

Define an **OpenStackBaremetalSet** custom resource (CR) to create Compute nodes from bare-metal machines that the Red Hat OpenShift Container Platform (RHOCP) manages.

TIP

For descriptions of the values you can use to configure your **OpenStackBareMetalSet** CR, view the **OpenStackBareMetalSet** CRD specification schema:

```
$ oc describe crd openstackbaremetalset
```

Prerequisites

- You have used the **OpenStackNetConfig** CR to create a control plane network and any additional isolated networks.
- You have created a control plane with the ` **OpenStackControlPlane** CRD.

Procedure

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

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackBaremetalSet
metadata:
  name: compute (1)
  namespace: openstack (2)
spec: (3)
  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 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 (4)
```

(1) The name of the Compute node bare-metal set, for example, **compute**.

(2) The OSPdO namespace, for example, **openstack**.

(3) The configuration for the Compute nodes.

(4) Optional: The **Secret** resource that provides root access on each node to users with the password.

2. Save the **openstack-compute.yaml** file.

3. 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 RHOCP manages to verify the creation of the Compute nodes:

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

5.3. CREATING A PROVISIONING SERVER WITH THE OPENSTACKPROVISIONSERVER CRD

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** CR is automatically created for any **OpenStackBaremetalSet** CRs you create. You can create the **OpenStackProvisionServer** CR manually and provide the name to any **OpenStackBaremetalSet** CRs that you create.

The **OpenStackProvisionServer** CRD creates an Apache server on the Red Hat OpenShift Container Platform (RHOCP) provisioning network for a specific RHEL QCOW2 image.

Procedure

1. Create a file named **openstack-provision.yaml** on your workstation. Include the resource specification for the Provisioning server. The following example defines a 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 1
  namespace: openstack 2
spec:
  baseImageUrl: http://<source_host>/rhel-guest-image-8.4-992.x86_64.qcow2 3
  port: 8080 4
```

1. The name that identifies the **OpenStackProvisionServer** CR.
2. The OSPdO namespace, for example, **openstack**.
3. The initial source of the RHEL QCOW2 image for the Provisioning server. The image is downloaded from this remote source when the server is created.
4. The Provisioning server port, set to 8080 by default. You can change it for a specific port configuration.

   For further descriptions of the values you can use to configure your **OpenStackProvisionServer** CR, view the **OpenStackProvisionServer** CRD specification schema:

```
$ oc describe crd openstackprovisionserver
```

2. Save the **openstack-provision.yaml** file.
3. Create the Provisioning Server:

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

4. Verify that the resource for the Provisioning server is created:

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

CHAPTER 6. CONFIGURING AND DEPLOYING THE OVERCLOUD WITH DIRECTOR OPERATOR

You can configure your overcloud nodes 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 THE OPENSTACKCONFIGGENERATOR CRD

After you provision the overcloud infrastructure, you must create a set of Ansible playbooks to configure Red Hat OpenStack Platform (RHOSP) on the overcloud nodes. You use the **OpenStackConfigGenerator** custom resource definition (CRD) to create these playbooks. The **OpenStackConfigGenerator** CRD uses the RHOSP director **config-download** feature to convert heat configuration to playbooks.

TIP

For descriptions of the values you can use to configure your **OpenStackConfigGenerator** custom resource (CR), view the **OpenStackConfigGenerator** CRD specification schema:

```
$ oc describe crd openstackconfiggenerator
```

Prerequisites

- You have created a control plane with the ` **OpenStackControlPlane** CRD.
- You have created Compute nodes with the ` **OpenStackBaremetalSets** CRD.
- You have created a **ConfigMap** object that contains your custom heat templates.
- You have created a **ConfigMap** object 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. The following example defines a specification to generate the playbooks:

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackConfigGenerator
metadata:
  name: default (1)
  namespace: openstack
spec:
  enableFencing: true (2)
  gitSecret: git-secret (3)
  imageURL: registry.redhat.io/rhosp-rhel8/openstack-tripleoclient:17.1
  heatEnvConfigMap: heat-env-config (4)
  # List of heat environment files to include from tripleo-heat-templates/environments
  heatEnvs: (5)
```

```
    - ssl/tls-endpoints-public-dns.yaml
    - ssl/enable-tls.yaml
  tarballConfigMap: tripleo-tarball-config 6
```

1. The name of the config generator, which is **default** by default.
2. Set to **true** to enable the automatic creation of required heat environment files to enable fencing. Production RHOSP environments must have fencing enabled. Virtual machines running Pacemaker require the **fence-agents-kubevirt** package.
3. Set to the **ConfigMap** object that contains the Git authentication credentials, by default **git-secret**.
4. The **ConfigMap** object that contains your custom environment files, by default **heat-env-config**.
5. A list of the default heat environment files, provided by TripleO in the **tripleo-heat-templates/environments** directory, to use to generate the playbooks.
6. The **ConfigMap** object that contains the tarball with your custom heat templates, by default **tripleo-tarball-config**.

2. Optional: To change the location of the container images the **OpenStackConfigGenerator** CR uses to create the ephemeral heat service, add the following configuration to your **openstack-config-generator.yaml** file:

```
spec:
  ...
  ephemeralHeatSettings:
    heatAPIImageURL: <heat_api_image_location>
    heatEngineImageURL: <heat_engine_image_location>
    mariadbImageURL: <mariadb_image_location>
    rabbitImageURL: <rabbitmq_image_location>
```

   - Replace **<heat_api_image_location>** with the path to the directory where you host your heat API image, **openstack-heat-api**.
   - Replace **<heat_engine_image_location>** with the path to the directory where you host your heat engine image, **openstack-heat-engine**.
   - Replace **<mariadb_image_location>** with the path to the directory where you host your MariaDB image, **openstack-mariadb**.
   - Replace **<rabbitmq_image_location>** with the path to the directory where you host your RabbitMQ image, **openstack-rabbitmq**.

3. Optional: To create the Ansible playbooks for configuration generation in debug mode, add the following configuration to your **openstack-config-generator.yaml** file:

```
spec:
  ...
  interactive: true
```

   For more information on debugging an **OpenStackConfigGenerator** pod in interactive mode, see Debugging configuration generation.

4. Save the **openstack-config-generator.yaml** file.
5. Create the Ansible config generator:

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

6. Verify that the resource for the config generator is created:

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

6.2. REGISTERING THE OPERATING SYSTEM OF YOUR OVERCLOUD

Before director Operator (OSPdO) configures the overcloud 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 part of the **OpenStackControlPlane** CR, OSPdO creates an **OpenStackClient** pod that you access through a Remote Shell (RSH) to 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 OpenStackClient pod.

Procedure

1. Open an RSH connection to the **OpenStackClient** pod:

```
$ 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-9.2-for-x86_64-baseos-eus-rpms
      - rhel-9.2-for-x86_64-appstream-eus-rpms
      - rhel-9.2-for-x86_64-highavailability-eus-rpms
      - openstack-17.1-for-rhel-9-x86_64-rpms
      - fast-datapath-for-rhel-9-x86_64-rpms
      - rhceph-5-tools-for-rhel-9-x86_64-rpms
  tasks:
    - name: Register system 1
      redhat_subscription:
        username: myusername
        password: p@55w0rd!
```

```
    org_id: 1234567
    release: {rhelvernum}
    pool_ids: 1a85f9223e3d5e43013e3d6e8ff506fd
  - name: Disable all repos 2
    command: "subscription-manager repos --disable *"
  - name: Enable Controller node repos 3
    command: "subscription-manager repos --enable {{ item }}"
    with_items: "{{ repos }}"
```

1. Task that registers the node.
2. Task that disables any auto-enabled repositories.
3. Task that enables only the repositories relevant to the Controller node. The repositories are listed with the **repos** variable.

4. Register the overcloud nodes to the 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
- Overcloud repositories

6.3. APPLYING OVERCLOUD CONFIGURATION WITH DIRECTOR OPERATOR

You can configure the overcloud with director Operator (OSPdO) 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** custom resource (CR), OSPdO creates a job that runs the Ansible playbooks to configure the overcloud.

TIP

For descriptions of the values you can use to configure your **OpenStackDeploy** CR, view the **OpenStackDeploy** CRD specification schema:

```
$ oc describe crd openstackdeploy
```

Prerequisites

- You have created a control plane with the **OpenStackControlPlane** CRD.
- You have created Compute nodes with the **OpenStackBaremetalSets** CRD.
- You have used the **OpenStackConfigGenerator** CRD to create the Ansible playbook configuration for your overcloud.

Procedure

1. Retrieve the **hash/digest** of the latest **OpenStackConfigVersion** object, which represents the Ansible playbooks that should be used to configure the overcloud:

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

2. Create a file named **openstack-deployment.yaml** on your workstation and include the resource specification to the Ansible playbooks:

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackDeploy
metadata:
  name: default
spec:
  configVersion: <config_version>
  configGenerator: default
```

   - Replace **<config_version>** with the Ansible playbooks **hash/digest** retrieved in step 1, for example, **n5fch96h548h75hf4hbdhb8hfdh676h57bh96h5c5h59hf4h88h...**.

3. Save the **openstack-deployment.yaml** file.

4. 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 view the logs of the job to watch the Ansible playbooks running:

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

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

6.4. DEBUGGING CONFIGURATION GENERATION

To debug configuration generation operations, you can set the **OpenStackConfigGenerator** CR to use interactive mode. In interactive mode, the **OpenStackConfigGenerator** CR creates the environment to start rendering the playbooks, but does not automatically render the playbooks.

Prerequisites

- Your **OpenStackConfigGenerator** CR was created in interactive mode:

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

```
spec:
  …
  interactive: true
```

- The **OpenStackConfigGenerator** pod with the prefix **generate-config** has started.

Procedure

1. Open a Remote Shell (RSH) connection to the **OpenStackConfigGenerator** pod:

```
$ oc rsh $(oc get pod -o name -l job-name=generate-config-default)
```

2. Inspect the files and playbook rendering:

```
$ 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 that stores the files auto-rendered by OSPdO.
2. Directory that stores the environment files specified with the **heatEnvConfigMap** option.
3. Directory that stores the overcloud service passwords created by OSPdO.
4. Script that renders the Ansible playbooks.
5. Internal script used by **create-playbooks** to replicate the undocumented heat client merging of map parameters.
6. Directory that stores the tarball specified with the **tarballConfigMap** option.

CHAPTER 7. DEPLOYING A RHOSP HYPERCONVERGED INFRASTRUCTURE (HCI) WITH DIRECTOR OPERATOR

You can use director Operator (OSPdO) to deploy an overcloud with hyperconverged infrastructure (HCI). An overcloud with HCI colocates Compute and Red Hat Ceph Storage OSD services on the same nodes.

7.1. PREREQUISITES

- Your Compute HCI nodes require extra disks to use as OSDs.
- You have installed and prepared OSPdO on an operational Red Hat OpenShift Container Platform (RHOCP) cluster. For more information, see Installing and preparing director Operator.
- You have created the overcloud networks by using the **OpenStackNetConfig** custom resource definition (CRD), including the control plane and any isolated networks. For more information, see Creating networks with director Operator .
- You have created **ConfigMaps** to store any custom heat templates and environment files for your overcloud. For more information, see Customizing the overcloud with director Operator .
- You have created a control plane and bare-metal Compute nodes for your overcloud. For more information, see Creating overcloud nodes with director Operator .
- You have created and applied an **openstackconfiggenerator** resource to render Ansible playbooks for overcloud configuration.

7.2. CREATING A ROLES_DATA.YAML FILE WITH THE COMPUTE HCI ROLE FOR 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 that you use **roles_data.yaml** as the file name.

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 -o roles_data.yaml Controller ComputeHCI
```

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

Next steps

- Configuring HCI networking in director Operator

7.3. CONFIGURING HCI NETWORKING IN DIRECTOR OPERATOR

Create directories on your workstation to store your custom templates and environment files, and configure the NIC templates for your Compute HCI role.

Procedure

1. Create a directory for your custom templates:

```
$ mkdir custom_templates
```

2. Create a custom template file named **multiple_nics_vlans_dvr.j2** in your **custom_templates** directory.

3. Add configuration for the NICs of your bare-metal nodes to your **multiple_nics_vlans_dvr.j2** file. For an example NIC configuration file, see Custom NIC heat template for HCI Compute nodes.

4. Create a directory for your custom environment files:

```
$ mkdir custom_environment_files
```

5. Map the NIC template for your overcloud role in the **network-environment.yaml** environment file in your **custom_environment_files** directory:

```
parameter_defaults:
  ComputeHCINetworkConfigTemplate: 'multiple_nics_vlans_dvr.j2'
```

Additional resources

- Custom network interface templates

Next steps

- Adding custom templates to the overcloud configuration

7.4. 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. The configuration in the heat template maps the networks to the following bridges and interfaces:

Networks	Bridge	Interface
Control Plane, Storage, Internal API	N/A	**nic3**
External, Tenant	**br-ex**	**nic4**

To use the following template in your deployment, copy the example to **multiple_nics_vlans_dvr.j2** in your **custom_templates** directory on your workstation. You can modify this configuration for the NIC configuration of your bare-metal nodes.

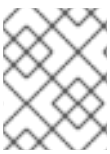
Example

```
{% set mtu_list = [ctlplane_mtu] %}
{% for network in role_networks %}
{{ mtu_list.append(lookup('vars', networks_lower[network] ~ '_mtu')) }}
{%- endfor %}
{% set min_viable_mtu = mtu_list | max %}
network_config:
# BMH provisioning interface used for ctlplane
- type: interface
  name: nic1
  mtu: 1500
  use_dhcp: false
  dns_servers: {{ ctlplane_dns_nameservers }}
  domain: {{ dns_search_domains }}
  addresses:
  - ip_netmask: {{ ctlplane_ip }}/{{ ctlplane_subnet_cidr }}
  routes: {{ ctlplane_host_routes }}
# Disable OCP cluster interface
- type: interface
  name: nic2
  mtu: 1500
  use_dhcp: false
{% for network in networks_all if network not in networks_skip_config|default([]) %}
{% if network == 'External' %}
- type: ovs_bridge
  name: {{ neutron_physical_bridge_name }}
  mtu: {{ lookup('vars', networks_lower[network] ~ '_mtu') }}
  dns_servers: {{ ctlplane_dns_nameservers }}
  use_dhcp: false
```

```
{% if network in role_networks %}
  addresses:
  - ip_netmask:
      {{ lookup('vars', networks_lower[network] ~ '_ip') }}/{{ lookup('vars', networks_lower[network] ~
'_cidr') }}
  routes: {{ lookup('vars', networks_lower[network] ~ '_host_routes') }}
{% endif %}
  members:
  - type: interface
    name: nic3
    mtu: {{ lookup('vars', networks_lower[network] ~ '_mtu') }}
    primary: true
{% endif %}
{% endfor %}
- type: ovs_bridge
  name: br-tenant
  mtu: {{ min_viable_mtu }}
  use_dhcp: false
  members:
  - type: interface
    name: nic4
    mtu: {{ min_viable_mtu }}
    use_dhcp: false
    primary: true
{% for network in networks_all if network not in networks_skip_config|default([]) %}
{% if network not in ["External"] and network in role_networks %}
  - type: vlan
    mtu: {{ lookup('vars', networks_lower[network] ~ '_mtu') }}
    vlan_id: {{ lookup('vars', networks_lower[network] ~ '_vlan_id') }}
    addresses:
    - ip_netmask:
        {{ lookup('vars', networks_lower[network] ~ '_ip') }}/{{ lookup('vars', networks_lower[network] ~
'_cidr') }}
    routes: {{ lookup('vars', networks_lower[network] ~ '_host_routes') }}
{% endif %}
{% endfor %}
```

7.5. ADDING CUSTOM TEMPLATES TO THE OVERCLOUD CONFIGURATION

Director Operator (OSPdO) converts a core set of overcloud heat templates into Ansible playbooks that you apply to provisioned nodes when you are ready to configure the Red Hat OpenStack Platform (RHOSP) 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** object named **tripleo-tarball-config**. This tarball file can contain complex directory structures to extend the core set of templates. OSPdO 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.

Prerequisites

- The custom overcloud 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 gzipped tarball:

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

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

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

4. Verify that the **ConfigMap** CR is created:

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

Additional resources

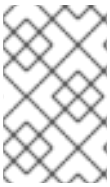
- Creating and using config maps
- Understanding heat templates

Next steps

- Adding custom environment files to the overcloud configuration

7.6. CUSTOM ENVIRONMENT FILE FOR CONFIGURING HYPERCONVERGED INFRASTRUCTURE (HCI) STORAGE IN DIRECTOR OPERATOR

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

```
resource_registry:
  OS::TripleO::Services::CephMgr: deployment/cephadm/ceph-mgr.yaml
  OS::TripleO::Services::CephMon: deployment/cephadm/ceph-mon.yaml
```

```
  OS::TripleO::Services::CephOSD: deployment/cephadm/ceph-osd.yaml
  OS::TripleO::Services::CephClient: deployment/cephadm/ceph-client.yaml

parameter_defaults:
  CephDynamicSpec: true
  CephSpecFqdn: true
  CephConfigOverrides:
    rgw_swift_enforce_content_length: true
    rgw_swift_versioning_enabled: true
    osd:
      osd_memory_target_autotune: true
      osd_numa_auto_affinity: true
    mgr:
      mgr/cephadm/autotune_memory_target_ratio: 0.2

  CinderEnableIscsiBackend: false
  CinderEnableRbdBackend: true
  CinderBackupBackend: ceph
  CinderEnableNfsBackend: false
  NovaEnableRbdBackend: true
  GlanceBackend: rbd
  CinderRbdPoolName: "volumes"
  NovaRbdPoolName: "vms"
  GlanceRbdPoolName: "images"
  CephPoolDefaultPgNum: 32
  CephPoolDefaultSize: 2
```

7.7. ADDING CUSTOM ENVIRONMENT FILES TO THE OVERCLOUD CONFIGURATION

To enable features or set parameters in the overcloud, you must include environment files with your overcloud deployment. Director Operator (OSPdO) uses a **ConfigMap** object named **heat-env-config** to store and retrieve environment files. The **ConfigMap** object stores the environment files in the following format:

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

For example, the following **ConfigMap** contains two environment files:

```
...
data:
  network_environment.yaml: |+
    parameter_defaults:
      ComputeNetworkConfigTemplate: 'multiple_nics_vlans_dvr.j2'
  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
```

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

Prerequisites

- The custom environment files for your overcloud deployment.

Procedure

1. Create the **heat-env-config ConfigMap** object:

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

   - Replace **<dir_custom_environment_files>** with the directory that contains the environment files you want to use in your overcloud deployment. The **ConfigMap** object stores these as individual **data** entries.

2. Verify that the **heat-env-config ConfigMap** object contains all the required environment files:

```
$ oc get configmap/<configmap_name> -n openstack
```

7.8. CREATING HCI COMPUTE NODES AND DEPLOYING THE OVERCLOUD

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

Define an **OpenStackBaremetalSet** custom resource (CR) to create Compute nodes from bare-metal machines that the Red Hat OpenShift Container Platform (RHOCP) manages.

TIP

For descriptions of the values you can use to configure your **OpenStackBareMetalSet** CR, view the **OpenStackBareMetalSet** CRD specification schema:

```
$ oc describe crd openstackbaremetalset
```

Prerequisites

- You have used the **OpenStackNetConfig** CR to create a control plane network and any additional isolated networks.
- You have created a control plane with the **OpenStackControlPlane** CRD.

Procedure

1. Create a file named **openstack-hcicompute.yaml** on your workstation. Include the resource specification for the HCI Compute nodes. For example, the specification for 3 HCI Compute nodes is as follows:

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackBaremetalSet
metadata:
  name: computehci (1)
  namespace: openstack (2)
spec: (3)
  count: 3
  baseImageUrl: http://host/images/rhel-image-9.2.x86_64.qcow2
  deploymentSSHSecret: osp-controlplane-ssh-keys
  ctlplaneInterface: enp8s0
  networks:
    - ctlplane
    - internal_api
    - tenant
    - storage
    - storage_mgmt
  roleName: ComputeHCI
  passwordSecret: userpassword (4)
```

(1) The name of the HCI Compute node bare metal set, for example, **computehci**.

(2) The OSPdO namespace, for example, **openstack**.

(3) The configuration for the HCI Compute nodes.

(4) Optional: The **Secret** resource that provides root access on each node to users with the password.

2. Save the **openstack-hcicompute.yaml** file.

3. Create the HCI Compute nodes:

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

4. Verify that the resource for the HCI Compute nodes is created:

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

5. To verify the creation of the HCI Compute nodes, view the bare-metal machines that RHOCP manages:

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

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

7. Register the operating system of your overcloud. For more information, see Registering the operating system of your overcloud.

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

CHAPTER 8. DEPLOYING RHOSP WITH AN EXTERNAL RED HAT CEPH STORAGE CLUSTER WITH DIRECTOR OPERATOR

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

Prerequisites

- You have an external Red Hat Ceph Storage cluster.
- You have installed and prepared OSPdO on an operational Red Hat OpenShift Container Platform (RHOCP) cluster. For more information, see Installing and preparing director Operator.
- You have created the overcloud networks by using the **OpenStackNetConfig** custom resource definition (CRD), including the control plane and any isolated networks. For more information, see Creating networks with director Operator .
- You have created **ConfigMaps** to store any custom heat templates and environment files for your overcloud. For more information, see Customizing the overcloud with director Operator .
- You have created a control plane and bare-metal Compute nodes for your overcloud. For more information, see Creating overcloud nodes with director Operator .
- You have created and applied an **openstackconfiggenerator** resource to render Ansible playbooks for overcloud configuration.

8.1. CONFIGURING NETWORKING FOR THE COMPUTE ROLE IN DIRECTOR OPERATOR

Create directories on your workstation to store your custom templates and environment files, and configure the NIC templates for your Compute role.

Procedure

1. Create a directory for your custom templates:

```
$ mkdir custom_templates
```

2. Create a custom template file named **multiple_nics_vlans_dvr.j2** in your **custom_templates** directory.
3. Add configuration for the NICs of your bare-metal Compute nodes to your **multiple_nics_vlans_dvr.j2** file. For an example NIC configuration file, see Custom NIC heat template for Compute nodes.
4. Create a directory for your custom environment files:

```
$ mkdir custom_environment_files
```

5. Map the NIC template for your overcloud role in the **network-environment.yaml** environment file in your **custom_environment_files** directory:

```
parameter_defaults:
  ComputeNetworkConfigTemplate: 'multiple_nics_vlans_dvr.j2'
```

Additional resources

- Custom network interface templates

8.2. CUSTOM NIC HEAT TEMPLATE FOR COMPUTE NODES

The following example is a heat template that contains NIC configuration for the Compute bare-metal nodes in an overcloud that connects to an external Red Hat Ceph Storage cluster. The configuration in the heat template maps the networks to the following bridges and interfaces:

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

To use the following template in your deployment, copy the example to **multiple_nics_vlans_dvr.j2** in your **custom_templates** directory on your workstation. You can modify this configuration for the NIC configuration of your bare-metal nodes.

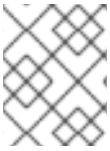
Example

```
{% set mtu_list = [ctlplane_mtu] %}
{% for network in role_networks %}
{{ mtu_list.append(lookup('vars', networks_lower[network] ~ '_mtu')) }}
{%- endfor %}
{% set min_viable_mtu = mtu_list | max %}
network_config:
# BMH provisioning interface used for ctlplane
- type: interface
  name: nic1
  mtu: 1500
  use_dhcp: false
  dns_servers: {{ ctlplane_dns_nameservers }}
  domain: {{ dns_search_domains }}
  addresses:
  - ip_netmask: {{ ctlplane_ip }}/{{ ctlplane_subnet_cidr }}
  routes: {{ ctlplane_host_routes }}
# Disable OCP cluster interface
- type: interface
  name: nic2
  mtu: 1500
  use_dhcp: false
{% for network in networks_all if network not in networks_skip_config|default([]) %}
{% if network == 'External' %}
- type: ovs_bridge
  name: {{ neutron_physical_bridge_name }}
  mtu: {{ lookup('vars', networks_lower[network] ~ '_mtu') }}
```

```
    dns_servers: {{ ctlplane_dns_nameservers }}
    use_dhcp: false
{% if network in role_networks %}
    addresses:
    - ip_netmask:
        {{ lookup('vars', networks_lower[network] ~ '_ip') }}/{{ lookup('vars', networks_lower[network] ~
'_cidr') }}
    routes: {{ lookup('vars', networks_lower[network] ~ '_host_routes') }}
{% endif %}
    members:
    - type: interface
      name: nic3
      mtu: {{ lookup('vars', networks_lower[network] ~ '_mtu') }}
      primary: true
{% endif %}
{% endfor %}
- type: ovs_bridge
    name: br-tenant
    mtu: {{ min_viable_mtu }}
    use_dhcp: false
    members:
    - type: interface
      name: nic4
      mtu: {{ min_viable_mtu }}
      use_dhcp: false
      primary: true
{% for network in networks_all if network not in networks_skip_config|default([]) %}
{% if network not in ["External"] and network in role_networks %}
    - type: vlan
      mtu: {{ lookup('vars', networks_lower[network] ~ '_mtu') }}
      vlan_id: {{ lookup('vars', networks_lower[network] ~ '_vlan_id') }}
      addresses:
      - ip_netmask:
          {{ lookup('vars', networks_lower[network] ~ '_ip') }}/{{ lookup('vars', networks_lower[network] ~
'_cidr') }}
      routes: {{ lookup('vars', networks_lower[network] ~ '_host_routes') }}
{% endif %}
{% endfor %}
```

8.3. ADDING CUSTOM TEMPLATES TO THE OVERCLOUD CONFIGURATION

Director Operator (OSPdO) converts a core set of overcloud heat templates into Ansible playbooks that you apply to provisioned nodes when you are ready to configure the Red Hat OpenStack Platform (RHOSP) 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** object named **tripleo-tarball-config**. This tarball file can contain complex directory structures to extend the core set of templates. OSPdO 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.

Prerequisites

- The custom overcloud 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 gzipped tarball:

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

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

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

4. Verify that the **ConfigMap** CR is created:

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

Additional resources

- Creating and using config maps
- Understanding heat templates

Next steps

- Adding custom environment files to the overcloud configuration

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

To integrate with an external Red Hat Ceph Storage cluster, include an environment file with parameters and values similar to those shown in the following example. The example enables the **CephExternal** and **CephClient** services on your overcloud nodes, and sets the pools for different RHOSP services.

NOTE

You can modify this configuration to suit your storage configuration.

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

```
resource_registry:
```

```
  OS::TripleO::Services::CephExternal: deployment/cephadm/ceph-client.yaml

parameter_defaults:
  CephClusterFSID: '4b5c8c0a-ff60-454b-a1b4-9747aa737d19' 1
  CephClientKey: 'AQDLOh1VgEp6FRAAFzT7Zw+Y9V6JJExQAsRnRQ==' 2
  CephExternalMonHost: '172.16.1.7, 172.16.1.8' 3
  ExternalCeph: true

  # the following parameters enable Ceph backends for Cinder, Glance, Gnocchi and Nova
  NovaEnableRbdBackend: true
  CinderEnableRbdBackend: true
  CinderBackupBackend: ceph
  GlanceBackend: rbd
  # Uncomment below if enabling legacy telemetry
  # GnocchiBackend: rbd
  # If the Ceph pools which host VMs, Volumes and Images do not match these
  # names OR the client keyring to use is not named 'openstack',  edit the
  # following as needed.
  NovaRbdPoolName: vms
  CinderRbdPoolName: volumes
  CinderBackupRbdPoolName: backups
  GlanceRbdPoolName: images
  # Uncomment below if enabling legacy telemetry
  # GnocchiRbdPoolName: metrics
  CephClientUserName: openstack

  # finally we disable the Cinder LVM backend
  CinderEnableIscsiBackend: false
```

1. The file system ID of your external Red Hat Ceph Storage cluster.
2. The Red Hat Ceph Storage client key for your external Red Hat Ceph Storage cluster.
3. A comma-delimited list of the IPs of all MON hosts in your external Red Hat Ceph Storage cluster.

Additional resources

- Integrating the overcloud with an existing Red Hat Ceph Storage Cluster
- Red Hat Container Registry Authentication

8.5. ADDING CUSTOM ENVIRONMENT FILES TO THE OVERCLOUD CONFIGURATION

To enable features or set parameters in the overcloud, you must include environment files with your overcloud deployment. Director Operator (OSPdO) uses a **ConfigMap** object named **heat-env-config** to store and retrieve environment files. The **ConfigMap** object stores the environment files in the following format:

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

For example, the following **ConfigMap** contains two environment files:

```
...
data:
  network_environment.yaml: |+
    parameter_defaults:
      ComputeNetworkConfigTemplate: 'multiple_nics_vlans_dvr.j2'
  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
```

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

Prerequisites

- The custom environment files for your overcloud deployment.

Procedure

1. Create the **heat-env-config ConfigMap** object:

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

   - Replace **<dir_custom_environment_files>** with the directory that contains the environment files you want to use in your overcloud deployment. The **ConfigMap** object stores these as individual **data** entries.

2. Verify that the **heat-env-config ConfigMap** object contains all the required environment files:

```
$ oc get configmap/<configmap_name> -n openstack
```

8.6. CREATING COMPUTE NODES AND DEPLOYING THE OVERCLOUD

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

Define an **OpenStackBaremetalSet** custom resource (CR) to create Compute nodes from bare-metal machines that the Red Hat OpenShift Container Platform (RHOCP) manages.

TIP

For descriptions of the values you can use to configure your **OpenStackBareMetalSet** CR, view the **OpenStackBareMetalSet** CRD specification schema:

```
$ oc describe crd openstackbaremetalset
```

Prerequisites

- You have used the **OpenStackNetConfig** CR to create a control plane network and any additional isolated networks.
- You have created a control plane with the **OpenStackControlPlane** CRD.

Procedure

1. Create your Compute nodes by using the **OpenStackBaremetalSet** CRD. For more information, see Creating Compute nodes with the OpenStackBaremetalSet CRD .
2. Create the Ansible playbooks for overcloud configuration with the **OpenStackConfigGenerator** CRD. For more information, see Creating Ansible playbooks for overcloud configuration with the OpenStackConfigGenerator CRD.
3. Register the operating system of your overcloud. For more information, see Registering the operating system of your overcloud.
4. Apply the overcloud configuration. For more information, see Applying overcloud configuration with director Operator.

CHAPTER 9. ACCESSING AN OVERCLOUD DEPLOYED WITH DIRECTOR OPERATOR

After you deploy the overcloud with director Operator (OSPdO), 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 OSPdO deploys as a part of the **OpenStackControlPlane** resource that you created.

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 director Operator (OSPdO), 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 OSPdO.

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 and managing instances
- Configuring Red Hat OpenStack Platform networking

9.2. ACCESSING THE OVERCLOUD DASHBOARD

You access the dashboard of an overcloud that you deploy with director Operator (OSPdO) by using the same method as a standard overcloud: access the virtual IP address reserved by the control plane by using a web browser.

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. Retrieve the IP address reserved for the control plane from your **OpenStackNetConfig** CR:

```
spec:
  ...
  reservations:
    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
```

3. Open a web browser.

4. Enter the IP address for the control plane in the URL field.

5. Log in to the dashboard with your 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 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, you create a new **OpenStackConfigGenerator** resource to generate a new set of Ansible playbooks, then use the **OpenStackConfigVersion** to create or update the **OpenStackDeploy** object to reapply the Ansible configuration to your overcloud.

Procedure

1. Check that you have enough hosts in a ready state in the **openshift-machine-api** namespace:

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

   For more information on managing your bare-metal hosts, see Managing bare metal hosts.

2. Increase the **count** parameter for the **compute OpenStackBaremetalSet** resource:

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

   The **OpenStackBaremetalSet** resource automatically provisions the new nodes with the Red Hat Enterprise Linux base operating system.

3. 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
   ```

4. Generate the Ansible playbooks by using **OpenStackConfigGenerator** and apply the overcloud configuration. For more information, see Configuring and deploying the overcloud with director Operator.

Additional resources

- Managing bare metal hosts

10.2. REMOVING COMPUTE NODES FROM YOUR OVERCLOUD WITH 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 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. If Instance HA is enabled, choose one of the following options:

   - If the Compute node is accessible, log in to the Compute node as the **root** user and perform a clean shutdown with the **shutdown -h now** command.
   - If the Compute node is not accessible, log in to a Controller node as the **root** user, disable the STONITH device for the Compute node, and shut down the bare metal node:

```
[root@controller-0 ~]# pcs stonith disable <stonith_resource_name>
[stack@undercloud ~]$ source stackrc
[stack@undercloud ~]$ openstack baremetal node power off <UUID>
```

5. Exit from **openstackclient**:

```
$ exit
```

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 deleted node.
- Director Operator flags the IP reservation entry in the **OpenStackNet** resource as deleted.

```
$ oc get osnet ctlplane -o json -n openstack | jq .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. DEPLOYING TLS FOR PUBLIC ENDPOINTS USING DIRECTOR OPERATOR

Deploy the overcloud using TLS to create public endpoint IPs or DNS names for director Operator (OSPdO).

Prerequisites

- You have installed OSPdO on an operational Red Hat OpenShift Container Platform (RHOSP) cluster.
- You have installed the **oc** command line tool on your workstation.
- You have created the certificate authority, key, and certificate. For information, see Enabling SSL/TLS on overcloud public endpoints.

11.1. TLS FOR PUBLIC ENDPOINT IP ADDRESSES

To reference public endpoint IP addresses, add your CA certificates to the **openstackclient** pod by creating a **ConfigMap** resource to store the CA certificates, then referencing that **ConfigMap** resource in the **OpenStackControlPlane** resource.

Procedure

1. Create a **ConfigMap** resource to store the CA certificates:

```
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** resource and reference the **ConfigMap** resource:

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

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

3. Create a file in the **~/custom_environment_files** directory named **tls-certs.yaml**, that specifies the generated certificates for the deployment by using the **SSLCertificate**, **SSLIntermediateCertificate**, **SSLKey**, and **CAMap** parameters.

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** resource 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. Generate the Ansible playbooks by using **OpenStackConfigGenerator** and apply the overcloud configuration. For more information, see Configuring and deploying the overcloud with director Operator.

11.2. TLS FOR PUBLIC ENDPOINT DNS NAMES

To reference public endpoint DNS names, add your CA certificates to the **openstackclient** pod by creating a **ConfigMap** resource to store the CA certificates, then referencing that **ConfigMap** resource in the **OpenStackControlPlane** resource.

Procedure

1. Create a **ConfigMap** resource to store the CA certificates:

```
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** resource and reference the **ConfigMap** resource:

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

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

3. Create a file in the **~/custom_environment_files** directory named **tls-certs.yaml**, that specifies the generated certificates for the deployment by using the **SSLCertificate**, **SSLIntermediateCertificate**, **SSLKey**, and **CAMap** parameters.

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** resource 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. Generate the Ansible playbooks by using **OpenStackConfigGenerator** and apply the overcloud configuration. For more information, see Configuring and deploying the overcloud with director Operator.

CHAPTER 12. 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**.

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

Define your **OpenStackNetConfig** custom resource (CR) and specify the subnets for the overcloud networks. Red Hat OpenStack Platform (RHOSP) director Operator (OSPdO) then renders the configuration and creates, or updates, the network topology.

Prerequisites

- You have installed OSPdO on an operational Red Hat OpenShift Container Platform (RHOCP) cluster.
- You have installed the **oc** command line tool on your workstation.

Procedure

1. Create a configuration file named **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
```

3. Verify that the resources and child resources for the **OpenStackNetConfig** resource are created:

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

12.2. 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. If the leaf network roles have different NIC configurations, you can create Ansible NIC templates for each role to configure the spine-leaf networking, register the NIC templates, and create the **ConfigMap** custom resource.

NOTE

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

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:
    - compute
    - 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:
    - compute
    - external_bridge
  networks:
    InternalApi:
      subnet: internal_api_leaf2
    Tenant:
      subnet: tenant_leaf2
    Storage:
      subnet: storage_leaf2
  HostnameFormatDefault: '%stackname%-novacompute-leaf2-%index%'
...
```

2. Create a NIC template for each Compute role. For example Ansible NIC templates, see https://github.com/openstack/tripleo-ansible/tree/stable/wallaby/tripleo_ansible/roles/tripleo_network_config/templates.

3. Add the NIC templates for the new nodes to an environment file:

```
parameter_defaults:
  ComputeNetworkConfigTemplate: 'multiple_nics_vlans_dvr.j2'
  ComputeLeaf1NetworkConfigTemplate: 'multiple_nics_vlans_dvr.j2'
  ComputeLeaf2NetworkConfigTemplate: 'multiple_nics_compute_leaf_2_vlans_dvr.j2'
```

4. In the **~/custom_environment_files** directory, archive the **roles_data.yaml** file, the environment file, and the NIC templates into a tarball:

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

5. Create the **tripleo-tarball-config ConfigMap** resource:

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

12.3. 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 the spine-leaf network, and then render and apply the Ansible playbooks. To create the control plane, specify the resources for the Controller nodes. To create the Compute nodes for the leafs from bare-metal machines, include the resource specification in the **OpenStackBaremetalSet** custom resource.

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-rhel9/openstack-tripleoclient:17.1
  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: 50
      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
```

3. Wait until Red Hat OpenShift Container Platform (RHOCP) creates the resources related to the **OpenStackControlPlane** resource.

4. Create a file on your workstation for each Compute leaf, for example, **openstack-computeleaf1.yaml**. Include the resource specification for the Compute nodes for the leaf. The following example shows a specification for one Compute leaf that includes one Compute 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-9.2.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
```

5. Create the Compute nodes for each leaf:

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

6. Generate the Ansible playbooks by using **OpenStackConfigGenerator** and apply the overcloud configuration. For more information, see Configuring and deploying the overcloud with director Operator.

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 (VM) set:

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

3. View the VM resources to verify the creation of the control plane VMs in OpenShift Virtualization:

```
$ oc get virtualmachines
```

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

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

5. View the resource for each Compute leaf:

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

6. View the bare-metal machines managed by RHOCP to verify the creation of the Compute nodes:

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

CHAPTER 13. BACKING UP AND RESTORING DIRECTOR OPERATOR

Red Hat OpenStack Platform (RHOSP) director Operator (OSPdO) provides custom resource definitions (CRDs) for backing up and restoring a deployment. You do not have to manually export and import multiple configurations. OSPdO knows which custom resources (CRs), including the **ConfigMap** and **Secret** CRs, that it needs to create a complete backup because it is aware of the state of all resources. Therefore, OSPdO does not backup any configuration that is in an incomplete or error state.

To backup and restore an OSPdO deployment, you create an **OpenStackBackupRequest** CR to initiate the creation or restoration of a backup. Your **OpenStackBackupRequest** CR creates the **OpenStackBackup** CR that stores the backup of the custom resources (CRs), the **ConfigMap** and the **Secret** configurations for the specified namespace.

13.1. BACKING UP DIRECTOR OPERATOR

To create a backup you must create an **OpenStackBackupRequest** custom resource (CR) for the namespace. The **OpenStackBackup** CR is created when the **OpenStackBackupRequest** object is created in **save** mode.

Procedure

1. Create a file named **openstack_backup.yaml** on your workstation.
2. Add the following configuration to your **openstack_backup.yaml** file to create the **OpenStackBackupRequest** custom resource (CR):

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackBackupRequest
metadata:
  name: openstackbackupsave
  namespace: openstack
spec:
  mode: save (1)
  additionalConfigMaps: [] (2)
  additionalSecrets: [] (3)
```

(1) Set the **mode** to **save** to request creation of an **OpenStackBackup** CR.

(2) Optional: Include any **ConfigMap** resources that you created manually.

(3) Optional: Include any **Secret** resources that you created manually.

NOTE

OSPdO attempts to include all **ConfigMap** and **Secret** objects associated with the OSPdO CRs in the namespace, such as **OpenStackControlPlane** and **OpenStackBaremetalSet**. You do not need to include those in the additional lists.

3. Save the **openstack_backup.yaml** file.

4. Create the **OpenStackBackupRequest** CR:

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

5. Monitor the creation status of the **OpenStackBackupRequest** CR:

```
$ oc get openstackbackuprequest openstackbackupsave -n openstack
```

- The **Quiescing** state indicates that OSPdO is waiting for the CRs to reach their finished state. The number of CRs can affect how long it takes to finish creating the backup.

```
NAME                   OPERATION   SOURCE   STATUS      COMPLETION TIMESTAMP
openstackbackupsave    save                 Quiescing
```

  If the status remains in the **Quiescing** state for longer than expected, you can investigate the OSPdO 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.

- The **Saved** state indicates that the **OpenStackBackup** CR is created.

```
NAME                   OPERATION   SOURCE   STATUS   COMPLETION TIMESTAMP
openstackbackupsave    save                 Saved    2022-01-11T19:12:58Z
```

- The **Error** state indicates the backup has failed to create. Review the request contents to find the error:

```
$ oc get openstackbackuprequest openstackbackupsave -o yaml -n openstack
```

6. View the **OpenStackBackup** resource to confirm it exists:

```
$ oc get openstackbackup -n openstack
NAME                              AGE
openstackbackupsave-1641928378    6m7s
```

13.2. RESTORING DIRECTOR OPERATOR FROM A BACKUP

When you request to restore a backup, Red Hat OpenStack Platform (RHOSP) director Operator (OSPdO) takes the contents of the specified **OpenStackBackup** resource and attempts to apply them to all existing custom resources (CRs), **ConfigMap** and **Secret** resources present within the namespace. OSPdO overwrites any existing resources in the namespace, and creates new resources for those not found within the namespace.

Procedure

1. List the available backups:

```
$ oc get osbackup
```

2. 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.

3. Create a file named **openstack_restore.yaml** on your workstation.

4. Add the following configuration to your **openstack_restore.yaml** file to create the **OpenStackBackupRequest** custom resource (CR):

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

   - Replace **<mode>** with one of the following options:
     - **restore**: Requests a restore from an existing **OpenStackBackup**.
     - **cleanRestore**: Completely wipes the existing OSPdO resources within the namespace before restoring and creating new resources from the existing **OpenStackBackup**.
   - Replace **<restore_source>** with the ID of the **OpenStackBackup** to restore, for example, **openstackbackupsave-1641928378**.

5. Save the **openstack_restore.yaml** file.

6. Create the **OpenStackBackupRequest** CR:

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

7. Monitor the creation status of the **OpenStackBackupRequest** CR:

```
$ oc get openstackbackuprequest openstackbackuprestore -n openstack
```

   - The **Loading** state indicates that all resources from the **OpenStackBackup** are being applied against the cluster.

```
NAME                    OPERATION  SOURCE                       STATUS     COMPLETION TIMESTAMP
openstackbackuprestore  restore    openstackbackupsave-1641928378  Loading
```

   - The **Reconciling** state indicates that all resources are loaded and OSPdO has begun reconciling to attempt to provision all resources.

```
NAME                    OPERATION  SOURCE                       STATUS      COMPLETION TIMESTAMP
openstackbackuprestore  restore    openstackbackupsave-1641928378  Reconciling
```

-

- The **Restored** state indicates that the **OpenStackBackup** CR has been restored.

```
NAME                    OPERATION  SOURCE                          STATUS     COMPLETION TIMESTAMP
openstackbackuprestore  restore    openstackbackupsave-1641928378  Restored   2022-01-12T13:48:57Z
```

- The **Error** state indicates the restoration has failed. Review the request contents to find the error:

```
$ oc get openstackbackuprequest openstackbackuprestore -o yaml -n openstack
```

CHAPTER 14. CHANGE RESOURCES ON VIRTUAL MACHINES USING DIRECTOR OPERATOR

To change the CPU, RAM, and disk resources of an **OpenStackVMSet** custom resource (CR), use the **OpenStackControlPlane** CRD.

14.1. CHANGE THE CPU OR RAM OF AN OPENSTACKVMSET CR

You can use the **OpenStackControlPlane** CRD to change the CPU or RAM of an **OpenStackVMSet** custom resource (CR).

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.

14.2. ADD ADDITIONAL DISKS TO AN OPENSTACKVMSET CR

You can use the **OpenStackControlPlane** CRD 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 15. 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.

15.1. PREREQUISITES

- An operational Red Hat Openshift Container Platform (RHOCP) cluster, version 4.12 or later. The cluster must contain a **provisioning** network, and the following Operators:
  - A **baremetal** cluster Operator. The **baremetal** cluster Operator must be enabled. For more information on **baremetal** cluster Operators, see Bare-metal cluster Operators.
  - OpenShift Virtualization Operator. For more information on installing the OpenShift Virtualization Operator, see Installing OpenShift Virtualization using the web console .
  - SR-IOV Network Operator.
- 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.
- The **oc** command line tool is installed on your workstation.
- You have access to a local git repository to store deployment artifacts.
- You have installed the **podman** and **skopeo** command line tools on your workstation.

15.2. 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 .

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 apply -f osp-director-operator.yaml
```

7. Copy the required overcloud images to the repository:

```
$ for i in $(podman search --limit 1000 "registry.redhat.io/rhosp-rhel9/openstack" --format="{{
.Name }}" | awk '{print $1 ":" "17.1.0"}' | 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 Installing and preparing director Operator.

Verification

1. Confirm that you have successfully installed 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