Red Hat OpenStack Services on OpenShift 18.0

Deploying Red Hat OpenStack Services on OpenShift

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

Last Updated: 2024-09-13

Red Hat OpenStack Services on OpenShift 18.0 Deploying Red Hat OpenStack Services on OpenShift

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

OpenStack Team
rhos-docs@redhat.com

Abstract

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

Table of Contents

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 for Red Hat OpenStack Services on OpenShift (RHOSO) or earlier releases of Red Hat OpenStack Platform (RHOSP). When you create an issue for RHOSO or RHOSP documents, the issue is recorded in the RHOSO Jira project, where you can track the progress of your feedback.

To complete the Create Issue form, ensure that you are logged in to Jira. If you do not have a Red Hat Jira account, you can create an account at https://issues.redhat.com.

1. Click the following link to open a **Create Issue** page: Create Issue
2. 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.
3. Click **Create**.

CHAPTER 1. INSTALLING AND PREPARING THE OPERATORS

You install the Red Hat OpenStack Services on OpenShift (RHOSO) OpenStack Operator (**openstack-operator**) and create the RHOSO control plane on an operational Red Hat OpenShift Container Platform (RHOCP) cluster. You install the OpenStack Operator by using the RHOCP web console. You perform the control plane installation tasks and all data plane creation tasks on a workstation that has access to the RHOCP cluster.

1.1. PREREQUISITES

- An operational RHOCP cluster, version 4.16. For the RHOCP system requirements, see Red Hat OpenShift Container Platform cluster requirements in *Planning your deployment*.
- The **oc** command line tool is installed on your workstation.
- You are logged in to the RHOCP cluster as a user with **cluster-admin** privileges.

1.2. INSTALLING THE OPENSTACK OPERATOR

You use OperatorHub on the Red Hat OpenShift Container Platform (RHOCP) web console to install the OpenStack Operator (**openstack-operator**) on your RHOCP cluster.

Procedure

1. Log in to the RHOCP web console as a user with **cluster-admin** permissions.
2. Select **Operators → OperatorHub**.
3. In the **Filter by keyword** field, type **OpenStack**.
4. Click the **OpenStack Operator** tile with the **Red Hat** source label.
5. Read the information about the Operator and click **Install**.
6. On the **Install Operator** page, select "Operator recommended Namespace: openstack-operators" from the **Installed Namespace** list.
7. Click **Install** to make the Operator available to the **openstack-operators** namespace. The Operators are deployed and ready when the Status of the OpenStack Operator is **Succeeded**.

CHAPTER 2. PREPARING RED HAT OPENSHIFT CONTAINER PLATFORM FOR RED HAT OPENSTACK SERVICES ON OPENSHIFT

You install Red Hat OpenStack Services on OpenShift (RHOSO) on an operational Red Hat OpenShift Container Platform (RHOCP) cluster. 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 OPENSHIFT 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.

Therefore, you can pin the **cinder-api**, **cinder-volume**, and **cinder-backup** services to dedicated nodes and let the OpenStack Operator place the **cinder-scheduler** service on a node that has capacity.

Additional resources

- Placing pods on specific nodes using node selectors
- Machine configuration overview
- Node Feature Discovery Operator

2.2. 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 (LVM) Storage 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. Red Hat recommends that you use a storage back end based on SSD or NVMe drives for the storage class.

You must wait until the LVM Storage Operator announces that the storage is available before creating the control plane. The LVM Storage Operator announces that the cluster and LVMS storage configuration is complete through the annotation for the volume group to the worker node object. If you deploy pods before all the control plane nodes are ready, then multiple PVCs and pods are scheduled on the same nodes.

To check that the storage is ready, you can query the nodes in your **lvmclusters.lvm.topolvm.io** object. For example, run the following command if you have three worker nodes and your device class for the LVM Storage Operator is named "local-storage":

```
# oc get node -l "topology.topolvm.io/node in ($(oc get nodes -l node-role.kubernetes.io/worker -o
name | cut -d '/' -f 2 | tr '\n' ',' | sed 's/.\{1\}$//'))" -
o=jsonpath='{.items[*].metadata.annotations.capacity\.topolvm\.io/local-storage}' | tr ' ' '\n'
```

The storage is ready when this command returns three non-zero values

For more information about how to configure the LVM Storage storage class, see Persistent storage using Logical Volume Manager Storage in the RHOCP *Storage* guide.

2.3. CREATING THE OPENSTACK NAMESPACE

You must create a namespace within your Red Hat OpenShift Container Platform (RHOCP) environment for the service pods of your Red Hat OpenStack Services on OpenShift (RHOSO) deployment. The service pods of each RHOSO deployment exist in their own namespace within the RHOCP environment.

Prerequisites

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

Procedure

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

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

2. Ensure the **openstack** namespace is labeled to enable privileged pod creation by the OpenStack Operators:

   ```
   $ oc get namespace openstack -ojsonpath='{.metadata.labels}' | jq
   {
     "kubernetes.io/metadata.name": "openstack",
     "pod-security.kubernetes.io/enforce": "privileged",
     "security.openshift.io/scc.podSecurityLabelSync": "false"
   }
   ```

   If the security context constraint (SCC) is not "privileged", use the following commands to change it:

   ```
   $ oc label ns openstack security.openshift.io/scc.podSecurityLabelSync=false --overwrite
   $ oc label ns openstack pod-security.kubernetes.io/enforce=privileged --overwrite
   ```

3. Optional: To remove the need to specify the namespace when executing commands on the **openstack** namespace, set the default **namespace** to **openstack**:

```
$ oc project openstack
```

2.4. PROVIDING SECURE ACCESS TO THE RED HAT OPENSTACK SERVICES ON OPENSHIFT SERVICES

You must create a **Secret** custom resource (CR) to provide secure access to the Red Hat OpenStack Services on OpenShift (RHOSO) service pods.

> **WARNING**
>
> You cannot change a service password once the control plane is deployed. If a service password is changed in **osp-secret** after deploying the control plane, the service is reconfigured to use the new password but the password is not updated in the Identity service (keystone). This results in a service outage.

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_fernet_key>
  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>
  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 32-character key that is base64 encoded. You can use the following command to manually generate a base64 encoded password:

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

  Alternatively, if you are using a Linux workstation and you are generating the **Secret** CR definition file by using a Bash command such as **cat**, you can replace **<base64_password>** with the following command to auto-generate random passwords for each service:

  ```
  $(tr -dc 'A-Za-z0-9' < /dev/urandom | head -c 32 | base64)
  ```

- Replace the **<base64_fernet_key>** with a fernet key that is base64 encoded. You can use the following command to manually generate the fernet key:

  ```
  python3 -c "from cryptography.fernet import Fernet;
  print(Fernet.generate_key().decode('UTF-8'))" | base64
  ```

NOTE

The **HeatAuthEncryptionKey** password must be a 32-character key for Orchestration service (heat) encryption. If you increase the length of the passwords for all other services, ensure that the **HeatAuthEncryptionKey** password remains at length 32.

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

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

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

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

CHAPTER 3. PREPARING NETWORKS FOR RED HAT OPENSTACK SERVICES ON OPENSHIFT

To prepare for configuring and deploying your Red Hat OpenStack Services on OpenShift (RHOSO) environment, you must configure the Red Hat OpenShift Container Platform (RHOCP) networks on your RHOCP cluster.

3.1. DEFAULT RED HAT OPENSTACK SERVICES ON OPENSHIFT NETWORKS

The following physical data center networks are typically implemented for a Red Hat OpenStack Services on OpenShift (RHOSO) deployment:

- Control plane network: This network is used by the OpenStack Operator for Ansible SSH access to deploy and connect to the data plane nodes from the Red Hat OpenShift Container Platform (RHOCP) environment. This network is also used by data plane nodes for live migration of instances.
- 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 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 3.1. Default RHOSO networks

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

3.2. PREPARING RHOCP FOR RHOSO NETWORKS

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** (**net-attach-def**) 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 an **L2Advertisement** resource to define how the Virtual IPs (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.

NOTE

The examples in the following procedure use IPv4 addresses. You can use IPv6 addresses instead of IPv4 addresses. Dual stack IPv4/6 is not available. For information about how to configure IPv6 addresses, see the following resources in the RHOCP *Networking* guide:

- Installing the Kubernetes NMState Operator
- Configuring MetalLB address pools

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 Services on OpenShift networks.
In the following example, the **nncp** CR configures the **enp6s0** interface for worker node 1, **osp-enp6s0-worker-1**, to use VLAN interfaces with IPv4 addresses 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: internalapi
      state: up
      type: vlan
      vlan:
        base-iface: enp6s0
        id: 20
        reorder-headers: true
    - description: storage vlan interface
      ipv4:
        address:
        - ip: 172.18.0.10
          prefix-length: 24
        enabled: true
        dhcp: false
      ipv6:
        enabled: false
      name: storage
      state: up
      type: vlan
      vlan:
        base-iface: enp6s0
        id: 21
        reorder-headers: true
    - description: tenant vlan interface
      ipv4:
        address:
        - ip: 172.19.0.10
          prefix-length: 24
        enabled: true
        dhcp: false
      ipv6:
        enabled: false
      name: tenant
      state: up
      type: vlan
      vlan:
        base-iface: enp6s0
        id: 22
        reorder-headers: true
    - 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-1
    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** (**net-attach-def**) CR file on your workstation, for example, **openstack-net-attach-def.yaml**.

8. In the **NetworkAttachmentDefinition** CR file, configure a **NetworkAttachmentDefinition** resource for each isolated network to attach a service deployment pod to the network. The following examples create a **NetworkAttachmentDefinition** 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": "internalapi", 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:
  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": "storage",
      "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:
  name: tenant
  namespace: openstack
spec:
  config: |
    {
      "cniVersion": "0.3.1",
      "name": "tenant",
      "type": "macvlan",
      "master": "tenant",
      "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 node interface name associated with the network, as defined in the **nncp** CR.

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 **NetworkAttachmentDefinition** CR in the cluster:

```
$ oc apply -f openstack-net-attach-def.yaml
```

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

    ```
    $ oc get net-attach-def -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 in the RHOCP *Networking* guide.

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

```
apiVersion: metallb.io/v1beta1
kind: L2Advertisement
metadata:
  name: internalapi
  namespace: metallb-system
spec:
  ipAddressPools:
  - internalapi
  interfaces:
  - internalapi [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:
```

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

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

For information about how to configure the other **L2Advertisement** resource parameters, see Configuring MetalLB with a L2 advertisement and label in the RHOCP *Networking* guide.

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

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

18. Verify that the **L2Advertisement** CRs are created:

```
$ oc get -n metallb-system L2Advertisement
NAME          IPADDRESSPOOLS    IPADDRESSPOOL SELECTORS   INTERFACES
ctlplane      ["ctlplane"]                                ["enp6s0"]
internalapi   ["internalapi"]                             ["internalapi"]
storage       ["storage"]                                 ["storage"]
tenant        ["tenant"]                                  ["tenant"]
```

19. If your cluster 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
    ```

3.3. CREATING 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 **openstack_netconfig.yaml** on your workstation.
2. Add the following configuration to **openstack_netconfig.yaml** to create the **NetConfig** CR:

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

3. In the **openstack_netconfig.yaml** file, define the topology for each data plane network. To use the default Red Hat OpenStack Services on OpenShift (RHOSO) networks, you must define a specification for each network. For information about the default RHOSO networks, see Default Red Hat OpenStack Services on OpenShift 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: (5)
         - 172.17.0.10
         - 172.17.0.12
         cidr: 172.17.0.0/24
         vlan: 20 (6)
     - 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: 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, **CtlPlane**.

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 Optional: List of IP addresses from the allocation range that must not be used by data plane nodes.

6 The network VLAN. For information about the default RHOSO networks, see Default Red Hat OpenStack Services on OpenShift networks.

4. Save the **openstack_netconfig.yaml** definition file.

5. Create the data plane network:

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

6. To verify that the data plane network is created, view the **openstacknetconfig** resource:

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

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

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

CHAPTER 4. 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.

NOTE

Creating the control plane also creates an **OpenStackClient** pod that you can access through a remote shell (**rsh**) to run OpenStack CLI commands.

4.1. PREREQUISITES

- The OpenStack Operator (**openstack-operator**) is installed. For more information, see Installing and preparing the Operators.
- The RHOCP cluster is prepared for RHOSO networks. For more information, see Preparing RHOCP for RHOSO networks.
- The RHOCP cluster is not configured with any network policies that prevent communication between the **openstack-operators** namespace and the control plane namespace (default **openstack**). Use the following command to check the existing network policies on the cluster:

  ```
  $ oc get networkpolicy -n openstack
  ```

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

4.2. CREATING THE CONTROL PLANE

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

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

The following procedure creates an initial control plane with the recommended configurations for each service. The procedure helps you quickly create an operating control plane environment that you can use to troubleshoot issues and test the environment before adding all the customizations you require. You can add service customizations to a deployed environment. For more information about how to customize your control plane after deployment, see the Customizing the Red Hat OpenStack Services on OpenShift deployment guide.

For an example **OpenStackControlPlane** CR, see Example **OpenStackControlPlane** CR.

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
     namespace: openstack
   ```

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
     namespace: openstack
   spec:
     secret: osp-secret
   ```

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

   ```
   spec:
     secret: osp-secret
     storageClass: <RHOCP_storage_class>
   ```

   - Replace **<RHOCP_storage_class>** with the storage class you created for your RHOCP cluster storage back end. For information about storage classes, see Creating a storage class.

4. Add the following service configurations:

   - Block Storage service (cinder):

     ```
     cinder:
       apiOverride:
         route: {}
       template:
         databaseInstance: openstack
         secret: osp-secret
         cinderAPI:
           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
     ```

```
    cinderScheduler:
      replicas: 1
    cinderBackup:
      networkAttachments:
      - storage
      replicas: 0 1
    cinderVolumes:
      volume1:
        networkAttachments:
        - storage
        replicas: 0 2
```

1. You can deploy the initial control plane without activating the **cinderBackup** service. To deploy the service, you must set the number of **replicas** for the service and configure the back end for the service. For information about the recommended replicas for each service and how to configure a back end for the Block Storage service and the backup service, see Configuring the Block Storage backup service in *Configuring persistent storage*.

2. You can deploy the initial control plane without activating the **cinderVolumes** service. To deploy the service, you must set the number of **replicas** for the service and configure the back end for the service. For information about the recommended replicas for the **cinderVolumes** service and how to configure a back end for the service, see Configuring the volume service in *Configuring persistent storage*.

- Compute service (nova):

```
nova:
  apiOverride:
    route: {}
  template:
    apiServiceTemplate:
      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
    metadataServiceTemplate:
      replicas: 3
      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
    schedulerServiceTemplate:
```

```
  replicas: 3
  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
cellTemplates:
  cell0:
    cellDatabaseAccount: nova-cell0
    cellDatabaseInstance: openstack
    cellMessageBusInstance: rabbitmq
    hasAPIAccess: true
  cell1:
    cellDatabaseAccount: nova-cell1
    cellDatabaseInstance: openstack-cell1
    cellMessageBusInstance: rabbitmq-cell1
    noVNCProxyServiceTemplate:
      enabled: true
      networkAttachments:
      - internalapi
      - ctlplane
    hasAPIAccess: true
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.

- DNS service for the data plane:

```
dns:
  template:
    options: (1)
    - key: server (2)
      values: (3)
      - 192.168.122.1
    - key: server
      values:
      - 192.168.122.2
    override:
      service:
        metadata:
          annotations:
            metallb.universe.tf/address-pool: ctlplane
            metallb.universe.tf/allow-shared-ip: ctlplane
            metallb.universe.tf/loadBalancerIPs: 192.168.122.80
        spec:
          type: LoadBalancer
    replicas: 2
```

1. Defines the dnsmasq instances required for each DNS server by using key-value pairs. In this example, there are two key-value pairs defined because there are two DNS servers configured to forward requests to.

2. Specifies the dnsmasq parameter to customize for the deployed dnsmasq instance. Set to one of the following valid values:

   - **server**
   - **rev-server**
   - **srv-host**
   - **txt-record**
   - **ptr-record**
   - **rebind-domain-ok**
   - **naptr-record**
   - **cname**
   - **host-record**
   - **caa-record**
   - **dns-rr**
   - **auth-zone**
   - **synth-domain**
   - **no-negcache**
   - **local**

3. Specifies the values for the dnsmasq parameter. You can specify a generic DNS server as the value, for example, **1.1.1.1**, or a DNS server for a specific domain, for example, **/google.com/8.8.8.8**.

- 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
      replicas: 3
```

- Image service (glance):

```
  glance:
    apiOverrides:
      default:
        route: {}
    template:
      databaseInstance: openstack
      storage:
        storageRequest: 10G
      secret: osp-secret
      keystoneEndpoint: default
      glanceAPIs:
        default:
          replicas: 0 (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
```

1. You can deploy the initial control plane without activating the Image service (glance). To deploy the service, you must set the number of **replicas** for the service and configure the back end for the service. For information about the recommended

replicas for the Image service and how to configure a back end for the service, see Configuring the Image service (glance) in *Configuring persistent storage*.

- Key Management service (barbican):

```
barbican:
  apiOverride:
    route: {}
  template:
    databaseInstance: openstack
    secret: osp-secret
    barbicanAPI:
      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
    barbicanWorker:
      replicas: 3
    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: 1
      secret: osp-secret
    swiftRing:
      ringReplicas: 1
    swiftStorage:
      networkAttachments:
      - storage
      replicas: 1
      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
```

- 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
    replicas: 3
    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: 1
      enabled: false
      aodh:
        databaseAccount: aodh
        databaseInstance: openstack
        passwordSelector:
          aodhService: AodhPassword
        rabbitMqClusterName: rabbitmq
        serviceUser: aodh
        secret: osp-secret
      heatInstance: heat
    ceilometer:
      enabled: true
      secret: osp-secret
    logging:
      enabled: false
      ipaddr: 172.17.0.80
```

1 You must have the **autoscaling** field present, even if autoscaling is disabled.

5. Create the control plane:

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

NOTE

Creating the control plane also creates an **OpenStackClient** pod that you can access through a remote shell (**rsh**) to run OpenStack CLI commands.

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

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-control-plane  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.

NOTE

Creating the control plane also creates an **OpenStackClient** pod that you can access through a remote shell (**rsh**) to run RHOSO CLI commands.

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

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  | https://glance-internal.openstack.svc                         |
| glance       | public    | https://glance-default-public-openstack.apps.ostest.test.metalkube.org
|
+--------------+-----------+---------------------------------------------------------------+
```

3. Exit the **OpenStackClient** pod:

```
$ exit
```

4.3. EXAMPLE OPENSTACKCONTROLPLANE CR

The following example **OpenStackControlPlane** CR is a complete 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
  namespace: openstack
spec:
  secret: osp-secret
  storageClass: your-RHOCP-storage-class 1
  cinder: 2
    apiOverride:
      route: {}
    template:
      databaseInstance: openstack
      secret: osp-secret
      cinderAPI:
        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
      cinderScheduler:
        replicas: 1
      cinderBackup: ③
        networkAttachments:
        - storage
        replicas: 0 # backend needs to be configured to activate the service
      cinderVolumes: ④
        volume1:
          networkAttachments: ⑤
          - storage
          replicas: 0 # backend needs to be configured to activate the service
  nova: ⑥
    apiOverride: ⑦
      route: {}
    template:
      apiServiceTemplate:
        replicas: 3
        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:
        replicas: 3
        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
      schedulerServiceTemplate:
        replicas: 3
        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
    cellTemplates:
      cell0:
        cellDatabaseAccount: nova-cell0
        cellDatabaseInstance: openstack
        cellMessageBusInstance: rabbitmq
        hasAPIAccess: true
      cell1:
        cellDatabaseAccount: nova-cell1
        cellDatabaseInstance: openstack-cell1
        cellMessageBusInstance: rabbitmq-cell1
        noVNCProxyServiceTemplate:
          enabled: true
          networkAttachments:
          - internalapi
          - ctlplane
        hasAPIAccess: true
    secret: osp-secret
dns:
  template:
    options:
    - key: server
      values:
      - 192.168.122.1
    - key: server
      values:
      - 192.168.122.2
    override:
      service:
        metadata:
          annotations:
            metallb.universe.tf/address-pool: ctlplane
            metallb.universe.tf/allow-shared-ip: ctlplane
            metallb.universe.tf/loadBalancerIPs: 192.168.122.80
        spec:
          type: LoadBalancer
    replicas: 2
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
    replicas: 3
glance:
  apiOverrides:
    default:
      route: {}
  template:
    databaseInstance: openstack
    storage:
      storageRequest: 10G
    secret: osp-secret
    keystoneEndpoint: default
    glanceAPIs:
      default:
        replicas: 0 # backend needs to be configured to activate the service
        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: 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
    barbicanWorker:
      replicas: 3
    barbicanKeystoneListener:
      replicas: 1
memcached:
  templates:
      memcached:
        replicas: 3
  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
  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: 1
      swiftRing:
        ringReplicas: 1
      swiftStorage:
        networkAttachments:
        - storage
        replicas: 1
        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
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
    replicas: 3
    secret: osp-secret
rabbitmq: 10
  templates:
    rabbitmq:
      replicas: 3
      override:
        service:
          metadata:
            annotations:
              metallb.universe.tf/address-pool: internalapi
              metallb.universe.tf/loadBalancerIPs: 172.17.0.85 11
          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 12
          spec:
            type: LoadBalancer
telemetry:
  enabled: true
  template:
    metricStorage:
      enabled: true
      monitoringStack:
        alertingEnabled: true
        scrapeInterval: 30s
        storage:
          strategy: persistent
          retention: 24h
          persistent:
```

```
        pvcStorageRequest: 20G
    autoscaling:
      enabled: false
      aodh:
        databaseAccount: aodh
        databaseInstance: openstack
        passwordSelector:
          aodhService: AodhPassword
        rabbitMqClusterName: rabbitmq
        serviceUser: aodh
        secret: osp-secret
      heatInstance: heat
    ceilometer:
      enabled: true
      secret: osp-secret
    logging:
      enabled: false
      ipaddr: 172.17.0.80
```

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 persistent storage guide.

4 The Block Storage service configuration. For more information on configuring storage services, see the Configuring persistent 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 The RabbitMQ instances exposed to an isolated network with distinct IP addresses defined in the **loadBalancerIPs** annotation, as indicated in 11 and 12.

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.

11 The distinct IP address for a RabbitMQ instance that is exposed to an isolated network.

12 The distinct IP address for a RabbitMQ instance that is exposed to an isolated network.

4.4. REMOVING A SERVICE FROM THE CONTROL PLANE

You can completely remove a service and the service database from the control plane after deployment by disabling the service. Many services are enabled by default, which means that the OpenStack Operator creates resources such as the service database and Identity service (keystone) users, even if no service pod is created because **replicas** is set to **0**.

WARNING

Remove a service with caution. Removing a service is not the same as stopping service pods. Removing a service is irreversible. Disabling a service removes the service database and any resources that referenced the service are no longer tracked. Red Hat recommends creating a backup of the service database before removing a service.

Procedure

1. Open the **OpenStackControlPlane** CR file on your workstation.
2. Locate the service you want to remove from the control plane and disable it:

```
  cinder:
    enabled: false
    apiOverride:
      route: {}
      ...
```

3. Update the control plane:

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

4. Wait until RHOCP removes the resource related to the disabled service. Run the following command to check the status:

```
$ oc get openstackcontrolplane -n openstack
NAME        STATUS  MESSAGE
openstack-control-plane  Unknown  Setup started
```

The **OpenStackControlPlane** resource is updated with the disabled service when the status is "Setup complete".

TIP

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

5. Optional: Confirm that the pods from the disabled service are no longer listed by reviewing the pods in the **openstack** namespace:

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

6. Check that the service is removed:

```
$ oc get cinder -n openstack
```

This command returns the following message when the service is successfully removed:

```
No resources found in openstack namespace.
```

7. Check that the API endpoints for the service are removed from the Identity service (keystone):

```
$ oc rsh -n openstack openstackclient
$ openstack endpoint list --service volumev3
```

This command returns the following message when the API endpoints for the service are successfully removed:

```
No service with a type, name or ID of 'volumev3' exists.
```

4.5. 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
- Configuring the Block Storage backup service in *Configuring persistent storage*.
- Configuring the Image service (glance) in *Configuring persistent storage*.

CHAPTER 5. 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. An **OpenStackDataPlaneNodeSet** CR is a logical grouping of nodes of a similar type. A data plane typically consists of multiple **OpenStackDataPlaneNodeSet** CRs to define groups of nodes with different configurations and roles. You can use pre-provisioned or unprovisioned nodes in an **OpenStackDataPlaneNodeSet** CR:

- Pre-provisioned node: You have used your own tooling to install the operating system on the node before adding it to the data plane.
- Unprovisioned node: The node does not have an operating system installed before you add it to the data plane. The node is provisioned by using the Cluster Baremetal Operator (CBO) as part of the data plane creation and deployment process.

NOTE

You cannot include both pre-provisioned and unprovisioned nodes in the same OpenStackDataPlaneNodeSet CR.

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

1. Create a **Secret** CR for each node set 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** CR that triggers the Ansible execution that deploys and configures the software for the specified list of **OpenStackDataPlaneNodeSet** CRs.

The following procedures create two simple node sets, one with pre-provisioned nodes, and one with bare-metal nodes that must be provisioned during the node set deployment. The procedures aim to get you up and running quickly with a data plane environment that you can use to troubleshoot issues and test the environment before adding all the customizations you require. You can add additional node sets to a deployed environment, and you can customize your deployed environment by updating the common configuration in the default **ConfigMap** CR for the service, and by creating custom services. For more information about how to customize your data plane after deployment, see the Customizing the Red Hat OpenStack Services on OpenShift deployment guide.

5.1. PREREQUISITES

- A functional control plane, created with the OpenStack Operator. For more information, see Creating the control plane.
- You are logged on to a workstation that has access to the Red Hat OpenShift Container Platform (RHOCP) cluster as a user with **cluster-admin** privileges.

5.2. CREATING THE DATA PLANE SECRETS

The data plane requires several **Secret** custom resources (CRs) to operate. The **Secret** CRs are used by the data plane nodes for the following functionality:

- To enable secure access between nodes:
  - You must generate an SSH key and create an SSH key **Secret** CR for each key to enable Ansible to manage the RHEL nodes on the data plane. Ansible executes commands with this user and key. You can create an SSH key for each node set in your data plane.
  - You must generate an SSH key and create an SSH key **Secret** CR for each key to enable migration of instances between Compute nodes.
- To register the operating system of the nodes that are not registered to the Red Hat Customer Portal.
- To enable repositories for the nodes.
- To provide access to libvirt.

Prerequisites

- Pre-provisioned nodes are configured with an SSH public key in the **$HOME/.ssh/authorized_keys** file for a user with passwordless **sudo** privileges. For information, see Configuring reserved user and group IDs in the RHEL *Configuring basic system settings* guide.

Procedure

1. For unprovisioned nodes, create the SSH key pair for Ansible:

   ```
   $ 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:

   ```
   $ oc create secret generic dataplane-ansible-ssh-private-key-secret \
   --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 **<key_file_name>** with the name and location of your SSH key pair file.
   - 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 ./nova-migration-ssh-key -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 \
   --save-config \
   ```

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

5. Create a file on your workstation named **secret_subscription.yaml** that contains the **subscription-manager** credentials for registering the operating system of the nodes that are not registered to the Red Hat Customer Portal:

   ```
   apiVersion: v1
   kind: Secret
   metadata:
     name: subscription-manager
   data:
     username: <base64_encoded_username>
     password: <base64_encoded_password>
   ```

6. Create the **Secret** CR:

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

7. Create a file on your workstation named **secret_registry.yaml** that contains the Red Hat registry credentials:

   ```
   apiVersion: v1
   kind: Secret
   metadata:
     name: redhat-registry
   data:
     username: <registry_username>
     password: <registry_password>
   ```

8. Create the **Secret** CR:

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

9. Create a file on your workstation named **secret_libvirt.yaml** to define the libvirt secret:

   ```
   apiVersion: v1
   data:
    LibvirtPassword: <base64_password>
   kind: Secret
   metadata:
    name: libvirt-secret
    namespace: openstack
   type: Opaque
   ```

   - Replace **<base64_password>** with a base64 encoded string with maximum length 63 characters. Use the following command to generate a base64 encoded password:

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

10. Create the **Secret** CR:

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

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

```
$ oc describe secret dataplane-ansible-ssh-private-key-secret
$ oc describe secret nova-migration-ssh-key
$ oc describe secret subscription-manager
$ oc describe secret redhat-registry
$ oc describe secret libvirt-secret
```

5.3. CREATING A SET OF DATA PLANE NODES WITH PRE-PROVISIONED NODES

Define an **OpenStackDataPlaneNodeSet** custom resource (CR) for each logical grouping of pre-provisioned 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 you customize your control plane to include additional Compute cells, you must specify the cell to which the node set is connected. For more information on adding Compute cells, see Connecting an **OpenStackDataPlaneNodeSet** CR to a Compute cell in the *Customizing the Red Hat OpenStack Services on OpenShift deployment* guide.

You use the **nodeTemplate** field to configure the properties that all nodes in an **OpenStackDataPlaneNodeSet** CR share, and the **nodeTemplate.nodes** field for node-specific properties. Node-specific configurations override the inherited values from the **nodeTemplate**.

For an example **OpenStackDataPlaneNodeSet** CR that creates a node set from pre-provisioned Compute nodes, see Example **OpenStackDataPlaneNodeSet** CR for pre-provisioned nodes.

Procedure

1. Create a file on your workstation named **openstack_preprovisioned_node_set.yaml** to define the **OpenStackDataPlaneNodeSet** CR:

```
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneNodeSet
metadata:
  name: openstack-data-plane 1
  namespace: openstack
spec:
  env:
    - name: ANSIBLE_FORCE_COLOR
      value: "True"
```

1 The **OpenStackDataPlaneNodeSet** CR name must be unique, must consist of lower case alphanumeric characters, **-** (hyphen) or **.** (period), must start and end with an alphanumeric character, and must have a maximum length of 20 characters. Update the name in this example to a name that reflects the nodes in the set.

2. Specify the services to apply to the nodes:

```
spec:
  ...
```

```
services:
- bootstrap
- configure-network
- validate-network
- install-os
- configure-os
- ssh-known-hosts
- run-os
- reboot-os
- install-certs
- ovn
- neutron-metadata
- libvirt
- nova
- telemetry
```

NOTE

The order of the services in the list is important. Do not change the order of the service deployments

3. Connect the data plane to the control plane network:

```
spec:
  ...
  networkAttachments:
    - ctlplane
```

4. Specify that the nodes in this set are pre-provisioned:

```
preProvisioned: true
```

5. Add the SSH key secret that you created to enable Ansible to connect to the data plane nodes:

```
nodeTemplate:
  ansibleSSHPrivateKeySecret: <secret-key>
```

- Replace **<secret-key>** with the name of the SSH key **Secret** CR you created for this node set in Creating the data plane secrets, for example, **dataplane-ansible-ssh-private-key-secret**.

6. Create a Persistent Volume Claim (PVC) on your Red Hat OpenShift Container Platform (RHOCP) cluster to store logs. Set the **volumeType** to **Filesystem** and **accessModes** to **ReadWriteOnce**. For information on how to create a PVC, see Understanding persistent storage in the RHOCP *Storage* guide.

7. Enable persistent logging for the data plane nodes:

```
nodeTemplate:
  ansibleSSHPrivateKeySecret: <secret-key>
  extraMounts:
    - extraVolType: Logs
      volumes:
      - name: ansible-logs
```

```
          persistentVolumeClaim:
            claimName: <pvc_name>
        mounts:
        - name: ansible-logs
          mountPath: "/runner/artifacts"
```

- Replace **<pvc_name>** with the name of the Persistent Volume Claim (PVC) storage on your RHOCP cluster.

8. Specify the management network:

```
nodeTemplate:
  ...
  managementNetwork: ctlplane
```

9. Specify the **Secret** CRs used to source the usernames and passwords to register the operating system of the nodes that are not registered to the Red Hat Customer Portal, and enable repositories for your nodes. The following example demonstrates how to register your nodes to CDN. For details on how to register your nodes with Red Hat Satellite 6.13, see Managing Hosts.

```
nodeTemplate:
  ...
  ansible:
    ansibleUser: cloud-admin
    ansiblePort: 22
    ansibleVarsFrom:
      - prefix: subscription_manager_
        secretRef:
          name: subscription-manager
      - prefix: registry_
        secretRef:
          name: redhat-registry
    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-18.0-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
      edpm_bootstrap_release_version_package: []
```

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.

10. Add the network configuration template to apply to your data plane nodes. The following example applies the single NIC VLANs network configuration to the data plane nodes:

```
nodeTemplate:
  ...
```

```
    ansible:
      ...
      ansibleVars:
        ...
        edpm_bootstrap_release_version_package: []
        edpm_network_config_os_net_config_mappings:
          edpm-compute-0:
            nic1: 52:54:04:60:55:22 (1)
        neutron_physical_bridge_name: br-ex
        neutron_public_interface_name: eth0
        edpm_network_config_template: |
          ---
          {% set mtu_list = [ctlplane_mtu] %}
          {% for network in nodeset_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_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 nodeset_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 %}
```

(1) Update the **nic1** to the MAC address assigned to the NIC to use for network configuration on the Compute node.

For alternative templates, see **roles/edpm_network_config/templates**. For more information about data plane network configuration, see Customizing data plane networks in the *Configuring network services* guide.

11. Add the common configuration for the set of nodes in this group under the **nodeTemplate** section. Each node in this **OpenStackDataPlaneNodeSet** inherits this configuration. For information about the properties you can use to configure common node attributes, see **OpenStackDataPlaneNodeSet** CR **spec** properties.

12. Define each node in this node set:

```
nodes:
  edpm-compute-0: 1
    hostName: edpm-compute-0
    networks: 2
      - name: ctlplane
        subnetName: subnet1
        defaultRoute: true
        fixedIP: 192.168.122.100 3
      - 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: 4
        fqdn_internal_api: edpm-compute-0.example.com
  edpm-compute-1:
    hostName: edpm-compute-1
    networks:
      - name: ctlplane
        subnetName: subnet1
        defaultRoute: true
        fixedIP: 192.168.122.101
      - name: internalapi
        subnetName: subnet1
        fixedIP: 172.17.0.101
      - name: storage
        subnetName: subnet1
        fixedIP: 172.18.0.101
      - name: tenant
        subnetName: subnet1
        fixedIP: 172.19.0.101
    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.

- Many **ansibleVars** include **edpm** in the name, which stands for "External Data Plane Management".

For information about the properties you can use to configure node attributes, see **OpenStackDataPlaneNodeSet** CR properties.

13. Save the **openstack_preprovisioned_node_set.yaml** definition file.

14. Create the data plane resources:

```
$ oc create --save-config -f openstack_preprovisioned_node_set.yaml -n openstack
```

15. Verify that the data plane resources have been created by confirming that the status is **SetupReady**:

```
$ oc wait openstackdataplanenodeset openstack-data-plane --for condition=SetupReady --
timeout=10m
```

When the status is **SetupReady** the command returns a **condition met** message, otherwise it returns a timeout error.

For information about the data plane conditions and states, see Data plane conditions and states.

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

```
$ oc get secret | grep openstack-data-plane
dataplanenodeset-openstack-data-plane Opaque 1 3m50s
```

17. Verify the services were created:

```
$ oc get openstackdataplaneservice -n openstack
NAME                AGE
bootstrap           46m
ceph-client         46m
ceph-hci-pre        46m
configure-network   46m
configure-os        46m
...
```

5.3.1. Example **OpenStackDataPlaneNodeSet** CR for pre-provisioned nodes

The following example **OpenStackDataPlaneNodeSet** CR creates a node set from pre-provisioned Compute nodes with some node-specific configuration. Update the name of the **OpenStackDataPlaneNodeSet** CR in this example to a name that reflects the nodes in the set. The **OpenStackDataPlaneNodeSet** CR name must be unique, must consist of lower case alphanumeric characters, **-** (hyphen) or **.** (period), must start and end with an alphanumeric character, and must have a maximum length of 20 characters. Update the name in this example to a name that reflects the nodes in the set.

```
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneNodeSet
metadata:
  name: openstack-data-plane
  namespace: openstack
spec:
  env: 1
    - name: ANSIBLE_FORCE_COLOR
      value: "True"
  services:
  - bootstrap
  - configure-network
  - validate-network
  - install-os
  - configure-os
  - ssh-known-hosts
  - run-os
  - reboot-os
  - install-certs
  - ovn
  - neutron-metadata
  - libvirt
  - nova
  - telemetry
  networkAttachments:
    - ctlplane
  preProvisioned: true 2
  nodeTemplate: 3
    ansibleSSHPrivateKeySecret: dataplane-ansible-ssh-private-key-secret 4
    extraMounts:
      - extraVolType: Logs
        volumes:
        - name: ansible-logs
          persistentVolumeClaim:
            claimName: <pvc_name>
        mounts:
        - name: ansible-logs
          mountPath: "/runner/artifacts"
    managementNetwork: ctlplane
    ansible:
      ansibleUser: cloud-admin 5
      ansiblePort: 22
      ansibleVarsFrom:
        - prefix: subscription_manager_
          secretRef:
            name: subscription-manager
        - prefix: registry_
```

```
     secretRef:
      name: redhat-registry
   ansibleVars: 6
    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-18.0-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
    edpm_bootstrap_release_version_package: []
    edpm_network_config_os_net_config_mappings:
     edpm-compute-1:
      nic1: 52:54:04:60:55:22 7
    neutron_physical_bridge_name: br-ex
    neutron_public_interface_name: eth0
    edpm_network_config_template: |
     ---
     {% set mtu_list = [ctlplane_mtu] %}
     {% for network in nodeset_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_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 nodeset_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 %}
 nodes:
  edpm-compute-0: 8
   hostName: edpm-compute-0
   ansible:
        ansibleHost: 192.168.122.100
        ansibleUser: cloud-admin
        ansibleVars:
          fqdn_internal_api: edpm-compute-0.example.com
      networks:
      - name: ctlplane
        subnetName: subnet1
        defaultRoute: true
        fixedIP: 192.168.122.100
      - 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
    edpm-compute-1:
      hostName: edpm-compute-1
      ansible:
        ansibleHost: 192.168.122.101
        ansibleUser: cloud-admin
        ansibleVars:
          fqdn_internal_api: edpm-compute-1.example.com
      networks:
      - name: ctlplane
        subnetName: subnet1
        defaultRoute: true
        fixedIP: 192.168.122.101
      - name: internalapi
        subnetName: subnet1
        fixedIP: 172.17.0.101
      - name: storage
        subnetName: subnet1
        fixedIP: 172.18.0.101
      - name: tenant
        subnetName: subnet1
        fixedIP: 172.19.0.101
```

1 Optional: A list of environment variables to pass to the pod.

2 Specify that the nodes in this set are pre-provisioned.

3 The common configuration to apply to all nodes in this set of nodes.

4 The name of the secret that you created in Creating the data plane secrets .

5 The user associated with the secret you created in Creating the data plane secrets .

6 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/.

7 The MAC address assigned to the NIC to use for network configuration on the Compute node.

8 The node definition reference, for example, **edpm-compute-0**. Each node in the node set must have a node definition.

5.4. CREATING A SET OF DATA PLANE NODES WITH UNPROVISIONED NODES

Define an **OpenStackDataPlaneNodeSet** custom resource (CR) for each logical grouping of unprovisioned 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 you customize your control plane to include additional Compute cells, you must specify the cell to which the node set is connected. For more information on adding Compute cells, see Connecting an **OpenStackDataPlaneNodeSet** CR to a Compute cell in the *Customizing the Red Hat OpenStack Services on OpenShift deployment* guide.

You use the **nodeTemplate** field to configure the properties that all nodes in an **OpenStackDataPlaneNodeSet** CR share, and the **nodeTemplate.nodes** field for node-specific properties. Node-specific configurations override the inherited values from the **nodeTemplate**.

For more information about provisioning bare-metal nodes, see Provisioning bare-metal data plane nodes.

For an example **OpenStackDataPlaneNodeSet** CR that creates a node set from unprovisioned Compute nodes, see Example **OpenStackDataPlaneNodeSet** CR for unprovisioned nodes.

Prerequisites

- Cluster Baremetal Operator (CBO) is installed and configured for provisioning. For more information, see Provisioning bare-metal data plane nodes.
- A **Provisioning** CR is created in RHOCP. For more information about creating a **Provisioning** CR, see Configuring a provisioning resource to scale user-provisioned clusters in the Red Hat OpenShift Container Platform (RHOCP) *Installing* guide.
- 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.

Procedure

1. Create a file on your workstation named **openstack_unprovisioned_node_set.yaml** to define the **OpenStackDataPlaneNodeSet** CR:

```
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneNodeSet
metadata:
  name: openstack-data-plane 1
  namespace: openstack
spec:
  tlsEnabled: true
  env:
    - name: ANSIBLE_FORCE_COLOR
      value: "True"
```

1 The **OpenStackDataPlaneNodeSet** CR name must be unique, must consist of lower case alphanumeric characters, **-** (hyphen) or **.** (period), must start and end with an alphanumeric character, and must have a maximum length of 20 characters. Update the name in this

example to a name that reflects the nodes in the set.

2. Specify the services to apply to the nodes:

```
spec:
  ...
  services:
  - bootstrap
  - configure-network
  - validate-network
  - install-os
  - configure-os
  - ssh-known-hosts
  - run-os
  - reboot-os
  - install-certs
  - ovn
  - neutron-metadata
  - libvirt
  - nova
  - telemetry
```

NOTE

The order of the services in the list is important. Do not change the order of the service deployments

3. Connect the data plane to the control plane network:

```
spec:
  ...
  networkAttachments:
    - ctlplane
```

4. Define the **baremetalSetTemplate** field to describe the configuration of the bare-metal nodes:

```
  preProvisioned: false
  baremetalSetTemplate:
    deploymentSSHSecret: dataplane-ansible-ssh-private-key-secret
    bmhNamespace: <bmh_namespace>
    cloudUserName: <ansible_ssh_user>
    bmhLabelSelector:
      app: <bmh_label>
    ctlplaneInterface: <interface>
    dnsSearchDomains:
    - osptest.openstack.org
```

- Replace **<bmh_namespace>** with the namespace defined in the corresponding **BareMetalHost** CR for the node, for example, **openshift-machine-api**.

- Replace **<ansible_ssh_user>** with the username of the Ansible SSH user, for example, **cloud-admin**.

- Replace **<bmh_label>** with the label defined in the corresponding **BareMetalHost** CR for the node, for example, **openstack**.
- Replace **<interface>** with the control plane interface the node connects to, for example, **enp6s0**.

5. The BMO manages **BareMetalHost** CRs in the **openshift-machine-api** namespace by default. You must update the **Provisioning** CR to watch all namespaces:

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

6. Add the SSH key secret that you created to enable Ansible to connect to the data plane nodes:

```
nodeTemplate:
  ansibleSSHPrivateKeySecret: <secret-key>
```

- Replace **<secret-key>** with the name of the SSH key **Secret** CR you created in Creating the data plane secrets, for example, **dataplane-ansible-ssh-private-key-secret**.

7. Create a Persistent Volume Claim (PVC) on your RHOCP cluster to store logs. Set the **volumeType** to **Filesystem** and **accessModes** to **ReadWriteOnce**. For information on how to create a PVC, see Understanding persistent storage in the RHOCP *Storage* guide.

8. Enable persistent logging for the data plane nodes:

```
nodeTemplate:
  ansibleSSHPrivateKeySecret: <secret-key>
  extraMounts:
    - extraVolType: Logs
      volumes:
      - name: ansible-logs
        persistentVolumeClaim:
          claimName: <pvc_name>
      mounts:
      - name: ansible-logs
        mountPath: "/runner/artifacts"
```

- Replace **<pvc_name>** with the name of the Persistent Volume Claim (PVC) storage on your RHOCP cluster.

9. Specify the management network:

```
nodeTemplate:
  ...
  managementNetwork: ctlplane
```

10. Specify the **Secret** CRs used to source the usernames and passwords to register the operating system of the nodes that are not registered to the Red Hat Customer Portal, and enable repositories for your nodes. The following example demonstrates how to register your nodes to CDN. For details on how to register your nodes with Red Hat Satellite 6.13, see Managing Hosts.

```
nodeTemplate:
  ansible:
    ansibleUser: cloud-admin
```

```
      ansiblePort: 22
      ansibleVarsFrom:
        - prefix: subscription_manager_
          secretRef:
            name: subscription-manager
        - prefix: registry_
          secretRef:
            name: redhat-registry
      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-18.0-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
        edpm_bootstrap_release_version_package: []
```

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.

11. Add the network configuration template to apply to your data plane nodes. The following example applies the single NIC VLANs network configuration to the data plane nodes:

```
nodeTemplate:
  ...
  ansible:
    ...
    ansibleVars:
      ...
      edpm_bootstrap_release_version_package: []
      edpm_network_config_os_net_config_mappings:
        edpm-compute-0:
          nic1: 52:54:04:60:55:22 1
        edpm-compute-1:
          nic1: 52:54:04:60:55:22
      neutron_physical_bridge_name: br-ex
      neutron_public_interface_name: eth0
      edpm_network_config_template: |
        ---
        {% set mtu_list = [ctlplane_mtu] %}
        {% for network in nodeset_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_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 nodeset_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 %}
```

1 Update the **nic1** to the MAC address assigned to the NIC to use for network configuration on the Compute node.

For alternative templates, see **roles/edpm_network_config/templates**. For more information about data plane network configuration, see Customizing data plane networks in the *Configuring network services* guide.

12. Add the common configuration for the set of nodes in this group under the **nodeTemplate** section. Each node in this **OpenStackDataPlaneNodeSet** inherits this configuration. For information about the properties you can use to configure common node attributes, see **OpenStackDataPlaneNodeSet** CR properties.

13. Define each node in this node set:

```
nodes:
  edpm-compute-0: 1
    hostName: edpm-compute-0
    networks: 2
      - name: ctlplane
        subnetName: subnet1
        defaultRoute: true
        fixedIP: 192.168.122.100 3
      - name: internalapi
        subnetName: subnet1
      - name: storage
        subnetName: subnet1
      - name: tenant
        subnetName: subnet1
    ansible:
      ansibleHost: 192.168.122.100
      ansibleUser: cloud-admin
      ansibleVars: 4
        fqdn_internal_api: edpm-compute-0.example.com
  edpm-compute-1:
      hostName: edpm-compute-1
      networks:
        - name: ctlplane
          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.
- Many **ansibleVars** include **edpm** in the name, which stands for "External Data Plane Management".

For information about the properties you can use to configure node attributes, see **OpenStackDataPlaneNodeSet** CR properties.

14. Save the **openstack_unprovisioned_node_set.yaml** definition file.

15. Create the data plane resources:

```
$ oc create --save-config -f openstack_unprovisioned_node_set.yaml -n openstack
```

16. Verify that the data plane resources have been created by confirming that the status is **SetupReady**:

```
$ oc wait openstackdataplanenodeset openstack-data-plane --for condition=SetupReady --
timeout=10m
```

When the status is **SetupReady** the command returns a **condition met** message, otherwise it returns a timeout error.

For information about the data plane conditions and states, see Data plane conditions and states.

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

```
$ oc get secret -n openstack | grep openstack-data-plane
dataplanenodeset-openstack-data-plane Opaque 1 3m50s
```

18. Verify that the nodes have transitioned to the **provisioned** state:

```
$ oc get bmh
NAME           STATE         CONSUMER               ONLINE   ERROR   AGE
edpm-compute-0  provisioned   openstack-data-plane   true             3d21h
```

19. Verify that the services were created:

```
$ oc get openstackdataplaneservice -n openstack
NAME                  AGE
bootstrap             8m40s
ceph-client           8m40s
ceph-hci-pre          8m40s
configure-network     8m40s
configure-os          8m40s
...
```

5.4.1. Example OpenStackDataPlaneNodeSet CR for unprovisioned nodes

The following example **OpenStackDataPlaneNodeSet** CR creates a node set from unprovisioned Compute nodes with some node-specific configuration. The unprovisioned Compute nodes are provisioned when the node set is created. Update the name of the **OpenStackDataPlaneNodeSet** CR in this example to a name that reflects the nodes in the set. The **OpenStackDataPlaneNodeSet** CR name must be unique, must consist of lower case alphanumeric characters, **-** (hyphen) or **.** (period), must start and end with an alphanumeric character, and must have a maximum length of 20 characters. Update the name in this example to a name that reflects the nodes in the set.

```
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneNodeSet
metadata:
  name: openstack-data-plane
  namespace: openstack
spec:
  env: (1)
    - name: ANSIBLE_FORCE_COLOR
      value: "True"
  services:
  - bootstrap
  - configure-network
  - validate-network
  - install-os
  - configure-os
  - ssh-known-hosts
  - run-os
  - reboot-os
  - install-certs
  - ovn
  - neutron-metadata
  - libvirt
  - nova
  - telemetry
  networkAttachments:
    - ctlplane
  preProvisioned: false 2
  baremetalSetTemplate: 3
    deploymentSSHSecret: dataplane-ansible-ssh-private-key-secret
    bmhNamespace: openshift-machine-api 4
    cloudUserName: <ansible_ssh_user>
    bmhLabelSelector:
      app: openstack 5
    ctlplaneInterface: enp1s0
    dnsSearchDomains:
      - osptest.openstack.org
  nodeTemplate: 6
    ansibleSSHPrivateKeySecret: dataplane-ansible-ssh-private-key-secret 7
    extraMounts:
      - extraVolType: Logs
        volumes:
        - name: ansible-logs
          persistentVolumeClaim:
            claimName: <pvc_name>
        mounts:
        - name: ansible-logs
          mountPath: "/runner/artifacts"
    managementNetwork: ctlplane
    ansible:
      ansibleUser: cloud-admin 8
      ansiblePort: 22
      ansibleVarsFrom:
        - prefix: subscription_manager_
          secretRef:
            name: subscription-manager
        - prefix: registry_
          secretRef:
            name: redhat-registry
      ansibleVars: 9
        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-18.0-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
    edpm_bootstrap_release_version_package: []
    edpm_network_config_os_net_config_mappings:
     edpm-compute-0:
      nic1: 52:54:04:60:55:22 10
     edpm-compute-1:
      nic1: 52:54:04:60:55:22
    neutron_physical_bridge_name: br-ex
    neutron_public_interface_name: eth0
    edpm_network_config_template: |
     ---
     {% set mtu_list = [ctlplane_mtu] %}
     {% for network in nodeset_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_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 nodeset_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 %}
 nodes:
  edpm-compute-0: 11
   hostName: edpm-compute-0
   ansible:
    ansibleHost: 192.168.122.100
    ansibleUser: cloud-admin
    ansibleVars:
     fqdn_internal_api: edpm-compute-0.example.com
   networks: 12
   - name: ctlplane
    subnetName: subnet1
    defaultRoute: true
    fixedIP: 192.168.122.100 13
   - name: internalapi
```

```
        subnetName: subnet1
      - name: storage
        subnetName: subnet1
      - name: tenant
        subnetName: subnet1
  edpm-compute-1:
    hostName: edpm-compute-1
    ansible: 14
      ansibleHost: 192.168.122.101
      ansibleUser: cloud-admin
      ansibleVars:
        fqdn_internal_api: edpm-compute-1.example.com
    networks:
    - name: ctlplane
      subnetName: subnet1
      defaultRoute: true
      fixedIP: 192.168.122.101
    - name: internalapi
      subnetName: subnet1
    - name: storage
      subnetName: subnet1
    - name: tenant
      subnetName: subnet1
```

1 Optional: A list of environment variables to pass to the pod.

2 Specify that the nodes in this set are unprovisioned and must be provisioned when creating the resource.

3 Configure the bare-metal template for bare-metal nodes that must be provisioned when creating the resource.

4 The namespace defined in the corresponding **BareMetalHost** CR for the node.

5 The label defined in the corresponding **BareMetalHost** CR for the node.

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 data plane secrets .

8 The user associated with the secret you created in Creating the data plane 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 MAC address assigned to the NIC to use for network configuration on the Compute node.

11 The node definition reference, for example, **edpm-compute-0**. Each node in the node set must have a node definition.

12 Defines the IPAM and the DNS records for the node.

13 14 Defines the predictable IP addresses for each network.

5.4.2. 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.

5.5. OPENSTACKDATAPLANENODESET CR SPEC PROPERTIES

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

5.5.1. nodeTemplate

Defines the common attributes for the nodes in this **OpenStackDataPlaneNodeSet**. You can override these common attributes in the definition for each individual node.

Table 5.1. **nodeTemplate** properties

Field	Description

Field	Description
ansibleSSHPrivateKeySecret	Name of the private SSH key secret that contains the private SSH key for connecting to nodes. Secret name format: Secret.data.ssh-privatekey For more information, see Creating an SSH authentication secret. Default: **dataplane-ansible-ssh-private-key-secret**
managementNetwork	Name of the network to use for management (SSH/Ansible). Default: **ctlplane**
networks	Network definitions for the **OpenStackDataPlaneNodeSet**.
ansible	Ansible configuration options. For more information, see **ansible** properties.
extraMounts	The files to mount into an Ansible Execution Pod.
userData	UserData configuration for the **OpenStackDataPlaneNodeSet**.
networkData	NetworkData configuration for the **OpenStackDataPlaneNodeSet**.

5.5.2. nodes

Defines the node names and node-specific attributes for the nodes in this **OpenStackDataPlaneNodeSet**. Overrides the common attributes defined in the **nodeTemplate**.

Table 5.2. **nodes** properties

Field	Description
ansible	Ansible configuration options. For more information, see **ansible** properties.
extraMounts	The files to mount into an Ansible Execution Pod.
hostName	The node name.
managementNetwork	Name of the network to use for management (SSH/Ansible).
networkData	NetworkData configuration for the node.

Field	Description
networks	Instance networks.
preprovisioningNetworkDataName	NetworkData secret name in the local namespace for pre-provisioning.
userData	Node-specific user data.

5.5.3. ansible

Defines the group of Ansible configuration options.

Table 5.3. **ansible** properties

Field	Description
ansibleUser	The user associated with the secret you created in Creating the data plane secrets. Default: **rhel-user**
ansibleHost	SSH host for the Ansible connection.
ansiblePort	SSH port for the Ansible connection.
ansibleVars	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 **edpm-ansible** role. For a complete list of Ansible variables by role, see the **edpm-ansible** documentation. NOTE The **ansibleVars** parameters that you can configure for an **OpenStackDataPlaneNodeSet** CR are determined by the services defined for the **OpenStackDataPlaneNodeSet**. The **OpenStackDataPlaneService** CRs call the Ansible playbooks from the **edpm-ansible** playbook collection, which include the roles that are executed as part of the data plane service.
ansibleVarsFrom	A list of sources to populate Ansible variables from. Values defined by an **AnsibleVars** with a duplicate key take precedence. For more information, see **ansibleVarsFrom** properties.

5.5.4. ansibleVarsFrom

Defines the list of sources to populate Ansible variables from.

Table 5.4. **ansibleVarsFrom** properties

Field	Description
prefix	An optional identifier to prepend to each key in the **ConfigMap**. Must be a C_IDENTIFIER.
configMapRef	The **ConfigMap** CR to select the **ansibleVars** from.
secretRef	The **Secret** CR to select the **ansibleVars** from.

5.6. DEPLOYING THE DATA PLANE

You use the **OpenStackDataPlaneDeployment** CRD to configure the services on the data plane nodes and deploy the data plane. You control the execution of Ansible on the data plane by creating **OpenStackDataPlaneDeployment** custom resources (CRs). Each **OpenStackDataPlaneDeployment** CR models a single Ansible execution. When the **OpenStackDataPlaneDeployment** successfully completes execution, it does not automatically execute the Ansible again, even if the **OpenStackDataPlaneDeployment** or related **OpenStackDataPlaneNodeSet** resources are changed. To start another Ansible execution, you must create another **OpenStackDataPlaneDeployment** CR.

Create an **OpenStackDataPlaneDeployment** (CR) that deploys each of your **OpenStackDataPlaneNodeSet** CRs.

Procedure

1. Create a file on your workstation named **openstack_data_plane_deploy.yaml** to define the **OpenStackDataPlaneDeployment** CR:

```
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneDeployment
metadata:
  name: data-plane-deploy (1)
  namespace: openstack
```

(1) The **OpenStackDataPlaneDeployment** CR name must be unique, must consist of lower case alphanumeric characters, **-** (hyphen) or **.** (period), must start and end with an alphanumeric character, and must have a maximum length of 20 characters. Update the name in this example to a name that reflects the node sets in the deployment.

2. Add all the **OpenStackDataPlaneNodeSet** CRs that you want to deploy:

```
spec:
  nodeSets:
    - openstack-data-plane
    - <nodeSet_name>
    - ...
    - <nodeSet_name>
```

- Replace **<nodeSet_name>** with the names of the **OpenStackDataPlaneNodeSet** CRs that you want to include in your data plane deployment.

3. Save the **openstack_data_plane_deploy.yaml** deployment file.

4. Deploy the data plane:

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

   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
   ```

5. Verify that the data plane is deployed:

   ```
   $ oc get openstackdataplanedeployment -n openstack
   NAME             STATUS   MESSAGE
   data-plane-deploy   True     Setup Complete


   $ oc get openstackdataplanenodeset -n openstack
   NAME             STATUS   MESSAGE
   openstack-data-plane   True     NodeSet Ready
   ```

   For information about 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.

6. 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**.

7. 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
   ```

5.7. 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 5.5. OpenStackDataPlaneNodeSet CR conditions

Condition	Description
Ready	• "True": The **OpenStackDataPlaneNodeSet** CR is successfully deployed. • "False": The deployment is not yet requested or has failed, or there are other failed conditions.
SetupReady	"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.
DeploymentReady	"True": The NodeSet has been successfully deployed.
InputReady	"True": The required inputs are available and ready.
NodeSetDNSDataReady	"True": DNSData resources are ready.
NodeSetIPReservationReady	"True": The IPSet resources are ready.
NodeSetBaremetalProvisionReady	"True": Bare-metal nodes are provisioned and ready.

Table 5.6. **OpenStackDataPlaneNodeSet** status fields

Status field	Description
Deployed	• "True": The **OpenStackDataPlaneNodeSet** CR is successfully deployed. • "False": The deployment is not yet requested or has failed, or there are other failed conditions.
DNSClusterAddresses	
CtlplaneSearchDomain	

Table 5.7. **OpenStackDataPlaneDeployment** CR conditions

Condition	Description

Condition	Description
Ready	• "True": The data plane is successfully deployed. • "False": The data plane deployment failed, or there are other failed conditions.
DeploymentReady	"True": The data plane is successfully deployed.
InputReady	"True": The required inputs are available and ready.
<NodeSet> Deployment Ready	"True": The deployment has succeeded for the named **NodeSet**, indicating all services for the**NodeSet** have succeeded.
<NodeSet> <Service> Deployment Ready	"True": The deployment has succeeded for the named **NodeSet** and **Service**. Each **<NodeSet> <Service> Deployment Ready** specific condition is set to "True" as that service completes successfully for the named **NodeSet**. Once all services are complete for a **NodeSet**, the **<NodeSet> Deployment Ready** condition is set to "True". The service conditions indicate which services have completed their deployment, or which services failed and for which **NodeSets**.

Table 5.8. **OpenStackDataPlaneDeployment** status fields

Status field	Description
Deployed	• "True": The data plane is successfully deployed. All Services for all NodeSets have succeeded. • "False": The deployment is not yet requested or has failed, or there are other failed conditions.

Table 5.9. **OpenStackDataPlaneService** CR conditions

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

5.8. TROUBLESHOOTING DATA PLANE CREATION AND DEPLOYMENT

To troubleshoot a deployment when services are not deploying or operating correctly, you can check the job condition message for the service, and you can check the logs for a node set.

5.8.1. Checking the job condition message for a service

Each data plane deployment in the environment has associated services. Each of these services have a job condition message that matches 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
   data-plane-deploy ["openstack-data-plane-1"] False   Deployment in progress
   data-plane-deploy ["openstack-data-plane-2"] 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     Job complete
   download-cache-openstack-edpm ["ctlplane"]         False    Job is running
   repo-setup-openstack-edpm     ["ctlplane"]         True     Job complete
   validate-network-another-osdpd ["ctlplane"]        False    Job is running
   ```

   For information on the job condition messages, see Job condition messages.

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

   ```
   $ oc get openstackansibleee -l \
   openstackdataplanedeployment=<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 **data-plane-deploy** deployment:

     ```
     $ oc get openstackansibleee -l \
     openstackdataplanedeployment=data-plane-deploy

     NAME                          NETWORKATTACHMENTS   STATUS   MESSAGE
     ```

```
bootstrap-openstack-edpm        ["ctlplane"]         True     Job complete
download-cache-openstack-edpm   ["ctlplane"]         False    Job is running
repo-setup-openstack-edpm       ["ctlplane"]         True     Job complete
```

5.8.1.1. 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:

- **Job not started**: The job has not started.
- **Job not found**: The job could not be found.
- **Job is running**: The job is currently running.
- **Job complete**: The job execution is complete.
- **Job error occured <error_message>**: The job stopped executing unexpectedly. The **<error_message>** is replaced with a specific error message.

To further investigate a service that is displaying a particular job condition message, view its logs by using the command **oc logs job/<service>**. For example, to view the logs for the **repo-setup-openstack-edpm** service, use the command **oc logs job/repo-setup-openstack-edpm**.

5.8.2. Checking the logs for a node set

You can access the logs for a node set to check for deployment issues.

Procedure

1. Retrieve pods with the **OpenStackAnsibleEE** label:

```
$ oc get pods -l app=openstackansibleee
configure-network-edpm-compute-j6r4l   0/1     Completed          0         3m36s
validate-network-edpm-compute-6g7n9    0/1     Pending            0         0s
validate-network-edpm-compute-6g7n9    0/1     ContainerCreating  0         11s
validate-network-edpm-compute-6g7n9    1/1     Running            0         13s
```

2. SSH into the pod you want to check:
   a. Pod that is running:

   ```
   $ oc rsh validate-network-edpm-compute-6g7n9
   ```

   b. Pod that is not running:

   ```
   $ oc debug configure-network-edpm-compute-j6r4l
   ```

3. List the directories in the **/runner/artifacts** mount:

```
$ ls /runner/artifacts
configure-network-edpm-compute
validate-network-edpm-compute
```

4. View the **stdout** for the required artifact:

```
$ cat /runner/artifacts/configure-network-edpm-compute/stdout
```

CHAPTER 6. 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.

6.1. 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 networking services*

6.2. 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.