



# Red Hat OpenStack Services on OpenShift 18.0-beta

## Deploying Red Hat OpenStack Services on OpenShift

Deploying a Red Hat OpenStack Services on OpenShift (RHOSO) environment on a  
Red Hat OpenShift Container Platform cluster



# Red Hat OpenStack Services on OpenShift 18.0-beta Deploying Red Hat OpenStack Services on OpenShift

---

Deploying a Red Hat OpenStack Services on OpenShift (RHOSO) environment on a Red Hat OpenShift Container Platform cluster

OpenStack Team  
rhos-docs@redhat.com

## Legal Notice

Copyright © 2024 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

## Abstract

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

## Table of Contents

<b>PROVIDING FEEDBACK ON RED HAT DOCUMENTATION</b> .....	<b>4</b>
<b>CHAPTER 1. INSTALLING AND PREPARING THE OPERATORS</b> .....	<b>5</b>
1.1. PREREQUISITES	5
1.2. INSTALLING THE OPENSTACK OPERATOR	5
<b>CHAPTER 2. PREPARING RED HAT OPENSIFT CONTAINER PLATFORM FOR RED HAT OPENSTACK SERVICES ON OPENSIFT</b> .....	<b>9</b>
2.1. CONFIGURING RED HAT OPENSIFT CONTAINER PLATFORM NODES FOR A RED HAT OPENSTACK PLATFORM DEPLOYMENT	9
2.2. PROVIDING SECURE ACCESS TO THE RED HAT OPENSTACK SERVICES ON OPENSIFT SERVICES	9
2.3. DEFAULT RED HAT OPENSTACK PLATFORM NETWORKS	11
2.4. PREPARING RHOCN FOR RHOSO NETWORK ISOLATION	12
2.5. CONFIGURING THE DATA PLANE NETWORK	19
2.6. CREATING A STORAGE CLASS	21
<b>CHAPTER 3. CREATING THE CONTROL PLANE</b> .....	<b>22</b>
3.1. PREREQUISITES	22
3.2. CREATING THE CONTROL PLANE	22
3.3. EXAMPLE OPENSTACKCONTROLPLANE CR FOR A CORE CONTROL PLANE	30
3.4. ADDING THE BARE METAL PROVISIONING SERVICE (IRONIC) TO THE CONTROL PLANE	36
3.5. ADDING COMPUTE CELLS TO THE CONTROL PLANE	38
3.6. ENABLING THE DASHBOARD SERVICE (HORIZON) INTERFACE	41
3.7. ADDITIONAL RESOURCES	42
<b>CHAPTER 4. CREATING THE DATA PLANE</b> .....	<b>43</b>
4.1. PREREQUISITES	43
4.2. CREATING THE SSH KEY SECRETS	44
4.3. CREATING A SET OF DATA PLANE NODES	45
4.4. DATA PLANE SERVICES	53
4.4.1. Creating a custom service	55
4.4.2. Configuring a node set for a Compute feature or workload	57
4.4.3. Building a custom ansible-runner image	58
4.5. DEPLOYING THE DATA PLANE	59
4.6. OPENSTACKDATAPLANENODESET CR PROPERTIES	60
4.7. EXAMPLE OPENSTACKDATAPLANENODESET CR FOR PRE-PROVISIONED NODES	62
4.8. EXAMPLE OPENSTACKDATAPLANENODESET CR FOR BARE-METAL NODES	65
4.9. DATA PLANE CONDITIONS AND STATES	67
4.10. PROVISIONING BARE-METAL DATA PLANE NODES	69
4.11. TROUBLESHOOTING DATA PLANE CREATION AND DEPLOYMENT	70
<b>CHAPTER 5. CUSTOMIZING RED HAT OPENSTACK ON OPENSIFT OBSERVABILITY</b> .....	<b>72</b>
5.1. CONFIGURING RED HAT OPENSTACK ON OPENSIFT OBSERVABILITY	72
<b>CHAPTER 6. ADDING CUSTOM TLS CERTIFICATES FOR RED HAT OPENSTACK SERVICES ON OPENSIFT</b>	<b>75</b>
6.1. UPDATING THE CONTROL PLANE WITH CUSTOM CERTIFICATES FOR PUBLIC SERVICES	75
<b>CHAPTER 7. ACCESSING THE RHOSO CLOUD</b> .....	<b>78</b>
7.1. ACCESSING THE OPENSTACKCLIENT POD	78
7.2. ACCESSING THE DASHBOARD SERVICE (HORIZON) INTERFACE	78
<b>CHAPTER 8. MONITORING HIGH AVAILABILITY SERVICES</b> .....	<b>80</b>
8.1. RHOSO GALERA CLUSTERS	80

8.1.1. Monitoring Galera startup	81
8.2. RHOSO RABBITMQ CLUSTERS	83
8.2.1. Monitoring the RabbitMQ operator's startup	83
8.3. RHOSO MEMCACHED CLUSTERS	84
8.3.1. Monitoring memached startup	84
8.4. LISTING RHOSO CONTROL PLANE SERVICES PODS	85
8.5. LISTING THE RHOSO HIGH AVAILABILITY OPERATORS	85
8.6. RETRIEVING INFORMATION ABOUT AN OPERATOR'S CUSTOM RESOURCE	86
8.7. RETRIEVING INFORMATION ABOUT AN OPERATOR'S STATEFULSET	86
8.8. RETRIEVING MORE INFORMATION ABOUT AN OPERATOR'S STATEFULSET	87
8.8.1. Basic information about a service's statefulset	87
8.8.2. Information about actual container of a service's statefulset	88
8.8.3. Information about the volumes of a service's statefulset	89
8.8.4. Information about Event details of a service's statefulset	90
8.9. CHECKING THE STATUS OF THE CONTROL PLANE	90
8.9.1. Checking the status of a pod	91
8.10. EXPOSURE OF EACH SERVICE THROUGH CLUSTERIP OR LOADBALANCER	91
8.11. TESTING THE RESILIENCE OF THE CONTROL PLANE	92
8.11.1. The Taint-Based Evictions feature	93
<b>CHAPTER 9. COLLECTING DIAGNOSTIC INFORMATION FOR SUPPORT</b> .....	<b>94</b>
9.1. COLLECTING DATA ON THE RHOSO CONTROL PLANE	94
9.2. COLLECTING DATA ON THE RHOSO DATA PLANE NODES	95



# PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

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

## Providing documentation feedback in Jira

Use the [Create Issue](#) form to provide feedback on the documentation. The Jira issue will be created in the Red Hat OpenStack Services on OpenShift Jira project, where you can track the progress of your feedback.

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



### IMPORTANT

This content in this guide is available in this release as *Technology Preview*, and therefore is not fully supported by Red Hat. It should only be used for testing, and should not be deployed in a production environment. For more information, see [Technology Preview](#).



# CHAPTER 1. INSTALLING AND PREPARING THE OPERATORS

You install the Red Hat OpenStack Services on OpenShift (RHOSO) 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 RHOSO deployment. You must use the dedicated user, **stack**, with passwordless sudo rights and only ssh-key login enabled.

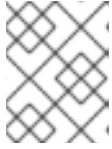
## 1.1. PREREQUISITES

- 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 RHOSO 18.0 Development Preview code cannot be located on a public repository.
- You are logged in to the RHOCP cluster 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 a **metalb** instance. For more information, see [Installing the MetalLB Operator](#) in the *RHOCP Networking* guide.
- You have installed the cert-manager Operator. For more information, see [cert-manager Operator for Red Hat OpenShift](#) in the *RHOCP Security and compliance* 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, 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>

## 1.2. INSTALLING THE OPENSTACK OPERATOR

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

- **openstack-operators**: Create this project for the Red Hat OpenStack Services on OpenShift (RHOSO) service Operators.
- **openstack**: Create this project for the deployed RHOSO 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 RHOSO catalog. For more information on **CatalogSource**, see [CatalogSource](#) in the *Operator Lifecycle Manager* documentation.
- An **OperatorGroup**, which defines the Operator group for RHOSO and restricts RHOSO to a target namespace. For more information on **OperatorGroup**, see [OperatorGroup](#) in the *Operator Lifecycle Manager* documentation.
- A **Subscription**, which tracks changes in the RHOSO 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 RHOSO commands.

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

**Procedure**

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

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

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

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

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

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

```
$ opm index add -u podman --pull-tool podman --tag <your_registry>:<port>/rhoso-podified-beta/openstack-operator-index:1.0.0 \
--bundles "registry.redhat.io/rhoso-podified-beta/openstack-operator-bundle:1.0.0,registry.redhat.io/rhoso-podified-beta/swift-operator-bundle:1.0.0,registry.redhat.io/rhoso-podified-beta/glance-operator-bundle:1.0.0,registry.redhat.io/rhoso-podified-beta/infra-operator-bundle:1.0.0,registry.redhat.io/rhoso-podified-beta/ironic-operator-bundle:1.0.0,registry.redhat.io/rhoso-podified-beta/keystone-operator-bundle:1.0.0,registry.redhat.io/rhoso-podified-beta/ovn-operator-bundle:1.0.0,registry.redhat.io/rhoso-podified-beta/placement-operator-bundle:1.0.0,registry.redhat.io/rhoso-podified-beta/telemetry-operator-bundle:1.0.0,registry.redhat.io/rhoso-podified-beta/heat-operator-bundle:1.0.0,registry.redhat.io/rhoso-podified-beta/cinder-operator-
```

```
bundle:1.0.0,registry.redhat.io/rhoso-podified-beta/manila-operator-
bundle:1.0.0,registry.redhat.io/rhoso-podified-beta/neutron-operator-
bundle:1.0.0,registry.redhat.io/rhoso-podified-beta/nova-operator-
bundle:1.0.0,registry.redhat.io/rhoso-edpm-beta/openstack-ansibleee-operator-
bundle:1.0.0,registry.redhat.io/rhoso-podified-beta/mariadb-operator-
bundle:1.0.0,registry.redhat.io/rhoso-podified-beta/openstack-baremetal-operator-
bundle:1.0.0,registry.redhat.io/rhoso-podified-beta/rabbitmq-cluster-operator-
bundle:1.0.0,registry.redhat.io/rhoso-podified-beta/horizon-operator-
bundle:1.0.0,registry.redhat.io/rhoso-podified-beta/octavia-operator-
bundle:1.0.0,registry.redhat.io/rhoso-podified-beta/barbican-operator-
bundle:1.0.0,registry.redhat.io/rhoso-podified-beta/designate-operator-bundle:1.0.0" --mode
semver
```

- Replace **<your\_registry>** with your registry account details. If you are using [quay.io](https://quay.io) or a private Quay instance as your registry, then include your account in **<your\_registry>**, for example, **quay.io/my\_quay\_account/**.

5. Push the index image to your private registry:

```
$ podman push <your_registry>[:<port>]/rhoso-podified-beta/openstack-operator-index:1.0.0
```

- Replace **<your\_registry>** with your registry account details. If you are using a registry other than [quay.io](https://quay.io) or a private Quay instance, then include the registry **<port>**.

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

7. 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: <your_registry>[:<port>]/rhoso-podified-beta/openstack-operator-index:1.0.0
```

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](#).



#### NOTE

You must create the secret that enables pull access to your container image registry in the **openstack-operators** namespace.

8. 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
```

9. 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
```

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

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

11. 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
```

12. 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 2. PREPARING RED HAT OPENSIFT CONTAINER PLATFORM FOR RED HAT OPENSTACK SERVICES ON OPENSIFT

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

### 2.1. CONFIGURING RED HAT OPENSIFT CONTAINER PLATFORM NODES FOR A RED HAT OPENSTACK PLATFORM DEPLOYMENT

Red Hat OpenStack Services on OpenShift (RHOSO) services run on Red Hat OpenShift Container Platform (RHOCP) worker nodes. By default, the OpenStack Operator deploys RHOSO services on any worker node. You can use node labels in your **OpenStackControlPlane** custom resource (CR) to specify which RHOCP nodes host the RHOSO services. By pinning some services to specific infrastructure nodes rather than running the services on all of your RHOCP worker nodes, you optimize the performance of your deployment. You can create labels for the RHOCP nodes, or you can 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 creating a volume from an image.
- The **cinder-backup** service has high memory, network, and CPU requirements.

#### Additional resources

- [Placing pods on specific nodes using node selectors](#)
- [Post-installation machine configuration tasks](#)
- [Node Feature Discovery Operator](#)

### 2.2. PROVIDING SECURE ACCESS TO THE RED HAT OPENSTACK SERVICES ON OPENSIFT SERVICES

You must create a **Secret** custom resource (CR) to provide secure access to the Red Hat OpenStack Services on OpenShift (RHOSO) 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>
  AodhPassword: <base64_password>
  AodhDatabasePassword: <base64_password>
  BarbicanDatabasePassword: <base64_password>
  BarbicanPassword: <base64_password>
  BarbicanSimpleCryptoKEK: <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 Orchestration service (heat) authentication that is at least 32 characters long.

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

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

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

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

## 2.3. DEFAULT RED HAT OPENSTACK PLATFORM NETWORKS

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

- **Control plane network:** This network is used by the DataPlane Operator for Ansible SSH access to deploy and connect to the data plane nodes from the Red Hat OpenShift Container Platform (RHOCP) environment.
- **External network:** (Optional) You can configure an external network if one is required for your environment. For example, you might create an external network for any of the following purposes:
  - To provide virtual machine instances with Internet access.
  - To create flat provider networks that are separate from the control plane.
  - To configure VLAN provider networks on a separate bridge from the control plane.
  - To provide access to virtual machine instances with floating IPs on a network other than the control plane network.
- **Internal API network:** This network is used for internal communication between Red Hat OpenStack Services on OpenShift (RHOSO) components.
- **Storage network:** This network is used for block storage, RBD, NFS, FC, and iSCSI.
- **Tenant (project) network:** This network is used for data communication between virtual machine instances within the cloud deployment.
- **Storage Management network:** (Optional) This network is used by storage components. For example, Red Hat Ceph Storage uses the Storage Management network in a hyperconverged infrastructure (HCI) environment as the **cluster\_network** to replicate data.



### NOTE

For more information on Red Hat Ceph Storage network configuration, see [Ceph network configuration](#) in the *Red Hat Ceph Storage Configuration Guide*.

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



## NOTE

By default, the control plane and external networks do not use VLANs. Networks that do not use VLANs must be placed on separate NICs. You can use a VLAN for the control plane network on new RHOSO deployments. You can also use the Native VLAN on a trunked interface as the non-VLAN network. For example, you can have the control plane and the internal API on one NIC, and the external network with no VLAN on a separate NIC.

Table 2.1. Default RHOSO networks

Network name	VLAN	CIDR	NetConfig allocation range	MetalLB IP Address Pool range	nad ipam range	OCP worker nncp range
<b>ctlplane</b>	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
<b>external</b>	n/a	10.0.0.0/24	10.0.0.100 - 10.0.0.250	n/a	n/a	
<b>internalapi</b>	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
<b>storage</b>	21	172.18.0.0/24	172.18.0.100 - 172.18.0.250	n/a	172.18.0.30 - 172.18.0.70	172.18.0.10 - 172.18.0.20
<b>tenant</b>	22	172.19.0.0/24	172.19.0.100 - 172.19.0.250	n/a	172.19.0.30 - 172.19.0.70	172.19.0.10 - 172.19.0.20
<b>storageMgmt</b>	23	172.20.0.0/24	172.20.0.100 - 172.20.0.250	n/a	172.20.0.30 - 172.20.0.70	172.20.0.10 - 172.20.0.20

## 2.4. PREPARING RHOCP FOR RHOSO NETWORK ISOLATION

The Red Hat OpenStack Services on OpenShift (RHOSO) 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 about 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": "192.168.122.0/24",
        "range_start": "192.168.122.30",
        "range_end": "192.168.122.70"
      }
    }
  ---
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 -n openstack
```

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 about 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
```

19. If your cluster is RHOCP 4.14 or later and it has OVNKubernetes as the network back end, then you must enable global forwarding so that MetalLB can work on a secondary network interface.

- a. Check the network back end used by your cluster:

```
$ oc get network.operator cluster --output=jsonpath='{.spec.defaultNetwork.type}'
```

- b. If the back end is OVNKubernetes, then run the following command to enable global IP forwarding:

```
$ oc patch network.operator cluster -p '{"spec":{"defaultNetwork":{"ovnKubernetesConfig":{"gatewayConfig":{"ipForwarding": "Global"}}}}}' --type=merge
```

## 2.5. 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 RHOSO networks, you must define a specification for each network. For information about the default RHOSO networks, see [Default Red Hat OpenStack Platform networks](#). The following example creates isolated networks for the data plane:

-

```
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
    - name: External
      dnsDomain: external.example.com
      subnets:
      - name: subnet1
        allocationRanges:
        - end: 10.0.0.250
          start: 10.0.0.100
        cidr: 10.0.0.0/24
        gateway: 10.0.0.1
    - name: Storage
      dnsDomain: storage.example.com
      subnets:
      - name: subnet1
        allocationRanges:
        - end: 172.18.0.250
          start: 172.18.0.100
        cidr: 172.18.0.0/24
        vlan: 21
    - name: StorageMgmt
      dnsDomain: storagemgmt.example.com
      subnets:
      - name: subnet1
        allocationRanges:
        - end: 172.20.0.250
          start: 172.20.0.100
        cidr: 172.20.0.0/24
        vlan: 23
    - name: Tenant
      dnsDomain: tenant.example.com
      subnets:
```



```
- name: subnet1
  allocationRanges:
  - end: 172.19.0.250
    start: 172.19.0.100
  cidr: 172.19.0.0/24
  vlan: 22
```

- 1 The name of the network, for example, **CtiPlane**.
- 2 The IPv4 subnet specification.
- 3 The name of the subnet, for example, **subnet1**.
- 4 The **NetConfig allocationRange**. The **allocationRange** must not overlap with the MetalLB **IpAddressPool** range and the IP address pool range.
- 5 The network VLAN. For information about the default RHOSO 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 dataplane network is created, view the **openstacknetconfig** resource:

```
$ oc get netconfig/openstacknetconfig -n openstack
```

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
```

## 2.6. CREATING A STORAGE CLASS

You must create a storage class for your Red Hat OpenShift Container Platform (RHOCP) cluster storage back end, to provide persistent volumes to Red Hat OpenStack Services on OpenShift (RHOSO) pods. Red Hat recommends that you use the Logical Volume Manager Storage (LVMS) storage class with RHOSO, although you can use other implementations, such as Container Storage Interface (CSI) or OpenShift Data Foundation (ODF). You specify this storage class as the cluster storage back end for the RHOSO deployment.

For more information about how to configure the LVMS storage class, see [Persistent storage using Logical Volume Manager Storage](#) in *Configuring and managing storage in OpenShift Container Platform*.

## CHAPTER 3. CREATING THE CONTROL PLANE

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

### 3.1. PREREQUISITES

- The RHOCP cluster is prepared for RHOSO network isolation. For more information, see [Preparing RHOCP for RHOSO network isolation](#).
- The OpenStack Operator (**openstack-operator**) is installed. For more information, see [Installing and preparing the Operators](#).
- You are logged on to a workstation that has access to the RHOCP cluster, as a user with **cluster-admin** privileges.

### 3.2. CREATING THE CONTROL PLANE

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

- Create the control plane.
- Enable the core, mandatory Red Hat OpenStack Services on OpenShift (RHOSO) services.

#### TIP

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

```
$ oc describe crd openstackcontrolplane
$ oc explain openstackcontrolplane.spec
```

#### Procedure

1. Create a file on your workstation named **openstack\_control\_plane.yaml** to define the **OpenStackControlPlane** CR:

```
apiVersion: core.openstack.org/v1beta1
kind: OpenStackControlPlane
metadata:
  name: openstack-control-plane
```

2. Specify the **Secret** CR you created to provide secure access to the RHOSO service pods in [Providing secure access to the Red Hat OpenStack Services on OpenShift services](#) :

```
apiVersion: core.openstack.org/v1beta1
kind: OpenStackControlPlane
metadata:
  name: openstack-control-plane
spec:
  secret: osp-secret
```

- Specify the **storageClass** you created for your Red Hat OpenShift Container Platform (RHOCP) cluster storage back end:

```

apiVersion: core.openstack.org/v1beta1
kind: OpenStackControlPlane
metadata:
  name: openstack-control-plane
spec:
  secret: osp-secret
  storageClass: your-RHOCP-storage-class

```



#### NOTE

For information about storage classes, see [Creating a storage class](#).

- Add configuration for the following core, mandatory services:

- Block Storage service (cinder):

```

cinder:
  apiOverride:
    route: {}
  template:
    databaseInstance: openstack
    secret: osp-secret
    cinderAPI:
      override:
        service:
          internal:
            metadata:
              annotations:
                metallb.universe.tf/address-pool: internalapi
                metallb.universe.tf/allow-shared-ip: internalapi
                metallb.universe.tf/loadBalancerIPs: 172.17.0.80
            spec:
              type: LoadBalancer
    cinderScheduler:
      replicas: 1
    cinderBackup:
      networkAttachments:
        - storage
      replicas: 0 # backend needs to be configured
    cinderVolumes:
      volume1:
        networkAttachments:
          - storage
        replicas: 0 # backend needs to be configured

```

- Compute service (nova):

```

nova:
  apiOverride:
    route: {}
  template:

```

```

apiServiceTemplate:
  override:
    service:
      internal:
        metadata:
          annotations:
            metallb.universe.tf/address-pool: internalapi
            metallb.universe.tf/allow-shared-ip: internalapi
            metallb.universe.tf/loadBalancerIPs: 172.17.0.80
        spec:
          type: LoadBalancer
metadataServiceTemplate:
  override:
    service:
      metadata:
        annotations:
          metallb.universe.tf/address-pool: internalapi
          metallb.universe.tf/allow-shared-ip: internalapi
          metallb.universe.tf/loadBalancerIPs: 172.17.0.80
        spec:
          type: LoadBalancer
secret: osp-secret

```



## NOTE

A full set of Compute services (nova) are deployed by default for each of the default cells, **cell0** and **cell1**: **nova-api**, **nova-metadata**, **nova-scheduler**, and **nova-conductor**. The **novncproxy** service is also enabled for **cell1** by default.

- A Galera cluster for use by all RHOSO services (**openstack**), and a Galera cluster for use by the Compute service for **cell1** (**openstack-cell1**):

```

galera:
  templates:
    openstack:
      storageRequest: 5000M
      secret: osp-secret
      replicas: 3
    openstack-cell1:
      storageRequest: 5000M
      secret: osp-secret
      replicas: 3

```

- Identity service (keystone)

```

keystone:
  apiOverride:
    route: {}
  template:
    override:
      service:
        internal:
          metadata:

```

```

annotations:
  metallb.universe.tf/address-pool: internalapi
  metallb.universe.tf/allow-shared-ip: internalapi
  metallb.universe.tf/loadBalancerIPs: 172.17.0.80
spec:
  type: LoadBalancer
databaseInstance: openstack
secret: osp-secret

```

- Image service (glance):

```

glance:
  apiOverrides:
    default:
      route: {}
  template:
    databaseInstance: openstack
    storageClass: ""
    storageRequest: 10G
    secret: osp-secret
    keystoneEndpoint: default
  glanceAPIs:
    default:
      type: single
      replicas: 1
      override:
        service:
          internal:
            metadata:
              annotations:
                metallb.universe.tf/address-pool: internalapi
                metallb.universe.tf/allow-shared-ip: internalapi
                metallb.universe.tf/loadBalancerIPs: 172.17.0.80
            spec:
              type: LoadBalancer
  networkAttachments:
    - storage

```



#### NOTE

You must configure a back end for the Image service. If you do not configure a back end for the Image service, then the service is deployed but not activated (**replicas: 0**). For information about configuring a back end for the Image service, see the [Configuring storage](#) guide.

- Key Management service (barbican):

```

barbican
apiOverride:
  route: {}
template:
  databaseInstance: openstack
  secret: osp-secret
  barbicanAPI:

```

```

replicas: 1
override:
  service:
    internal:
      metadata:
        annotations:
          metallb.universe.tf/address-pool: internalapi
          metallb.universe.tf/allow-shared-ip: internalapi
          metallb.universe.tf/loadBalancerIPs: 172.17.0.80
      spec:
        type: LoadBalancer
barbicanWorker:
  replicas: 1
barbicanKeystoneListener:
  replicas: 1

```

- Memcached:

```

memcached:
  templates:
    memcached:
      replicas: 3

```

- Networking service (neutron):

```

neutron:
  apiOverride:
    route: {}
  template:
    replicas: 3
    override:
      service:
        internal:
          metadata:
            annotations:
              metallb.universe.tf/address-pool: internalapi
              metallb.universe.tf/allow-shared-ip: internalapi
              metallb.universe.tf/loadBalancerIPs: 172.17.0.80
          spec:
            type: LoadBalancer
    databaseInstance: openstack
    secret: osp-secret
    networkAttachments:
      - internalapi

```

- Object Storage service (swift):

```

swift:
  enabled: true
  proxyOverride:
    route: {}
  template:
    swiftProxy:
      networkAttachments:
        - storage

```

```

override:
  service:
    internal:
      metadata:
        annotations:
          metallb.universe.tf/address-pool: internalapi
          metallb.universe.tf/allow-shared-ip: internalapi
          metallb.universe.tf/loadBalancerIPs: 172.17.0.80
      spec:
        type: LoadBalancer
      replicas: 2
    swiftRing:
      ringReplicas: 3
    swiftStorage:
      networkAttachments:
      - storage
      replicas: 3
      storageClass: local-storage
      storageRequest: 10Gi

```

- OVN:

```

ovn:
  template:
    ovnDBCluster:
      ovndbcluster-nb:
        replicas: 3
        dbType: NB
        storageRequest: 10G
        networkAttachment: internalapi
      ovndbcluster-sb:
        dbType: SB
        storageRequest: 10G
        networkAttachment: internalapi
    ovnNorthd:
      networkAttachment: internalapi
    ovnController:
      networkAttachment: tenant
      nicMappings:
        <network_name: nic_name>

```

- Replace **<network\_name>** with the name of the network your gateway is on.
- Replace **<nic\_name>** with the name of the NIC connecting to the gateway network.
- Optional: Add additional **<network\_name>:<nic\_name>** pairs under `nicMappings` as required.

- Placement service (placement):

```

placement:
  apiOverride:
    route: {}
  template:
    override:
      service:

```

```

internal:
  metadata:
    annotations:
      metallb.universe.tf/address-pool: internalapi
      metallb.universe.tf/allow-shared-ip: internalapi
      metallb.universe.tf/loadBalancerIPs: 172.17.0.80
  spec:
    type: LoadBalancer
databaseInstance: openstack
secret: osp-secret

```

- RabbitMQ:

```

rabbitmq:
  templates:
    rabbitmq:
      replicas: 3
      override:
        service:
          metadata:
            annotations:
              metallb.universe.tf/address-pool: internalapi
              metallb.universe.tf/loadBalancerIPs: 172.17.0.85
          spec:
            type: LoadBalancer
    rabbitmq-cell1:
      replicas: 3
      override:
        service:
          metadata:
            annotations:
              metallb.universe.tf/address-pool: internalapi
              metallb.universe.tf/loadBalancerIPs: 172.17.0.86
          spec:
            type: LoadBalancer

```

- Telemetry service (ceilometer, prometheus):

```

telemetry:
  enabled: true
  template:
    metricStorage:
      enabled: true
    monitoringStack:
      alertingEnabled: true
      scrapeInterval: 30s
    storage:
      strategy: persistent
      retention: 24h
      persistent:
        pvcStorageRequest: 20G
  autoscaling:
    enabled: false
  aodh:
    passwordSelectors:

```



```

databaseUser: aodh
databaseInstance: openstack
memcachedInstance: memcached
secret: osp-secret
heatInstance: heat
ceilometer:
  enabled: true
  secret: osp-secret
logging:
  enabled: false
  network: internalapi
  ipaddr: <ip_address>

```

- Replace **<ip\_address>** with the IP address for your environment.

5. Create the control plane:

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

6. 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-galera-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.

7. Optional: 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.

### 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.metakube.org |
+-----+-----+-----+
```

- 3. Exit the **OpenStackClient** pod:

```
$ exit
```

### 3.3. EXAMPLE OPENSTACKCONTROLPLANE CR FOR A CORE CONTROL PLANE

The following example **OpenStackControlPlane** CR is a complete core control plane configuration that includes all the key services that must always be enabled for a successful deployment.

```
apiVersion: core.openstack.org/v1beta1
kind: OpenStackControlPlane
metadata:
  name: openstack-control-plane
spec:
  secret: osp-secret
  storageClass: your-RHOCP-storage-class 1

  cinder: 2
    apiOverride:
      route: {}
    template:
      databaseInstance: openstack
      secret: osp-secret
      cinderAPI:
        override:
          service:
            internal:
              metadata:
                annotations:
                  metallb.universe.tf/address-pool: internalapi
                  metallb.universe.tf/allow-shared-ip: internalapi
                  metallb.universe.tf/loadBalancerIPs: 172.17.0.80
              spec:
                type: LoadBalancer
      cinderScheduler:
        replicas: 1
      cinderBackup: 3
        networkAttachments:
          - storage
        replicas: 0 # backend needs to be configured
      cinderVolumes: 4
        volume1:
          networkAttachments: 5
            - storage
          replicas: 0 # backend needs to be configured
  nova: 6
    apiOverride: 7
      route: {}
    template:
      apiServiceTemplate:
        override:
```

```

service:
  internal:
    metadata:
      annotations:
        metallb.universe.tf/address-pool: internalapi 8
        metallb.universe.tf/allow-shared-ip: internalapi
        metallb.universe.tf/loadBalancerIPs: 172.17.0.80 9
    spec:
      type: LoadBalancer
metadataServiceTemplate:
  override:
    service:
      metadata:
        annotations:
          metallb.universe.tf/address-pool: internalapi
          metallb.universe.tf/allow-shared-ip: internalapi
          metallb.universe.tf/loadBalancerIPs: 172.17.0.80
      spec:
        type: LoadBalancer
  secret: osp-secret
galera:
  templates:
    openstack:
      storageRequest: 5000M
      secret: osp-secret
      replicas: 3
    openstack-cell1:
      storageRequest: 5000M
      secret: osp-secret
      replicas: 3
keystone:
  apiOverride:
    route: {}
  template:
    override:
      service:
        internal:
          metadata:
            annotations:
              metallb.universe.tf/address-pool: internalapi
              metallb.universe.tf/allow-shared-ip: internalapi
              metallb.universe.tf/loadBalancerIPs: 172.17.0.80
          spec:
            type: LoadBalancer
      databaseInstance: openstack
      secret: osp-secret
glance:
  apiOverrides:
    default:
      route: {}
  template:
    databaseInstance: openstack
    storageClass: ""
    storageRequest: 10G
    secret: osp-secret
    keystoneEndpoint: default

```

```

glanceAPIs:
  default:
    type: single
    replicas: 1
    override:
      service:
        internal:
          metadata:
            annotations:
              metallb.universe.tf/address-pool: internalapi
              metallb.universe.tf/allow-shared-ip: internalapi
              metallb.universe.tf/loadBalancerIPs: 172.17.0.80
          spec:
            type: LoadBalancer
    networkAttachments:
      - storage
barbican
  apiOverride:
    route: {}
  template:
    databaseInstance: openstack
    secret: osp-secret
  barbicanAPI:
    replicas: 1
    override:
      service:
        internal:
          metadata:
            annotations:
              metallb.universe.tf/address-pool: internalapi
              metallb.universe.tf/allow-shared-ip: internalapi
              metallb.universe.tf/loadBalancerIPs: 172.17.0.80
          spec:
            type: LoadBalancer
  barbicanWorker:
    replicas: 1
  barbicanKeystoneListener:
    replicas: 1
memcached:
  templates:
    memcached:
      replicas: 3
neutron:
  apiOverride:
    route: {} 10
  template:
    replicas: 3
    override:
      service:
        internal:
          metadata:
            annotations:
              metallb.universe.tf/address-pool: internalapi
              metallb.universe.tf/allow-shared-ip: internalapi
              metallb.universe.tf/loadBalancerIPs: 172.17.0.80
          spec:

```

```

    type: LoadBalancer
  databaseInstance: openstack
  secret: osp-secret
  networkAttachments:
  - internalapi
swift:
  enabled: true
  proxyOverride:
    route: {}
  template:
    swiftProxy:
      networkAttachments:
      - storage
    override:
      service:
        internal:
          metadata:
            annotations:
              metallb.universe.tf/address-pool: internalapi
              metallb.universe.tf/allow-shared-ip: internalapi
              metallb.universe.tf/loadBalancerIPs: 172.17.0.80
          spec:
            type: LoadBalancer
        replicas: 2
    swiftRing:
      ringReplicas: 3
    swiftStorage:
      networkAttachments:
      - storage
      replicas: 3
      storageClass: local-storage
      storageRequest: 10Gi
ovn:
  template:
    ovnDBCluster:
      ovndbcluster-nb:
        replicas: 3
        dbType: NB
        storageRequest: 10G
        networkAttachment: internalapi
      ovndbcluster-sb:
        dbType: SB
        storageRequest: 10G
        networkAttachment: internalapi
    ovnNorthd:
      networkAttachment: internalapi
    ovnController:
      networkAttachment: tenant
      nicMappings:
      <network_name: nic_name>
  placement:
    apiOverride:
      route: {}
  template:
    override:
      service:

```

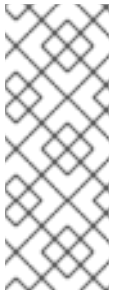
```

internal:
  metadata:
    annotations:
      metallb.universe.tf/address-pool: internalapi
      metallb.universe.tf/allow-shared-ip: internalapi
      metallb.universe.tf/loadBalancerIPs: 172.17.0.80
  spec:
    type: LoadBalancer
databaseInstance: openstack
secret: osp-secret
rabbitmq: 11
  templates:
    rabbitmq:
      replicas: 3
      override:
        service:
          metadata:
            annotations:
              metallb.universe.tf/address-pool: internalapi
              metallb.universe.tf/loadBalancerIPs: 172.17.0.85
          spec:
            type: LoadBalancer
    rabbitmq-cell1:
      replicas: 3
      override:
        service:
          metadata:
            annotations:
              metallb.universe.tf/address-pool: internalapi
              metallb.universe.tf/loadBalancerIPs: 172.17.0.86
          spec:
            type: LoadBalancer
telemetry:
  enabled: true
  template:
    metricStorage:
      enabled: true
    monitoringStack:
      alertingEnabled: true
      scrapInterval: 30s
    storage:
      strategy: persistent
      retention: 24h
    persistent:
      pvcStorageRequest: 20G
autoscaling:
  enabled: false
aodh:
  passwordSelectors:
    databaseUser: aodh
  databaseInstance: openstack
  memcachedInstance: memcached
  secret: osp-secret
  heatInstance: heat
ceilometer:
  enabled: true

```

```
secret: osp-secret
logging:
  enabled: false
network: internalapi
ipaddr: <ip_address>
```

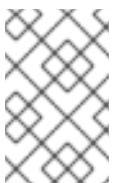
- 1 The storage class that you created for your Red Hat OpenShift Container Platform (RHOCP) cluster storage back end.
- 2 Service-specific parameters for the Block Storage service (cinder).
- 3 The Block Storage service back end. For more information on configuring storage services, see the [Configuring storage](#) guide.
- 4 The Block Storage service configuration. For more information on configuring storage services, see the [Configuring storage](#) guide.
- 5 The list of networks that each service pod is directly attached to, specified by using the **NetworkAttachmentDefinition** resource names. 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 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.

- 6 Service-specific parameters for the Compute service (nova).
- 7 Service API route definition. You can customize the service route by using route-specific annotations. For more information, see [Route-specific annotations](#) in the RHOCP *Networking* guide. Set **route:** to **{}** to apply the default route template.
- 8 The internal service API endpoint registered as a MetalLB service with the **IPAddressPool internalapi**.
- 9 The virtual IP (VIP) address for the service. The IP is shared with other services by default.
- 10 Customized service API route definition. For more information, see [Route-specific annotations](#) in the RHOCP *Networking* guide.
- 11 The RabbitMQ instances exposed to an isolated network.



#### 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.

### 3.4. ADDING THE BARE METAL PROVISIONING SERVICE (IRONIC) TO THE CONTROL PLANE

If you want your cloud users to be able to launch bare-metal instances, you must configure the control plane with the Bare Metal Provisioning service (ironic).

#### Procedure

1. Open your **OpenStackControlPlane** custom resource (CR) file, **openstack\_control\_plane.yaml**, on your workstation.
2. Add the following **cellTemplates** configuration to the **nova** service configuration:

```
nova:
  apiOverride:
    route: {}
  template:
    ...
  secret: osp-secret
  cellTemplates:
    cell0:
      cellDatabaseUser: nova_cell0
      hasAPIAccess: true
    cell1:
      cellDatabaseUser: nova_cell1
      cellDatabaseInstance: openstack-cell1
      cellMessageBusInstance: rabbitmq-cell1
      hasAPIAccess: true
      novaComputeTemplates:
        compute-ironic: 1
        computeDriver: ironic.IronicDriver
```

- 1** The name of the Compute service. The name has a limit of 20 characters, and must contain only lowercase alphanumeric characters and the - symbol.

3. Create the network that the **ironic** service pod attaches to, for example, **baremetal**. For more information about how to create an isolated network, see [Preparing RHOCF for RHOSO network isolation](#).
4. Enable and configure the **ironic** service:

```
spec:
  ...
  ironic:
    enabled: true
    template:
      rpcTransport: oslo
      databaseInstance: openstack
      ironicAPI:
        replicas: 1
      override:
        service:
          internal:
            metadata:
```



```

    annotations:
      metallb.universe.tf/address-pool: internalapi
      metallb.universe.tf/allow-shared-ip: internalapi
      metallb.universe.tf/loadBalancerIPs: 172.17.0.80
    spec:
      type: LoadBalancer
  ironicConductors:
  - replicas: 1
    storageRequest: 10G
    networkAttachments:
    - baremetal 1
    provisionNetwork: baremetal
    customServiceConfig: |
      [neutron]
      cleaning_network = provisioning
      provisioning_network = provisioning
      rescuing_network = provisioning
  ironicInspector:
    replicas: 0
    networkAttachments:
    - baremetal
    inspectionNetwork: baremetal
  ironicNeutronAgent:
    replicas: 1
    secret: osp-secret

```

**1** The **NetworkAttachmentDefinition** CR for your **baremetal** network.

- Update the control plane:

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

- 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-ironic 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.

- 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.

### 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 ironic
+-----+-----+-----+
| Service Name | Interface | URL |
+-----+-----+-----+
| ironic      | internal  | http://ironic-internal.openstack.svc:9292 |
| ironic      | public    | http://ironic-public-openstack.apps.ostest.test.metalkube.org |
+-----+-----+-----+
```

3. Exit the **openstackclient** pod:

```
$ exit
```

### 3.5. ADDING COMPUTE CELLS TO THE CONTROL PLANE

You can use cells to divide Compute nodes in large deployments into groups. Each cell has a dedicated message queue, runs standalone copies of the cell-specific Compute services and databases, and stores instance metadata in a database dedicated to instances in that cell.

By default, the control plane creates two cells:

- **cell0**: The controller cell that manages global components and services, such as the Compute scheduler and the global conductor. This cell also contains a dedicated database to store information about instances that failed to be scheduled to a Compute node. You cannot connect Compute nodes to this cell.
- **cell1**: The default cell that Compute nodes are connected to when you don't create and configure additional cells.

You can add cells to your Red Hat OpenStack Services on OpenShift (RHOSO) environment when you create your control plane or at any time afterwards.

#### Procedure

1. Open your **OpenStackControlPlane** custom resource (CR) file, **openstack\_control\_plane.yaml**, on your workstation.
2. Create a database server for each new cell that you want to add to your RHOSO environment:

```
apiVersion: core.openstack.org/v1beta1
kind: OpenStackControlPlane
metadata:
  name: openstack-galera-network-isolation
spec:
  secret: osp-secret
  storageClass: local-storage
  ...
galera:
  enabled: true
```

```

templates:
  openstack: 1
    storageRequest: 5G
    secret: cell0-secret
    replicas: 1
  openstack-cell1: 2
    storageRequest: 5G
    secret: cell1-secret
    replicas: 1
  openstack-cell2: 3
    storageRequest: 5G
    secret: cell2-secret
    replicas: 1

```

- 1 The database used by most of the RHOSO services, including the Compute services **nova-api** and **nova-scheduler**, and **cell0**.
- 2 The database to be used by **cell1**.
- 3 The database to be used by **cell2**.

3. Create a message bus with unique IPs for the load balancer for each new cell that you want to add to your RHOSO environment:

```

apiVersion: core.openstack.org/v1beta1
kind: OpenStackControlPlane
metadata:
  name: openstack-galera-network-isolation
spec:
  secret: osp-secret
  storageClass: local-storage
  ...
  rabbitmq:
    templates:
      rabbitmq: 1
        override:
          service:
            metadata:
              annotations:
                metallb.universe.tf/address-pool: internalapi
                metallb.universe.tf/loadBalancerIPs: 172.17.0.85
            spec:
              type: LoadBalancer
      rabbitmq-cell1: 2
        override:
          service:
            metadata:
              annotations:
                metallb.universe.tf/address-pool: internalapi
                metallb.universe.tf/loadBalancerIPs: 172.17.0.86
            spec:
              type: LoadBalancer
      rabbitmq-cell2: 3
        override:

```

```

service:
  metadata:
    annotations:
      metallb.universe.tf/address-pool: internalapi
      metallb.universe.tf/loadBalancerIPs: 172.17.0.87
  spec:
    type: LoadBalancer

```

- 1 The message bus used by most of the RHOSO services, including the Compute services **nova-api** and **nova-scheduler**, and **cell0**.
- 2 The message bus to be used by **cell1**.
- 3 The message bus to be used by **cell2**.

4. Add the new cells to the **cellTemplates** configuration in the **nova** service configuration:

```

nova:
  apiOverride:
    route: {}
  template:
    ...
  secret: osp-secret
  cellTemplates:
    cell0:
      cellDatabaseUser: nova_cell0
      hasAPIAccess: true
    cell1:
      cellDatabaseInstance: openstack-cell1
      cellDatabaseUser: nova_cell1
      cellMessageBusInstance: rabbitmq-cell1
      hasAPIAccess: true
    cell2: 1
      cellDatabaseInstance: openstack-cell2
      cellDatabaseUser: nova_cell1
      cellMessageBusInstance: rabbitmq-cell2
      hasAPIAccess: true

```

- 1 The name of the new Compute cell. The name has a limit of 20 characters, and must contain only lowercase alphanumeric characters and the - symbol. For more information about the properties you can configure for a cell, view the definition for the **Nova** CRD:

```
$ oc describe crd nova
```

5. Update the control plane:

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

6. 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-galera-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.

- Optional: Confirm that the control plane is deployed by reviewing the pods in the **openstack** namespace for each of the cells you created:

```
$ oc get pods -n openstack | grep cell2
nova-cell2-conductor-0      1/1   Running   2          5d20h
nova-cell2-novncproxy-0    1/1   Running   2          5d20h
openstack-cell2-galera-0   1/1   Running   2          5d20h
rabbitmq-cell2-server-0    1/1   Running   2          5d20h
```

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

- Optional: Confirm that the new cells are created:

```
$ oc exec -it nova-cell0-conductor-0 /bin/bash
# nova-manage cell_v2 list_cells
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| Name | UUID |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
Transport URL | Database Connection | Disabled |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| cell0 | 00000000-0000-0000-0000-000000000000 |
rabbit: | mysql+pymysql://nova_cell0:****@openstack/nova_cell0 | False || cell1 | c5bf5e35-
6677-40aa-80d0-33a440cac14e | rabbit://default_user_CuUVnXz-
PvgzXvPxypU:****@rabbitmq-cell1.openstack.svc:5672 |
mysql+pymysql://nova_cell1:****@openstack-cell1/nova_cell1 | False || cell2 | c5bf5e35-
6677-40aa-80d0-33a440cac14e | rabbit://default_user_CuUVnXz-
PvgzXvPxypU:****@rabbitmq-cell2.openstack.svc:5672 |
mysql+pymysql://nova_cell2:****@openstack-cell2/nova_cell2 | False |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
```

## 3.6. ENABLING THE DASHBOARD SERVICE (HORIZON) INTERFACE

You can enable the Dashboard service (horizon) interface for cloud user access to the cloud through a web browser.

### Procedure

- Obtain the **OpenStackControlPlane** CR name:

```
$ oc get openstackcontrolplanes
```

- 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-galera-network-isolation**.

3. Retrieve the Dashboard service endpoint URL:

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

Use this URL to access the Horizon interface.

## Verification

1. 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
```

2. Open a web browser.
3. Enter the Dashboard endpoint URL.
4. Log in to the dashboard with your username and password.

## 3.7. 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 4. CREATING THE DATA PLANE

The Red Hat OpenStack Services on OpenShift (RHOSO) data plane consists of RHEL 9.4 nodes. Use the **OpenStackDataPlaneNodeSet** custom resource definition (CRD) to create the custom resources (CRs) that define the nodes and the layout of the data plane. You can use pre-provisioned nodes, or provision bare-metal nodes as part of the data plane creation and deployment process.

To create and deploy a data plane, you must perform the following tasks:

1. Create a **Secret** CR for Ansible to use to execute commands on the data plane nodes.
2. Create the **OpenStackDataPlaneNodeSet** CRs that define the nodes and layout of the data plane.
3. Create the **OpenStackDataPlaneDeployment** CRs that trigger the Ansible execution to deploy and configure software.

### 4.1. PREREQUISITES

- A functional control plane, created with the OpenStack Operator. For more information, see [Creating the control plane](#).
- Pre-provisioned nodes must be configured with an SSH public key in the **\$HOME/.ssh/authorized\_keys** file for a user with passwordless **sudo** privileges.
- For bare-metal nodes that are provisioned when creating the **OpenStackDataPlaneNodeSet** resource:
  - Cluster Baremetal Operator (CBO) is installed and configured for provisioning. For more information, see [Provisioning bare-metal data plane nodes](#).
  - A **BareMetalHost** CR is registered and inspected for each bare-metal data plane node. Each bare-metal node must be in the **Available** state after inspection. For more information about configuring bare-metal nodes, see [Bare metal configuration](#) in the *RHOCP Postinstallation configuration* guide.
  - Update the **edpm-hardened-uefi-rhel9:18.0.0** image in the operator bundles from the default version 10 to version 9:

```
$ oc edit csv openstack-operator.v0.1.3
apiVersion: operators.coreos.com/v1alpha1
kind: ClusterServiceVersion
...
value: registry.redhat.io/rhoso-beta/edpm-hardened-uefi-rhel9:18.0.0-9
value: edpm-hardened-uefi.qcow2
...
```

```
$ oc edit csv openstack-baremetal-operator.v0.1.3
apiVersion: operators.coreos.com/v1alpha1
kind: ClusterServiceVersion
...
value: registry.redhat.io/rhoso-beta/edpm-hardened-uefi-rhel9:18.0.0-9
value: edpm-hardened-uefi.qcow2
...
```

- You are logged on to a workstation that has access to the RHOCV cluster as a user with **cluster-admin** privileges.

## 4.2. CREATING THE SSH KEY SECRETS

You must generate SSH keys and create an SSH key **Secret** custom resource (CR) for each key to enable the following functionality:

- You must generate an SSH key to enable Ansible to manage the RHEL nodes on the data plane. Ansible executes commands with this user and key.
- You must generate an SSH key to enable migration of instances between Compute nodes.

The **Secret** CRs are used by the data plane nodes to enable secure access between nodes.

### Procedure

1. Create the SSH key pair for Ansible:

```
$ KEY_FILE_NAME=<key_file_name>
$ 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 **Secret** CR for Ansible and apply it to the cluster:

```
$ SECRET_NAME=<secret_name>
$ 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 \
  -n openstack \
  -o yaml | oc apply -f-
```

- Replace **<secret\_name>** with the name you want to use for the **Secret** resource.
- Include the **--from-file=authorized\_keys** option for bare-metal nodes that must be provisioned when creating the data plane.

3. Create the SSH key pair for instance migration:

```
$ ssh-keygen -f ./id -t ecdsa-sha2-nistp521 -N "
```

4. Create the **Secret** CR for migration and apply it to the cluster:

```
$ oc create secret generic nova-migration-ssh-key \
  --from-file=ssh-privatekey=id \
  --from-file=ssh-publickey=id.pub \
  -n openstack \
  -o yaml | oc apply -f-
```

5. Verify that the **Secret** CRs are created:

-



```
$ oc describe secret $SECRET_NAME
```

### 4.3. CREATING A SET OF DATA PLANE NODES

You use the **OpenStackDataPlaneNodeSet** CRD to define the data plane and the data plane nodes. An **OpenStackDataPlaneNodeSet** custom resource (CR) represents a set of nodes of the same type that have similar configuration, comparable to the concept of a "role" in a director-deployed Red Hat OpenStack Services on OpenShift (RHOSO) environment.

Create an **OpenStackDataPlaneNodeSet** CR for each logical grouping of nodes in your data plane, for example, nodes grouped by hardware, location, or networking. You can define as many node sets as necessary for your deployment. Each node can be included in only one **OpenStackDataPlaneNodeSet** CR. Each node set can be connected to only one Compute cell. By default, node sets are connected to **cell1**. If your control plane includes additional Compute cells, you must specify the cell to which the node set is connected.

#### Procedure

1. Copy the sample **OpenStackDataPlaneNodeSet** CR and save it to a file named **openstack-edpm.yaml** on your workstation:

```
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneNodeSet
metadata:
  name: openstack-edpm
spec:
  tlsEnabled: true
  env:
    - name: ANSIBLE_FORCE_COLOR
      value: "True"
  services:
    - bootstrap
    - download-cache
    - configure-network
    - validate-network
    - install-os
    - configure-os
    - ssh-known-hosts
    - run-os
    - reboot-os
    - install-certs
    - ovn
    - neutron-metadata
    - libvirt
    - nova
    - telemetry
  preProvisioned: true
  networkAttachments:
    - ctlplane
  nodes:
    edpm-compute-0:
      hostName: edpm-compute-0
      ansible:
        ansibleHost: 192.168.122.100
      networks:
```

```

- name: ctlplane
  subnetName: subnet1
  defaultRoute: true
  fixedIP: 192.168.122.100
- name: internalapi
  subnetName: subnet1
- name: storage
  subnetName: subnet1
- name: tenant
  subnetName: subnet1
nodeTemplate:
  ansibleSSHPrivateKeySecret: dataplane-ansible-ssh-private-key-secret
  ansible:
    ansibleVarsFrom:
      - prefix: edpm_
        configMapRef:
          name: network-config-template
      - prefix: neutron_
        configMapRef:
          name: neutron-edpm
    # CHANGE ME -- see https://access.redhat.com/solutions/253273
    # - prefix: subscription_manager_
    #   secretRef:
    #     name: subscription-manager
    # - prefix: registry_
    #   secretRef:
    #     name: redhat-registry
  ansibleVars:
    # CHANGE ME -- see https://access.redhat.com/solutions/253273
    # edpm_bootstrap_command: |
    #   subscription-manager register --username {{ subscription_manager_username }} -
    # -password {{ subscription_manager_password }}
    #   podman login -u {{ registry_username }} -p {{ registry_password }}
  registry.redhat.io
    edpm_nodes_validation_validate_controllers_icmp: false
    edpm_nodes_validation_validate_gateway_icmp: false
    gather_facts: false
    enable_debug: false
    # edpm firewall, change the allowed CIDR if needed
    edpm_sshd_allowed_ranges: ['192.168.122.0/24']

```

2. The sample **OpenStackDataPlaneNodeSet** CR is connected to **cell1** by default. If you added additional Compute cells to the control plane and you want to connect the node set to one of the other cells, then you must create a custom service for the node set that includes the **Secret** CR for the cell:

- a. Create a custom **nova** service that includes the **Secret** CR for the cell to connect to:

```

apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneService
metadata:
  name: nova-cell-custom
spec:
  label: dataplane-deployment-custom-service
  playbook: osp.edpm.nova

```

```
...
secrets:
  - nova-cell2-compute-config 1
```

- 1** The **Secret** CR generated by the control plane for the cell.

For information about how to create a custom service, see [Creating a custom service](#).

- b. Replace the **nova** service in your **OpenStackDataPlaneNodeSet** CR with your custom **nova** service:

```
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneNodeSet
metadata:
  name: openstack-edpm-ipam
spec:
  services:
    - download-cache
    - bootstrap
    - configure-network
    - validate-network
    - install-os
    - configure-os
    - run-os
    - ovn
    - neutron-metadata
    - libvirt
    - nova-cell-custom
    - telemetry
```



#### NOTE

Do not change the order of the default services.

- c. If you are deploying multiple nodesets, ensure that you add the **ssh-known-hosts** service to exactly one node set of the deployment because this service is deployed globally:

```
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneNodeSet
metadata:
  name: openstack-edpm-ipam
spec:
  services:
    - download-cache
    - bootstrap
    - configure-network
    - validate-network
    - install-os
    - configure-os
    - ssh-known-hosts
    - run-os
    - ovn
    - neutron-metadata
```

- libvirt
- nova-cell-custom
- telemetry

3. Update the **Secret** to the SSH key secret that you created to enable Ansible to connect to the data plane nodes:

```
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneNodeSet
metadata:
  name: openstack-edpm-ipam
spec:
  nodeTemplate:
    ansibleSSHPrivateKeySecret: <secret-key>
```

- Replace **<secret-key>** with the name of the SSH key **Secret** CR you created in [Creating the SSH key secrets](#), for example, **dataplane-ansible-ssh-private-key-secret**.
4. Optional: Configure the node set for a Compute feature or workload. For more information, see [Configuring a node set for a Compute feature or workload](#).
  5. Optional: The sample **OpenStackDataPlaneNodeSet** CR that you copied includes the minimum common configuration required for a set of nodes in this group under the **nodeTemplate** section. Each node in this **OpenStackDataPlaneNodeSet** inherits this configuration. You can edit the configured values as required, and you can add additional configuration. For information about the properties you can use to configure common node attributes, see [OpenStackDataPlaneNodeSet CR properties](#).

For example **OpenStackDataPlaneNodeSet** CR **nodeTemplate** definitions, see [Example OpenStackDataPlaneNodeSet CR for pre-provisioned nodes](#) or [Example OpenStackDataPlaneNodeSet CR for bare-metal nodes](#).

6. Optional: The sample **OpenStackDataPlaneNodeSet** CR you copied applies the single NIC VLANs network configuration by default to the data plane nodes. You can edit the template that is applied. For example, to configure the data plane for multiple NICs, copy the contents of the **roles/edpm\_network\_config/templates/multiple\_nics/multiple\_nics.j2** file and add it to your **openstack-edpm.yaml** file:

```
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneNodeSet
metadata:
  name: openstack-edpm-ipam
spec:
  ...
  nodeTemplate:
    ...
    ansible:
      ansibleVars:
        edpm_network_config_template: |
          ---
          network_config:
            - type: interface
              name: nic1
              mtu: {{ ctlplane_mtu }}
              dns_servers: {{ ctlplane_dns_nameservers }}
              domain: {{ dns_search_domains }}
```

```

routes: {{ ctlplane_host_routes }}
use_dhcp: false
addresses:
- ip_netmask: {{ ctlplane_ip }}/{{ ctlplane_subnet_cidr }}
{% for network in networks_all if network not in networks_skip_config %}
{% if network not in ["External", "Tenant"] and network in role_networks %}
- type: interface
  name: nic{{ loop.index + 1 }}
  mtu: {{ lookup('vars', networks_lower[network] ~ '_mtu') }}
  use_dhcp: false
  addresses:
  - ip_netmask:
    {{ lookup('vars', networks_lower[network] ~ '_ip') }}/{{ lookup('vars',
networks_lower[network] ~ '_cidr') }}
    routes: {{ lookup('vars', networks_lower[network] ~ '_host_routes') }}
    {% elif network in role_networks or 'external_bridge' in role_tags %}
  - type: ovs_bridge
    {% if network == 'External' %}
    name: {{ neutron_physical_bridge_name }}
    {% else %}
    name: {{ 'br-' ~ networks_lower[network] }}
    {% endif %}
    mtu: {{ lookup('vars', networks_lower[network] ~ '_mtu') }}
    dns_servers: {{ ctlplane_dns_nameservers }}
    use_dhcp: false
    addresses:
    - ip_netmask:
      {{ lookup('vars', networks_lower[network] ~ '_ip') }}/{{ lookup('vars',
networks_lower[network] ~ '_cidr') }}
      routes: {{ lookup('vars', networks_lower[network] ~ '_host_routes') }}
    members:
    - type: interface
      name: nic{{ loop.index + 1 }}
      mtu: {{ lookup('vars', networks_lower[network] ~ '_mtu') }}
      use_dhcp: false
      primary: true
    {% endif %}
  {% endfor %}

```

You can copy a sample template from <https://github.com/openstack-k8s-operators/dataplane-operator/tree/main/config/samples/nic-config-samples>.

7. Register the operating system of the nodes that are not registered to the Red Hat Customer Portal, and enable repositories for your nodes:

```

apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneNodeSet
metadata:
  name: openstack-edpm-ipam
spec:
  preProvisioned: True
  ...
nodeTemplate:
  ansible:
    ...
  ansibleVars:

```

```

edpm_bootstrap_command: |
  subscription-manager register --username <subscription_manager_username> --
password <subscription_manager_password>
  subscription-manager release --set=9.4
  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=fast-datapath-for-rhel-9-x86_64-rpms --enable=rhoso-18-beta-for-rhel-9-
x86_64-rpms --enable=rhceph-7-tools-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>. For information about how to log into **registry.redhat.io**, see <https://access.redhat.com/RegistryAuthentication#creating-registry-service-accounts-6>.

8. If your nodes are not pre-provisioned, you must configure the bare-metal template:

```

apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneNodeSet
metadata:
  name: openstack-edpm-ipam
spec:
  preProvisioned: false
  baremetalSetTemplate:
    deploymentSSHSecret: dataplane-ansible-ssh-private-key-secret
    bmhNamespace: openshift-machine-api 1
    cloudUserName: <ansible_ssh_user>
    bmhLabelSelector:
      app: openstack 2
    ctlplaneInterface: enp1s0
    dnsSearchDomains:
      - osptest.openstack.org

```

- 1** The namespace defined in the corresponding **BareMetalHost** CR for the node.
- 2** The label defined in the corresponding **BareMetalHost** CR for the node.

For more information about provisioning bare-metal nodes, see [Provisioning bare-metal data plane nodes](#).



## NOTE

The BMO manages **BareMetalHost** CRs in the **openshift-machine-api** namespace by default. If the BMO must also manage **BareMetalHost** CRs in other namespaces, you must update the **Provisioning** CR to watch all namespaces:

```
$ oc patch provisioning provisioning-configuration --type merge -p '{"spec": {"watchAllNamespaces": true }}'
```

9. Optional: The sample **OpenStackDataPlaneNodeSet** CR that you copied includes default node configurations in the **nodes** section. If necessary, you can add additional nodes and edit the configured values. For example, to add node-specific Ansible variables that customize the node, add the following configuration to your **openstack-edpm.yaml** file:

```
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneNodeSet
metadata:
  name: openstack-edpm-ipam
spec:
  ...
  nodes:
    edpm-compute-0: ❶
      hostName: edpm-compute-0
      networks: ❷
        - name: ctplane
          subnetName: subnet1
          defaultRoute: true
          fixedIP: 192.168.122.100 ❸
        - name: internalapi
          subnetName: subnet1
          fixedIP: 172.17.0.100
        - name: storage
          subnetName: subnet1
          fixedIP: 172.18.0.100
        - name: tenant
          subnetName: subnet1
          fixedIP: 172.19.0.100
      ansible:
        ansibleHost: 192.168.122.100
        ansibleUser: cloud-admin
        ansibleVars: ❹
          fqdn_internal_api: edpm-compute-0.example.com
    edpm-compute-1:
      hostName: edpm-compute-1
      networks:
        - name: ctplane
          subnetName: subnet1
          defaultRoute: true
          fixedIP: 192.168.122.101
        - name: internalapi
          subnetName: subnet1
        - name: storage
          subnetName: subnet1
```

```

- name: tenant
  subnetName: subnet1
ansible:
  ansibleHost: 192.168.122.101
  ansibleUser: cloud-admin
  ansibleVars:
    fqdn_internal_api: edpm-compute-1.example.com

```

- 1 The node definition reference, for example, **edpm-compute-0**. Each node in the node set must have a node definition.
- 2 Defines the IPAM and the DNS records for the node.
- 3 Defines the predictable IP addresses for each network.
- 4 Node-specific Ansible variables that customize the node.



## NOTE

- Nodes defined within the **nodes** section can configure the same Ansible variables that are configured in the **nodeTemplate** section. Where an Ansible variable is configured for both a specific node and within the **nodeTemplate** section, the node-specific values override those from the **nodeTemplate** section.
- You do not need to replicate all the **nodeTemplate** Ansible variables for a node to override the default and set some node-specific values. You only need to configure the Ansible variables you want to override for the node.

For information about the properties you can use to configure node attributes, see [OpenStackDataPlaneNodeSet CR properties](#). For example [OpenStackDataPlaneNodeSet CR nodes](#) definitions, see [Example OpenStackDataPlaneNodeSet CR for pre-provisioned nodes](#) or [Example OpenStackDataPlaneNodeSet CR for bare-metal nodes](#).

10. Optional: Customize the container images used by the **edpm-ansible** roles. The following example shows the default images:

```

spec:
  ...
  nodeTemplate:
    ...
    ansible:
      ...
    ansibleVars:
      edpm_iscsid_image: "registry.redhat.io/rhoso-beta/openstack-iscsid-rhel9:18.0"
      edpm_logrotate_cron_image: "registry.redhat.io/rhoso-beta/openstack-cron-rhel9:18.0"
      edpm_ovn_controller_agent_image: "registry.redhat.io/rhoso-beta/openstack-frr-rhel9:18.0"
      edpm_ovn_metadata_agent_image: "registry.redhat.io/rhoso-beta/openstack-neutron-metadata-agent-ovn-rhel9:18.0"
      edpm_frr_image: "registry.redhat.io/rhoso-beta/openstack-frr-rhel9:18.0"
      edpm_ovn_bgp_agent_image: "registry.redhat.io/rhoso-beta/openstack-ovn-bgp-agent-rhel9:18.0"
      telemetry_node_exporter_image: "redhat.registry.io/prometheus/node-exporter:v1.5.0"
      edpm_libvirt_image: "registry.redhat.io/rhoso-beta/openstack-nova-libvirt-rhel9:18.0"

```



```

    edpm_nova_compute_image: "registry.redhat.io/rhoso-beta/openstack-nova-compute-
    rhel9:18.0"
    edpm_neutron_sriov_image: "registry.redhat.io/rhoso-beta/openstack-neutron-sriov-
    agent-rhel9:18.0"
    edpm_multipathd_image: "registry.redhat.io/rhoso-beta/openstack-multipathd-
    rhel9:18.0"

```

11. Save the **openstack-edpm.yaml** definition file.

12. Create the data plane resources:

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

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

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

14. Verify that the **Secret** resource was created for the node set:

```
$ oc get secret | grep openstack-edpm-ipam
dataplanenodeset-openstack-edpm-ipam Opaque 1 3m50s
```

15. Verify the services were created:

```
$ oc get openstackdataplaneservice
NAME          AGE
configure-network 6d7h
configure-os    6d6h
install-os     6d6h
run-os         6d6h
validate-network 6d6h
ovn            6d6h
libvirt        6d6h
nova           6d6h
telemetry      6d6h
```

## 4.4. DATA PLANE SERVICES

A data plane service is an Ansible execution that manages the installation, configuration, and execution of a software deployment on data plane nodes. Each service is a resource instance of the **OpenStackDataPlaneService** CRD.

The DataPlane Operator provides core services that are deployed by default on data plane nodes. If the **services** field is omitted from the **OpenStackDataPlaneNodeSet** specification, then the following services are applied by default in the following order:

```

services:
- configure-network
- validate-network
- install-os
- configure-os

```

- run-os
- ovn
- libvirt
- nova
- telemetry

The DataPlane Operator also includes the following services that are not enabled by default:

Service	Description
<b>ceph-client</b>	<p>Include this service to configure data plane nodes as clients of a Red Hat Ceph Storage server. Include between the <b>install-os</b> and <b>configure-os</b> services. The <b>OpenStackDataPlaneNodeSet</b> CR must include the following configuration to access the Red Hat Ceph Storage secrets:</p> <pre> apiVersion: dataplane.openstack.org/v1beta1 kind: OpenStackDataPlaneNodeSet spec:   ...   nodeTemplate:     extraMounts:       - extraVolType: Ceph         volumes:           - name: ceph             secret:               secretName: ceph-conf-files         mounts:           - name: ceph             mountPath: "/etc/ceph"             readOnly: true </pre>
<b>ceph-hci-pre</b>	<p>Include this service to prepare data plane nodes to host Red Hat Ceph Storage in an HCI configuration. For more information, see <a href="#">assembly_configuring-a-hyperconverged-infrastructure-environment[Configuring a Hyperconverged Infrastructure environment]</a>.</p>
<b>neutron-dhcp</b>	<p>Include this service to run a Neutron DHCP agent on the data plane nodes.</p>
<b>neutron-metadata</b>	<p>Include this service to run the Neutron OVN Metadata agent on the data plane nodes. This agent is required to provide metadata services to the Compute nodes.</p>
<b>neutron-ovn</b>	<p>Include this service to run the Neutron OVN agent on the data plane nodes. This agent is required to provide QoS to hardware offloaded ports on the Compute nodes.</p>
<b>neutron-sriov</b>	<p>Include this service to run a Neutron SR-IOV NIC agent on the data plane nodes.</p>

For more information about the available default services, see <https://github.com/openstack-k8s-operators/dataplane-operator/tree/main/config/services>.

You can enable and disable services for an **OpenStackDataPlaneNodeSet** resource.



#### NOTE

Do not change the order of the default service deployments.

You can use the **OpenStackDataPlaneService** CRD to create custom services that you can deploy on your data plane nodes. You add your custom services to the default list of services where the service must be executed. For more information, see [Creating a custom service](#).

You can view the details of a service by viewing the YAML representation of the resource:

```
$ oc get openstackdataplaneservice configure-network -o yaml
```

### 4.4.1. Creating a custom service

You can use the **OpenStackDataPlaneService** CRD to create custom services to deploy on your data plane nodes.



#### NOTE

Do not create a custom service with the same name as one of the default services. If a custom service name matches a default service name, the default service values overwrite the custom service values during **OpenStackDataPlaneNodeSet** reconciliation.

You specify the Ansible execution for your service with either an Ansible playbook or by including the free-form play contents directly in the **spec** section of the service.



#### NOTE

You cannot use both an Ansible playbook and an Ansible play in the same service.

#### Procedure

1. Create an **OpenStackDataPlaneService** CR and save it to a YAML file on your workstation, for example **custom-service.yaml**:

```
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneService
metadata:
  name: custom-service
spec:
  label: dataplane-deployment-custom-service
```

2. Specify the Ansible commands to create the custom service, by referencing an Ansible playbook or by including the Ansible play in the **spec**:
  - Specify the Ansible playbook to use:

```
apiVersion: dataplane.openstack.org/v1beta1
```

```
kind: OpenStackDataPlaneService
metadata:
  name: custom-service
spec:
  label: dataplane-deployment-custom-service
  playbook: osp.edpm.configure_os
```

For information about how to create an Ansible playbook, see [Creating a playbook](#).

- Specify the Ansible play as a string that uses Ansible playbook syntax:

```
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneService
metadata:
  name: custom-service
spec:
  label: dataplane-deployment-custom-service
  play: |
    - hosts: all
      tasks:
        - name: Hello World!
          shell: "echo Hello World!"
          register: output
        - name: Show output
          debug:
            msg: "{{ output.stdout }}"
        - name: Hello World role
          import_role: hello_world
```

3. Optional: To override the default container image used by the **ansible-runner** execution environment with a custom image that uses additional Ansible content for a custom service, build and include a custom **ansible-runner** image. For information, see [Building a custom ansible-runner image](#).
4. Optional: Designate and configure a node set for a Compute feature or workload. For more information, see [Configuring a node set for a Compute feature or workload](#).
5. Optional: Specify the names of **Secret** resources to use to pass secrets into the **OpenStackAnsibleEE** job:

```
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneService
metadata:
  name: custom-service
spec:
  ...
  play: |
    ...
  secrets:
    - hello-world-secret-0
    - hello-world-secret-1
```

A mount is created for each **secret** in the **OpenStackAnsibleEE** pod with a filename that matches the **secret** value. The mounts are created under `/var/lib/openstack/configs/<service name>`.

6. Create the custom service:

```
$ oc apply -f custom-service.yaml
```

7. Verify that the custom service is created:

```
$ oc get openstackdataplaneservice <custom_service_name> -o yaml
```

#### 4.4.2. Configuring a node set for a Compute feature or workload

You can designate a node set for a particular Compute feature or workload. To designate and configure a node set for a feature, complete the following tasks:

1. Create a **ConfigMap** CR to configure the Compute nodes.
2. Create a custom **nova** service for the feature that runs the **osp.edpm.nova** playbook.
3. Include the **ConfigMap** CR in the custom **nova** service.

##### Procedure

1. Create a **ConfigMap** CR to configure the Compute nodes. For example, to enable CPU pinning on the Compute nodes, create the following **ConfigMap** object:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: nova-cpu-pinning-configmap
  namespace: openstack
data:
  25-nova-cpu-pinning.conf: |
    [compute]
    cpu_shared_set = 2,6
    cpu_dedicated_set = 1,3,5,7
```

When the service is deployed it adds the configuration to **etc/nova/nova.conf.d/** in the **nova\_compute** container.

For more information on creating **ConfigMap** objects, see [Creating and using config maps](#).

##### TIP

You can use a **Secret** to create the custom configuration instead if the configuration includes sensitive information, such as passwords or certificates that are required for certification.

2. Create a custom **nova** service for the feature. For information about how to create a custom service, see [Creating a custom service](#).
3. Add the **ConfigMap** CR to the custom **nova** service:

```
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneService
metadata:
  name: nova-cpu-pinning-service
```

```
spec:
  label: dataplane-deployment-custom-service
  playbook: osp.edpm.nova
  configMaps:
    - nova-cpu-pinning-configmap
```

- Specify the **Secret** CR for the cell that the node set that runs this service connects to:

```
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneService
metadata:
  name: nova-cpu-pinning-service
spec:
  label: dataplane-deployment-custom-service
  playbook: osp.edpm.nova
  configMaps:
    - nova-cpu-pinning-configmap
  secrets:
    - nova-migration-ssh-key
    - nova-cell1-compute-config
```

#### 4.4.3. Building a custom ansible-runner image

You can override the default container image used by the **ansible-runner** execution environment with your own custom image when you need additional Ansible content for a custom service.

##### Procedure

- Create a **Containerfile** that adds the custom content to the default image:

```
FROM quay.io/openstack-k8s-operators/openstack-ansibleee-runner:latest
COPY my_custom_role /usr/share/ansible/roles/my_custom_role
```

- Build and push the image to a container registry:

```
$ podman build -t quay.io/example_user/my_custom_image:latest .
$ podman push quay.io/example_user/my_custom_role:latest
```

- Specify your new container image as the image that the **ansible-runner** execution environment must use to add the additional Ansible content that your custom service requires, such as Ansible roles or modules:

```
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneService
metadata:
  name: custom-service
spec:
  label: dataplane-deployment-custom-service
  openStackAnsibleEERunnerImage: quay.io/openstack-k8s-operators/openstack-ansibleee-
runner:latest 1
  play: |
```

- 1 Your container image that the **ansible-runner** execution environment uses to execute Ansible.

## 4.5. DEPLOYING THE DATA PLANE

You use the **OpenStackDataPlaneDeployment** CRD to configure the services on the data plane nodes and deploy the data plane. Create an **OpenStackDataPlaneDeployment** custom resource (CR) that deploys each of your **OpenStackDataPlaneNodeSet** CRs.

### Procedure

1. Create a file called **libvirt-secret.yaml** and add contents similar to the following:

```
apiVersion: v1
data:
  LibvirtPassword: cGFZc3dvcmQ=
kind: Secret
metadata:
  name: libvirt-secret
  namespace: openstack
type: Opaque
```

2. Create the secret:

```
oc apply -f libvirt-secret.yaml
```

3. Copy the sample **OpenStackDataPlaneDeployment** CR from [https://github.com/openstack-k8s-operators/dataplane-operator/blob/main/config/samples/dataplane\\_v1beta1\\_openstackdataplanedeployment.yaml](https://github.com/openstack-k8s-operators/dataplane-operator/blob/main/config/samples/dataplane_v1beta1_openstackdataplanedeployment.yaml) and save it to a file named **openstack-edpm-deploy.yaml** on your workstation.
4. Optional: Update **nodeSets** to include all the **OpenStackDataPlaneNodeSet** CRs that you want to deploy:

```
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneDeployment
metadata:
  name: openstack-edpm-ipam
spec:
  nodeSets:
    - openstack-edpm-ipam
    - <nodeSet_name>
    - ...
    - <nodeSet_name>
```

- Replace **<nodeSet\_name>** with the names of the **OpenStackDataPlaneNodeSet** CRs that you want to include in your data plane deployment.
5. Save the **openstack-edpm-deploy.yaml** deployment file.
  6. Deploy the data plane:

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

You can view the Ansible logs while the deployment executes:

```
$ oc get pod -l app=openstackansibleee -w
$ oc logs -l app=openstackansibleee -f --max-log-requests 10
```

- Verify that the data plane is deployed:

```
$ oc get openstackdataplanedeployment
NAME          STATUS MESSAGE
openstack-edpm-ipam True   Setup Complete
```

```
$ oc get openstackdataplanenodeset
NAME          STATUS MESSAGE
openstack-edpm-ipam True   NodeSet Ready
```

For information on the meaning of the returned status, see [Data plane conditions and states](#).

If the status indicates that the data plane has not been deployed, then troubleshoot the deployment. For information, see [Troubleshooting the data plane creation and deployment](#).

- Map the Compute nodes to the Compute cell that they are connected to:

```
$ oc rsh nova-cell0-conductor-0 nova-manage cell_v2 discover_hosts --verbose
```

If you did not create additional cells, this command maps the Compute nodes to **cell1**.

- Access the remote shell for the **openstackclient** pod and verify that the deployed Compute nodes are visible on the control plane:

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

## 4.6. OPENSTACKDATAPLANENODESET CR PROPERTIES

The following tables detail the **OpenStackDataPlaneNodeSet** CR properties you can configure.

**Table 4.1. nodeTemplate properties**

Properties	Description
<b>ansibleSSHPrivateKeySecret</b>	<p>Name of the private SSH key secret that contains the private SSH key for connecting to nodes.</p> <p>Secret name format: <code>Secret.data.ssh-privatekey</code></p> <p>For more information, see <a href="#">Creating an SSH authentication secret</a>.</p> <p>Default: <b>dataplane-ansible-ssh-private-key-secret</b></p>
<b>managementNetwork</b>	<p>Name of the network to use for management (SSH/Ansible).</p> <p>Default: <b>ctlplane</b></p>




Properties	Description
<b>networks</b>	Network definitions for the <b>OpenStackDataPlaneNodeSet</b> .
<b>ansible</b>	Ansible configuration options. For more information, see <a href="#">ansible properties</a> .
<b>extraMounts</b>	The files to mount into an Ansible Execution Pod.
<b>userData</b>	UserData configuration.
<b>networkData</b>	NetworkData configuration for the <b>OpenStackDataPlaneNodeSet</b> .

Table 4.2. nodes properties

Properties	Description
<b>ansible</b>	Ansible configuration options. For more information, see <a href="#">ansible properties</a> .
<b>extraMounts</b>	The files to mount into an Ansible Execution Pod.
<b>hostName</b>	The node name.
<b>managementNetwork</b>	Name of the network to use for management (SSH/Ansible).
<b>networkData</b>	NetworkData configuration for the node.
<b>networks</b>	Instance networks.
<b>userData</b>	Node-specific user data.

Table 4.3. ansible properties

Properties	Description
<b>ansibleUser</b>	The user associated with the secret you created in <a href="#">Creating the SSH key secrets</a> . Default: <b>rhel-user</b>
<b>ansibleHost</b>	SSH host for the Ansible connection.
<b>ansiblePort</b>	SSH port for the Ansible connection.

Properties	Description
<p><b>ansibleVars</b></p>	<p>The Ansible variables that customize the set of nodes. You can use this property to configure any custom Ansible variable, including the Ansible variables available for each <b>edpm-ansible</b> role. For a complete list of Ansible variables by role, see the <a href="#">edpm-ansible documentation</a>.</p> <div data-bbox="687 443 798 792" style="border: 1px solid #ccc; padding: 5px; width: fit-content;">  </div> <p><b>NOTE</b></p> <p>The <b>ansibleVars</b> parameters that you can configure for an <b>OpenStackDataPlaneNodeSet</b> CR are determined by the services defined for the <b>OpenStackDataPlaneNodeSet</b>. The <b>OpenStackDataPlaneService</b> CRs call the Ansible playbooks from the <a href="#">edpm-ansible playbook collection</a>, which include the roles that are executed as part of the data plane service.</p>

## 4.7. EXAMPLE OPENSTACKDATAPLANENODESET CR FOR PRE-PROVISIONED NODES

The following example **OpenStackDataPlaneNodeSet** CR creates a set of generic Compute nodes with some node-specific configuration.

```

apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneNodeSet
metadata:
  name: openstack-edpm-ipam
spec:
  env: 1
    - name: ANSIBLE_FORCE_COLOR
      value: "True"
  preProvisioned: true 2
  services: 3
    - bootstrap
    - configure-network
    - validate-network
    - install-os
    - configure-os
    - run-os
    - install-certs
    - ovn
    - neutron-metadata
    - libvirt
    - ssh-known-hosts
    - nova
    - telemetry
  nodes:
    edpm-compute-0: 4
      hostname: edpm-compute-0

```

```

ansible:
  ansibleHost: 192.168.122.100
  ansibleUser: cloud-admin
networks:
- name: CtlPlane
  subnetName: subnet1
  defaultRoute: true
  fixedIP: 192.168.122.100
- name: InternalApi
  subnetName: subnet1
- name: Storage
  subnetName: subnet1
- name: Tenant
  subnetName: subnet1
edpm-compute-1:
  hostName: edpm-compute-1
  ansible:
    ansibleHost: 192.168.122.101
    ansibleUser: cloud-admin
  networks:
- name: CtlPlane
  subnetName: subnet1
  defaultRoute: true
  fixedIP: 192.168.122.101
- name: InternalApi
  subnetName: subnet1
- name: Storage
  subnetName: subnet1
- name: Tenant
  subnetName: subnet1
networkAttachments: 5
- ctlplane
nodeTemplate: 6
  ansibleSSHPrivateKeySecret: dataplane-ansible-ssh-private-key-secret 7
  managementNetwork: ctlplane
  ansible:
    ansibleUser: cloud-admin 8
    ansiblePort: 22
    ansibleVars: 9
      edpm_bootstrap_release_version_package: "rhoso-release"
      edpm_neutron_dhcp_image: "registry.redhat.io/rhoso-beta/openstack-neutron-dhcp-agent-
rhel9:18.0.0"
    service_net_map:
      nova_api_network: internal_api
      nova_libvirt_network: internal_api
    timesync_ntp_servers:
      - hostname: pool.ntp.org
    # edpm_network_config
    # Default nic config template for a EDPM compute node
    # These vars are edpm_network_config role vars
    edpm_network_config_template: | 10
      ---
      {% set mtu_list = [ctlplane_mtu] %}
      {% for network in role_networks %}
      {{ mtu_list.append(lookup('vars', networks_lower[network] ~ '_mtu')) }}

```

```

{%%- endfor %}
{% set min_viable_mtu = mtu_list | max %}
network_config:
- type: ovs_bridge
  name: {{ neutron_physical_bridge_name }}
  mtu: {{ min_viable_mtu }}
  use_dhcp: false
  dns_servers: {{ ctlplane_dns_nameservers }}
  domain: {{ dns_search_domains }}
  addresses:
  - ip_netmask: {{ ctlplane_ip }}/{{ ctlplane_subnet_cidr }}
  routes: {{ ctlplane_host_routes }}
  members:
  - type: interface
    name: nic1
    mtu: {{ min_viable_mtu }}
    # force the MAC address of the bridge to this interface
    primary: true
{% for network in role_networks %}
- type: vlan
  mtu: {{ lookup('vars', networks_lower[network] ~ '_mtu') }}
  vlan_id: {{ lookup('vars', networks_lower[network] ~ '_vlan_id') }}
  addresses:
  - ip_netmask:
    {{ lookup('vars', networks_lower[network] ~ '_ip') }}/{{ lookup('vars',
networks_lower[network] ~ '_cidr') }}
    routes: {{ lookup('vars', networks_lower[network] ~ '_host_routes') }}
{% endfor %}
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
# edpm_nodes_validation
edpm_nodes_validation_validate_controllers_icmp: false
edpm_nodes_validation_validate_gateway_icmp: false
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 Optional: A list of environment variables to pass to the pod.
- 2 Specify if the nodes in this set are pre-provisioned, or if they are provisioned when creating the resource.
- 3 The services that are deployed on the data plane nodes in this **OpenStackDataPlaneNodeSet** CR.
- 4 The node definition reference, for example, **edpm-compute-0**. Each node in the node set must have a node definition.
- 5 The networks the **ansible-runner** connects to, specified as a list of **netattach** resource names.

- 6 The common configuration to apply to all nodes in this set of nodes.
- 7 The name of the secret that you created in [Creating the SSH key secrets](#) .
- 8 The user associated with the secret you created in [Creating the SSH key secrets](#) .
- 9 The Ansible variables that customize the set of nodes. For a list of Ansible variables that you can use, see <https://openstack-k8s-operators.github.io/edpm-ansible/>.
- 10 The network configuration template to apply to nodes in the set. For sample templates, see <https://github.com/openstack-k8s-operators/dataplane-operator/tree/main/config/samples/nic-config-samples>.

## 4.8. EXAMPLE OPENSTACKDATAPLANENODESET CR FOR BARE-METAL NODES

The following example **OpenStackDataPlaneNodeSet** CR creates a set of generic Compute nodes with some node-specific configuration.

```

apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneNodeSet
metadata:
  name: openstack-edpm-ipam
spec:
  env: 1
    - name: ANSIBLE_FORCE_COLOR
      value: "True"
  services: 2
    - download-cache
    - bootstrap
    - configure-network
    - validate-network
    - install-os
    - configure-os
    - run-os
    - install-certs
    - ovn
    - neutron-metadata
    - libvirt
    - ssh-known-hosts
    - nova
    - telemetry
  baremetalSetTemplate: 3
    bmhLabelSelector:
      app: openstack
    ctlplaneInterface: enp1s0
    cloudUserName: cloud-admin
  nodes:
    edpm-compute-0: 4
      hostName: edpm-compute-0
  networkAttachments: 5
    - ctlplane
  nodeTemplate: 6

```

```

ansibleSSHPublicKeySecret: dataplane-ansible-ssh-public-key-secret 7
networks: 8
- name: CtlPlane
  subnetName: subnet1
  defaultRoute: true
- name: InternalApi
  subnetName: subnet1
- name: Storage
  subnetName: subnet1
- name: Tenant
  subnetName: subnet1
managementNetwork: ctlplane
ansible:
  ansibleVars: 9
    edpm_bootstrap_release_version_package: "rhoso-release"
    edpm_neutron_dhcp_image: "registry.redhat.io/rhoso-beta/openstack-neutron-dhcp-agent-
rhel9:18.0.0"
    edpm_network_config_template: | 10
      ---
      {% set mtu_list = [ctlplane_mtu] %}
      {% for network in role_networks %}
      {{ mtu_list.append(lookup('vars', networks_lower[network] ~ '_mtu')) }}
      {%- endfor %}
      {% set min_viable_mtu = mtu_list | max %}
      network_config:
      - type: ovs_bridge
        name: {{ neutron_physical_bridge_name }}
        mtu: {{ min_viable_mtu }}
        use_dhcp: false
        dns_servers: {{ ctlplane_dns_nameservers }}
        domain: {{ dns_search_domains }}
        addresses:
        - ip_netmask: {{ ctlplane_ip }}/{{ ctlplane_subnet_cidr }}
          routes: {{ ctlplane_host_routes }}
          members:
          - type: interface
            name: nic1
            mtu: {{ min_viable_mtu }}
            # force the MAC address of the bridge to this interface
            primary: true
          {% for network in role_networks %}
          - type: vlan
            mtu: {{ lookup('vars', networks_lower[network] ~ '_mtu') }}
            vlan_id: {{ lookup('vars', networks_lower[network] ~ '_vlan_id') }}
            addresses:
            - ip_netmask:
                {{ lookup('vars', networks_lower[network] ~ '_ip') }}/{{ lookup('vars',
networks_lower[network] ~ '_cidr') }}
              routes: {{ lookup('vars', networks_lower[network] ~ '_host_routes') }}
          {% endfor %}
      # 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
      # edpm_nodes_validation
      edpm_nodes_validation_validate_controllers_icmp: false

```

```

edpm_nodes_validation_validate_gateway_icmp: false
gather_facts: false
enable_debug: false
# edpm firewall, change the allowed CIDR if needed
edpm_sshd_allowed_ranges: ['192.168.122.0/24']

```

- 1 Optional: A list of environment variables to pass to the pod.
- 2 The services that are deployed on the data plane nodes in this **OpenStackDataPlaneNodeSet** CR.
- 3 Configure the bare-metal template for bare-metal nodes that must be provisioned when creating the resource.
- 4 The node definition reference, for example, **edpm-compute-0**. Each node in the node set must have a node definition.
- 5 The networks the **ansibleee-runner** connects to, specified as a list of **netattach** resource names.
- 6 The common configuration to apply to all nodes in this set of nodes.
- 7 The name of the secret that you created in [Creating the SSH key secrets](#).
- 8 Networks for the bare-metal nodes.
- 9 The Ansible variables that customize the set of nodes. For a list of Ansible variables that you can use, see <https://openstack-k8s-operators.github.io/edpm-ansible/>.
- 10 The network configuration template to apply to nodes in the set. For sample templates, see [https://github.com/openstack-k8s-operators/edpm-ansible/tree/main/roles/edpm\\_network\\_config/templates](https://github.com/openstack-k8s-operators/edpm-ansible/tree/main/roles/edpm_network_config/templates).

## 4.9. DATA PLANE CONDITIONS AND STATES

Each data plane resource has a series of conditions within their **status** subresource that indicates the overall state of the resource, including its deployment progress.

For an **OpenStackDataPlaneNodeSet**, until an **OpenStackDataPlaneDeployment** has been started and finished successfully, the **Ready** condition is **False**. When the deployment succeeds, the **Ready** condition is set to **True**. A subsequent deployment sets the **Ready** condition to **False** until the deployment succeeds, when the **Ready** condition is set to **True**.

Table 4.4. **OpenStackDataPlaneNodeSet** CR conditions

Condition	Description
<b>Ready</b>	<ul style="list-style-type: none"> <li>● "True": The <b>OpenStackDataPlaneNodeSet</b> CR is successfully deployed.</li> <li>● "False": The deployment is not yet requested or has failed, or there are other failed conditions.</li> </ul>

Condition	Description
<b>SetupReady</b>	"True": All setup tasks for a resource are complete. Setup tasks include verifying the SSH key secret, verifying other fields on the resource, and creating the Ansible inventory for each resource. Each service-specific condition is set to "True" when that service completes deployment. You can check the service conditions to see which services have completed their deployment, or which services failed.
<b>DeploymentReady</b>	"True": The NodeSet has been successfully deployed.
<b>InputReady</b>	"True": The required inputs are available and ready.
<b>NodeSetDNSDataReady</b>	"True": DNSData resources are ready.
<b>NodeSetIPReservationReady</b>	"True": The IPSet resources are ready.
<b>NodeSetBaremetalProvisionReady</b>	"True": Bare-metal nodes are provisioned and ready.

Table 4.5. **OpenStackDataPlaneNodeSet** status fields

Status field	Description
<b>Deployed</b>	<ul style="list-style-type: none"> <li>• "True": The <b>OpenStackDataPlaneNodeSet</b> CR is successfully deployed.</li> <li>• "False": The deployment is not yet requested or has failed, or there are other failed conditions.</li> </ul>
<b>DNSClusterAddresses</b>	
<b>CtlplaneSearchDomain</b>	

Table 4.6. **OpenStackDataPlaneDeployment** CR conditions

Condition	Description
<b>Ready</b>	<ul style="list-style-type: none"> <li>• "True": The data plane is successfully deployed.</li> <li>• "False": The data plane deployment failed, or there are other failed conditions.</li> </ul>
<b>DeploymentReady</b>	"True": The data plane is successfully deployed.
<b>InputReady</b>	"True": The required inputs are available and ready.



Condition	Description
<b>&lt;NodeSet&gt; Deployment Ready</b>	"True": The deployment has succeeded for the named <b>NodeSet</b> , indicating all services for the <b>NodeSet</b> have succeeded.
<b>&lt;NodeSet&gt; &lt;Service&gt; Deployment Ready</b>	"True": The deployment has succeeded for the named <b>NodeSet</b> and <b>Service</b> . Each <b>&lt;NodeSet&gt; &lt;Service&gt; Deployment Ready</b> specific condition is set to "True" as that service completes successfully for the named <b>NodeSet</b> . Once all services are complete for a <b>NodeSet</b> , the <b>&lt;NodeSet&gt; Deployment Ready</b> condition is set to "True". The service conditions indicate which services have completed their deployment, or which services failed and for which <b>NodeSets</b> .

Table 4.7. OpenStackDataPlaneDeployment status fields

Status field	Description
<b>Deployed</b>	<ul style="list-style-type: none"> <li>• "True": The data plane is successfully deployed. All Services for all NodeSets have succeeded.</li> <li>• "False": The deployment is not yet requested or has failed, or there are other failed conditions.</li> </ul>

Table 4.8. OpenStackDataPlaneService CR conditions

Condition	Description
<b>Ready</b>	"True": The service has been created and is ready for use. "False": The service has failed to be created.

## 4.10. PROVISIONING BARE-METAL DATA PLANE NODES

Provisioning bare-metal nodes on the data plane is supported with the Red Hat OpenShift Container Platform (RHOCP) Cluster Baremetal Operator (CBO). The CBO deploys the components required to provision bare-metal nodes within the RHOCP cluster, including the Bare Metal Operator (BMO) and Ironic containers.

The BMO manages the available hosts on clusters and performs the following operations:

- Inspects node hardware details and reports them to the corresponding **BareMetalHost** CR. This includes information about CPUs, RAM, disks, and NICs.
- Provisions nodes with a specific image.
- Cleans node disk contents before and after provisioning.

The availability of the CBO depends on which of the following installation methods was used for the RHOCP cluster:

## Assisted Installer

You can enable CBO on clusters installed with the Assisted Installer, and you can manually add the provisioning network to the Assisted Installer cluster after installation.

## Installer-provisioned infrastructure

CBO is enabled by default on RHOCP clusters that are installed with the bare-metal installer-provisioned infrastructure. You can configure installer-provisioned clusters with a provisioning network to enable both virtual media and network boot installations. Alternatively, you can configure an installer-provisioned cluster without a provisioning network so that only virtual media provisioning is available. For more information about installer-provisioned clusters on bare metal, see [Deploying installer-provisioned clusters on bare metal](#).

## User-provisioned infrastructure

You can activate CBO on RHOCP clusters installed with user-provisioned infrastructure by creating a Provisioning CR. You cannot add a provisioning network to a user-provisioned cluster. For more information about how to create a Provisioning CR, see [Scaling a user-provisioned cluster with the Bare Metal Operator](#).

## 4.11. TROUBLESHOOTING DATA PLANE CREATION AND DEPLOYMENT

Each data plane deployment in the environment has associated services. Each of these services have a job condition message that matches to the current status of the AnsibleEE job executing for that service. This information can be used to troubleshoot deployments when services are not deploying or operating correctly.

### Procedure

1. Determine the name and status of all deployments:

```
$ oc get openstackdataplanedeployment
```

The following example output shows two deployments currently in progress:

```
$ oc get openstackdataplanedeployment

NAME                NODESETS                STATUS MESSAGE
openstack-edpm-ipam1 ["openstack-edpm"] False  Deployment in progress
openstack-edpm-ipam2 ["openstack-edpm"] False  Deployment in progress
```

2. Determine the name and status of all services and their job condition:

```
$ oc get openstackansibleee
```

The following example output shows all services and their job condition for all current deployments:

```
$ oc get openstackansibleee

NAME                                NETWORKATTACHMENTS STATUS MESSAGE
bootstrap-openstack-edpm            ["ctlplane"]      True  AnsibleExecutionJob complete
download-cache-openstack-edpm       ["ctlplane"]      False AnsibleExecutionJob is running
repo-setup-openstack-edpm           ["ctlplane"]      True  AnsibleExecutionJob complete
validate-network-another-osdpd      ["ctlplane"]      False AnsibleExecutionJob is running
```

3. Filter for the name and service for a specific deployment:

```
$ oc get openstackansibleee -l osdpd=<deployment_name>
```

- Replace **<deployment\_name>** with the name of the deployment to use to filter the services list.  
The following example filters the list to only show services and their job condition for the **openstack-edpm-ipam1** deployment:

```
$ oc get openstackansibleee -l osdpd=openstack-edpm-ipam1
```

NAME	NETWORKATTACHMENTS	STATUS	MESSAGE
bootstrap-openstack-edpm	["ctlplane"]	True	AnsibleExecutionJob complete
download-cache-openstack-edpm	["ctlplane"]	False	AnsibleExecutionJob is running
repo-setup-openstack-edpm	["ctlplane"]	True	AnsibleExecutionJob complete

### Job Condition Messages

AnsibleEE jobs have an associated condition message that indicates the current state of the service job. This condition message is displayed in the **MESSAGE** field of the **oc get openstackansibleee** command output. Jobs return one of the following conditions when queried:

- **AnsibleExecutionJob not started:** The job has not started.
- **AnsibleExecutionJob not found:** The job could not be found.
- **AnsibleExecutionJob is running:** The job is currently running.
- **AnsibleExecutionJob complete:** The job execution is complete.
- **AnsibleExecutionJob error occurred <error\_message>:** The job stopped executed unexpectedly. The **<error\_message>** is replaced with a specific error message.

To further investigate a service displaying a particular job condition message, use the command **oc logs job/<service>** to display the logs associated with that service. For example, to display the logs for the **repo-setup-openstack-edpm** service, use the command **oc logs job/repo-setup-openstack-edpm**.

## CHAPTER 5. CUSTOMIZING RED HAT OPENSTACK ON OPENSIFT OBSERVABILITY

Use observability with Red Hat OpenStack Services on OpenShift (RHOSO) to get insight into the metrics, logs, and alerts from your deployment.

The observability architecture in RHOSO is composed of services within OpenShift, as well as services on your Compute nodes that expose metrics, logs and alerts. You can use the OpenShift observability ecosystem for insight into the RHOSO environment. Additionally, you have access to the logging infrastructure for collecting, storing, and searching through logs. RHOSO services such as ceilometer and sg-core make metrics from your compute nodes and associated virtual infrastructure available to the OpenShift Observability framework.

### 5.1. CONFIGURING RED HAT OPENSTACK ON OPENSIFT OBSERVABILITY

The Telemetry service (ceilometer, prometheus) is enabled by default in a Red Hat OpenStack Services on OpenShift (RHOSO) deployment. You can configure observability by editing the `openstack_control_plane.yaml` CR file.

#### Prerequisites

- The Cluster Observability Operator is installed from OperatorHub. For more information, see [Installing the Cluster Observability Operator](#).
- Optional: If you plan to enable logging, the Cluster Logging Operator is installed from OperatorHub.
  - A LokiStack instance must be running. For more information, see [Configuring the LokiStack log store](#).
  - A ClusterLogging instance must be running. For more information, see [Configuring the logging collector](#).
  - The syslog receiver must be enabled. For more information, see [Forwarding logs using the syslog protocol](#).



#### NOTE

You do not need these Operators to expose and query OpenStack metrics in Prometheus format. If you do not disable ceilometer, then a Prometheus metrics exporter is created and exposed from inside the cluster at the following URL: <http://ceilometer-internal.openstack.svc:3000/metrics>

#### Procedure

1. Use a text editor of your choice to open the `openstack_control_plane.yaml` file.
2. Create the `telemetry` section based on the needs of your environment:

```
telemetry:
  enabled: true
  template:
    metricStorage:
```

```

enabled: true
monitoringStack:
  dashboardsEnabled: true
  alertingEnabled: true
  scrapInterval: 30s 1
  storage:
    strategy: persistent
    retention: 24h 2
    persistent:
      pvcStorageRequest: 20G 3
  autoscaling: 4
    enabled: false
  aodh:
    databaseUser: aodh
    databaseInstance: openstack
    secret: osp-secret
    heatInstance: heat
  ceilometer:
    enabled: true
    secret: osp-secret
  logging:
    enabled: false
  ipaddr: <ip_address> 5

```

- 1** Use the `scrapInterval` field to control the amount of time that passes before new metrics are gathered. Changing this parameter can affect performance.
- 2** Use the `retention` field to adjust the length of time telemetry metrics are stored. This field affects the amount of storage required.
- 3** You can change the amount of storage to be allocated for the Prometheus time series database.
- 4** You must have the **autoscaling** field present, even if you keep it disabled. For information on enabling and configuring autoscaling, see [link](#).
- 5** Replace `<ip_address>` with the IP address on the internal network you would like to configure.

3. Update the control plane with the Telemetry configurations that you set in **openstack\_control\_plane.yaml**:

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

## Verification

1. Access the remote shell for the **OpenStackClient** pod from your workstation:

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

2. Confirm that you can query prometheus and that the scrape endpoints are active with the following command:

```
$ openstack metric query up --disable-rbac -c container -c instance -c value
```

Example output:

```
+-----+-----+-----+
| container | instance           | value |
+-----+-----+-----+
| alertmanager | 10.217.1.112:9093 | 1 |
| prometheus   | 10.217.1.63:9090  | 0 |
| proxy-httpd  | 10.217.1.52:3000  | 1 |
|              | 192.168.122.100:9100 | 1 |
|              | 192.168.122.101:9100 | 1 |
+-----+-----+-----+
```



#### NOTE

Each entry in the value field should be "1", except for the prometheus container. The prometheus container reports a value of "0" due to TLS, which is enabled by default.

#### Additional resource

- [Installing Logging](#)

## CHAPTER 6. ADDING CUSTOM TLS CERTIFICATES FOR RED HAT OPENSTACK SERVICES ON OPENSIFT

If you decide to apply trusted certificates from your own internal certificate authority (CA), you will need the following information.

### DNS names

For each service you apply your own custom certificate to, you will need its DNS hostname for the process of generating the certificate. You can get a list of public hostnames using the following command: **oc get -n openstack routes**



### NOTE

To use a single certificate for two or more services, use a wildcard in the DNS name field, or list multiple DNS names in the subject alt names field. **If you do not use a wildcard, then you must update the certificate in the event of a route hostname change.**

### Duration

To update a service's certificate in OpenShift, the service must be restarted. The duration for the certificate is the longest amount of time a service can stay live without being restarted, subject to your internal security policies.

### Usages

You must include - **key encipherment**, **digital signature**, and **server auth** within the list of usages in your certificate.

Updating TLS to use custom certificates requires edits to both the control plane and the data plane.

## 6.1. UPDATING THE CONTROL PLANE WITH CUSTOM CERTIFICATES FOR PUBLIC SERVICES

When you deploy Red Hat OpenStack Services on OpenShift (RHOSO), most API connections are protected by TLS.



### NOTE

TLS is not currently available for the internal Alert Manager Web UI service endpoint.

You might be required to protect public APIs using your own internal certificate authority. In order to replace the automatically generated certificates you must create a secret that contains your additional ca certs, including all certificates in needed chains of trust.

### Prerequisites

- You have a service certificate for the public services

### Procedure

1. Create a manifest file called **myAdditionalCACerts.yaml** that includes all CA certificates. Include all certificates in chains of trust if applicable:

■

```

apiVersion: v1
kind: Secret
metadata:
  name: myAdditionalCACerts
  namespace: openstack
type: Opaque
data:
  myBundleExample: <cat mybundle.pem | base64 -w0>
  CACertExample: <cat cacert.pem | base64 -w0>

```

2. Create the secret from the manifest file:

```
oc apply -f myAdditionalCACerts.yaml
```

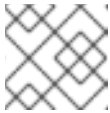
3. Create a manifest file for each API certificate secret called **api\_certificate\_<service>\_secret.yaml**:

```

apiVersion: v1
kind: Secret
metadata:
  name: api_certificate_<service>_secret
  namespace: openstack
type: kubernetes.io/tls
data:
  tls.crt: <cat tls.crt.pem | base64 -w0>
  tls.key: <cat tls.key.pem | base64 -w0>
  ca.crt: <cat ca.crt.pem | base64 -w0>

```

- Replace **<service>** with the name of the service that this secret is for.



#### NOTE

You can use the same secret for multiple services.

4. Create the secret

```
oc apply -f api_certificate_<service>_secret.yaml
```

5. Edit the **openstack\_control\_plane.yaml** custom resource and add your bundle as the parameter for **caBundleSecretName**:

```

apiVersion: core.openstack.org/v1beta1
kind: OpenStackControlPlane
metadata:
  name: myctlplane
spec:
  tls:
    podLevel:
      enabled: true
    caBundleSecretName: myAdditionalCACerts

```

6. Apply the secret service certificates to each of the public services under the `apiOverride` field. For example enter the following for the Identity service:

■



```
apiVersion: core.openstack.org/v1beta1
kind: OpenStackControlPlane
metadata:
  name: myctlplane
  namespace: openstack
spec:
  ...
  keystone:
    apiOverride:
      tls:
        secretName: api_certificate_keystone_secret
```

7. Apply the control plane changes

```
oc apply -f openstack_control_plane.yaml
```

## CHAPTER 7. ACCESSING THE RHOSO CLOUD

You can access your Red Hat OpenStack Services on OpenShift (RHOSO) cloud to perform actions on your data plane by either accessing the OpenStackClient pod through a remote shell from your workstation, or by using a web browser to access the Dashboard service (horizon) interface.

### 7.1. ACCESSING THE OPENSTACKCLIENT POD

You can execute Red Hat OpenStack Services on OpenShift (RHOSO) commands on the deployed data plane by using the **OpenStackClient** pod through a remote shell from your workstation. The OpenStack Operator created the **OpenStackClient** pod as a part of the **OpenStackControlPlane** resource. The **OpenStackClient** pod contains the client tools and authentication details that you require to perform actions on your data plane.

#### 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](#)

### 7.2. ACCESSING THE DASHBOARD SERVICE (HORIZON) INTERFACE

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

#### Procedure

1. 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
```

2. Retrieve the Dashboard service endpoint URL:

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

3. Open a web browser.

4. Enter the Dashboard endpoint URL.
5. Log in to the dashboard with your username and password.

## CHAPTER 8. MONITORING HIGH AVAILABILITY SERVICES

Red Hat OpenStack on OpenShift (RHOSO) high availability (HA) uses Red Hat OpenShift Container Platform (RHOCP) operations to orchestrate failover and recovery deployment. When you plan your deployment, ensure that you review the considerations for different aspects of the environment, such as hardware assignments and network configuration.

The following shared control plane services are required to implement HA:

- Galera Cluster
- RabbitMQ
- memcached

These services run as pods, and they are managed and monitored by RHOCP.

You can use the OpenShift client command line interface (“oc”) to interact with the platform and retrieve information about the status of the OpenStack control plane services.

You can use the OpenShift Client (oc) to complete the following actions:

- List the pods
- Learn more about the pods' configuration
- Retrieve information about the pods' runtime

### 8.1. RHOSO GALERA CLUSTERS

RHOSO deploys the two following Galera clusters:

- **openstack**. This cluster hosts the databases for all OpenStack services.
- **openstack-cell1**. This cluster hosts the databases specific to Nova cell.

Galera Custom Resources configures both clusters.

To retrieve more information about the Galera’s Custom Resources, use the **oc get galera** command as shown in the following example:

```
$ oc get galera
NAME          READY MESSAGE
openstack    True  Setup complete
openstack-cell1 True  Setup complete
```

The **Message** and **Ready** columns show the startup state and the service availability of the Galera CR. When the **Ready** condition is **True**, the pods are started and ready to accept traffic as shown in the following example:

```
$ oc get pod -l galera/name=openstack
NAME          READY STATUS RESTARTS AGE
openstack-galera-0 1/1 Running 0    4h22m
openstack-galera-1 1/1 Running 0    4h22m
openstack-galera-2 1/1 Running 0    4h22m
```

The mariadb operator performs the following Galera cluster operations:

- Creates the pods that host the mysqld servers.
- Runs the logic for bootstrapping a Galera cluster. For example, the mariadb operator starts the cluster using the most recent copy of the Galera database.
- Monitors the running Galera pods.
- Restarts the pods when the pods fail the healthcheck.

To expose the database service, the mariadb operator creates an OpenShift service object called **openstack**. The OpenStack service object components access the database through the IP provided by the service:

```
$ oc get service -l mariadb/name=openstack
NAME      TYPE      CLUSTER-IP  EXTERNAL-IP  PORT(S)  AGE
openstack ClusterIP  10.217.5.210 <none>      3306/TCP  7h
```

The incoming traffic is load-balanced to any available Galera pod. OpenShift marks a pod as available based on its Readiness healthcheck. If a pod misbehaves or if a pod begins to stop, it is removed from the service's list of endpoints.

- To view a list of available endpoints, use the following command:



#### NOTE

The mariadb operator creates a 'headless' service for the Galera pods. This service is a DNS service between Galera pods for internal Galera cluster communication.

### 8.1.1. Monitoring Galera startup

- To monitor the startup of the Galera pods, use the **oc describe galera** command.

Galera CR's status records the status of a Galera cluster's startup. The CR's conditions report the status of the prerequisites that Galera pods need to start as shown in the following example:



#### NOTE

The **Ready** condition is true only when all the other conditions are **True**.

```
Status:
Conditions:
Last Transition Time: 2024-04-22T07:32:06Z
Message:      Setup complete
Reason:       Ready
Status:       True
Type:         Ready
Last Transition Time: 2024-04-22T07:31:49Z
Message:      Deployment completed
Reason:       Ready
Status:       True
Type:         DeploymentReady
Last Transition Time: 2024-04-22T07:31:11Z
```

```

Message:      Exposing service completed
Reason:      Ready
Status:      True
Type:      ExposeServiceReady
Last Transition Time: 2024-04-22T07:31:11Z
Message:      Input data complete
Reason:      Ready
Status:      True
Type:      InputReady
Last Transition Time: 2024-04-22T07:31:11Z
Message:      RoleBinding created
Reason:      Ready
Status:      True
Type:      RoleBindingReady
Last Transition Time: 2024-04-22T07:31:11Z
Message:      Role created
Reason:      Ready
Status:      True
Type:      RoleReady
Last Transition Time: 2024-04-22T07:31:11Z
Message:      ServiceAccount created
Reason:      Ready
Status:      True
Type:      ServiceAccountReady
Last Transition Time: 2024-04-22T07:31:11Z
Message:      Service config create completed
Reason:      Ready
Status:      True
Type:      ServiceConfigReady
Last Transition Time: 2024-04-22T07:31:11Z
Message:      Input data complete
Reason:      Ready
Status:      True
Type:      TLSInputReady

```

When the mariadb operator bootstraps a Galera cluster, it gathers information from every database replica, and then stores it in transient attributes. The transient attributes appear in the Galera CR's status if the cluster is being inspected while the Galera cluster is stopped and being restarted:

```

Status:
  Attributes:
    openstack-galera-0:
      Seqno: 1232
    openstack-galera-1:
      Container ID: cri-o://f56ec2389e878b462a54f5255dad83db29daf4d8e8cda338904bfd353b370165
      Gcomm: gcomm://
      Seqno: 1232
    openstack-galera-2:
      Seqno: 1231
  Bootstrapped: false

```

Before starting a Galera Cluster, the MariaDB operator starts all Galera pod replicas in a **waiting** state. Even if you can see the pods using **oc get pods** command, they have not started mysqld servers yet. The mariadb operator introspects the content of each pod's database copy to extract the database sequence number (Seqno). Once the mariadb operator retrieves all of the pods' Seqno information, it decides which pod holds the most recent version of the database and bootstraps a new Galera cluster

from this pod. This pod starts a mysqld server and a transient attribute **Gcomm://** appears in the Galera CR's status. When the first mysqld server is ready to serve traffic, the attribute **Bootstrapped** becomes true, and transient **Attributes** for this pod are removed from the Galera CR's status.

## 8.2. RHOSO RABBITMQ CLUSTERS

RHOSO deploys the two following RabbitMQ clusters:

- **rabbitmq**. This cluster is used for messaging between OpenStack services.
- **rabbitmq-cell1**. This cluster is used by only Nova.

RabbitMQ Custom Resources configures both clusters.

- To retrieve more information about the RabbitMQ operator, use the following command:

```
$ oc get rabbitmq --show-labels
NAME      ALLREPLICASREADY  RECONCILESUCCESS  AGE  LABELS
rabbitmq  True              True               25h  <none>
rabbitmq-cell1  True              True               25h  <none>
```

The RabbitMQ-cluster operator completes the following tasks:

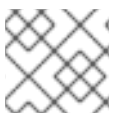
- Creates the pods that run the rabbitmq servers.
- Monitors the pods that run the rabbitmq servers.
- Restarts the pods that run the rabbitmq servers when healthchecks fail.

### 8.2.1. Monitoring the RabbitMQ operator's startup

The state and the service availability of the rabbitmq-cluster operator and the rabbitmq clusters are exposed in the output of the Rabbitmq CR.

#### Procedure

- To retrieve information about the state and service availability of the rabbitmq-cluster operator and the rabbitmq clusters, use the following command:



#### NOTE

Each RabbitMQ replica runs in a dedicated pod.

```
$ oc get pods -l app.kubernetes.io/name=rabbitmq
NAME          READY  STATUS  RESTARTS  AGE
rabbitmq-server-0  1/1  Running  0          46h
rabbitmq-server-1  1/1  Running  0          46h
rabbitmq-server-2  1/1  Running  0          46h
```

The rabbitmq-cluster operator creates two Openstack service objects for a rabbitmq cluster. One service provides a DNS name resolution to the rabbitmq servers for internal rabbitmq communication. The RabbitMQ messaging service is exposed using an Openshift service managed by MetalLB:

```
$ oc get service -l app.kubernetes.io/name=rabbitmq
NAME      TYPE      CLUSTER-IP  EXTERNAL-IP  PORT(S)              AGE
rabbitmq  LoadBalancer  172.30.170.63  172.17.0.85  5671:31331/TCP,15671:31404/TCP,15691:31453/TCP  47h
rabbitmq-nodes ClusterIP  None         <none>       4369/TCP,25672/TCP  47h
```

For example, this MetalLB-managed service is called **rabbitmq**. This service acts as a load balancer across the RabbitMQ pods. It has an IP address that listens to the internal API network so it is accessible from the Openstack dataplane and controlplane. The MetalLB receives incoming traffic from the internal API on 172.17.0.85, and forwards it to the service's IP 172.30.170.63 which balances traffic to rabbitmq pods:

```
$ oc describe service rabbitmq
Name:          rabbitmq
Namespace:    openstack
Labels:        app.kubernetes.io/component=rabbitmq
               app.kubernetes.io/name=rabbitmq
               app.kubernetes.io/part-of=rabbitmq
Annotations:   dnsmasq.network.openstack.org/hostname: rabbitmq.openstack.svc
               metallb.universe.tf/address-pool: internalapi
               metallb.universe.tf/ip-allocated-from-pool: internalapi
               metallb.universe.tf/loadBalancerIPs: 172.17.0.85
Selector:      app.kubernetes.io/name=rabbitmq
Type:          LoadBalancer
IP Family Policy:  SingleStack
IP Families:    IPv4
IP:            172.30.170.63
IPs:           172.30.170.63
LoadBalancer Ingress: 172.17.0.85
Port:          amqps 5671/TCP
TargetPort:    5671/TCP
NodePort:      amqps 31331/TCP
Endpoints:     192.168.16.69:5671,192.168.20.54:5671,192.168.24.45:5671
```

## 8.3. RHOSO MEMCACHED CLUSTERS

By default, all the OpenStack services in the control plane target a single memcached cluster that contains three memcached servers. This cluster is configured using a single memcached resource created by the openstack operator. The infra operator creates the pods that host the memcached servers and the OpenShift service objects that expose the memcached service.

### 8.3.1. Monitoring memached startup

#### Procedure

- To monitor the memcached startup, use the **oc get memached** command. You can view the the startup state and service availability in the **Message** and **Ready** column:

```
$ oc get memcached
NAME      READY MESSAGE
memcached True Setup complete
```



When a memcached CR is marked as **Ready**, its associated pods are started and ready to accept traffic. For example, here is a memcached cluster that is ready to accept traffic:

```
$ oc get pods -l memcached/name=memcached
NAME      READY  STATUS   RESTARTS  AGE
memcached-0  1/1    Running  0         2d4h
memcached-1  1/1    Running  0         15m
memcached-2  1/1    Running  0         15m

$ oc get service memcached
NAME      TYPE        CLUSTER-IP  EXTERNAL-IP  PORT(S)          AGE
memcached ClusterIP   None        <none>       11211/TCP,11212/TCP 2d4h
```

The memcached pods are accessed directly by name through the Openstack components. The memcached service is used only to maintain a list of DNS records for each memcached pod.

## 8.4. LISTING RHOSO CONTROL PLANE SERVICES PODS

You can list your control plane service pods to understand which pods are running on your control plane.

### Procedure

- Use the **oc get pods** command to list the pods:

```
$ oc get pods |egrep -e "galera|rabbit|memcache"
NAME                READY  STATUS   RESTARTS  AGE
memcached-0         1/1    Running  0         28m
memcached-1         1/1    Running  0         28m
memcached-2         1/1    Running  0         28m
openstack-cell1-galera-0  1/1    Running  0         28m
openstack-cell1-galera-1  1/1    Running  0         28m
openstack-cell1-galera-2  1/1    Running  0         28m
openstack-galera-0     1/1    Running  0         28m
openstack-galera-1     1/1    Running  0         28m
openstack-galera-2     1/1    Running  0         28m
rabbitmq-cell1-server-0  1/1    Running  0         28m
rabbitmq-cell1-server-1  1/1    Running  0         28m
rabbitmq-cell1-server-2  1/1    Running  0         28m
rabbitmq-server-0     1/1    Running  0         28m
rabbitmq-server-1     1/1    Running  0         28m
rabbitmq-server-2     1/1    Running  0         28m
```

## 8.5. LISTING THE RHOSO HIGH AVAILABILITY OPERATORS

You can view the operators that your environment currently uses.

### Procedure

- Use the following command to list these services:

```
$ oc get operators
NAME          AGE
...
```

```

infra-operator.openstack-operators      9h
...
mariadb-operator.openstack-operators    9h
...
rabbitmq-cluster-operator.openstack-operators  9h

```



## NOTE

The **infra-operator** is responsible for the **Memcached** service.

## 8.6. RETRIEVING INFORMATION ABOUT AN OPERATOR'S CUSTOM RESOURCE

### Procedure

1. Use the following command to view the custom resource definition that an operator implements:

```

$ oc describe operator/infra-operator.openstack-operators |less
...
Status:
Components:
...
Kind:          CustomResourceDefinition
Name:          memcacheds.memcached.openstack.org
...

```

2. Use the following command to retrieve information about a custom resource's definition:

```

$ oc describe crd/galeras.mariadb.openstack.org
Name:      galeras.mariadb.openstack.org
Namespace:
Labels:    operators.coreos.com/mariadb-operator.openstack-operators=
Annotations: controller-gen.kubebuilder.io/version: v0.11.1
            operatorframework.io/installed-alongside-96a31840a95472ca: openstack-operators/mariadb-
operator.v0.0.1
API Version: apiextensions.k8s.io/v1
Kind:       CustomResourceDefinition
Metadata:
  Creation Timestamp: 2024-03-21T22:08:06Z
  Generation:        1
  Resource Version:  64637
  UID:               f68caee7-b4ec-4713-8095-c4ee9b1fd13e
Spec:
...

```

For more information about operators, see [What are Operators?](#)

## 8.7. RETRIEVING INFORMATION ABOUT AN OPERATOR'S STATEFULSET

A statefulset manages the deployment and scaling of a set of pods. Each of the shared services Operators are responsible for creating and managing a **statefulset**.

### Procedure

- Use the **oc get statefulset** command to retrieve information about the Operators' **statefulset**:

```
$ oc get statefulset |egrep -e "galera|rabbit|memcache"
NAME                READY  AGE
memcached           1/1   174m
openstack-cell1-galera  3/3   174m
openstack-galera     3/3   174m
rabbitmq-cell1-server  3/3   174m
rabbitmq-server      3/3   174m
```

## 8.8. RETRIEVING MORE INFORMATION ABOUT AN OPERATOR'S STATEFULSET

You can retrieve the following information about the **statefulset** of each service:

- Basic information about the service. For example, the number of the replicas
- Actual container details. For example, environment variables
- Volume details
- Event details

### Procedure

- To retrieve more information about the Operator's statefulset, use the **oc describe statefulset/<operator\_name>**

Replace <opeartore\_name> with the name of the operator you want to retrieve more information about.

### 8.8.1. Basic information about a service's statefulset

The following example shows the basic information that you can retrieve about an operator:

```
Name:      openstack-galera
Namespace: openstack
CreationTimestamp: Thu, 21 Mar 2024 08:39:59 -0400
Selector:  app=galera,cr=galera-
openstack,galera/name=openstack,galera/namespace=openstack,galera/uid=1c93b3a3-1ac3-4f18-
984d-34e9ce9dc12f,owner=mariadb-operator
Labels:    <none>
Annotations: <none>
Replicas:  3 desired | 3 total
Update Strategy: RollingUpdate
  Partition: 0
Pods Status:  3 Running / 0 Waiting / 0 Succeeded / 0 Failed
Pod Template:
  Labels:  app=galera
          cr=galera-openstack
```

```

    galera/name=openstack
    galera/namespace=openstack
    galera/uid=1c93b3a3-1ac3-4f18-984d-34e9ce9dc12f
    owner=mariadb-operator
Service Account: galera-openstack
Init Containers:
  mysql-bootstrap:
Image: quay.io/podified-antelope-centos9/openstack-
mariadb@sha256:7fa37f7dcdd850fb6e401c4d5f76d16ad53ecdd14d6a130dbf61f02b819dcdf6
Port: <none>
Host Port: <none>
Command:
  bash
  /var/lib/operator-scripts/mysql_bootstrap.sh
Environment:
  KOLLA_BOOTSTRAP: True
  KOLLA_CONFIG_STRATEGY: COPY_ALWAYS
  DB_ROOT_PASSWORD: <set to the key 'DbRootPassword' in secret 'osp-secret'> Optional:
false
Mounts:
  /var/lib/config-data/default from config-data-default (ro)
  /var/lib/config-data/generated from config-data-generated (rw)
  /var/lib/kolla/config_files from kolla-config (ro)
  /var/lib/mysql from mysql-db (rw)
  /var/lib/operator-scripts from operator-scripts (ro)
  /var/lib/secrets from secrets (ro)
... [cont]

```

## 8.8.2. Information about actual container of a service's statefulset

The following example shows the information about the actual container that you can retrieve about an operator:

```

Containers:
  galera:
Image: quay.io/podified-antelope-centos9/openstack-
mariadb@sha256:7fa37f7dcdd850fb6e401c4d5f76d16ad53ecdd14d6a130dbf61f02b819dcdf6
Ports: 3306/TCP, 4567/TCP
Host Ports: 0/TCP, 0/TCP
Command:
  /usr/bin/dumb-init
  --
  /usr/local/bin/kolla_start
Liveness: exec [/bin/bash /var/lib/operator-scripts/mysql_probe.sh liveness] delay=0s timeout=1s
period=10s #success=1 #failure=3
Readiness: exec [/bin/bash /var/lib/operator-scripts/mysql_probe.sh readiness] delay=0s timeout=1s
period=10s #success=1 #failure=3
Startup: exec [/bin/bash /var/lib/operator-scripts/mysql_probe.sh startup] delay=0s timeout=1s
period=10s #success=1 #failure=30
Environment:
  CR_CONFIG_HASH:
n558hf6h557hcfh589h688h684hb6h687h679h659h554h64fh77h76h568h695h5b6h8fh79h5c8h648h67
4hdch556h56bh655h64bh655h66ch5h5c4q
  KOLLA_CONFIG_STRATEGY: COPY_ALWAYS
  DB_ROOT_PASSWORD: <set to the key 'DbRootPassword' in secret 'osp-secret'> Optional:

```

```

false
Mounts:
  /etc/pki/ca-trust/extracted/pem/tls-ca-bundle.pem from combined-ca-bundle (ro,path="tls-ca-
bundle.pem")
  /var/lib/config-data/default from config-data-default (ro)
  /var/lib/config-data/generated from config-data-generated (rw)
  /var/lib/config-data/tls/certs/galera.crt from galera-tls-certs (ro,path="tls.crt")
  /var/lib/config-data/tls/private/galera.key from galera-tls-certs (ro,path="tls.key")
  /var/lib/kolla/config_files from kolla-config (ro)
  /var/lib/mysql from mysql-db (rw)
  /var/lib/operator-scripts from operator-scripts (ro)
  /var/lib/secrets from secrets (ro)
... [cont]

```

### 8.8.3. Information about the volumes of a service's statefulset

The following example shows the information about the volumes of a service that you can retrieve about an operator:

```

Volumes:
  secrets:
    Type: Secret (a volume populated by a Secret)
    SecretName: osp-secret
    Optional: false
  kolla-config:
    Type: ConfigMap (a volume populated by a ConfigMap)
    Name: openstack-config-data
    Optional: false
  config-data-generated:
    Type: EmptyDir (a temporary directory that shares a pod's lifetime)
  Medium:
  SizeLimit: <unset>
  config-data-default:
    Type: ConfigMap (a volume populated by a ConfigMap)
    Name: openstack-config-data
    Optional: false
  operator-scripts:
    Type: ConfigMap (a volume populated by a ConfigMap)
    Name: openstack-scripts
    Optional: false
  galera-tls-certs:
    Type: Secret (a volume populated by a Secret)
    SecretName: cert-galera-openstack-svc
    Optional: false
  combined-ca-bundle:
    Type: Secret (a volume populated by a Secret)
    SecretName: combined-ca-bundle
    Optional: false
Volume Claims:
  Name: mysql-db
  StorageClass: local-storage
  Labels: app=galera
         cr=galera-openstack
         galera/name=openstack
         galera/namespace=openstack

```

```

galera/uid=1c93b3a3-1ac3-4f18-984d-34e9ce9dc12f
owner=mariadb-operator
Annotations: <none>
Capacity: 5G
Access Modes: [ReadWriteOnce]
... [cont]

```

### 8.8.4. Information about Event details of a service's statefulset

The following example shows the Event details that you can retrieve about an operator:

```

Events:
  Type Reason      Age From          Message
  ---- -
Normal SuccessfulCreate 179m statefulset-controller create Claim mysql-db-openstack-galera-0
Pod openstack-galera-0 in statefulset openstack-galera success
Normal SuccessfulCreate 179m statefulset-controller create Pod openstack-galera-0 in
statefulset openstack-galera successful
Normal SuccessfulCreate 179m statefulset-controller create Claim mysql-db-openstack-galera-1
Pod openstack-galera-1 in statefulset openstack-galera success
Normal SuccessfulCreate 179m statefulset-controller create Claim mysql-db-openstack-galera-2
Pod openstack-galera-2 in statefulset openstack-galera success
Normal SuccessfulCreate 179m statefulset-controller create Pod openstack-galera-1 in
statefulset openstack-galera successful
Normal SuccessfulCreate 179m statefulset-controller create Pod openstack-galera-2 in
statefulset openstack-galera successful

```

## 8.9. CHECKING THE STATUS OF THE CONTROL PLANE

Each of the operators monitors the status of the pods that they manage. If necessary, they will take appropriate actions with the target of keeping one replica with a status of "ready" and "running".

### Procedure

- Use the **oc get pods** command to check the status of your control plane shared services:

```

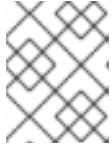
oc get pods |egrep -e "galera|rabbit|memcache"
NAME                READY STATUS RESTARTS AGE
memcached-0         1/1 Running 0       3h11m
memcached-1         1/1 Running 0       3h11m
memcached-2         1/1 Running 0       3h11m
openstack-cell1-galera-0 1/1 Running 0       3h11m
openstack-cell1-galera-1 1/1 Running 0       3h11m
openstack-cell1-galera-2 1/1 Running 0       3h11m
openstack-galera-0   1/1 Running 0       3h11m
openstack-galera-1   1/1 Running 0       3h11m
openstack-galera-2   1/1 Running 0       3h11m
rabbitmq-cell1-server-0 1/1 Running 0       3h11m
rabbitmq-cell1-server-1 1/1 Running 0       3h11m
rabbitmq-cell1-server-2 1/1 Running 0       3h11m
rabbitmq-server-0    1/1 Running 0       3h11m
rabbitmq-server-1    1/1 Running 0       3h11m
rabbitmq-server-2    1/1 Running 0       3h11m

```

### 8.9.1. Checking the status of a pod

#### Procedure

- You can retrieve more information about a pod using the **oc describe pod/<pod-name>** command.



#### NOTE

Replace <pod-name> with the name of the pod that you want to retrieve more information about.

```
$ oc describe pod/rabbitmq-server-0
Name:      rabbitmq-server-0
Namespace: openstack
Priority:   0
Service Account: rabbitmq-server
Node:      master-2/192.168.111.22
Start Time: Thu, 21 Mar 2024 08:39:57 -0400
Labels:    app.kubernetes.io/component=rabbitmq
           app.kubernetes.io/name=rabbitmq
           app.kubernetes.io/part-of=rabbitmq
           controller-revision-hash=rabbitmq-server-5c886b79b4
           statefulset.kubernetes.io/pod-name=rabbitmq-server-0
Annotations: k8s.ovn.org/pod-networks:
             {"default":{"ip_addresses":
["192.168.16.35/22"],"mac_address":"0a:58:c0:a8:10:23","gateway_ips":["192.168.16.1"],"routes":
[{"dest":"192.16...
             k8s.v1.cni.cncf.io/network-status:
             [{
               "name": "ovn-kubernetes",
               "interface": "eth0",
               "ips": [
                 "192.168.16.35"
               ],
               "mac": "0a:58:c0:a8:10:23",
               "default": true,
               "dns": {}
             }]
             openshift.io/scc: restricted-v2
             seccomp.security.alpha.kubernetes.io/pod: runtime/default
Status:     Running
...
```

## 8.10. EXPOSURE OF EACH SERVICE THROUGH CLUSTERIP OR LOADBALANCER

The ClusterIP or the LoadBalancer exposes each service.

- To retrieve more information about the clustertips of the loadbalancers that expose a service, use the following command:

```
$ oc get svc |egrep -e "rabbit|galera|memcache"
```

memcached	ClusterIP	None	<none>	11211/TCP	openstack-cell1-galera
ClusterIP	None	<none>	3306/TCP		
openstack-galera	ClusterIP	None	<none>	3306/TCP	
rabbitmq	LoadBalancer	172.30.21.129	172.17.0.85		
		5672:31952/TCP,15672:30111/TCP,15692:30081/TCP			
rabbitmq-cell1	LoadBalancer	172.30.97.190	172.17.0.86		
		5672:30043/TCP,15672:30645/TCP,15692:32654/TCP			
rabbitmq-cell1-nodes	ClusterIP	None	<none>	4369/TCP,25672/TCP	
rabbitmq-nodes	ClusterIP	None	<none>	4369/TCP,25672/TCP	

For more information about the OpenShift capabilities that you can use to expose the services, see [About networking](#).

- Use the following command to retrieve more information about a service:

```
$ oc describe svc/rabbitmq
Name:          rabbitmq
Namespace:    openstack
Labels:       app.kubernetes.io/component=rabbitmq
              app.kubernetes.io/name=rabbitmq
              app.kubernetes.io/part-of=rabbitmq
Annotations:  dnsmasq.network.openstack.org/hostname: rabbitmq.openstack.svc
              metallb.universe.tf/address-pool: internalapi
              metallb.universe.tf/ip-allocated-from-pool: internalapi
              metallb.universe.tf/loadBalancerIPs: 172.17.0.85
Selector:     app.kubernetes.io/name=rabbitmq
Type:         LoadBalancer
IP Family Policy:  SingleStack
IP Families:   IPv4
IP:           172.30.21.129
IPs:          172.30.21.129
LoadBalancer Ingress: 172.17.0.85
Port:         amqp 5672/TCP
TargetPort:   5672/TCP
NodePort:     amqp 31952/TCP
Endpoints:    192.168.16.43:5672,192.168.20.69:5672,192.168.24.53:5672
Port:         management 15672/TCP
TargetPort:   15672/TCP
NodePort:     management 30111/TCP
Endpoints:    192.168.16.43:15672,192.168.20.69:15672,192.168.24.53:15672
Port:         prometheus 15692/TCP
TargetPort:   15692/TCP
NodePort:     prometheus 30081/TCP
Endpoints:    192.168.16.43:15692,192.168.20.69:15692,192.168.24.53:15692
Session Affinity:  None
External Traffic Policy: Cluster
Events:       <none>
```

## 8.11. TESTING THE RESILIENCE OF THE CONTROL PLANE

To test that the control plane shared services are resilient to container failures, you can simulate a failure.

### Procedure



- To simulate a failure, you can use the following command to delete one of the pods:

```
$ oc delete pod/rabbitmq-server-1
pod "rabbitmq-server-1" deleted
```

After you delete the pod, the “rabbitmq-server-1” pod is immediately rescheduled:

```
$ oc get pods |grep -rabbit
rabbitmq-cell1-server-0          1/1 Running 0    4h20m
rabbitmq-cell1-server-1          1/1 Running 0    4h20m
rabbitmq-cell1-server-2          1/1 Running 0    4h20m
rabbitmq-server-0                1/1 Running 0    4h20m
rabbitmq-server-1                0/1 Init:0/1 0    2s
rabbitmq-server-2                1/1 Running 0    4h20m
```

After a few seconds, the pod should have the status of **running**:

```
[zuul@controller-0 ~]$ oc get pods |grep rabbit
rabbitmq-cell1-server-0          1/1 Running 0    4h23m
rabbitmq-cell1-server-1          1/1 Running 0    4h23m
rabbitmq-cell1-server-2          1/1 Running 0    4h23m
rabbitmq-server-0                1/1 Running 0    4h23m
rabbitmq-server-1                1/1 Running 0    3m8s
rabbitmq-server-2                1/1 Running 0    4h23m
```

### 8.11.1. The Taint-Based Evictions feature

By default, The Taint-Based Evictions feature evicts pods from a node that experiences specific conditions like **not-ready** and **unreachable**. When a node experiences one of these conditions, OCP adds taints to the node, evicts the pods, and then reschedules the pods on different nodes.

Also, Taint-Based Evictions have a **NoExecute effect**. Any pod that does not tolerate the taint is evicted immediately and any pod that does tolerate the taint will never be evicted, unless the pod uses the **tolerationSeconds** parameter.

Use the **tolerationSeconds** parameter to specify how long a pod stays bound to a node that has a node condition. If the condition still exists after the **tolerationSeconds** period, the taint remains on the node and the pods with a matching toleration are evicted. If the condition clears before the tolerationSeconds period, pods with matching tolerations are not removed.

OpenShift Container Platform adds a toleration for `node.kubernetes.io/not-ready` and `node.kubernetes.io/unreachable` with `tolerationSeconds=300`, unless the Pod configuration specifies either toleration.



#### IMPORTANT

RHOSO 18.0 operators do not modify the default **tolerationSeconds** values. Pods that run on a faulty worker node take more than five minutes to be rescheduled.

For more information about Remediation, fencing, and maintenance, see [Remediation, fencing, and maintenance](#)

## CHAPTER 9. COLLECTING DIAGNOSTIC INFORMATION FOR SUPPORT

Use the Red Hat OpenStack Services on OpenShift (RHOSO) **must-gather** tool to collect diagnostic information about your Red Hat OpenShift Container Platform (RHOCP) cluster, including the RHOSO control plane and the deployed RHOSO services. Use the RHOCP **sosreport** tool to collect diagnostic information about your RHOSO data plane.

### 9.1. COLLECTING DATA ON THE RHOSO CONTROL PLANE

You can use the Red Hat OpenStack Services on OpenShift (RHOSO) **must-gather** tool to collect the following information about your Red Hat OpenShift Container Platform (RHOCP) cluster to troubleshoot service failures:

- The RHOSO control plane service logs.
- The configuration of RHOSO control plane services, such as the RHOCP **Secrets** and **ConfigMaps**.
- Status of the services that are deployed in the RHOSO control plane.
- The RHOSO generated Custom Resource Definitions (CRDs).
- The RHOSO control plane applied Custom Resources (CRs).
- The **openstack** and **openstack-operators** namespaces.
- RHOCP Events that are related to the RHOSO namespaces.

#### Prerequisites

- Access to the cluster as a user with **cluster-admin** privileges.

#### Procedure

1. Navigate to the directory where you want to store the **must-gather** data.
2. Pass one or more images or image streams to the **must-gather** tool to specify the data to collect. For example, the following command gathers both the default cluster data and the information that is specific to the deployed RHOSO control plane:

```
$ oc adm must-gather \  
--image-stream=openshift/must-gather \ 1  
--image=registry.redhat.io/openstack-k8s-operators/openstack-must-gather 2
```

**1** The default RHOCP **must-gather** image that is used to gather RHOCP cluster information.

**2** The RHOSO **must-gather** image.

This command creates a local directory that stores the logs, services configuration, and the status of the RHOSO control plane services.

## 9.2. COLLECTING DATA ON THE RHOSO DATA PLANE NODES

The data plane nodes are RHEL nodes where the Compute service (nova) runs, and the Ceph daemons in an HCI environment. The **must-gather** tool collects the Red Hat OpenShift Container Platform (RHOCP) information that is generated in the control plane, but it does not gather the logs for the data plane nodes. To diagnose and troubleshoot issues on the data plane, Red Hat Support requires regular SOS reports to gather the data of the services that are deployed in the data plane nodes.

For information on how to use the SOS report tool, see [Getting the most from your Support experience](#) .