Red Hat OpenStack Services on OpenShift 18.0

Customizing the Red Hat OpenStack Services on OpenShift deployment

Customizing a deployed 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 Customizing the Red Hat OpenStack Services on OpenShift deployment

Customizing a deployed Red Hat OpenStack Services on OpenShift environment on a Red Hat OpenShift Container Platform cluster

OpenStack Team
rhos-docs@redhat.com

Legal Notice

Abstract

Learn how to customize your Red Hat OpenStack Services on OpenShift control plane on a Red Hat OpenShift Container Platform cluster, and customize the 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. CUSTOMIZING 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. You can customize your deployed control plane with the services required for your environment.

1.1. PREREQUISITES

- The RHOSO environment is deployed on a RHOCP cluster. For more information, see Deploying Red Hat OpenStack Services on OpenShift .
- You are logged on to a workstation that has access to the RHOCP cluster, as a user with **cluster-admin** privileges.

1.2. ENABLING DISABLED SERVICES

If you enable a service that is disabled by setting **enabled: true**, you must either create an empty template for the service by adding **template: {}** to the service definition to ensure that the default values for the service are set, or specify some or all of the template parameter values. For example, to enable the Dashboard service (horizon) with the default service values, add the following configuration to your **OpenStackControlPlane** custom resource (CR):

```
spec:
  ...
  horizon:
    apiOverride: {}
    enabled: true
    template: {}
```

If you want to set the values for specific service parameters, then add the following configuration to your **OpenStackControlPlane** custom resource (CR):

```
spec:
  ...
  horizon:
    apiOverride: {}
    enabled: true
    template:
      customServiceConfig: ""
      memcachedInstance: memcached
      override: {}
      preserveJobs: false
      replicas: 2
      resources: {}
      secret: osp-secret
      tls: {}
```

Any parameters that you do not specify are set to the default value from the service template.

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

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

Procedure

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

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

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

3. Create the network that the **ironic** service pod attaches to, for example, **baremetal**. For more information about how to create an isolated network, see Preparing RHOCP for RHOSO networks in the *Deploying Red Hat OpenStack Services on OpenShift* guide.
4. Enable and configure the **ironic** service:

```
spec:
  ...
  ironic:
    enabled: true
    template:
      rpcTransport: oslo
      databaseInstance: openstack
      ironicAPI:
        replicas: 1
        override:
          service:
            internal:
              metadata:
                annotations:
                  metallb.universe.tf/address-pool: internalapi
                  metallb.universe.tf/allow-shared-ip: internalapi
                  metallb.universe.tf/loadBalancerIPs: 172.17.0.80
```

```
        spec:
          type: LoadBalancer
  ironicConductors:
  - replicas: 1
    storageRequest: 10G
    networkAttachments:
    - baremetal (1)
    provisionNetwork: baremetal
    customServiceConfig: |
      [neutron]
      cleaning_network = provisioning
      provisioning_network = provisioning
      rescuing_network = provisioning
  ironicInspector:
    replicas: 0
    networkAttachments:
    - baremetal
    inspectionNetwork: baremetal
  ironicNeutronAgent:
    replicas: 1
  secret: osp-secret
```

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

5. Update the control plane:

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

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

```
$ oc get openstackcontrolplane -n openstack
NAME       STATUS  MESSAGE
openstack-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.

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

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

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

Verification

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

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

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

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

3. Exit the **openstackclient** pod:

```
$ exit
```

1.4. ADDING COMPUTE CELLS TO THE CONTROL PLANE

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

By default, the control plane creates two cells:

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

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

Procedure

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

```
apiVersion: core.openstack.org/v1beta1
kind: OpenStackControlPlane
metadata:
  name: openstack-control-plane
  namespace: openstack
spec:
  secret: osp-secret
  ...
  galera:
    enabled: true
    templates:
      openstack: 1
        storageRequest: 5G
        secret: cell0-secret
```

```
    replicas: 1
  openstack-cell1: (2)
    storageRequest: 5G
    secret: cell1-secret
    replicas: 1
  openstack-cell2: (3)
    storageRequest: 5G
    secret: cell2-secret
    replicas: 1
```

(1) The database used by most of the RHOSO services, including the Compute services **nova-api** and **nova-scheduler**, and **cell0**.

(2) The database to be used by **cell1**.

(3) The database to be used by **cell2**.

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

```
apiVersion: core.openstack.org/v1beta1
kind: OpenStackControlPlane
metadata:
  name: openstack-control-plane
spec:
  secret: osp-secret
  ...
  rabbitmq:
    templates:
      rabbitmq: (1)
        override:
          service:
            metadata:
              annotations:
                metallb.universe.tf/address-pool: internalapi
                metallb.universe.tf/loadBalancerIPs: 172.17.0.85
            spec:
              type: LoadBalancer
      rabbitmq-cell1: (2)
        override:
          service:
            metadata:
              annotations:
                metallb.universe.tf/address-pool: internalapi
                metallb.universe.tf/loadBalancerIPs: 172.17.0.86
            spec:
              type: LoadBalancer
      rabbitmq-cell2: (3)
        override:
          service:
            metadata:
              annotations:
                metallb.universe.tf/address-pool: internalapi
```

```
      metallb.universe.tf/loadBalancerIPs: 172.17.0.87
  spec:
    type: LoadBalancer
```

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

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

```
nova:
  apiOverride:
    route: {}
  template:
    ...
    secret: osp-secret
    apiDatabaseAccount: nova-api
    cellTemplates:
      cell0:
        hasAPIAccess: true
        cellDatabaseAccount: nova-cell0
        cellDatabaseInstance: openstack
        cellMessageBusInstance: rabbitmq
      cell1:
        hasAPIAccess: true
        cellDatabaseAccount: nova-cell1
        cellDatabaseInstance: openstack-cell1
        cellMessageBusInstance: rabbitmq-cell1
      cell2: 1
        hasAPIAccess: true
        cellDatabaseAccount: nova-cell2
        cellDatabaseInstance: openstack-cell2
        cellMessageBusInstance: rabbitmq-cell2
```

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

```
$ oc describe crd nova
```

5. Update the control plane:

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

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

```
$ oc get openstackcontrolplane -n openstack
NAME       STATUS  MESSAGE
openstack-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.

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

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

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

8. Optional: Confirm that the new cells are created:

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

1.5. ENABLING THE DASHBOARD SERVICE (HORIZON) INTERFACE

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

Procedure

1. Open your **OpenStackControlPlane** custom resource (CR) file, **openstack_control_plane.yaml**, on your workstation.

2. Enable and configure the **horizon** service:

```
spec:
  ...
  horizon:
    apiOverride: {}
    enabled: true
    template:
```

```
    customServiceConfig: ""
    memcachedInstance: memcached
    override: {}
    preserveJobs: false
    replicas: 2 1
    resources: {}
    secret: osp-secret
    tls: {}
```

1 Set **replicas** to a minimum of **2** for high availability.

3. Update the control plane:

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

4. 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.

5. 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.

6. Retrieve the Dashboard service endpoint URL:

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

Use this URL to access the Horizon interface.

Verification

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

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

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

1.6. ENABLING THE ORCHESTRATION SERVICE (HEAT)

You can enable the Orchestration service (heat) in your Red Hat OpenStack Services on OpenShift (RHOSO) environment. Cloud users can use the Orchestration service to create and manage cloud resources such as storage, networking, instances, or applications.

Procedure

1. Open your **OpenStackControlPlane** custom resource (CR) file, **openstack_control_plane.yaml**, on your workstation.
2. Enable and configure the **heat** service:

```
spec:
  ...
  heat:
    apiOverride:
      route: {}
    cnfAPIOverride:
      route: {}
    enabled: true
    template:
      databaseAccount: heat
      databaseInstance: openstack
      heatAPI:
        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
        resources: {}
        tls:
          api:
            internal: {}
            public: {}
      heatCfnAPI:
        override: {}
        replicas: 1
        resources: {}
        tls:
          api:
            internal: {}
            public: {}
      heatEngine:
        replicas: 1
        resources: {}
      memcachedInstance: memcached
      passwordSelectors:
        authEncryptionKey: HeatAuthEncryptionKey
        service: HeatPassword
```

```
      preserveJobs: false
      rabbitMqClusterName: rabbitmq
      secret: osp-secret
      serviceUser: heat
```

3. Update the control plane:

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

4. 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.

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

3. Exit the **openstackclient** pod:

```
$ exit
```

1.7. ADDITIONAL RESOURCES

- Kubernetes NMState Operator

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

CHAPTER 2. CUSTOMIZING THE DATA PLANE

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

You can add additional node sets to your data plane by using the procedures in Creating the data plane in the *Deploying Red Hat OpenStack Services on OpenShift* guide.

You can also modify existing **OpenStackDataPlaneNodeSet** CRs, add Compute cells to your data plane, and customize your data plane by creating custom services.

2.1. PREREQUISITES

- The RHOSO environment is deployed on a Red Hat OpenShift Container Platform (RHOCP) cluster. For more information, see Deploying Red Hat OpenStack Services on OpenShift .
- You are logged on to a workstation that has access to the RHOCP cluster as a user with **cluster-admin** privileges.

2.2. MODIFYING AN OPENSTACKDATAPLANENODESET CR

You can modify an existing **OpenStackDataPlaneNodeSet** custom resource (CR), for example, to add a new node or update node configuration. Each node can be included in only one **OpenStackDataPlaneNodeSet** CR. Each node set can be connected to only one Compute cell. By default, node sets are connected to **cell1**. If your control plane includes additional Compute cells, you must specify the cell to which the node set is connected.

To apply the **OpenStackDataPlaneNodeSet** CR modifications to the data plane, you create an **OpenStackDataPlaneDeployment** CR that deploys the modified **OpenStackDataPlaneNodeSet** CR.

NOTE

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.

Procedure

1. Open the **OpenStackDataPlaneNodeSet** CR definition file for the node set you want to update, for example, **openstack_data_plane.yaml**.
2. Update or add the configuration you require. For information about the properties you can use to configure common node attributes, see **OpenStackDataPlaneNodeSet** CR spec properties in the *Deploying Red Hat OpenStack Services on OpenShift* guide.
3. Save the **OpenStackDataPlaneNodeSet** CR definition file.
4. Apply the updated **OpenStackDataPlaneNodeSet** CR configuration:

```
$ oc apply -f openstack_data_plane.yaml
```

5. Verify that the data plane resource has been updated 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 in *Deploying Red Hat OpenStack Services on OpenShift* .

6. Create a file on your workstation to define the **OpenStackDataPlaneDeployment** CR:

```
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneDeployment
metadata:
  name: <node_set_deployment_name>
```

TIP

Give the definition file and the **OpenStackDataPlaneDeployment** CR a unique and descriptive name that indicates the purpose of the modified node set.

7. Add the **OpenStackDataPlaneNodeSet** CR that you modified:

```
spec:
  nodeSets:
    - <nodeSet_name>
```

8. Save the **OpenStackDataPlaneDeployment** CR deployment file.

9. Deploy the modified **OpenStackDataPlaneNodeSet** CR:

```
$ 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
```

10. Verify that the modified **OpenStackDataPlaneNodeSet** CR is deployed:

```
$ oc get openstackdataplanedeployment -n openstack
NAME              STATUS   MESSAGE
openstack-data-plane   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 in the *Deploying Red Hat OpenStack Services on OpenShift* guide.

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 in the *Deploying Red Hat OpenStack Services on OpenShift* guide.

11. If you added a new node to the node set, then map the node to the Compute cell it is connected to:

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

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

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

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

2.3. DATA PLANE SERVICES

A data plane service is an Ansible execution that manages the installation, configuration, and execution of a software deployment on data plane nodes. Each service is a resource instance of the **OpenStackDataPlaneService** custom resource definition (CRD), which combines Ansible content and configuration data from **ConfigMap** and **Secret** CRs. You specify the Ansible execution for your service with Ansible play content, which can be an Ansible playbook from **edpm-ansible**, or any Ansible play content. The **ConfigMap** and **Secret** CRs can contain any configuration data that needs to be consumed by the Ansible content.

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

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

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

Service	Description

ceph-client	Include this service to configure data plane nodes as clients of a Red Hat Ceph Storage server. Include between the **install-os** and **configure-os** services. The **OpenStackDataPlaneNodeSet** CR must include the following configuration to access the Red Hat Ceph Storage secrets:
	
	`apiVersion: dataplane.openstack.org/v1beta1`
	`kind: OpenStackDataPlaneNodeSet`
	`spec:`
	`  ...`
	`  nodeTemplate:`
	`  extraMounts:`
	`  - extraVolType: Ceph`
	`  volumes:`
	`  - name: ceph`
	`  secret:`
	`  secretName: ceph-conf-files`
	`  mounts:`
	`  - name: ceph`
	`  mountPath: "/etc/ceph"`
	`  readOnly: true`

ceph-hci-pre	Include this service to prepare data plane nodes to host Red Hat Ceph Storage in a HCI configuration. For more information, see *Deploying a Hyperconverged Infrastructure environment*
neutron-dhcp	Include this service to run a Neutron DHCP agent on the data plane nodes.
neutron-metadata	Include this service to run the Neutron OVN Metadata agent on the data plane nodes. This agent is required to provide metadata services to the Compute nodes.
neutron-ovn	Include this service to run the Neutron OVN agent on the data plane nodes. This agent is required to provide QoS to hardware offloaded ports on the Compute nodes.
neutron-sriov	Include this service to run a Neutron SR-IOV NIC agent on the data plane nodes.

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

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

NOTE

Do not change the order of the default service deployments.

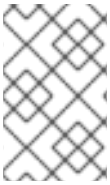
You can use the **OpenStackDataPlaneService** CRD to create a custom service that you can deploy on your data plane nodes. You add your custom service to the default list of services where the service must be executed. For more information, see Creating and enabling a custom service .

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

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

2.3.1. Creating and enabling a custom service

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



NOTE

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

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

NOTE

You cannot include an Ansible **playbook** and **playbookContents** in the same service.

Procedure

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

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

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

   - Specify the Ansible playbook to use:

```
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneService
metadata:
  name: custom-service
spec:
  playbook: osp.edpm.configure_os
```

   - Specify the Ansible play in the **playbookContents** field as a string that uses Ansible playbook syntax:

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

```
kind: OpenStackDataPlaneService
metadata:
  name: custom-service
spec:
  playbookContents: |
    - hosts: all
      tasks:
        - name: Hello World!
          shell: "echo Hello World!"
          register: output
        - name: Show output
          debug:
            msg: "{{ output.stdout }}"
        - name: Hello World role
          import_role: hello_world
```

For information about how to create an Ansible playbook, see Creating a playbook.

3. Optional: To override the default container image used by the **ansible-runner** execution environment with a custom image that uses additional Ansible content for a custom service, build and include a custom **ansible-runner** image. For information, see Building a custom **ansible-runner** image.

4. Optional: Specify the names of **Secret** or **ConfigMap** resources to use to pass secrets or configurations into the **OpenStackAnsibleEE** job:

```
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneService
metadata:
  name: custom-service
spec:
  ...
  playbookContents: |
    ...
  dataSources:
    - configMapRef:
        name: hello-world-cm-0
    - secretRef:
        name: hello-world-secret-0
    - secretRef:
        name: hello-world-secret-1
        optional: true (1)
```

(1) Optional: Set the **optional** field to "true" to mark the resource as optional so that an error is not thrown if it doesn't exist.

A mount is created for each **Secret** and **ConfigMap** CR in the **OpenStackAnsibleEE** pod with a filename that matches the resource value. The mounts are created under **/var/lib/openstack/configs/<service name>**. You can then use Ansible content to access the configuration or secret data.

5. Optional: Set the **deployOnAllNodeSets** field to true if the service must run on all node sets in the **OpenStackDataPlaneDeployment** CR, even if the service is not listed as a service in every node set in the deployment:

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

  playbookContents: |
  ...
  deployOnAllNodeSets: true
```

6. Optional: Specify the **edpmServiceType** field for the service. Different custom services may use the same Ansible content to manage the same data plane service, for example, **ovn** or **nova**. You must mount the DataSources, TLS certificates, and CA certificates at the same location so that Ansible content can locate them even when using a custom service. You use the **edpmServiceType** field to create this association. The value is the name of the default service that uses the same Ansible content as the custom service. For example, a custom service that uses the **edpm_ovn** Ansible content from **edpm-ansible** would set **edpmServiceType** to **ovn**, which matches the default **ovn** service name provided by the OpenStack Operator.

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

NOTE

The acroynm **edpm** used in fieldnames stands for "External Data Plane Management".

7. Create the custom service:

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

8. Verify that the custom service is created:

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

9. Add the custom service to the **services** field in the definition file for the node sets the service applies to. Add the service name in the order that it should be executed relative to the other services. If the **deployAllNodeSets** field is set to **true**, then you need to add the service to only one of the node sets in the deployment.

NOTE

When adding your custom service to the services list in a node set definition, you must include all the required services, including the default services. If you include only your custom service in the services list, then that is the only service that is deployed.

2.3.2. Building a custom ansible-runner image

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

Procedure

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

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

2. Build and push the image to a container registry:

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

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

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

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

2.4. CONFIGURING A NODE SET FOR A FEATURE OR WORKLOAD

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

1. Create the **ConfigMap** custom resources (CRs) to configure the nodes for the feature.
2. Create a custom service for the node set that runs the playbook for the service.
3. Include the **ConfigMap** CRs in the custom service.

NOTE

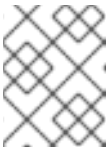
The Compute service (nova) provides a default **ConfigMap** CR named **nova-extra-config**, where you can add generic configuration that applies to all the node sets that use the default **nova** service. If you use this default **nova-extra-config ConfigMap** to add generic configuration to be applied to all the node sets, then you do not need to create a custom service.

Procedure

1. Create a **ConfigMap** CR that defines a new configuration file for the feature:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: feature-configmap
  namespace: openstack
data:
  <integer>-<feature>.conf: |
    <[config_grouping]>
    <config_option> = <value>
    <config_option> = <value>
```

- Replace **<integer>** with a number that indicates when to apply the configuration. The control plane services apply every file in their service directory, **/etc/<service>/<service>.conf.d/**, in lexicographical order. Therefore, configurations defined in later files override the same configurations defined in an earlier file. Each service operator generates the default configuration file with the name **01-<service>.conf**. For example, the default configuration file for the **nova-operator** is **01-nova.conf**.



NOTE

Numbers below 25 are reserved for the OpenStack services and Ansible configuration files.

- Replace **<feature>** with a string that indicates the feature being configured.

NOTE

Do not use the name of the default configuration file, because it would override the infrastructure configuration, such as the **transport_url**.

- Replace **<[config_grouping]>** with the name of the group the configuration options belong to in the service configuration file. For example, **[compute]** or **database**.
- Replace **<config_option>** with the option you want to configure, for example, **cpu_shared_set**.
- Replace **<value>** with the value for the configuration option, for example, **2,6**.
  When the service is deployed, it adds the configuration to the **etc/<service>/<service>.conf.d/** directory in the service container. For example, for a Compute feature, the configuration file is added to **etc/nova/nova.conf.d/** in the **nova_compute** container.

  For more information on creating **ConfigMap** objects, see Creating and using config maps in the RHOCP *Nodes* guide.

TIP

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

2. Create a custom service for the node set. For information about how to create a custom service, see Creating and enabling a custom service .

3. Add the **ConfigMap** CR to the custom service:

```
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneService
metadata:
  name: <nodeset>-service
spec:
  ...
  dataSources:
    - configMapRef:
        name: feature-configmap
```

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

```
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneService
metadata:
  name: <nodeset>-service
spec:
  ...
  dataSources:
    - configMapRef:
        name: feature-configmap
    - secretRef:
        name: nova-migration-ssh-key
    - secretRef:
        name: nova-cell1-compute-config
```

2.5. CONNECTING AN OPENSTACKDATAPLANENODESET CR TO A COMPUTE CELL

Each node set can be connected to only one Compute cell. By default, node sets are connected to **cell1**. If you added additional Compute cells to your control plane, you must specify to which cell the node set connects.

Procedure

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

```
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneService
metadata:
  name: nova-cell-custom
  spec:
    playbook: osp.edpm.nova
    ...
    dataSources:
    - secretRef:
        name: nova-cell2-compute-config (1)
    edpmServiceType: nova (2)
```

1. The **Secret** CR generated by the control plane for the cell.
2. Associate the **OpenStackDataPlaneService** CR with the **nova** service.

For information about how to create a custom service, see Creating and enabling a custom service.

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

```
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneNodeSet
metadata:
  name: openstack-cell2
spec:
  services:
    - download-cache
    - bootstrap
    - configure-network
    - validate-network
    - install-os
    - configure-os
    - ssh-known-hosts
    - run-os
    - ovn
    - libvirt
    - *nova-cell-custom*
    - telemetry
```

NOTE

Do not change the order of the default services.

3. Complete the configuration of your **OpenStackDataPlaneNodeSet** CR. For more information, see Creating the data plane.
4. Save the **OpenStackDataPlaneNodeSet** CR definition file.
5. Create the data plane resources:

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

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

```
$ oc wait openstackdataplanenodeset openstack-cell2 --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 in *Deploying Red Hat OpenStack Services on OpenShift* .

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

```
$ oc get secret | grep openstack-cell2
dataplanenodeset-openstack-cell2 Opaque 1 3m50s
```

8. Verify the services were created:

```
$ oc get openstackdataplaneservice -n openstack | grep nova-cell-custom
```

9. Create an **OpenStackDataPlaneDeployment** CR to deploy the **OpenStackDataPlaneNodeSet** CR. For more information, see Deploying the data plane in the *Deploying Red Hat OpenStack Services on OpenShift* guide.

CHAPTER 3. CUSTOMIZING RED HAT OPENSTACK SERVICES ON OPENSHIFT OBSERVABILITY

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

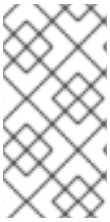
The observability architecture in RHOSO is composed of services within OpenShift, as well as services on your Compute nodes that expose metrics, logs, and alerts.You can use the OpenShift observability ecosystem for insight into the RHOSO environment. Additionally, you have access to the logging infrastructure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

3.1. CONFIGURING RED HAT OPENSTACK SERVICES ON OPENSHIFT OBSERVABILITY

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

Prerequisites

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



NOTE

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

Procedure

1. Open the **OpenStackControlPlane** CR definition file, **openstack_control_plane.yaml**, on your workstation.
2. Update the **telemetry** section based on the needs of your environment:

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

```
    monitoringStack:
      alertingEnabled: true
      scrapeInterval: 30s (1)
      storage:
        strategy: persistent
        retention: 24h (2)
        persistent:
          pvcStorageRequest: 20G (3)
  autoscaling:
    enabled: false
    aodh:
      databaseAccount: aodh
      databaseInstance: openstack
      secret: osp-secret
    heatInstance: heat
  ceilometer:
    enabled: true
    secret: osp-secret
  logging:
    enabled: false
    ipaddr: <ip_address>
```

(1) Use the **scrapeInterval** field to control the amount of time that passes before new metrics are gathered. Changing this parameter can affect performance.

(2) Use the **retention** field to adjust the length of time telemetry metrics are stored. This field affects the amount of storage required.

(3) Use the **pvcStorageRequest** field to change the amount of storage to be allocated for the Prometheus time series database.

3. Update the control plane:

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

4. 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.

5. Optional: Confirm that the control plane is deployed by reviewing the pods in the **openstack** namespace for each of your cells:

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

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

Verification

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

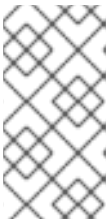
```
$ oc rsh -n openstack openstackclient
```

2. Confirm that you can query **prometheus** and that the scrape endpoints are active:

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

Example output:

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



NOTE

Each entry in the value field should be "1" when there are active workloads scheduled on the cluster, except for the **prometheus** container. The **prometheus** container reports a value of "0" due to TLS, which is enabled by default.

3. You can find RHOSO telemetry dashboards by clicking **Observe** and then **Dashboards** in the RHOCP console.

Additional resource

- Installing Logging