Red Hat OpenStack Platform 18.0-dev-preview

Deploying Red Hat OpenStack Platform 18.0 Development Preview 1 on Red Hat OpenShift Container Platform

Deploying a Red Hat OpenStack Platform environment on a Red Hat OpenShift Container Platform cluster

Last Updated: 2023-10-10

Red Hat OpenStack Platform 18.0-dev-preview Deploying Red Hat OpenStack Platform 18.0 Development Preview 1 on Red Hat OpenShift Container Platform

Deploying a Red Hat OpenStack Platform environment on a Red Hat OpenShift Container Platform cluster

OpenStack Team
rhos-docs@redhat.com

Legal Notice

Abstract

Learn how to install the Red Hat OpenStack Platform (RHOSP) control plane on a Red Hat OpenShift Container Platform (RHOCP) cluster, and use the OpenStack Operator to deploy a RHOSP data plane.

Table of Contents

DEVELOPER PREVIEW

This is a *Developer Preview* release. A Developer Preview release contains a preview of the functional set of features that will be included in the next version release of the software. Developer Preview versions are published before all features have been implemented and tested, therefore some features may be absent, incomplete, or not work as expected. Red Hat encourages customers to use the Developer Preview release to provide feedback.

Developer preview builds have the following constraints and limitations:

- Production use is not permitted.
- Installation and use is not eligible for Red Hat production support.
- Upgrades to, from, or between Developer Preview versions are not supported.

For more information about Developer Preview support scope, see Developer Preview Support Scope - Red Hat Customer Portal.

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.

CHAPTER 1. PLANNING YOUR RED HAT OPENSTACK PLATFORM DEPLOYMENT

To deploy and operate your Red Hat OpenStack Platform (RHOSP) environment, you use tools and container infrastructure provided by the Red Hat OpenShift Container Platform (RHOCP).

RHOCP uses a modular system of Operators to extend the functions of your RHOCP cluster. RHOSP OpenStack Operator (**openstack-operator**) installs and runs a RHOSP control plane within RHOCP. The RHOSP DataPlane Operator (**dataplane-operator**) automates the deployment of a RHOSP data plane. The data plane is the collection of nodes that are used for hosting RHOSP workloads. The DataPlane Operator prepares the nodes with the operating system configuration required for hosting the RHOSP services and workloads.

The OpenStack and DataPlane Operators manage a set of Custom Resource Definitions (CRDs) that deploy and manage the infrastructure and configuration of the RHOSP control plane and data plane nodes. The basic architecture of a RHOSP on RHOCP environment includes the following features:

Container-native application delivery

RHOSP is delivered by using a container-native approach that spans the RHOCP and RHEL platforms to deliver a container-native RHOSP deployment.

RHOCP hosted services

RHOCP hosts infrastructure services and RHOSP controller services, by leveraging RHOCP Operators to provide lifecycle management.

Ansible-managed RHEL-hosted services

RHOSP workloads run on RHEL worker nodes that are managed by a dedicated DataPlane Operator. The DataPlane Operator runs Ansible jobs to configure the RHEL data plane nodes, such as the Compute nodes. RHOCP manages provisioning, DNS, and configuration management.

Installer provisioned infrastructure (IPI)

The RHOSP installer enables installer-provisioned infrastructure that leverages RHOSP bare-metal machine management to provision the Compute nodes for the RHOSP cloud.

User provisioned infrastructure (UPI)

If you have your own machine ingest and provisioning workflow, you can use the RHOSP pre-provisioned model to add your pre-provisioned hardware into your RHOSP environment, while leveraging the benefits of a container-native workflow.

Hosted RHOSP client

RHOSP provides a host **openstackclient** pod that is preconfigured with administrator access to the deployed RHOSP environment.

To create a RHOSP cloud with a RHOCP hosted control plane, you must complete the following tasks:

1. Install OpenStack Operator (**openstack-operator**) on an operational RHOCP cluster.
2. Prepare the RHOCP cluster for your RHOSP deployment.
3. Create and configure the control plane network.
4. Create and configure the data plane networks.
5. Create and configure the control plane.
6. Create and configure the data plane nodes.

7. Optional: Configure a storage solution for the RHOSP deployment.

1.1. RHOSP OPERATORS

The OpenStack Operator (**openstack-operator**) installs all the required Red Hat OpenStack Platform (RHOSP) service Operators, and it is the interface that you use to manage those Operators.

The OpenStack Operator installs and manages the following Operators:

- **cinder-operator**
- **dataplane-operator**
- **glance-operator**
- **heat-operator**
- **horizon-operator**
- **infra-operator**
- **ironic-operator**
- **keystone-operator**
- **manila-operator**
- **mariadb-operator**
- **neutron-operator**
- **nova-operator**
- **openstack-ansibleee-operator**
- **openstack-baremetal-operator**
- **ovn-operator**
- **placement-operator**
- **rabbitmq-cluster-operator**
- **swift-operator**
- **telemetry-operator**

1.2. AVAILABLE RED HAT OPENSTACK PLATFORM SERVICES

The Dev Preview of Red Hat OpenStack Platform (RHOSP) 18.0 installs and supports the following services:

- Compute service (nova)
- Networking service (neutron)
- Block Storage service (cinder)

- Image service (glance)
- Shared File Systems service (manila)
- Object Storage service (swift)
- Identity service (keystone)
- Dashboard service (horizon)
- Telemetry service (ceilometer)
- Placement service (placement)
- Orchestration service (heat)
- Bare Metal Provisioning service (ironic)
- OVN
- RabbitMQ
- Galera
- MariaDB
- Memcached

1.3. AVAILABLE STORAGE AND NETWORKING FEATURES

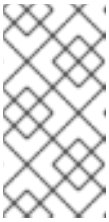
The following features are expected to work in the Dev Preview of Red Hat OpenStack Platform (RHOSP) 18.0:

- Red Hat Ceph Storage integration:
  - Ceph Block Device (RBD) with the Block Storage service (cinder), the Image service (glance), and the Compute service (nova)
  - Ceph File System (CephFS) with the Shared File Systems service (manila)
  - Hyper Converged Infrastructure (HCI): Compute nodes deployed on an existing Red Hat Ceph Storage cluster
  - HCI node scaling
- Networking service:
  - ML2/OVN with kernel OVS
- Transport protocols: Ceph/RBD, NFS, FC, and iSCSI work with the appropriate configuration or **MachineConfig**.
- Multipathing for SCSI based protocols is available on Controller nodes with the appropriate **MachineConfig**.

1.4. LIMITATIONS WITH THE DEV PREVIEW DEPLOYMENT

Only a limited set of functionalities exist and are tested for the Dev Preview. The following is a list of deployment or configuration options that are not available.

- Security:
  - TLS
  - FIPS
  - Key Management service (barbican)
- DCN/Edge
- Upgrades from this release to the next dev preview release, or the final GA release
- Compute service (nova):
  - Custom policy
  - Ironic driver
  - Declaring custom traits and resource classes in a **provider.yaml** file
  - PCI passthrough, SRIOV, VGPU, and hardware offloaded OVS



NOTE

While it is possible to use the **CustomServiceConfig** field to configure these features in the Dev Preview, hardware passthrough is planned for a later release and is currently untested. Networking service (neutron) SRIOV ports require Networking service support, which is not present in this release.

  - Cold and live migration are not enabled.
- Networking service (neutron):
  - OVS with DPDK
  - OVS hardware offload
  - SR-IOV
  - Bare metal IPv6 provisioning
  - IPv6 host deployment
  - Spine-leaf topology with DHCP relays
- Red Hat Ceph Storage integration:
  - Ceph Object Gateway (RGW)
  - Ceph File System (CephFS) through Ganesha NFS with the Shared File Systems service (manila)
- Block Storage service (cinder):
  - Back ends:

        - LVM
        - Third party vendor drivers have not been tested, but they should work if they use one of the supported transport protocols below. No container images exist for back ends with vendor dependencies, though manually built images can be used to add the required dependencies. Drivers with vendor kernel modules are unlikely to work right now.
    - Transport protocols other than Ceph/RBD, NFS, FC, and iSCSI might not work.
    - Multipathing for SCSI based protocols is not available on Compute nodes.
    - Volume encryption using the Key Manager service (barbican).
- Shared File System service (manila):
    - Only the CephFS back end driver has been extensively tested.
    - The CephFS back end driver cannot be used to provide NFS shares.
    - Hard multi-tenancy mode (**driver_handles_share_servers=True**).
    - Third party vendor drivers have been minimally tested. Vendor drivers that are shipped in Red Hat provided "manila-share" service container image with no external software dependencies will work, for example: the NetApp ONTAP **driver_handles_share_servers=False** driver.
    - Third party vendor drivers that depend on custom vendor provided container images are not available.
    - Configuring multiple back end drivers with a single "manila-share" service. You can instead configure multiple "manila-share" services, each with a single storage back end driver.
- Image Service (glance):
    - Using the Block Storage service (cinder) as an Image service (glance) back end only works with NFS as a storage transport protocol.
    - Using the Object Storage service (swift) as an Image service (glance) back end to store images.
    - Image import plugins such as image conversion and compression.
    - Image caching.
    - Image signing with the Key Manager service (barbican).

1.5. CUSTOM RESOURCE DEFINITIONS (CRDS)

The OpenStack and DataPlane Operators include a set of custom resource definitions (CRDs) that you can use to create and manage RHOSP resources.

- Use the following command to view a complete list of the RHOSP CRDs:
  $ oc get crd | grep "^openstack"
- Use the following command to view the definition for a specific CRD:

```
$ oc describe crd openstackcontrolplane
```

```
Name:         openstackcontrolplane.openstack.org
Namespace:
Labels:       operators.coreos.com/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
...
```

- Use the following command to view descriptions of the fields you can use to configure a specific CRD:

```
$ oc explain openstackcontrolplane.spec
KIND:     OpenStackControlPlane
VERSION:  core.openstack.org/v1beta1

RESOURCE: spec <Object>

DESCRIPTION:
    <empty>

FIELDS:
  ceilometer   <Object>
  cinder       <Object>
  dns  <Object>
  extraMounts  <[]Object>
...
```

Additional resources

- Managing resources from custom resource definitions

1.5.1. CRD naming conventions

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

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

```
apiVersion: core.openstack.org/v1beta1
kind: OpenStackControlPlane
...
```

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

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

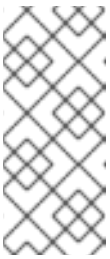
```
$ oc describe openstackcontrolplane/compute
```

1.6. PLANNING THE RED HAT OPENSTACK PLATFORM DEPLOYMENT

You must plan your Red Hat OpenStack Platform (RHOSP) deployment to determine the system requirements for your environment.

System requirements

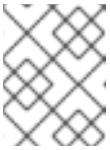
- Pre-provisioned 3-node Red Hat OpenShift Container Platform (RHOCP) compact cluster. Each node in the compact cluster must have the following resources:
  - 32 GB RAM
  - 8+ CPUs
  - 250 GB storage



NOTE

The images, volumes and root disks for the virtual machine instances running on the deployed RHOSP environment are hosted on dedicated external storage nodes. However, the RHOSP service logs, databases, and metadata are stored in a RHOCP Persistent Volume Claim (PVC). Starting with 150 GB should be sufficient for testing.

  - 2 physical NICs



NOTE

In a 6-node cluster with 3 controllers and 3 workers, only the worker nodes require 2 physical NICs.

- A set of data plane nodes. You can use pre-provisioned nodes or unprovisioned bare-metal nodes. Requirements for pre-provisioned nodes:
  - Pre-provisioned nodes must be RHEL 9.2.
  - Pre-provisioned nodes must be configured for SSH access with the SSH key created in Creating the SSH key secret . The SSH user must either be root or have unrestricted and password-less sudo enabled.
  - Pre-provisioned nodes must have a routable IP address on the control plane network to enable Ansible access through SSH.
- A dedicated NIC on RHOCP worker nodes for RHOSP isolated networks.
- Port switches with VLANs for the required isolated networks. For information on the required isolated networks, see Default Red Hat OpenStack Platform networks .

Supported topologies

- Storage:
  - The Image service (glance) with external Red Hat Ceph Storage or the Block Storage service (cinder) with NFS as the back end.
  - The Compute service (nova) with local file storage or external Red Hat Ceph Storage as the back end.
  - The Block Storage service (cinder) with external Red Hat Ceph Storage, third party FC, iSCSI or NFS as the back end.

- The Shared File Systems service (manila) with external CephFS or external NetApp ONTAP storage as the back end.
- The Object Storage service (swift) with Red Hat OpenShift Container Platform store using persistent volumes as the back end.

CHAPTER 2. INSTALLING AND PREPARING THE OPERATORS

You install the Red Hat OpenStack Platform (RHOSP) control plane on an operational Red Hat Openshift Container Platform (RHOCP) cluster. You perform the control plane installation tasks and all data plane creation tasks on a workstation that has access to the RHOCP cluster.

NOTE

Do not use the **root** user to interact with your RHOSP deployment. You must use the dedicated user, **stack**, with passwordless sudo rights and only ssh-key login enabled.

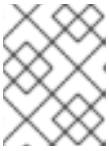
2.1. PREREQUISITES

- An operational Red Hat Openshift Container Platform (RHOCP) cluster, version 4.12 or later.
- The RHOCP environment supports Multus CNI.
- The **oc** command line tool is installed on your workstation.
- The **podman** command line tool is installed on your workstation.
- A private Red Hat Quay Container Registry account, https://quay.io/.
- Access to a private repository in your registry. The RHOSP 18.0 Development Preview code cannot be located on a public repository.
- You are logged in as a user with **cluster-admin** privileges.
- You have installed the Kubernetes NMState Operator, and started the Operator by creating an **nmstate** instance. For more information, see Installing the Kubernetes NMState Operator in the RHOCP *Networking* guide.
- You have installed the MetalLB Operator, and started the Operator by creating an **metallb** instance. For more information, see Installing the MetalLB Operator in the RHOCP *Networking* guide.
- You have configured the RHOCP storage backend and storage class. For more information, see Storage and Post-installation storage configuration.
- For Installer provisioned infrastructure (IPI), you must prepare an operating system image for use with bare-metal provisioning. You can use the following image as the bare-metal image: https://catalog.redhat.com/software/containers/rhel9/rhel-guest-image/6197bdceb4dcabca7fe351d5?container-tabs=overview

2.2. INSTALLING THE RED HAT OPENSTACK PLATFORM SERVICE OPERATORS

To install the OpenStack Operator (**openstack-operator**), you must create the following projects:

- **openstack-operators**: Create this project for the Red Hat OpenStack Platform (RHOSP) service Operators.
- **openstack**: Create this project for the deployed RHOSP services.



NOTE

Each project is a namespace with additional functionality to support multi-tenancy.

You must also create the following custom resources (CRs) within the project:

- A **CatalogSource**, which identifies the index image to use for the RHOSP catalog. For more information on **CatalogSource**, see CatalogSource in the *Operator Lifecycle Manager* documentation.
- An **OperatorGroup**, which defines the Operator group for RHOSP and restricts RHOSP to a target namespace. For more information on **OperatorGroup**, see OperatorGroup in the *Operator Lifecycle Manager* documentation.
- A **Subscription**, which tracks changes in the RHOSP catalog, and defines which version of the Operator is installed and from which **CatalogSource** to install it. For more information on **Subscription**, see Subscription in the *Operator Lifecycle Manager* documentation.

NOTE

Installing the OpenStack Operator also creates an **OpenStackClient** pod that you can access through a remote shell (**rsh**) to run RHOSP commands.

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

Procedure

1. Create the **openstack-operators** project for the RHOSP operators:

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

2. Create the **openstack** project for the deployed RHOSP environment:

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

3. To prevent issues with image signing, enter the following commands:

```
$ curl https://www.redhat.com/security/data/f21541eb.txt -o /etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-beta

$ podman image trust set -f /etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-beta registry.redhat.io/rhosp-dev-preview

$ cat /etc/containers/policy.json
{
    "default": [
        {
            "type": "insecureAcceptAnything"
        }
    ],
    "transports": {
        "docker": {
            "registry.access.redhat.com": [
```

```
            {
                "type": "signedBy",
                "keyType": "GPGKeys",
                "keyPath": "/etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release"
            }
        ],
        "registry.redhat.io": [
            {
                "type": "signedBy",
                "keyType": "GPGKeys",
                "keyPath": "/etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release"
            }
        ],
        "registry.redhat.io/rhosp-dev-preview": [
            {
                "type": "signedBy",
                "keyType": "GPGKeys",
                "keyPath": "/etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-beta"
            }
        ]
    },
    "docker-daemon": {
        "": [
            {
                "type": "insecureAcceptAnything"
            }
        ]
    }
  }
}
```

4. Download the Operator Package Manager (**opm**) tool from https://console.redhat.com/openshift/downloads.

5. Use the **opm** tool to create an index image:

```
$ opm index add -u podman --pull-tool podman --tag <your_registry>:<port>/rhosp-dev-preview/openstack-operator-index:0.1.0 \
 --bundle "registry.redhat.io/rhosp-dev-preview/openstack-operator-bundle:0.1.0,registry.redhat.io/rhosp-dev-preview/swift-operator-bundle:0.1.0,registry.redhat.io/rhosp-dev-preview/glance-operator-bundle:0.1.0,registry.redhat.io/rhosp-dev-preview/infra-operator-bundle:0.1.0,registry.redhat.io/rhosp-dev-preview/ironic-operator-bundle:0.1.0,registry.redhat.io/rhosp-dev-preview/keystone-operator-bundle:0.1.0,registry.redhat.io/rhosp-dev-preview/ovn-operator-bundle:0.1.0,registry.redhat.io/rhosp-dev-preview/placement-operator-bundle:0.1.0,registry.redhat.io/rhosp-dev-preview/telemetry-operator-bundle:0.1.0,registry.redhat.io/rhosp-dev-preview/heat-operator-bundle:0.1.0,registry.redhat.io/rhosp-dev-preview/cinder-operator-bundle:0.1.0,registry.redhat.io/rhosp-dev-preview/manila-operator-bundle:0.1.0,registry.redhat.io/rhosp-dev-preview/neutron-operator-bundle:0.1.0,registry.redhat.io/rhosp-dev-preview/nova-operator-bundle:0.1.0,registry.redhat.io/rhosp-dev-preview/openstack-ansibleee-operator-bundle:0.1.0,registry.redhat.io/rhosp-dev-preview/mariadb-operator-bundle:0.1.0,registry.redhat.io/rhosp-dev-preview/openstack-baremetal-operator-bundle:0.1.0,registry.redhat.io/rhosp-dev-preview/rabbitmq-cluster-operator-
```

```
bundle:0.1.0,registry.redhat.io/rhosp-dev-preview/rabbitmq-cluster-operator-
bundle:0.1.0,registry.redhat.io/rhosp-dev-preview/dataplane-operator-
bundle:0.1.0,registry.redhat.io/rhosp-dev-preview/horizon-operator-bundle:0.1.0" --mode
semver
```

6. Push the index image to your private registry:

```
$ podman push <your_registry>:<port>/rhosp-dev-preview/openstack-operator-index:0.1.0
```

7. Create an environment file to configure the **CatalogSource**, **OperatorGroup**, and **Subscription** CRs required to install the OpenStack Operator, for example, **openstack-operator.yaml**.

8. To configure the **CatalogSource** CR, add the following configuration to **openstack-operator.yaml**:

```
apiVersion: operators.coreos.com/v1alpha1
kind: CatalogSource
metadata:
  name: openstack-operator-index
  namespace: openstack-operators
spec:
  sourceType: grpc
  image: quay.io/<account>/openstack-operator-index:latest
```

   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.

9. To configure the **OperatorGroup** CR, add the following configuration to **openstack-operator.yaml**:

```
---
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: openstack
  namespace: openstack-operators
```

10. To configure the **Subscription** CR, add the following configuration to **openstack-operator.yaml**:

```
---
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: openstack-operator
  namespace: openstack-operators
spec:
  name: openstack-operator
  channel: alpha
  source: openstack-operator-index
  sourceNamespace: openstack-operators
```

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

```
$ oc apply -f openstack-operator.yaml
```

12. Confirm that you have installed the Openstack Operator, **openstack-operator.openstack-operators**:

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

13. Confirm that the Openstack Operator is deployed by reviewing the pods in the **openstack-operators** namespace:

```
$ oc get pods -n openstack-operators
```

The Openstack Operator is deployed when all the pods are either completed or running.

CHAPTER 3. PREPARING RED HAT OPENSHIFT CONTAINER PLATFORM FOR RED HAT OPENSTACK PLATFORM

You install Red Hat OpenStack Platform (RHOSP) on an operational Red Hat Openshift Container Platform (RHOCP) cluster, version 4.12 or later. You must configure the RHOCP worker nodes and the RHOCP networks on your RHOCP cluster to prepare for installing and deploying your RHOSP environment.

3.1. CONFIGURING RED HAT OPENSHIFT CONTAINER PLATFORM NODES FOR A RED HAT OPENSTACK PLATFORM DEPLOYMENT

Red Hat OpenStack Platform (RHOSP) services run on Red Hat OpenShift Container Platform (RHOCP) worker nodes. By default, the OpenStack Operator deploys RHOSP services on any worker node. You can use node labels in your **OpenStackControlPlane** custom resource (CR) to specify which RHOCP nodes host the RHOSP services, based on the service requirements. You can either create your own labels for the RHOCP nodes or use the existing labels, and then specify those labels in the **OpenStackControlPlane** CR by using the **nodeSelector** field.

For example, the Block Storage service (cinder) has different requirements for each of its services:

- The **cinder-scheduler** service is a very light service with low memory, disk, network, and CPU usage.
- The **cinder-api** service has high network usage due to resource listing requests.
- The **cinder-volume** service has high disk and network usage because many of its operations are in the data path, such as offline volume migration, and create a volume from an image.
- The **cinder-backup** service has high memory, network, and CPU requirements.

Given these requirements, and any hardware restrictions in your environment, you can pin some services to a set of infrastructure nodes rather than running services over all of your RHOCP worker nodes, which could impact other workloads.

Additional resources

- Placing pods on specific nodes using node selectors
- Post-installation machine configuration tasks
- Node Feature Discovery Operator

3.2. PROVIDING SECURE ACCESS TO THE RED HAT OPENSTACK PLATFORM SERVICES

You must create a **Secret** custom resource (CR) to provide secure access to the Red Hat OpenStack Platform (RHOSP) service pods.

Procedure

1. Create a **Secret** CR file on your workstation, for example, **openstack-service-secret.yaml**.
2. Add the following initial configuration to **openstack-service-secret.yaml**:
   -

```
apiVersion: v1
data:
  AdminPassword: <base64_password>
  CeilometerPassword: <base64_password>
  CinderDatabasePassword: <base64_password>
  CinderPassword: <base64_password>
  DatabasePassword: <base64_password>
  DbRootPassword: <base64_password>
  DesignateDatabasePassword: <base64_password>
  DesignatePassword: <base64_password>
  GlanceDatabasePassword: <base64_password>
  GlancePassword: <base64_password>
  HeatAuthEncryptionKey: <base64_password_heat>
  HeatDatabasePassword: <base64_password>
  HeatPassword: <base64_password>
  IronicDatabasePassword: <base64_password>
  IronicInspectorDatabasePassword: <base64_password>
  IronicInspectorPassword: <base64_password>
  IronicPassword: <base64_password>
  KeystoneDatabasePassword: <base64_password>
  ManilaDatabasePassword: <base64_password>
  ManilaPassword: <base64_password>
  MetadataSecret: <base64_password>
  NeutronDatabasePassword: <base64_password>
  NeutronPassword: <base64_password>
  NovaAPIDatabasePassword: <base64_password>
  NovaAPIMessageBusPassword: <base64_password>
  NovaCell0DatabasePassword: <base64_password>
  NovaCell0MessageBusPassword: <base64_password>
  NovaCell1DatabasePassword: <base64_password>
  NovaCell1MessageBusPassword: <base64_password>
  NovaPassword: <base64_password>
  OctaviaDatabasePassword: <base64_password>
  OctaviaPassword: <base64_password>
  PlacementDatabasePassword: <base64_password>
  PlacementPassword: <base64_password>
  SwiftPassword: <base64_password>
kind: Secret
metadata:
  name: osp-secret
  namespace: openstack
type: Opaque
```

- Replace **<base64_password>** with a base64 encoded string. Use the following command to generate a base64 encoded password:

  ```
  $ echo -n <password> | base64
  ```

- Replace **<base64_password_heat>** with a base64 encoded password for Heat authentication.

3. Create the **Secret** in the cluster:

   ```
   $ oc create -f openstack-service-secret.yaml
   ```

4. Verify that the **Secret** is created:

```
$ oc describe secret osp-secret -n openstack
```

3.3. DEFAULT RED HAT OPENSTACK PLATFORM NETWORKS

The following physical data center networks are typically implemented on top of the control plane:

- Internal API network: Used for internal communication between Red Hat OpenStack Platform (RHOSP) components.
- Tenant network: Used for data communication between virtual machine instances within the cloud deployment.
- External network: Used to provide virtual machine instances with Internet access.
- Storage network: Used for block storage, RBD, NFS, FC, and iSCSI.
- Storage Management network: Used by the storage. For example, in a hyperconverged infrastructure (HCI) environment, Red Hat Ceph Storage uses the Storage Management network as its **cluster_network** to replicate data.

The following table details the default networks used in a RHOSP deployment. If required, you can update the networks for your environment.

Table 3.1. Default RHOSP networks

Network name	VLAN	CIDR	NetConfig **allocationRange**	MetalLB **IPAddressPool** range	nad **ipam** range	OCP worker **nncp** range
ctlplane	n/a	192.168.122.0/24	192.168.122.100 - 192.168.122.250	192.168.122.80 - 192.168.122.90	192.168.122.30 - 192.168.122.70	192.168.122.10 - 192.168.122.20
external	n/a	10.0.0.0/24	10.0.0.100 - 10.0.0.250	n/a	n/a	
internalapi	20	172.17.0.0/24	172.17.0.100 - 172.17.0.250	172.17.0.80 - 172.17.0.90	172.17.0.30 - 172.17.0.70	172.17.0.10 - 172.17.0.20
storage	21	172.18.0.0/24	172.18.0.100 - 172.18.0.250	172.18.0.80 - 172.18.0.90	172.18.0.30 - 172.18.0.70	172.18.0.10 - 172.18.0.20
tenant	22	172.19.0.0/24	172.19.0.100 - 172.19.0.250	172.19.0.80 - 172.19.0.90	172.19.0.30 - 172.19.0.70	172.19.0.10 - 172.19.0.20
storageMgmt	23	172.20.0.0/24	172.20.0.100 - 172.20.0.250	172.20.0.80 - 172.20.0.90	172.20.0.30 - 172.20.0.70	172.20.0.10 - 172.20.0.20

3.4. PREPARING RHOCP FOR RHOSP NETWORK ISOLATION

The Red Hat OpenStack Platform (RHOSP) services run as a Red Hat OpenShift Container Platform (RHOCP) workload. You use the NMState Operator to connect the worker nodes to the required isolated networks. You create a **NetworkAttachmentDefinition** (**nad**) custom resource (CR) for each isolated network to attach service pods to the isolated networks, where needed. You use the MetalLB Operator to expose internal service endpoints on the isolated networks. By default, the public service endpoints are exposed as RHOCP routes.

You must also create a **L2Advertisement** resource to define how the VIPs are announced, and an **IpAddressPool** resource to configure which IPs can be used as VIPs. In layer 2 mode, one node assumes the responsibility of advertising a service to the local network.

Procedure

1. Create a **NodeNetworkConfigurationPolicy** (**nncp**) CR file on your workstation, for example, **openstack-nncp.yaml**.

2. Retrieve the names of the worker nodes in the RHOCP cluster:

```
$ oc get nodes -l node-role.kubernetes.io/worker -o jsonpath="{.items[*].metadata.name}"
```

3. Discover the network configuration:

```
$ oc get nns/<worker_node> -o yaml | more
```

   - Replace **<worker_node>** with the name of a worker node retrieved in step 2, for example, **worker-1**. Repeat this step for each worker node.

4. In the **nncp** CR file, configure the interfaces for each isolated network on each worker node in the RHOCP cluster. For information on the default physical data center networks that must be configured with network isolation, see Default Red Hat OpenStack Platform networks .
   In the following example, the **nncp** CR configures the **enp6s0** interface for worker node 1, **osp-enp6s0-worker-1**, to use VLANs for network isolation:

```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: osp-enp6s0-worker-1
spec:
  desiredState:
    interfaces:
    - description: internalapi vlan interface
      ipv4:
        address:
        - ip: 172.17.0.10
          prefix-length: 24
        enabled: true
        dhcp: false
      ipv6:
        enabled: false
      name: enp6s0.20
      state: up
      type: vlan
      vlan:
```

```
      base-iface: enp6s0
      id: 20
    - description: storage vlan interface
      ipv4:
        address:
        - ip: 172.18.0.10
          prefix-length: 24
        enabled: true
        dhcp: false
      ipv6:
        enabled: false
      name: enp6s0.21
      state: up
      type: vlan
      vlan:
        base-iface: enp6s0
        id: 21
    - description: tenant vlan interface
      ipv4:
        address:
        - ip: 172.19.0.10
          prefix-length: 24
        enabled: true
        dhcp: false
      ipv6:
        enabled: false
      name: enp6s0.22
      state: up
      type: vlan
      vlan:
        base-iface: enp6s0
        id: 22
    - description: Configuring enp6s0
      ipv4:
        address:
        - ip: 192.168.122.10
          prefix-length: 24
        enabled: true
        dhcp: false
      ipv6:
        enabled: false
      mtu: 1500
      name: enp6s0
      state: up
      type: ethernet
  nodeSelector:
    kubernetes.io/hostname: worker-10
    node-role.kubernetes.io/worker: ""
```

5. Create the **nncp** CR in the cluster:

```
$ oc apply -f openstack-nncp.yaml
```

6. Verify that the **nncp** CR is created:

```
$ oc get nncp -w
NAME                    STATUS       REASON
osp-enp6s0-worker-1   Progressing   ConfigurationProgressing
osp-enp6s0-worker-1   Progressing   ConfigurationProgressing
osp-enp6s0-worker-1   Available     SuccessfullyConfigured
```

7. Create a **NetworkAttachmentDefinition** (**nad**) CR file on your workstation, for example, **openstack-nad.yaml**.

8. In the **nad** CR file, configure a **nad** resource for each isolated network to attach a service deployment pod to the network. The following examples create a **nad** resource for the **internalapi**, **storage**, **ctlplane**, and **tenant** networks of type **macvlan**:

```
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  name: internalapi
  namespace: openstack 1
spec:
  config: |
    {
      "cniVersion": "0.3.1",
      "name": "internalapi",
      "type": "macvlan",
      "master": "enp6s0.20", 2
      "ipam": {  3
        "type": "whereabouts",
        "range": "172.17.0.0/24",
        "range_start": "172.17.0.30", 4
        "range_end": "172.17.0.70"
      }
    }
---
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  labels:
    osp/net: ctlplane
  name: ctlplane
  namespace: openstack
spec:
  config: |
    {
      "cniVersion": "0.3.1",
      "name": "ctlplane",
      "type": "macvlan",
      "master": "enp6s0",
      "ipam": {
        "type": "whereabouts",
        "range": "192.168.50.0/24",
        "range_start": "192.168.50.30",
        "range_end": "192.168.50.70"
      }
    }
---
```

```
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  name: storage
  namespace: openstack
spec:
  config: |
    {
      "cniVersion": "0.3.1",
      "name": "storage",
      "type": "macvlan",
      "master": "enp6s0.21",
      "ipam": {
        "type": "whereabouts",
        "range": "172.18.0.0/24",
        "range_start": "172.18.0.30",
        "range_end": "172.18.0.70"
      }
    }
---
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  labels:
    osp/net: tenant
  name: tenant
  namespace: openstack
spec:
  config: |
    {
      "cniVersion": "0.3.1",
      "name": "tenant",
      "type": "macvlan",
      "master": "enp6s0.22",
      "ipam": {
        "type": "whereabouts",
        "range": "172.19.0.0/24",
        "range_start": "172.19.0.30",
        "range_end": "172.19.0.70"
      }
    }
```

1. The namespace where the services are deployed.
2. The worker node interface to use for the VLAN.
3. The **whereabouts** CNI IPAM plugin to assign IPs to the created pods from the range ``.30 - .70`.
4. The IP address pool range must not overlap with the MetalLB **IPAddressPool** range and the **NetConfig allocationRange**.

9. Create the **nad** CR in the cluster:

```
$ oc apply -f openstack-nad.yaml
```

10. Verify that the **nad** CR is created:

```
$ oc get network-attachment-definitions
```

11. Create an **IPAddressPool** CR file on your workstation, for example, **openstack-ipaddresspools.yaml**.

12. In the **IPAddressPool** CR file, configure an **IPAddressPool** resource on the isolated network to specify the IP address ranges over which MetalLB has authority:

```
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: internalapi
  namespace: metallb-system
spec:
  addresses:
    - 172.17.0.80-172.17.0.90 (1)
  autoAssign: true
  avoidBuggyIPs: false
---
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  namespace: metallb-system
  name: ctlplane
spec:
  addresses:
    - 192.168.122.80-192.168.122.90
  autoAssign: true
  avoidBuggyIPs: false
---
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  namespace: metallb-system
  name: storage
spec:
  addresses:
    - 172.18.0.80-172.18.0.90
  autoAssign: true
  avoidBuggyIPs: false
---
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  namespace: metallb-system
  name: tenant
spec:
  addresses:
    - 172.19.0.80-172.19.0.90
  autoAssign: true
  avoidBuggyIPs: false
```

1 The **IPAddressPool** range must not overlap with the **whereabouts** IPAM range and the NetConfig **allocationRange**.

For information on how to configure the other **IPAddressPool** resource parameters, see Configuring MetalLB address pools.

13. Create the **IPAddressPool** CR in the cluster:

```
$ oc apply -f openstack-ipaddresspools.yaml
```

14. Verify that the **IPAddressPool** CR is created:

```
$ oc describe -n metallb-system IPAddressPool
```

15. Create a **L2Advertisement** CR file on your workstation, for example, **openstack-l2advertisement.yaml**.

16. In the **L2Advertisement** CR file, configure a **L2Advertisement** resource to define which node advertises a service to the local network. Create one L2Advertisement resource for each network.
In the following example, the **L2Advertisement** CR specifies that the VIPs requested from the **internalapi** address pool are announced on the interface that is attached to the **internalapi** VLAN:

```
apiVersion: metallb.io/v1beta1
kind: L2Advertisement
metadata:
  name: l2advertisement
  namespace: metallb-system
spec:
  ipAddressPools:
  - internalapi
  interfaces:
  - enp6s0.20 1
---
apiVersion: metallb.io/v1beta1
kind: L2Advertisement
metadata:
  name: ctlplane
  namespace: metallb-system
spec:
  ipAddressPools:
  - ctlplane
  interfaces:
  - enp6s0
---
apiVersion: metallb.io/v1beta1
kind: L2Advertisement
metadata:
  name: storage
  namespace: metallb-system
spec:
  ipAddressPools:
  - storage
```

```
  interfaces:
  - enp6s0.21
---
apiVersion: metallb.io/v1beta1
kind: L2Advertisement
metadata:
  name: tenant
  namespace: metallb-system
spec:
  ipAddressPools:
  - tenant
  interfaces:
  - enp6s0.22
```

1 The interface that the VIPs requested from the VLAN address pool are announced on.

17. Create the **L2Advertisement** CR in the cluster:

```
$ oc apply -f openstack-l2advertisement.yaml
```

18. Verify that the **L2Advertisement** CR is created:

```
$ oc describe -n metallb-system L2Advertisement l2advertisement
```

3.5. CONFIGURING THE DATA PLANE NETWORK

To create the data plane network, you define a **NetConfig** custom resource (CR) and specify all the subnets for the data plane networks. You must define at least one control plane network for your data plane. You can also 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.

TIP

Use the following commands to view the **NetConfig** CRD definition and specification schema:

```
$ oc describe crd netconfig

$ oc explain netconfig.spec
```

Procedure

1. Create a file named **openstacknetconfig.yaml** on your workstation.
2. Add the following configuration to **openstacknetconfig.yaml** to create the **NetConfig** CR:

```
apiVersion: network.openstack.org/v1beta1
kind: NetConfig
metadata:
  name: openstacknetconfig
  namespace: openstack
```

3. In the **openstacknetconfig.yaml** file, define the topology for each data plane network. To use

the default RHOSP networks, you must define a specification for each network. For information on the default RHOSP networks, see Default Red Hat OpenStack Platform networks . The following example creates a control plane network and an isolated network for **InternalApi** traffic:

```
spec:
  networks:
  - name: CtlPlane 1
    dnsDomain: ctlplane.example.com
    subnets: 2
    - name: subnet1 3
      allocationRanges: 4
      - end: 192.168.122.120
        start: 192.168.122.100
      - end: 192.168.122.200
        start: 192.168.122.150
      cidr: 192.168.122.0/24
      gateway: 192.168.122.1
  - name: InternalApi
    dnsDomain: internalapi.example.com
    subnets:
    - name: subnet1
      allocationRanges:
      - end: 172.17.0.250
        start: 172.17.0.100
      excludeAddresses:
      - 172.17.0.10
      - 172.17.0.12
      cidr: 172.17.0.0/24
      vlan: 20 5
```

1. The name of the network, for example, **CtlPlane**.
2. The subnet specifications.
3. The name of the subnet, for example, subnet1.
4. Details of the IPv4 subnet with allocationRanges, cidr, gateway, and an optional list of routes with destination and nexthop. The NetConfig allocationRange must not overlap with the MetalLB IpAddressPool range and the IP address pool range.
5. The network VLAN. For information on the default RHOSP networks, see Default Red Hat OpenStack Platform networks.

4. Save the **openstacknetconfig.yaml** definition file.
5. Create the data plane network:

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

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

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

7. View the **NetConfig** API and child resources:

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

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
```

CHAPTER 4. CREATING THE CONTROL PLANE

The Red Hat OpenStack Platform (RHOSP) control plane contains the RHOSP services that manage the cloud. The RHOSP services run as a Red Hat OpenShift Container Platform (RHOCP) workload.

4.1. PREREQUISITES

- The RHOCP cluster is prepared for RHOSP network isolation. For more information, see Preparing RHOCP for RHOSP network isolation.
- The OpenStack Operator (**openstack-operator**) is installed. For more information, see Installing and preparing the Operators.

4.2. CREATING THE CONTROL PLANE

Define an **OpenStackControlPlane** custom resource (CR) to perform the following tasks:

- Create the control plane.
- Enable the required Red Hat OpenStack Platform (RHOSP) services.
- Configure the control plane network.
- Configure service back ends.

TIP

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

```
$ oc describe crd openstackcontrolplane
```

Procedure

1. Copy the sample network isolation **OpenStackControlPlane** CR from https://github.com/openstack-k8s-operators/openstack-operator/blob/v0.1.0/config/samples/core_v1beta1_openstackcontrolplane_network_isolation.yam and save it to a file named **openstack_control_plane.yaml** on your workstation.

NOTE

For information about how the services are configured, see Example control plane service configuration.

2. Update the **storageClass** to the storage class you created for your Red Hat OpenShift Container Platform (RHOCP) cluster storage back end:

```
apiVersion: core.openstack.org/v1beta1
kind: OpenStackControlPlane
metadata:
  name: openstack-network-isolation
```

```
spec:
  secret: osp-secret
  storageClass: your-RHOCP-storage-class
```

3. Optional: Configure each service by using the **customServiceConfig** option to pass service configuration options to the service configuration file:

```
apiVersion: core.openstack.org/v1beta1
kind: OpenStackControlPlane
spec:
  extraMounts:
    ...
  glance:
    template:
      databaseInstance: openstack
      customServiceConfig: |
        [DEFAULT]
        enabled_backends = default_backend:rbd
        [glance_store]
        default_backend = default_backend
        [default_backend]
        rbd_store_ceph_conf = /etc/ceph/ceph.conf
        store_description = "RBD backend"
        rbd_store_pool = images
        rbd_store_user = openstack
```

   For an additional example on how to configure the Image service (glance), see Configuring the control plane to use the Red Hat Ceph Storage cluster.

   For information on the available configuration options for each service, see the Configuration reference.

4. Create the control plane:

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

5. Wait until RHOCP creates the resources related to the **OpenStackControlPlane** CR. Run the following command to check the status:

```
$ oc get openstackcontrolplane -n openstack
NAME STATUS MESSAGE
openstack-network-isolation Unknown Setup started
```

   The **OpenStackControlPlane** resources are created when the status is "Setup complete".

   TIP

   Append the **-w** option to the end of the **get** command to track deployment progress.

6. Confirm that the control plane is deployed by reviewing the pods in the **openstack** namespace:

```
$ oc get pods -n openstack
```

   The control plane is deployed when all the pods are either completed or running.

NOTE

The **glance-internal-api**, **glance-external-api**, and **placement-api** pods might show a status of **CrashLoopBackoff**, which indicates that the pod is stuck in a loop of trying to start but failing. This is a known issue in **glance-operator** that will be fixed in a subsequent release. You can ignore the issue.

Verification

1. Open a remote shell connection to the **OpenStackClient** pod:

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

2. Confirm that the internal service endpoints are registered with each service:

```
$ openstack endpoint list -c 'Service Name' -c Interface -c URL --service glance
+--------------+-----------+---------------------------------------------------------------+
| Service Name | Interface | URL                                                           |
+--------------+-----------+---------------------------------------------------------------+
| glance       | internal  | http://glance-internal.openstack.svc:9292                     |
| glance       | public    | http://glance-public-openstack.apps.ostest.test.metalkube.org |
+--------------+-----------+---------------------------------------------------------------+
```

3. Exit the **OpenStackClient** pod:

```
$ exit
```

4.3. EXAMPLE CONTROL PLANE SERVICE CONFIGURATION

The following example **OpenStackControlPlane** CR explains how services are configured on the control plane.

```
apiVersion: core.openstack.org/v1beta1
kind: OpenStackControlPlane
metadata:
  name: openstack-network-isolation
spec:
  secret: osp-secret
  storageClass: local-storage (1)
  dns: (2)
    template:
      externalEndpoints:
      - ipAddressPool: ctlplane
        loadBalancerIPs:
        - 192.168.122.80
      options:
      - key: server
        values:
        - 192.168.122.1
      replicas: 1
  cinder: (3)
    template:
      databaseInstance: openstack
```

```
    secret: osp-secret
    cinderAPI:
      externalEndpoints: 4
      - endpoint: internal
        ipAddressPool: internalapi 5
        loadBalancerIPs:
        - 172.17.0.80 6
    cinderScheduler:
      replicas: 1
    cinderBackup:
      networkAttachments: 7
      - storage
      replicas: 0 # backend needs to be configured
    cinderVolumes:
      volume1:
        networkAttachments:
        - storage
        replicas: 0 # backend needs to be configured
glance:
  template:
    databaseInstance: openstack
    storageClass: ""
    storageRequest: 10G
    glanceAPIInternal:
      externalEndpoints:
      - endpoint: internal
        ipAddressPool: internalapi
        loadBalancerIPs:
        - 172.17.0.80
      networkAttachments:
      - storage
    glanceAPIExternal:
      networkAttachments:
      - storage
...
rabbitmq: 8
  templates:
    rabbitmq:
      externalEndpoint:
        loadBalancerIPs:
        - 172.17.0.85
        ipAddressPool: internalapi
        sharedIP: false
    rabbitmq-cell1:
      externalEndpoint:
        loadBalancerIPs:
        - 172.17.0.86
        ipAddressPool: internalapi
        sharedIP: false
...
```

1 The storage class that you created for your Red Hat OpenShift Container Platform (RHOCP) cluster storage back end.

2 Service-specific parameters.

3. Service-specific parameters.
4. Expose the endpoints for each service to an isolated network by using the **externalEndpoints** field.
5. Register the internal service API endpoint as a MetalLB service by using the IPAddressPool **internalapi**.
6. Request to use the IP 172.17.0.80 as the VIP. The IP is shared with other services by default.
7. List the **NetworkAttachmentDefinition** resource names to configure the networks that each service pod is directly attached to. A NIC is configured for the service for each specified network attachment.

NOTE

If you do not configure the isolated networks that each service pod is attached to, then the default pod network is used. For example, the Block Storage service (cinder) uses the storage network to connect to a storage back end; the Identity service (keystone) uses an LDAP or Active Directory (AD) network; the **ovnDBCluster** and **ovnNorthd** services use the **internalapi** network; and the **ovnController** service uses the tenant network.

8. Expose the RabbitMQ instances to an isolated network by using the **externalEndpoint** parameter.

NOTE

Multiple RabbitMQ instances cannot share the same VIP as they use the same port. If you need to expose multiple RabbitMQ instances to the same network, then you must use distinct IP addresses.

4.4. ADDITIONAL RESOURCES

- Kubernetes NMState Operator
- The Kubernetes NMState project
- Load balancing with MetalLB
- MetalLB documentation
- MetalLB in layer 2 mode
- Specify network interfaces that LB IP can be announced from
- Multiple networks
- Using the Multus CNI in OpenShift
- macvlan plugin
- whereabouts IPAM CNI plugin - Extended configuration
- About advertising for the IP address pools

- Dynamic provisioning

CHAPTER 5. CREATING THE DATA PLANE

The Red Hat OpenStack Platform (RHOSP) data plane consists of RHEL 9 nodes. Use the **OpenStackDataPlane** custom resource definition (CRD) to create the custom resources (CRs) that define the layout of the data plane.

Prerequisites

- A functional control plane, created by the OpenStack Operator. For more information, see Chapter 4: Creating the control plane.
- Pre-provisioned nodes must be configured with an SSH public key for a user with **sudo** privileges that does not require a password.

TIP

Use the following commands to view the **OpenStackDataPlane** CRD definition and specification schema:

```
$ oc describe crd openstackdataplane

$ oc explain openstackdataplane.spec
```

5.1. CREATING THE SSH KEY SECRET

After you have installed the OpenStack Operator, you must generate an SSH key to enable Ansible to manage the Red Hat Enterprise Linux (RHEL) hosts on the data plane. Create an SSH key secret for pre-provisioned nodes. Ansible executes commands with this user and key.

Procedure

1. Create the SSH key pair:
   $ ssh-keygen -f <key_file_name> -N "" -t rsa -b 4096
   - Replace **<key_file_name>** with the name to use for the key pair.
2. Create the key secret for the cluster and apply it to the cluster:

```
$ oc create secret generic <secret_name> \
--save-config \
--dry-run=client \
--from-file=authorized_keys=<key_file_name>.pub \
--from-file=ssh-privatekey=<key_file_name> \
--from-file=ssh-publickey=<key_file_name>.pub \
-o yaml | \ oc apply -f-
```

   - Replace **<secret_name>** with the name you want to use for the secret.
3. Verify that the secret is created:
   $ oc describe secret <secret_name>

5.2. CREATING AND DEPLOYING THE DATA PLANE

You use the **OpenStackDataPlane** CRD to create and deploy the data plane. The **OpenStackDataPlane** CRD consists of the following resources:

- **OpenStackDataPlaneNode**
- **OpenStackDataPlaneService**
- **OpenStackDataPlaneRole**

An **OpenStackDataPlane** CR provides a single resource where the entire data plane is defined. The following procedure creates an **OpenStackDataPlane** resource to define the data plane.

Procedure

1. Copy the sample **OpenStackDataPlane** CR from https://github.com/openstack-k8s-operators/dataplane-operator/blob/v0.1.0/config/samples/dataplane_v1beta1_openstackdataplane.yaml and save it to file named **openstack-edpm.yaml** on your workstation.
2. Optional: Add roles to the **openstack-edpm.yaml** file. You can define as many roles as necessary for your deployment. For information about how the roles are configured, see Example data plane role configuration .
3. Retrieve the following OVN metadata agent values from your control plane:

```
$ oc get secret rabbitmq-transport-url-neutron-neutron-transport -o json | jq -r
.data.transport_url | base64 -d
$ oc get ovndbcluster ovndbcluster-sb -o json | jq -r .status.dbAddress
$ oc get svc nova-metadata-internal -o json |jq -r '.status.loadBalancer.ingress[0].ip'
$ oc get secret osp-secret -o json | jq -r .data.MetadataSecret  | base64 -d
$ oc get ovndbcluster ovndbcluster-sb -o json | jq -r
'.status.networkAttachments."openstack/internalapi"[0]'
```

4. Update the following Ansible variables in your **openstack-edpm.yaml** file to the values you retrieved in step 3:

```
spec:
  roles:
    edpm-compute:
      ...
      nodeTemplate:
        ansibleVars: |
          ...
          edpm_ovn_metadata_agent_DEFAULT_transport_url: <transport_url>
          edpm_ovn_metadata_agent_metadata_agent_ovn_ovn_sb_connection:
<sb_connection>
          edpm_ovn_metadata_agent_metadata_agent_DEFAULT_nova_metadata_host:
<compute_metadata_host>

edpm_ovn_metadata_agent_metadata_agent_DEFAULT_metadata_proxy_shared_secret:
<shared_secret>
          edpm_ovn_metadata_agent_DEFAULT_bind_host: 127.0.0.1
          ...
          edpm_ovn_dbs:
          - <ovn_dbs_network>
```

- Replace **<transport_url>** with the transport URL retrieved in step 3.
- Replace **<sb_connection>** with the south bound database of the OVN cluster retrieved in step 3.
- Replace **<compute_metadata_host>** with the IP address of the node retrieved in step 3 that hosts the Compute service (nova) metadata.
- Replace **<shared_secret>** with the password retrieved in step 3 for secure access to the Red Hat OpenStack Platform (RHOSP) service pods.
- Replace **<ovn_dbs_network>** with the IP address of the **internalapi** network used by the OVN cluster retrieved in step 3.

5. Add the following configuration to prevent the Neutron deployment error "Index search returned more than one value":

```
edpm_ovn_metadata_agent_neutron_config:
  DEFAULT:
    debug: True
    log_dir: '/var/log/neutron'
  agent:
    report_interval: '300'
  oslo_concurrency:
    lock_path: '$state_path/lock'
```

6. If your nodes are pre-provisioned, you must register the operating system of the nodes to the Red Hat Customer Portal, if not already registered. You must also enable the repositories for your nodes. For more information, see Registering pre-provisioned nodes to the data plane .
7. If your nodes must be provisioned when creating the **OpenStackDataPlane** resource, you must configure the bare-metal template. For more information, see Provisioning bare-metal nodes.
8. Optional: Add nodes to the **openstack-edpm.yaml** file. For information about how nodes are configured, see Example data plane node configuration .
9. Save the **openstack-edpm.yaml** definition file.
10. Set the **ANSIBLEEE_IMAGE_URL_DEFAULT** environment variable to **registry.redhat.io/rhosp-dev-preview/ee-openstack-ansible-ee-rhel8:0.1.0** in the RHOSP **OpenStackAnsibleEE ClusterServiceVersion** (**csv**) resource:

```
$ oc edit csv openstack-ansibleee-operator.v0.0.1 -n openstack-operators

apiVersion: operators.coreos.com/v1alpha1
kind: ClusterServiceVersion
metadata:
  annotations:
    alm-examples: |-
      [
        {
          "apiVersion": "ansibleee.openstack.org/v1alpha1",
          "kind": "OpenStackAnsibleEE",
          "metadata": {
            "name": "openstackansibleee-sample"
          },
```

```
          "spec": null
        }
      ]
[...]
              env:
              - name: ANSIBLEEE_IMAGE_URL_DEFAULT
                value: registry.redhat.io/rhosp-dev-preview/ee-openstack-ansible-ee-rhel8:0.1.0
              image: registry.redhat.io/rhosp-dev-preview/openstack-ansibleee-rhel9-
operator:0.1
[...]
```

11. Wait until the **csv PHASE** field moves from **Installing** to **Succeeded**:

```
$ oc get csv openstack-ansibleee-operator.v0.0.1 -n openstack-operators -w
NAME                                DISPLAY            VERSION   REPLACES   PHASE
openstack-ansibleee-operator.v0.0.1   OpenStackAnsibleEE   0.0.1               Succeeded
```

12. Create the data plane resources:

```
$ oc create -f openstack-edpm.yaml
```

13. Verify that the data plane resources have been created:

```
$ oc get openstackdataplane
NAME           STATUS MESSAGE
openstack-edpm False  Deployment not started
```

The command output shows your data plane resources with the message, **Deployment not started**.

14. Verify that the data plane roles have been created:

```
$ oc get openstackdataplanerole
NAME         STATUS MESSAGE
edpm-compute False  Deployment not started
```

The command output shows your role resources with the message, **Deployment not started**.

15. Verify that the data plane nodes have been created:

```
$ oc get openstackdataplanenode
NAME             STATUS   MESSAGE
edpm-compute-0   False    Deployment not started
edpm-compute-1   False    Deployment not started
```

The command output shows your node resources with the message, **Deployment not started**.

16. Deploy the data plane:

```
$ oc patch openstackdataplane openstack-edpm \
 -p='[{"op": "replace", "path": "/spec/deployStrategy/deploy", "value":true}]' \
 --type json
```

TIP

You can also use the **oc edit openstackdataplane openstack-edpm** command to directly edit the file and then apply the changes.

17. Verify that the data plane is deployed:

```
$ oc get pods
$ oc get openstackdataplane
NAME            STATUS   MESSAGE
openstack-edpm   True    DataPlane Ready

$ oc get openstackdataplanerole
NAME          STATUS   MESSAGE
edpm-compute   True    DataPlaneRole Ready

$ oc get openstackdataplanenode
NAME            STATUS   MESSAGE
edpm-compute-0   True    DataPlaneNode Ready
edpm-compute-1   True    DataPlaneNode Ready
```

18. Optional: Check if any jobs have not executed successfully:

```
$ oc get jobs -n openstack --field-selector status.successful=0
NAME                                          COMPLETIONS   DURATION   AGE
dataplane-deployment-configure-network-edpm-compute   0/1           10m        10m
```

5.3. EXAMPLE DATA PLANE ROLE CONFIGURATION

The following example **OpenStackDataPlane** CR creates one role named **edpm-compute** to illustrate how to configure roles on the data plane.

```
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlane
metadata:
  name: openstack-edpm
spec:
  ...
  roles:
    edpm-compute:
      preProvisioned: true (1)
      deployStrategy:
        deploy: false (2)
      nodeTemplate:
        nova: {} (3)
        ansibleUser: rhel-user (4)
        ansibleSSHPrivateKeySecret: dataplane-ansible-ssh-private-key-secret (5)
        ansibleVars: (6)
          service_net_map:
            nova_api_network: internal_api
            nova_libvirt_network: internal_api

          # edpm_network_config
```

```
        # Default nic config template for a EDPM compute node
        # These vars are edpm_network_config role vars
        edpm_network_config_template: templates/single_nic_vlans/single_nic_vlans.j2 (7)
        edpm_network_config_hide_sensitive_logs: false
        #
        # These vars are for the network config templates themselves and are
        # considered EDPM network defaults.
        neutron_physical_bridge_name: br-ex
        neutron_public_interface_name: eth0
        ctlplane_mtu: 1500
        ctlplane_subnet_cidr: 24
        ctlplane_gateway_ip: 192.168.122.1
        ctlplane_host_routes:
        - ip_netmask: 0.0.0.0/0
          next_hop: 192.168.122.1
        external_mtu: 1500
        external_vlan_id: 44
        external_cidr: '24'
        external_host_routes: []
        internal_api_mtu: 1500
        internal_api_vlan_id: 20
        internal_api_cidr: '24'
        internal_api_host_routes: []
        storage_mtu: 1500
        storage_vlan_id: 21
        storage_cidr: '24'
        storage_host_routes: []
        tenant_mtu: 1500
        tenant_vlan_id: 22
        tenant_cidr: '24'
        tenant_host_routes: []
        role_networks:
        - InternalApi
        - Storage
        - Tenant
        networks_lower:
          External: external
          InternalApi: internal_api
          Storage: storage
          Tenant: tenant

        # edpm_nodes_validation
        edpm_nodes_validation_validate_controllers_icmp: false
        edpm_nodes_validation_validate_gateway_icmp: false

        # Variables set with values from the controlplane
        edpm_ovn_metadata_agent_DEFAULT_transport_url:
rabbit://default_user@rabbitmq.openstack.svc:5672
        edpm_ovn_metadata_agent_metadata_agent_ovn_ovn_sb_connection: tcp:10.217.5.121:6642
        edpm_ovn_metadata_agent_metadata_agent_DEFAULT_nova_metadata_host: 127.0.0.1
        edpm_ovn_metadata_agent_metadata_agent_DEFAULT_metadata_proxy_shared_secret:
12345678
        edpm_ovn_metadata_agent_DEFAULT_bind_host: 127.0.0.1
        edpm_ovn_dbs:
        - 192.168.24.1
```

```
        edpm_chrony_ntp_servers:
        - clock.redhat.com
        - clock2.redhat.com

        ctlplane_dns_nameservers:
        - 192.168.122.80
        dns_search_domains: []

        edpm_ovn_controller_agent_image: registry.redhat.io/rhosp-dev-preview/openstack-ovn-controller-rhel9:18.0
        edpm_iscsid_image: registry.redhat.io/rhosp-dev-preview/openstack-iscsid-rhel9:18.0
        edpm_logrotate_crond_image: registry.redhat.io/rhosp-dev-preview/openstack-cron-rhel9:18.0
        edpm_nova_compute_container_image: registry.redhat.io/rhosp-dev-preview/openstack-nova-compute-rhel9:18.0
        edpm_nova_libvirt_container_image: registry.redhat.io/rhosp-dev-preview/openstack-nova-libvirt-rhel9:18.0
        edpm_ovn_metadata_agent_image: registry.redhat.io/rhosp-dev-preview/openstack-neutron-metadata-agent-ovn-rhel9:18.0

        gather_facts: false
        enable_debug: false
        # edpm firewall, change the allowed CIDR if needed
        edpm_sshd_configure_firewall: true
        edpm_sshd_allowed_ranges: ['192.168.122.0/24']
        # SELinux module
        edpm_selinux_mode: enforcing
```

1. Specify if the nodes with this role are pre-provisioned, or if they must be provisioned when creating the resource. For information on how to configure your **OpenStackDataPlane** CR to provision bare-metal nodes, see Provisioning bare-metal nodes.
2. Disable data plane deployment after the resources are created.
3. Specify that all nodes in the role are Compute nodes.
4. The user associated with the secret you created in Creating the SSH key secret .
5. The name of the secret that you created in Creating the SSH key secret .
6. The Ansible variables that customize the role. For a complete list of Ansible variables, see https://openstack-k8s-operators.github.io/edpm-ansible/.
7. The network configuration template to apply to nodes in the role. For sample templates, see https://github.com/openstack-k8s-operators/edpm-ansible/tree/main/roles/edpm_network_config/templates.

5.4. EXAMPLE DATA PLANE NODE CONFIGURATION

The following example **OpenStackDataPlane** CR creates one node named **edpm-compute-0** to illustrate how to configure nodes on the data plane.

```
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlane
metadata:
  name: openstack-edpm
```

```
spec:
  ...
  nodes:
    edpm-compute-0:
      role: edpm-compute (1)
      hostName: edpm-compute-0
      ansibleHost: 192.168.122.100
      node:
        ansibleVars: | (2)
          ctlplane_ip: 192.168.122.100
          internal_api_ip: 172.17.0.100
          storage_ip: 172.18.0.100
          tenant_ip: 172.19.0.100
          fqdn_internal_api: edpm-compute-0.example.com
      deployStrategy:
        deploy: false
```

1. The name of the node role. Each node must have a defined role.
2. The node-specific Ansible variables.

5.5. REGISTERING PRE-PROVISIONED NODES TO THE DATA PLANE

Before you deploy the data plane, you must register the operating system of any pre-provisoned data plane nodes that have not been previously registered to the Red Hat Customer Portal, and enable the repositories for your nodes.

Procedure

1. Add the **edpm_bootstrap_command** variable to the **ansibleVars** section of the role definition for the nodes in your **OpenStackDataPlane** CR file, **openstack-edpm.yaml**:

```
spec:
  roles:
    edpm-compute:
      preProvisioned: true
      deployStrategy:
        deploy: false
      ...
        ansibleVars: |
          service_net_map:
            nova_api_network: internal_api
            nova_libvirt_network: internal_api
            ...
          edpm_bootstrap_command: |
            subscription-manager register --username <subscription_manager_username> --
password <subscription_manager_password>
            subscription-manager release --set=9.2
            subscription-manager repos --disable=*
            subscription-manager repos --enable=rhel-9-for-x86_64-baseos-eus-rpms --
enable=rhel-9-for-x86_64-appstream-eus-rpms --enable=rhel-9-for-x86_64-highavailability-
eus-rpms --enable=openstack-17.1-for-rhel-9-x86_64-rpms --enable=fast-datapath-for-rhel-9-
x86_64-rpms --enable=openstack-dev-preview-for-rhel-9-x86_64-rpms
            podman login -u <registry_username> -p <registry_password> registry.redhat.io
```

- Replace **<subscription_manager_username>** with the applicable user name.
- Replace **<subscription_manager_password>** with the applicable password.
- Replace **<registry_username>** with the applicable user name.
- Replace **<registry_password>** with the applicable password.
  For a complete list of the Red Hat Customer Portal registration commands, see https://access.redhat.com/solutions/253273.

2. Save the changes to your **openstack-edpm.yaml** file.
3. Return to Creating and deploying the data plane to complete the creation of the data plane.

5.6. PROVISIONING BARE-METAL NODES

Bare-metal node provisioning is supported with the Red Hat OpenShift Container Platform (RHOCP) Cluster Baremetal Operator (CBO). CBO is an RHOCP Operator responsible for deploying all components required to provision bare-metal nodes.

RHOCP clusters that are installed with the installer-provisioned infrastructure (IPI) or assisted installation (AI) have CBO enabled. RHOCP clusters that are installed with user-provisioned infrastructure (UPI) might not have it enabled.

With AI or IPI clusters, available hosts are managed with the **BareMetalHost** CRD. The CBO performs the following operations:

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

A provisioning network needs to be configured for CBO. It is included by default with an IPI-installed RHOCP cluster. AI installed clusters do not have this provisioning network available by default. You must manually add it to the cluster after installation.

Prerequisites

- The networks are configured for provisioning and the control plane.
- CBO is installed and configured for provisioning.
- **BareMetalHosts** CRs are registered, inspected, and have the label **app:openstack**.

Procedure

1. Add the **baremetalSetTemplate** to the role definition for the bare-metal nodes in your **OpenStackDataPlane** custom resource (CR) file, **openstack-edpm.yaml**. The following example adds **baremetalSetTemplate** to the **edpm-compute** role:

```
spec:
  ...
  roles:
    edpm-compute:
```

```
    preProvisioned: false
    deployStrategy:
      deploy: false
    ...
    baremetalSetTemplate:
      deploymentSSHSecret: dataplane-ansible-ssh-private-key-secret
      bmhNameSpace: openshift-machine-api
      cloudUserName: <ansible_ssh_user>
      bmhLabelSelector:
        app: openstack
      ctlplaneInterface: enp1s0
      dnsSearchDomains:
        - osptest.openstack.org
      ...
```

2. Save the changes to your **openstack-edpm.yaml** file.

3. Return to Creating and deploying the data plane to complete the creation of the data plane.

5.7. TROUBLESHOOTING SOFTWARE CONFIGURATION

During role deployment, an **OpenStackAnsibleEE** resource is created for each service in the list of role services. You can use this resource during or after the deployment to troubleshoot software configuration by retrieving the Ansible execution logs.

Procedure

1. Retrieve the name and status of services:
   $ oc get openstackansibleee

   The following example output shows job execution based on a default services list:

```
NAME                                                NETWORKATTACHMENTS STATUS MESSAGE
dataplane-deployment-configure-network-edpm-compute                    True
AnsibleExecutionJob complete
dataplane-deployment-configure-os-edpm-compute                         True
AnsibleExecutionJob complete
dataplane-deployment-install-os-edpm-compute                           True   AnsibleExecutionJob
complete
dataplane-deployment-run-os-edpm-compute                               True   AnsibleExecutionJob
complete
Dataplane-deployment-validate-network-edpm-compute                     True
AnsibleExecutionJob complete
```

2. Retrieve the required logs using the service names obtained in the previous step:
   $ oc logs job/<openstackansibleee-name>

   - Replace **<openstackansibleee-name>** with the name of the required service.
     Example:

     $ oc logs job/dataplane-deployment-configure-network-edpm-compute

CHAPTER 6. ACCESSING THE DATA PLANE

After you deploy the data plane, you can access it and execute Red Hat OpenStack Platform (RHOSP) commands by using the **openstack** client tool. You access the data plane by using the **OpenStackClient** pod through a remote shell. The **OpenStackClient** pod contains the client tools and authentication details that you require to perform actions on your data plane. The OpenStack Operator deploys the **OpenStackClient** pod as a part of the **OpenStackControlPlane** resource that you created.

6.1. ACCESSING THE OPENSTACKCLIENT POD

To access the **OpenStackClient** pod from your workstation, use the **oc** command on your workstation to connect to the remote shell for the pod.

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

6.2. ACCESSING THE OPENSTACK DASHBOARD SERVICE (HORIZON) INTERFACE

You can access the OpenStack Dashboard service (horizon) interface by using a web browser to access the virtual IP address that is reserved by the control plane.

Procedure

1. Retrieve the name of the **OpenStackControlPlane** CR:

```
$ oc get openstackcontrolplanes
```

2. Enable the Dashboard service in the **OpenStackControlPlane** CR:

```
$ oc patch openstackcontrolplanes/<openstackcontrolplane_name> -p='[{"op": "replace",
"path": "/spec/horizon/enabled", "value": true}]' --type json
```

- Replace **<openstackcontrolplane_name>** with the name of your **OpenStackControlPlane** CR, for example, **openstack-network-isolation**.

3. Optional: To log in as the admin user, obtain the admin password from the **AdminPassword** parameter in the **osp-secret** secret:

```
$ oc get secret osp-secret -o jsonpath='{.data.AdminPassword}' | base64 -d
```

4. Retrieve the Dashboard service endpoint URL:

```
$ oc get horizons horizon -o jsonpath='{.status.endpoint}'
```

5. Open a web browser.
6. Enter the Dashboard endpoint URL.
7. Log in to the dashboard with your username and password.

CHAPTER 7. CONFIGURING RED HAT CEPH STORAGE AS THE BACK END FOR RED HAT OPENSTACK PLATFORM STORAGE

You can configure Red Hat OpenStack Platform (RHOSP) to use an external Red Hat Ceph Storage cluster. This configuration connects the Image service (glance), Block Storage service (cinder), Compute service (nova) and the Shared File Systems service (manila) to a Red Hat Ceph Storage cluster.

If you want to deploy a Red Hat Ceph Storage Hyper Converged Infrastructure (HCI), see Configuring a Hyperconverged Infrastructure environment.

If you want to configure RHOSP with NFS instead of Red Hat Ceph Storage, see Configuring alternate storage backends with OpenStack services.

To configure Red Hat Ceph Storage as the backend for the RHOSP storage, complete the following tasks:

1. Create the Red Hat Ceph Storage pools on the Red Hat Ceph Storage cluster.
2. Create a Red Hat Ceph Storage secret to provide RHOSP services access to the Red Hat Ceph Storage cluster.
3. Configure the **OpenStackControlPlane** CR to use the Red Hat Ceph Storage cluster as the back end.
4. Configure the **OpenStackDataPlane** CR to use the Red Hat Ceph Storage cluster as the back end.

Prerequisites

- Access to a Red Hat Ceph Storage cluster. If you intend to host Red Hat Ceph Storage on data plane nodes (HCI), then perform Configuring a Hyperconverged Infrastructure environment first.
- The RHOSP control plane is installed on an operational Red Hat OpenShift Platform cluster.

7.1. CREATING RED HAT CEPH STORAGE POOLS

Create Red Hat Ceph Storage cluster pools for each service that uses the cluster. You perform the configuration on the Red Hat Ceph Storage cluster server.

Procedure

1. Create pools for the Compute service (vms), the Block Storage service (volumes), and the Image service (images):

```
$ for P in vms volumes images; do
  cephadm shell -- ceph osd pool create $P;
  cephadm shell -- ceph osd pool application enable $P rbd;
done
```

2. Optional: Create the **cephfs** volume if the Shared File Systems service (manila) is enabled in the control plane:
   $ cephadm shell -- ceph fs volume create cephfs

3. Create a cephx key for RHOSP to use to access pools:

```
$ cephadm shell -- \
  ceph auth add client.openstack \
    mgr 'allow *' \
      mon 'allow r' \
      osd 'allow class-read object_prefix rbd_children, allow rwx pool=vms, allow rwx
pool=volumes, allow rwx pool=images'
```

4. Export the cephx key:
$ cephadm shell -- ceph auth get client.openstack > /etc/ceph/ceph.client.openstack.keyring

5. Export the configuration file:
$ cephadm shell -- ceph config generate-minimal-conf > /etc/ceph/ceph.conf

7.2. CREATING A RED HAT CEPH STORAGE SECRET

Create a Red Hat Ceph Storage secret so that services can access the Red Hat Ceph Storage cluster.

Procedure

1. Transfer the **cephx** key and configuration file created in the Creating Red Hat Ceph Storage pools procedure to a host that can create resources in the **openstack** namespace.

2. Base64 encode these files and store them in **KEY** and **CONF** environment variables:

```
$ KEY=$(cat /etc/ceph/ceph.client.openstack.keyring | base64 -w 0)
$ CONF=$(cat /etc/ceph/ceph.conf | base64 -w 0)
```

3. Create a YAML file to create the **Secret** resource.

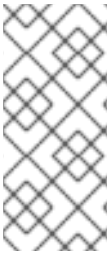
4. Using the environment variables, add the **Secret** configuration to the YAML file:

```
apiVersion: v1
data:
  ceph.client.openstack.keyring: $KEY
  ceph.conf: $CONF
kind: Secret
metadata:
  name: ceph-conf-files
  namespace: openstack
type: Opaque
```

5. Save the YAML file.

6. Create the **Secret** resource:
$ oc create -f <secret_configuration_file>

   - Replace **<secret_configuration_file>** with the name of the YAML file you created.



NOTE

The examples in this section use **openstack** as the name of the Red Hat Ceph Storage user. The file name in the **Secret** resource must match this user name.

For example, if the file name used is **/etc/ceph/ceph.client.openstack2.keyring**, then the secret data line should be **ceph.client.openstack2.keyring: $KEY**.

7.3. OBTAINING THE RED HAT CEPH STORAGE FSID

The Red Hat Ceph Storage File System Identifier (FSID) is a unique identifier for the cluster. The FSID is used in configuration and verification of cluster interoperability with RHOSP.

Procedure

- Extract the FSID from the Red Hat Ceph Storage secret:
  $ FSID=$(oc get secret ceph-conf-files -o json | jq -r '.data."ceph.conf"' | base64 -d | grep fsid | sed -e 's/fsid = //')

7.4. CONFIGURING THE CONTROL PLANE TO USE THE RED HAT CEPH STORAGE CLUSTER

You must configure the **OpenStackControlPlane** CR to use the Red Hat Ceph Storage cluster. Configuration includes the following tasks:

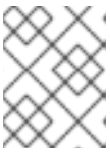
1. Confirming the Red Hat Ceph Storage cluster and associated services have the correct network configuration.
2. Configuring the control plane to use the Red Hat Ceph Storage secret.
3. Configuring the Image service (glance) to use the Red Hat Ceph Storage cluster.
4. Configuring the Block Storage service (cinder) to use the Red Hat Ceph Storage cluster.
5. Optional: Configuring the Shared File Systems service (manila) to use native CephFS with the Red Hat Ceph Storage cluster.

NOTE

This example does not include configuring **cinder-backup** with Red Hat Ceph Storage.

Procedure

1. Check the storage interface defined in your **NodeNetworkConfigurationPolicy** (**nncp**) custom resource to confirm that it has the same network configuration as the Red Hat Ceph Storage cluster **public_network**. This is required to enable access to the Red Hat Ceph Storage cluster through the **Storage** network.The **Storage** network should have the same network configuration as the Red Hat Ceph Storage cluster **public_network**.



NOTE

It is not necessary for RHOSP to access the Ceph Storage cluster **cluster_network**.

2. Check the **networkAttachments** for the Image service in the **OpenStackControlPlane** CR to confirm that the Image service is configured to access the Storage network:

```
glance:
    template:
          databaseInstance: openstack
      glanceAPIExternal:
        networkAttachments:
        - storage
      glanceAPIInternal:
        externalEndpoints:
        - endpoint: internal
          ipAddressPool: internalapi
          loadBalancerIPs:
          - 172.17.0.80
        networkAttachments:
        - storage
      storageClass: ""
      storageRequest: 10G
```

3. Confirm the Block Storage service is configured to access the **Storage** network through MetalLB.

4. Confirm the Compute service (nova) is configured to access the **Storage** network.

5. Confirm the Red Hat Ceph Storage configuration file, **/etc/ceph/ceph.conf**, contains the IP addresses of the Red Hat Ceph Storage cluster monitors. These IP addresses must be within the **Storage** network IP address range.

6. Open your **openstack_control_plane.yaml** file to edit the **OpenStackControlPlane** CR.

7. Add the **extraMounts** parameter to define the services that require access to the Red Hat Ceph Storage secret.
   The following is an example of using the **extraMounts** parameter for this purpose. You should only include **ManilaShare** in the propagation list if you are using the Shared File Systems service (manila):

```
apiVersion: core.openstack.org/v1beta1
kind: OpenStackControlPlane
spec:
  extraMounts:
    - name: v1
      region: r1
      extraVol:
        - propagation:
          - CinderVolume
          - GlanceAPI
          - ManilaShare
          extraVolType: Ceph
          volumes:
          - name: ceph
            projected:
              sources:
              - secret:
                  name: <ceph-conf-files 1>
          mounts:
```

```
        - name: ceph
          mountPath: "/etc/ceph"
          readOnly: true
```

- Replace <ceph-conf-files 1> with the name of your Secret CR created in Creating a Red Hat Ceph Storage secret.

8. Add the **customServiceConfig** parameter to the **glance** template to configure the Image service to use the Red Hat Ceph Storage cluster:

```
apiVersion: core.openstack.org/v1beta1
kind: OpenStackControlPlane
spec:
  extraMounts:
    ...
  glance:
    template:
      databaseInstance: openstack
      customServiceConfig: |
        [DEFAULT]
        enabled_backends = default_backend:rbd
        [glance_store]
        default_backend = default_backend
        [default_backend]
        rbd_store_ceph_conf = /etc/ceph/ceph.conf
        store_description = "RBD backend"
        rbd_store_pool = images
            rbd_store_user = openstack
```

9. Add the **customServiceConfig** parameter to the **cinder** template to configure the Block Storage service to use the Red Hat Ceph Storage cluster:

```
apiVersion: core.openstack.org/v1beta1
kind: OpenStackControlPlane
spec:
  extraMounts:
    ...
  cinder:
    template:
      cinderVolumes:
        ceph:
          customServiceConfig: |
            [DEFAULT]
            enabled_backends=ceph
            [ceph]
            volume_backend_name=ceph
            volume_driver=cinder.volume.drivers.rbd.RBDDriver
            rbd_ceph_conf=/etc/ceph/ceph.conf
            rbd_user=openstack
            rbd_pool=volumes
            rbd_flatten_volume_from_snapshot=False
            rbd_secret_uuid=$FSID 1
```

1 Set to the actual FSID. The FSID itself does not need to be considered secret. For more information, see Obtaining the Red Hat Ceph Storage FSID .

10. Optional: Add the **customServiceConfig** parameter to the **manila** template to configure the Shared File Systems service to use native CephFS with the Red Hat Ceph Storage cluster:

```
apiVersion: core.openstack.org/v1beta1
kind: OpenStackControlPlane
spec:
  extraMounts:
    ...
    manila:
      template:
        manilaShares:
          share1:
            customServiceConfig: |
              [DEFAULT]
              enabled_share_backends=cephfs
              enabled_share_protocols=cephfs
              [cephfs]
              driver_handles_share_servers=False
              share_backend_name=cephfs
              share_driver=manila.share.drivers.cephfs.driver.CephFSDriver
              cephfs_conf_path=/etc/ceph/ceph.conf
              cephfs_auth_id=openstack
              cephfs_cluster_name=ceph
              cephfs_volume_mode=0755
              cephfs_protocol_helper_type=CEPHFS
```

11. Apply the updates to the **OpenStackControlPlane** CR:
$ oc apply -f openstack_control_plane.yaml

7.5. CONFIGURING THE DATA PLANE TO USE THE CEPH STORAGE CLUSTER

Configure the OpenStackDataPlane CR to use the Red Hat Ceph Storage cluster.

Procedure

1. Open your **openstack_data_plane.yaml** file to edit the **OpenStackDataPlane** CR.
2. Add the **extraMounts** parameter to the **edpm-compute** role to specify the **Secret** CR:

```
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlane
spec:
  roles:
    edpm-compute:
      nodeTemplate:
        extraMounts:
        - extraVolType: Ceph
          volumes:
          - name: ceph
            secret:
              secretName: <ceph-conf-files 1>
          mounts:
```

```
      - name: ceph
        mountPath: "/etc/ceph"
        readOnly: true
```

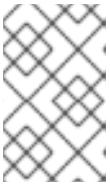
- Replace <ceph-conf-files 1> with the name of your **Secret** CR created in Creating a Red Hat Ceph Storage secret.

NOTE

When a CR is created with this configuration, Ansible mounts the files in the Ceph Storage secret and copies them to the data plane host using the edpm_ceph_client_files Ansible role. This gives the Compute service containers a copy of **/etc/ceph** to use to access the **cephx** key and Ceph configuration file.

3. Add the **customServiceConfig** parameter to the **edpm-compute** role to configure the Compute service (nova) to use the Red Hat Ceph Storage cluster:

```
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlane
spec:
...
  roles:
    edpm-compute:
    ...
        nova:
          cellName: cell1
          customServiceConfig: |
            [libvirt]
            images_type=rbd
            images_rbd_pool=vms
            images_rbd_ceph_conf=/etc/ceph/ceph.conf
            images_rbd_glance_store_name=default_backend
            images_rbd_glance_copy_poll_interval=15
            images_rbd_glance_copy_timeout=600
            rbd_user=openstack
            rbd_secret_uuid=$FSID
          deploy: true
          novaInstance: nova
```



NOTE

The **$FSID** value should contain the actual FSID as described in xref:proc_ceph-obtaining-ceph-fsid_ceph-back-endObtaining the Red Hat //Ceph Storage FSID]. The FSID itself does not need to be considered secret.

4. Apply the updates to the **OpenStackDataPlane** CR:
   $ oc apply -f <dataplane_cr_file>

- Replace **<dataplane_cr_file>** with the name of your file.

CHAPTER 8. CONFIGURING A HYPERCONVERGED INFRASTRUCTURE ENVIRONMENT

You can deploy data plane nodes that host both Red Hat Ceph Storage and the Compute service (nova). This type of deployment is known as a Hyperconverged Infrastructure (HCI) environment.

Creating an HCI environment consists of the following tasks:

1. Configuring the data plane node networking.
2. Installing Red Hat Ceph Storage on the data plane nodes.
3. Configuring Red Hat OpenStack Platform (RHOSP) to use the Red Hat Ceph Storage cluster.

8.1. DATA PLANE NODE SERVICES LIST

You configure data plane nodes by creating either an **OpenStackDataPlane** custom resource (CR) or an **OpenStackDataPlaneRole** and **OpenStackDataPlaneNode** CR. The **dataplane-operator** reconciles the CRs.

These CRs have a service list similar to the following example:

```
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlane
spec:
  ...
  roles:
    edpm-compute:
      ...
      services:
        - configure-network
        - validate-network
        - install-os
        - configure-os
        - run-os
```

Only the services in the services list will be configured.

You must deploy Red Hat Ceph Storage on the data plane node after the **Storage** and **Storage Management** networks are configured but before the Compute service (nova) is configured. This means you must edit the services list and make other changes to the CR.

8.2. CONFIGURING THE DATA PLANE NODE NETWORKS

You must configure the data plane node networks to accommodate the Red Hat Ceph Storage networking requirements.

Prerequisites

- Control plane deployment is complete but has not yet been modified to use Ceph Storage.
- The data plane nodes have been provisioned with an operating system.
- The data plane nodes are accessible through an SSH key that Ansible can use.

- The data plane nodes have disks available to be used as Ceph OSDs.
- There are a minimum of three data plane nodes available. Ceph Storage clusters must have a minimum of three nodes to ensure redundancy.

Procedure

1. Create an **OpenStackDataPlane** CR file to represent the data plane nodes.

NOTE

Do not create the CR in Red Hat OpenShift yet.

2. Add the **configure-network** and **validate-network** parameters to the **services** list and remove all other service listings.
   The following is an example of adding these parameters:

```
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlane
spec:
  ...
  roles:
    edpm-compute:
      ...
      services:
        - configure-network
        - validate-network
```

3. Ensure that the **roles** parameter in the CR file does not have a **nova** key.
   For example, if the **roles** parameter in the CR file has the following:

```
roles:
   edpm-compute:
      nova: {}
```

   Remove **nova** so that the **roles** parameter looks like this instead:

```
roles:
   edpm-compute:
```

4. Configure the Red Hat Ceph Storage **cluster_network** for storage management traffic between OSDs. This is accomplished by modifying the CR to set **edpm-ansible** variables so that the **edpm_network_config** role will configure a storage management network which Red Hat Ceph Storage will use as the **cluster_network**.
   The following example assumes the storage management network range is **172.20.0.0/24** and that it is on **VLAN23**. The bolded lines are additions for the **cluster_network**:

```
roles:
  edpm-compute:
    ansibleVars: |
    ctlplane_ip: 192.168.122.100
    internal_api_ip: 172.17.0.10
    storage_ip: 172.18.0.100
```

```
    storage_mgmt_ip: 172.20.0.100
      tenant_ip: 172.19.0.100
      fqdn_internal_api: {{ ansible_fqdn }}
role: edpm-compute
    ctlplane_ip: 192.168.122.101
    internal_api_ip: 172.17.0.101
    storage_ip: 172.18.0.101
    storage_mgmt_ip: 172.20.0.101
    tenant_ip: 172.19.0.101
    fqdn_internal_api: {{ ansible_fqdn }}
    openStackAnsibleEERunnerImage: quay.io/openstack-k8s-operators/openstack-
ansibleee-runner:latest

    ctlplane_ip: 192.168.122.102
    internal_api_ip: 172.17.0.102
    storage_ip: 172.18.0.102
    storage_mgmt_ip: 172.20.0.102
    tenant_ip: 172.19.0.102
    fqdn_internal_api: {{ ansible_fqdn }}

    storage_vlan_id: 21
    storage_mtu: 9000
    storage_cidr: 24
   storage_host_routes: []
    storage_mgmt_mtu: 9000 storage_mgmt_vlan_id: 23 storage_mgmt_cidr: 24
storage_mgmt_host_routes: []
    tenant_mtu: 1500
    tenant_vlan_id: 22
    tenant_cidr: 24

    role_networks:
    - InternalApi
    - Storage
    - StorageMgmt
    - Tenant
    networks_lower:
      External: external
        InternalApi: internal_api
        Storage: storage
        StorageMgmt: storage_mgmt
        Tenant: tenant
```

NOTE

It is not necessary to add the storage management network to the **networkAttachments** key.

5. Apply the CR.
   $ oc apply -f <dataplane_cr_file>

   Replace **<dataplane_cr_file>** with the name of your file.

NOTE

Ansible configures and validates the networks when the CR is created.

6. Confirm the network is configured.
    a. SSH into a data plane node.
    b. Use the **ip a** command to display configured networks.
    c. Confirm the storage networks are in the list of configured networks.

8.2.1. Red Hat Ceph Storage MTU settings

The example in this procedure changes the MTU of the **storage** and **storage_mgmt** networks from 1500 to 9000. An MTU of 9000 is known as a jumbo frame. Use jumbo frames for improved storage performance even though increasing the MTU is not mandatory. If you use jumbo frames, you must configure all network switch ports in the data path to support jumbo frames. You must also make MTU changes for services using the **Storage** network on Red Hat OpenShift.

To change the MTU for the Red Hat OpenShift services connecting to the data plane nodes, update the Node Network Configuration Policy (NNCP) for the base interface as well as the VLAN interface. It is not necessary to update the Network Attachment Definition (NAD) if the main NAD interface already has the MTU value you want. If the MTU of the underlying interface is set to 9000 and it is not specified for the VLAN interface on top of it, then it defaults to the value from the underlying interface.

If the MTU values are not consistent, issues can occur on the application layer that can cause the Red Hat Ceph Storage cluster to not reach quorum or not support authentication using the CephX protocol. If the MTU is changed and you observe these types of problems, verify all hosts that use the network using jumbo frames can communicate at the chosen MTU value with the ping command, for example:

```
ping -M do -s 8972 172.20.0.100
```

8.3. CONFIGURING AND DEPLOYING RED HAT CEPH STORAGE ON DATA PLANE NODES

You can use the **cephadm** utility to configure and deploy Red Hat Ceph Storage for a Hyperconverged Infrastructure (HCI) environment.

8.3.1. The cephadm utility

Use the **cephadm** utility to configure and deploy Red Hat Ceph Storage on the data plane nodes. The **cephadm** package must be deployed on at least one data plane node before proceeding; **edpm-ansible** does not deploy Red Hat Ceph Storage.

For additional information and procedures for deploying Red Hat Ceph Storage, see Red Hat Ceph Storage installation in the *Red Hat Ceph Storage Installation Guide* .

8.3.2. Configuring and deploying Red Hat Ceph Storage

Configure and deploy Red Hat Ceph Storage by editing the configuration file and using the **cephadm** utility.

Procedure

1. Edit the Red Hat Ceph Storage configuration file.

2. Add the **Storage** and **Storage Management** network ranges. Red Hat Ceph Storage uses the **Storage** network as the Red Hat Ceph Storage **public_network** and the **Storage Management** network as the **cluster_network**.
   The following example is for a configuration file entry where the **Storage** network range is **172.18.0.0/24** and the **Storage Management** network range is **172.20.0.0/24**:

```
[global]
public_network = 172.18.0.0/24
cluster_network = 172.20.0.0/24
```

3. Add collocation boundaries between the Compute service and Ceph OSD services. Set boundaries set between collocated Compute service and Ceph OSD services to reduce CPU and memory contention between them.
   The following is an example for a Ceph configuration file entry with these boundaries set:

```
[osd]
osd_memory_target_autotune = true
osd_numa_auto_affinity = true
[mgr]
mgr/cephadm/autotune_memory_target_ratio = 0.2
```

   In this example, the **osd_memory_target_autotune** parameter is set to **true** so that the OSD daemons adjust memory consumption based on the **osd_memory_target** option. The **autotune_memory_target_ratio** defaults to 0.7. This means 70% of the total RAM in the system is the starting point from which any memory consumed by non-autotuned Ceph daemons is subtracted, and then the remaining memory is divided between the OSDs; assuming all OSDs have **osd_memory_target_autotune** set to true. For HCI deployments, you can set **mgr/cephadm/autotune_memory_target_ratio** to 0.2 so that more memory is available for the Compute service.

   For additional information about service collocation, see Collocating services in a HCI environment for NUMA nodes.

NOTE

If these values need to be adjusted after the deployment, use the **ceph config set osd <key> <value>** command.

4. Deploy Ceph Storage with the edited configuration file on a data plane node:
   $ cephadm bootstrap --config <config_file> --mon-ip <data_plane_node_ip>

   - Replace **<config_file>** with the name of your Ceph configuration file.
   - Replace **<data_plane_node_ip>** with the **Storage** network IP address of the data plane node on which Red Hat Ceph Storage will be installed.

NOTE

Repeat this step for all data plane nodes on which you want to deploy Red Hat Ceph Storage.

8.3.2.1. Collocating services in a HCI environment for NUMA nodes

A two-NUMA node system can host a latency sensitive Compute service workload on one NUMA node and a Ceph OSD workload on the other NUMA node. To configure Ceph OSDs to use a specific NUMA node not being used by the the Compute service, use either of the following Ceph OSD configurations:

- **osd_numa_node** sets affinity to a NUMA node (-1 for none).
- **osd_numa_auto_affinity** automatically sets affinity to the NUMA node where storage and network match.

If there are network interfaces on both NUMA nodes and the disk controllers are on NUMA node 0, do the following:

1. Use a network interface on NUMA node 0 for the storage network
2. Host the Ceph OSD workload on NUMA node 0.
3. Host the Compute service workload on NUMA node 1 and have it use the network interfaces on NUMA node 1.

Set **osd_numa_auto_affinity** to true, as in the initial Ceph configuration file. Alternatively, set the **osd_numa_node** directly to 0 and clear the **osd_numa_auto_affinity** parameter so that it defaults to **false**.

When a hyperconverged cluster backfills as a result of an OSD going offline, the backfill process can be slowed down. In exchange for a slower recovery, the backfill activity has less of an impact on the collocated Compute service (nova) workload. Red Hat Ceph Storage has the following defaults to control the rate of backfill activity.

- **osd_recovery_op_priority = 3**
- **osd_max_backfills = 1**
- **osd_recovery_max_active_hdd = 3**
- **osd_recovery_max_active_ssd = 10**

8.3.3. Confirming Red Hat Ceph Storage deployed

Confirm Red Hat Ceph Storage is deployed before proceeding.

Procedure

1. Connect to a data plane node by using SSH.
2. View the status of the Red Hat Ceph Storage cluster:
   $ cephadm shell -- ceph -s

8.3.4. Confirming Red Hat Ceph Storage tuning

Ensure that Red Hat Ceph Storage is properly tuned before proceeding.

Procedure

1. Connect to a data plane node by using SSH.
2. Verify overall Red Hat Ceph Storage tuning with the following commands:

```
$ ceph config dump | grep numa
$ ceph config dump | grep autotune
$ ceph config dump | get mgr
```

3. Verify the tuning of a specific OSD with the following commands:

```
$ ceph config get <osd_number> osd_memory_target
$ ceph config get <osd_number> osd_memory_target_autotune
$ ceph config get <osd_number> osd_numa_auto_affinity
```

Replace **<osd_number>** with the number of a specific OSD. For example, to refer to OSD 11, use **osd.11**.

4. Verify the default backfill values of a specific OSD with the following commands:

```
$ ceph config get <osd_number> osd_recovery_op_priority
$ ceph config get <osd_number> osd_max_backfills
$ ceph config get <osd_number> osd_recovery_max_active_hdd
$ ceph config get <osd_number> osd_recovery_max_active_ssd
```

Replace **<osd_number>** with the number of a specific OSD. For example, to refer to OSD 11, use **osd.11**.

8.4. CONFIGURING THE DATA PLANE TO USE THE COLLOCATED CEPH STORAGE SERVER

Although the Red Hat Ceph Storage cluster is physically collocated with the Compute services on the data plane nodes, it is treated as logically separated. Red Hat Ceph Storage must be configured as the storage solution before the data plane nodes can use it.

Prerequisites

- Complete the procedures in the section Configuring Red Hat Ceph Storage as the backend for Red Hat OpenStack Platform (RHOSP) storage.

Procedure

1. Edit the **OpenStackDataPlane** CR.

2. Use the **extraMounts** parameter to define the **cephx** key and configuration file for the Compute service (nova).
The following is an example of using the **extraMounts** parameter for this purpose:

```
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlane
spec:
  roles:
    edpm-compute:
      nodeTemplate:
        extraMounts:
        - extraVolType: Ceph
          volumes:
          - name: ceph
```

```
      secret:
        secretName: ceph-conf-files
    mounts:
    - name: ceph
      mountPath: "/etc/ceph"
      readOnly: true
```

3. Locate the **services** list in the **OpenStackDataPlane** CR.

4. Edit the **services** list to restore all of the services removed in Configuring the data plane node networks. Restoring the full **services** list allows the remaining jobs to be run that complete the configuration of the HCI environment.
   The following is an example of a full **services** list:

```
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlane
spec:
  ...
  roles:
    edpm-compute:
      ...
      services:
        - configure-network
        - validate-network
        - install-os
        - configure-os
        - run-os
```

NOTE

It is not necessary to remove the **configure-network** or **validate-network** services because those Ansible jobs will not run again. After the configuration modification is complete, new jobs will be created for the newly added services.

5. Apply the CR changes.
   $ oc apply -f <dataplane_cr_file>

   Replace **<dataplane_cr_file>** with the name of your file.

8.5. EXAMPLE COMPUTE SERVICE HCI CONFIGURATION

Configure and tune the Compute service (nova) for collocation with Red Hat Ceph Storage after Red Hat Ceph Storage deployment. The following is an example of a Compute service configuration. It includes the directive **reserved_host_memory_mb** parameter. Use a value appropriate to your system for this parameter when configuring the Compute service for your environment.

```
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlane
spec:
  ...
  roles:
    edpm-compute:
    ...
      nova:
```

```
    cellName: cell1
    customServiceConfig: |
      [DEFAULT]
      reserved_host_memory_mb=75000
      [libvirt]
      images_type=rbd
      images_rbd_pool=vms
      images_rbd_ceph_conf=/etc/ceph/ceph.conf
      images_rbd_glance_store_name=default_backend
      images_rbd_glance_copy_poll_interval=15
      images_rbd_glance_copy_timeout=600
      rbd_user=openstack
      rbd_secret_uuid=$FSID
    deploy: true
    novaInstance: nova
```

You can set the value for the **reserved_host_memory_mb** so that the Compute service scheduler does not give memory to a virtual machine that is needed by a Ceph OSD on the same server. The example reserves 5 GB per OSD for 10 OSDs per host, in addition to the default reserved memory for the hypervisor. In an IOPS-optimized cluster, you can improve performance by reserving more memory per OSD. The 5 GB number is provided as a starting point which you can tune if necessary.

The **$FSID** value above should contain the actual FSID as described in Obtaining the Ceph FSID.

8.6. KNOWN ISSUE AND WORKAROUND

The firewall on external data plane management (EDPM) nodes currently only allows connections on port 22. This causes the Red Hat Ceph Storage cluster to be disconnected from the Red Hat Ceph Storage clients.

You can work around this condition by opening the firewall ports for Red Hat Ceph Storage on the EDPM nodes. For information on opening firewall ports, see Verifying firewall rules are configured for default Ceph ports in the *Red Hat Ceph Storage Configuration Guide* .

CHAPTER 9. CONFIGURING ALTERNATIVE STORAGE SOLUTIONS

You can configure storage solution alternatives to Red Hat Ceph Storage:

- Configure the Block Storage service (cinder) with a generic NFS back end.
- Configure the Block Storage service (cinder) with a third-party back end as described in OSP18 Cinder Alternative Storage.
- Configure the Image service (glance) with an NFS store.
- Configure the Shared File Systems service (manila) with a third-party storage system such as NetApp ONTAP.

9.1. CONFIGURING THE BLOCK STORAGE SERVICE WITH GENERIC NFS STORAGE

You can configure the Block Storage service (cinder) with a generic NFS back end to provide an alternative storage solution. To configure this storage solution, complete the following high level tasks:

1. Ensure network connectivity between the NFS server, the Red Hat OpenShift cluster, and the Compute nodes.
   a. Confirm all Block Storage services are operational.
   b. Create a test volume from an Image service image.
   c. Boot a VM from the test volume or attach a VM to the test volume.
2. Create a secret containing NFS server connection information.
3. Configure the **OpenStackControlPlane** custom resource (CR) to use the NFS storage as the back end for the Block Storage service.

9.1.1. Creating the NFS server connection secret

Create a server connection secret to prevent placing server connection information directly in the **OpenStackControlPlane** CRD.

Procedure

1. Create a configuration file that contains NFS server connection information.
   The following is an example of the contents of a configuration file:

```
[nfs]
nas_host=192.168.130.1
nas_share_path=/var/nfs/cinder
```

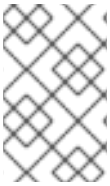
2. Save the configuration file.
3. Create the secret based on the configuration file:
   $ oc create secret generic <secret_name> --from-file=<configuration_file_name>

- Replace **<secret_name>** with the name you wish to assign to the secret.
- Replace **<configuration_file_name>** with the name of the configuration file you created.

4. Delete the configuration file.

9.1.2. Configuring the control plane to use the generic NFS driver

Configure the Block Storage service (cinder) in the **OpenStackControlPlane** CR to use NFS storage.



NOTE

Use a certified third-party NFS driver when using Red Hat OpenStack Platform (RHOSP) in a production environment. The generic NFS driver is not recommended for a production environment.

Procedure

1. Edit the **OpenStackControlPlane** CR.
2. Include NFS configuration in the relevant sections.
3. Add the **customServiceConfig** parameter to the **cinder** template to configure the Block Storage service.
   The following is an example of using the **customServiceConfig** parameter for this purpose:

```
apiVersion: core.openstack.org/v1beta1
kind: OpenStackControlPlane
metadata:
  name: openstack
spec:
  cinder:
    template:
      cinderVolumes:
        nfs:
          customServiceConfig: |
            [nfs]
            volume_backend_name=nfs
            volume_driver=cinder.volume.drivers.nfs.NfsDriver
            nfs_snapshot_support=true
            nas_secure_file_operations=false
            nas_secure_file_permissions=false
          customServiceConfigSecrets:
          - <nfs_secret_name> 1
```

1 The name of your secret created in Creating the NFS server connection secret .

4. Apply the CR changes:
   $ oc apply -f <control_plane_file>

- Replace **<control_plane_file>** with the name of your **OpenStackControlPlane** CR file.

9.2. CONFIGURING THE IMAGE SERVICE WITH THE BLOCK STORAGE SERVICE USING NFS

You can configure the Image service (glance) with the Block Storage service (cinder) by using NFS to provide an alternative storage solution. To configure this storage solution, complete the following high level tasks:

1. Ensure network connectivity between the NFS server, the Red Hat OpenShift cluster, and the Compute nodes.
2. Create a secret containing NFS server connection information.
3. Configure the **OpenStackControlPlane** custom resource (CR) to use the NFS storage as the back end for the Image service (glance).

9.2.1. Creating the Block Storage service connection secret

Create a server connection secret to prevent placing server connection information directly in the **OpenStackControlPlane** CRD.

Procedure

1. Retrieve the Block Storage service (cinder) password created at deployment and stored with the **osp-secret**:
   $ oc get secret osp-secret --template={{.data.CinderPassword}} | base64 -d
2. Create a configuration file that contains Block Storage service connection information. The following is an example of the contents of a configuration file:

```
[default_backend]
    cinder_store_user_name = glance
cinder_store_password = <cinder_password>
    cinder_store_project_name = service
```

   - Replace **<cinder_password>** with the password retrieved in the previous step.
3. Save the configuration file.
4. Create the secret based on the configuration file:
   $ oc create secret generic <secret_name> --from-file=<configuration_file_name>
   - Replace **<secret_name>** with the name you wish to assign to the secret.
   - Replace **<configuration_file_name>** with the name of the configuration file you created.
5. Delete the configuration file.

9.2.2. Configuring the control plane to use the Block Storage service with NFS

Configure the Block Storage service (cinder) in the **OpenStackControlPlane** CR to use NFS storage.

Procedure

1. Edit the **OpenStackControlPlane** CR.
2. Add the **customServiceConfig** parameter to the **glance** template to configure the Block Storage service and reference the previously created Block Storage service secret. The following is an example of using the **customServiceConfig** parameter for this purpose:

```
apiVersion: core.openstack.org/v1beta1
kind: OpenStackControlPlane
metadata:
  name: openstack
spec:
    …
   glance:
template:
       customServiceConfig: |
       [DEFAULT]
       enabled_backends = default_backend:cinder
       [glance_store]
       default_backend = default_backend
       [default_backend]
       rootwrap_config = /etc/glance/rootwrap.conf
   description = Default cinder backend
   cinder_catalog_info volumev3::publicURL
  customServiceConfigSecrets:
 - <cinder_secret_name> 1
```

1 The name of your secret created in Creating the Block Storage service connection secret .

3. Apply the CR changes:
   $ oc apply -f <control_plane_file>

   - Replace **<control_plane_file>** with the name of your **OpenStackControlPlane** CR file.

9.3. CONFIGURING THE SHARED FILE SYSTEM SERVICE WITH NETAPP ONTAP

You can configure the Shared File Systems service (manila) with NetApp ONTAP to provide an alternative storage solution. To configure this storage solution, complete the following high level tasks:

1. Prepare the NetApp ONTAP storage system for consumption by Red Hat OpenStack Platform (RHOSP).
2. Create the NetApp connection secrets.
3. Configure the **OpenStackControlPlane** CR to use the NetApp ONTAP storage system as the back end for the Shared File Systems service.

Prerequisites

- Administrative access to a NetApp ONTAP storage system to configure and consume the storage through the RHOSP Shared File Systems service.
- Network connectivity between the Red Hat OpenShift cluster, the Compute nodes, and the NetApp ONTAP storage system.

9.3.1. Preparing the NetApp ONTAP storage system

To prepare a NetApp ONTAP storage system for use, you must create ONTAP login users, vservers, and network interfaces and prepare the environment to provide the NFS or CIFS storage protocols. For information and procedures for these tasks, see the NetApp ONTAP guide: https://netapp-openstack-dev.github.io/openstack-docs.

9.3.2. Creating the NetApp server connection secret

Create a server connection secret to prevent placing server connection information directly in the **OpenStackControlPlane** custom resource definition (CRD).

Procedure

1. Create a configuration file that contains the NetApp server connection information. The following is an example of the contents of a configuration file:

```
[netapp]
netapp_server_hostname = <netapp_ip>
netapp_login = <netapp_user>
netapp_password = <netapp_password>
netapp_vserver = <netappvserver>
```

   - Replace **<netapp_ip>** with the IP address of the server.
   - Replace **<netapp_user>** with the login user name.
   - Replace **<netapp_password>** with the login password.
   - Replace **<netappvserver>** with the vserver name.

2. Save the configuration file.
3. Create the secret based on the configuration file:
   $ oc create secret generic <secret_name> --from-file=<configuration_file_name>
   - Replace **<secret_name>** with the name you wish to assign to the secret.
   - Replace **<configuration_file_name>** with the name of the configuration file you created.
4. Delete the configuration file.

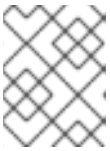
9.3.3. Configuring the control plane to use the NetApp ONTAP storage system

Configure the control plane to use the NetApp ONTAP cluster through the Shared File Systems service (manila). The configuration includes the following tasks:

- Confirming the NetApp ONTAP cluster has the correct network configuration.
- Configuring the Shared File Systems service to use native CephFS with the Red Hat Ceph Storage cluster.

Procedure

1. Confirm that the NetApp ONTAP cluster is accessible to the control plane and Compute workloads.



NOTE

NetApp ONTAP typically has separate management and data interfaces. Both interfaces must be accessible to the control plane and Compute nodes.

2. Edit the OpenStackControlPlane CR.

3. Include NetApp ONTAP configuration in the relevant sections.

4. Add the **customServiceConfig** parameter to the **manila** template to configure the Shared File Systems service.
   The following is an example of using the **customServiceConfig** parameter for this purpose:

```
apiVersion: core.openstack.org/v1beta1
kind: OpenStackControlPlane
spec:
  manila:
    template:
      manilaShares:
       share1:
         customServiceConfig: |
         [DEFAULT]
          enabled_share_backends=netapp
         [netapp]
         driver_handles_share_servers=False
         share_backend_name=netapp
share_driver=manila.share.drivers.netapp.common.NetAppDriver
         netapp_storage_family=ontap_cluster
         netapp_transport_type=http
        customServiceConfigSecrets:
         - <manila_netapp_secret> 1
```

   1 The name of your secret created in Creating the NetApp server connection secret.

5. Apply the CR changes:
   $ oc apply -f <control_plane_file>

   - Replace **<control_plane_file>** with the name of your **OpenStackControlPlane** CR file.