Red Hat OpenStack Services on OpenShift 18.0-beta

Adopting a Red Hat OpenStack Platform 17.1 deployment

Adopt a Red Hat OpenStack Platform 17.1 overcloud to a Red Hat OpenStack Services on OpenShift 18.0 data plane

Last Updated: 2024-06-25

Red Hat OpenStack Services on OpenShift 18.0-beta Adopting a Red Hat OpenStack Platform 17.1 deployment

Adopt a Red Hat OpenStack Platform 17.1 overcloud to a Red Hat OpenStack Services on OpenShift 18.0 data plane

OpenStack Team
rhos-docs@redhat.com

Legal Notice

Abstract

You can migrate your existing Red Hat OpenStack Platform 17.1 overcloud to a Red Hat OpenStack Services on OpenShift 18.0 data plane.

Table of Contents

CHAPTER 1. PLANNING THE NEW DEPLOYMENT

Just like you did back when you installed your director-deployed Red Hat OpenStack Platform, the upgrade/migration to the control plane requires planning various aspects of the environment such as node roles, planning your network topology, and storage.

This document covers some of this planning, but it is recommended to read the whole adoption guide before actually starting the process to be sure that there is a global understanding of the whole process.

1.1. SERVICE CONFIGURATIONS

There is a fundamental difference between the director and operator deployments regarding the configuration of the services.

In director deployments many of the service configurations are abstracted by director-specific configuration options. A single director option may trigger changes for multiple services and support for drivers, for example, the Block Storage service (cinder), that require patches to the director code base.

In operator deployments this approach has changed: reduce the installer specific knowledge and leverage Red Hat OpenShift Container Platform (RHOCP) and Red Hat OpenStack Platform (RHOSP) service specific knowledge whenever possible.

To this effect RHOSP services will have sensible defaults for RHOCP deployments and human operators will provide configuration snippets to provide necessary configuration, such as the Block Storage service backend configuration, or to override the defaults.

This shortens the distance between a service specific configuration file (such as **cinder.conf**) and what the human operator provides in the manifests.

These configuration snippets are passed to the operators in the different **customServiceConfig** sections available in the manifests, and then they are layered in the services available in the following levels. To illustrate this, if you were to set a configuration at the top Block Storage service level (**spec: cinder: template:**) then it would be applied to all the Block Storage services; for example to enable debug in all the Block Storage services you would do:

```
apiVersion: core.openstack.org/v1beta1
kind: OpenStackControlPlane
metadata:
  name: openstack
spec:
  cinder:
    template:
      customServiceConfig: |
        [DEFAULT]
        debug = True
< . . . >
```

If you only want to set it for one of the Block Storage services, for example the scheduler, then you use the **cinderScheduler** section instead:

```
apiVersion: core.openstack.org/v1beta1
kind: OpenStackControlPlane
metadata:
  name: openstack
spec:
```

```
  cinder:
    template:
      cinderScheduler:
        customServiceConfig: |
          [DEFAULT]
          debug = True
< . . . >
```

In Red Hat OpenShift Container Platform it is not recommended to store sensitive information like the credentials to the Block Storage service storage array in the CRs, so most RHOSP operators have a mechanism to use the Red Hat OpenShift Container Platform **Secrets** for sensitive configuration parameters of the services and then use them by reference in the **customServiceConfigSecrets** section which is analogous to the **customServiceConfig**.

The contents of the **Secret** references passed in the **customServiceConfigSecrets** will have the same format as **customServiceConfig**: a snippet with the section/s and configuration options.

When there are sensitive information in the service configuration then it becomes a matter of personal preference whether to store all the configuration in the **Secret** or only the sensitive parts. However, if you split the configuration between **Secret** and **customServiceConfig** you still need the section header (eg: **[DEFAULT]**) to be present in both places.

Attention should be paid to each service's adoption process as they may have some particularities regarding their configuration.

1.2. ABOUT NODE ROLES

In director deployments you had 4 different standard roles for the nodes: **Controller**, **Compute**, **Ceph Storage**, **Swift Storage**, but in the control plane you make a distinction based on where things are running, in Red Hat OpenShift Container Platform (RHOCP) or external to it.

When adopting a director Red Hat OpenStack Platform (RHOSP) your **Compute** nodes will directly become external nodes, so there should not be much additional planning needed there.

In many deployments being adopted the **Controller** nodes will require some thought because you have many RHOCP nodes where the Controller services could run, and you have to decide which ones you want to use, how you are going to use them, and make sure those nodes are ready to run the services.

In most deployments running RHOSP services on **master** nodes can have a seriously adverse impact on the RHOCP cluster, so it is recommended that you place RHOSP services on non **master** nodes.

By default RHOSP Operators deploy RHOSP services on any worker node, but that is not necessarily what's best for all deployments, and there may be even services that won't even work deployed like that.

When planing a deployment it's good to remember that not all the services on an RHOSP deployments are the same as they have very different requirements.

Looking at the Block Storage service (cinder) component you can clearly see different requirements for its services: the cinder-scheduler is a very light service with low memory, disk, network, and CPU usage; cinder-api service has a higher network usage due to resource listing requests; the cinder-volume service will have a high disk and network usage since many of its operations are in the data path (offline volume migration, create volume from image, etc.), and then you have the cinder-backup service which has high memory, network, and CPU (to compress data) requirements.

The Image Service (glance) and Object Storage service (swift) components are in the data path, as well as RabbitMQ and Galera services.

Given these requirements it may be preferable not to let these services wander all over your RHOCP worker nodes with the possibility of impacting other workloads, or maybe you don't mind the light services wandering around but you want to pin down the heavy ones to a set of infrastructure nodes.

There are also hardware restrictions to take into consideration, because if you are using a Fibre Channel (FC) Block Storage service backend you need the cinder-volume, cinder-backup, and maybe even the Image Service (glance) (if it's using the Block Storage service as a backend) services to run on a RHOCP host that has an HBA.

The RHOSP Operators allow a great deal of flexibility on where to run the RHOSP services, as you can use node labels to define which RHOCP nodes are eligible to run the different RHOSP services. Refer to the About node selector to learn more about using labels to define placement of the RHOSP services.

1.3. ABOUT NODE SELECTOR

There are a variety of reasons why you might want to restrict the nodes where Red Hat OpenStack Platform (RHOSP) services can be placed:

- Hardware requirements: System memory, Disk space, Cores, HBAs
- Limit the impact of the RHOSP services on other Red Hat OpenShift Container Platform workloads.
- Avoid collocating RHOSP services.

The mechanism provided by the RHOSP operators to achieve this is through the use of labels.

You either label the RHOCP nodes or use existing labels, and then use those labels in the RHOSP manifests in the **nodeSelector** field.

The **nodeSelector** field in the RHOSP manifests follows the standard RHOCP **nodeSelector** field. For more information, see About node selectors in *OpenShift Container Platform 4.15 Documentation*.

This field is present at all the different levels of the RHOSP manifests:

- Deployment: The **OpenStackControlPlane** object.
- Component: For example the **cinder** element in the **OpenStackControlPlane**.
- Service: For example the **cinderVolume** element within the **cinder** element in the **OpenStackControlPlane**.

This allows a fine grained control of the placement of the RHOSP services with minimal repetition.

Values of the **nodeSelector** are propagated to the next levels unless they are overwritten. This means that a **nodeSelector** value at the deployment level will affect all the RHOSP services.

For example, you can add label **type: openstack** to any 3 RHOCP nodes:

```
$ oc label nodes worker0 type=openstack
$ oc label nodes worker1 type=openstack
$ oc label nodes worker2 type=openstack
```

And then in our **OpenStackControlPlane** you can use the label to place all the services in those 3 nodes:

```
apiVersion: core.openstack.org/v1beta1
kind: OpenStackControlPlane
metadata:
  name: openstack
spec:
  secret: osp-secret
  storageClass: local-storage
  nodeSelector:
    type: openstack
< . . . >
```

You can use the selector for specific services. For example, you might want to place your the Block Storage service (cinder) volume and backup services on certain nodes if you are using FC and only have HBAs on a subset of nodes. The following example assumes that you have the label **fc_card: true**:

```
apiVersion: core.openstack.org/v1beta1
kind: OpenStackControlPlane
metadata:
  name: openstack
spec:
  secret: osp-secret
  storageClass: local-storage
  cinder:
    template:
      cinderVolumes:
        pure_fc:
          nodeSelector:
            fc_card: true
< . . . >
        lvm-iscsi:
          nodeSelector:
            fc_card: true
< . . . >
      cinderBackup:
        nodeSelector:
          fc_card: true
< . . . >
```

The Block Storage service operator does not currently have the possibility of defining the **nodeSelector** in **cinderVolumes**, so you need to specify it on each of the backends.

It is possible to leverage labels added by the Node Feature Discovery (NFD) Operator to place RHOSP services. For more information, see Node Feature Discovery Operator in *OpenShift Container Platform 4.15 Documentation*.

1.4. ABOUT MACHINE CONFIGS

Some services require you to have services or kernel modules running on the hosts where they run, for example **iscsid** or **multipathd** daemons, or the **nvme-fabrics** kernel module.

For those cases you use **MachineConfig** manifests, and if you are restricting the nodes that you are placing the Red Hat OpenStack Platform services on using the **nodeSelector** then you also want to limit where the **MachineConfig** is applied.

To define where the **MachineConfig** can be applied, you need to use a **MachineConfigPool** that links the **MachineConfig** to the nodes.

For example to be able to limit **MachineConfig** to the 3 Red Hat OpenShift Container Platform (RHOCP) nodes that you marked with the **type: openstack** label, you create the **MachineConfigPool** like this:

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  name: openstack
spec:
  machineConfigSelector:
    matchLabels:
      machineconfiguration.openshift.io/role: openstack
  nodeSelector:
    matchLabels:
      type: openstack
```

And then you use it in the **MachineConfig**:

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: openstack
< . . . >
```

Refer to the Postinstallation machine configuration tasks in *OpenShift Container Platform 4.15 Documentation*.

WARNING

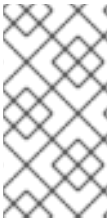
Applying a **MachineConfig** to an RHOCP node makes the node reboot.

1.5. KEY MANAGER SERVICE SUPPORT FOR CRYPTO PLUG-INS

The Key Manager service (barbican) does not yet support all of the crypto plug-ins available in director.

1.6. CONFIGURING THE NETWORK FOR THE RHOSO DEPLOYMENT

With Red Hat OpenShift Container Platform (RHOCP), the network is a very important aspect of the deployment, and it is important to plan it carefully. The general network requirements for the Red Hat OpenStack Platform (RHOSP) services are not much different from the ones in a director deployment, but the way you handle them is.



NOTE

For more information about the network architecture and configuration, see *Deploying Red Hat OpenStack Platform 18.0 Development Preview 3 on Red Hat OpenShift Container Platform* and About networking in *OpenShift Container Platform 4.15 Documentation*. This document will address concerns specific to adoption.

When adopting a new RHOSP deployment, it is important to align the network configuration with the adopted cluster to maintain connectivity for existing workloads.

The following logical configuration steps will incorporate the existing network configuration:

- configure **RHOCP worker nodes** to align VLAN tags and IPAM configuration with the existing deployment.
- configure **Control Plane services** to use compatible IP ranges for service and load balancing IPs.
- configure **Data Plane nodes** to use corresponding compatible configuration for VLAN tags and IPAM.

Specifically,

- IPAM configuration will either be reused from the **existing** deployment or, depending on IP address availability in the existing allocation pools, **new** ranges will be defined to be used for the new control plane services. If so, **IP routing** will be configured between the old and new ranges. For more information, see Planning your IPAM configuration .
- VLAN tags will be reused from the existing deployment.

1.6.1. Retrieving the network configuration from your existing deployment

Let's first determine which isolated networks are defined in the existing deployment. You can find the network configuration in the **network_data.yaml** file. For example,

```
- name: InternalApi
  mtu: 1500
  vip: true
  vlan: 20
  name_lower: internal_api
  dns_domain: internal.mydomain.tld.
  service_net_map_replace: internal
  subnets:
    internal_api_subnet:
      ip_subnet: '172.17.0.0/24'
      allocation_pools: [{'start': '172.17.0.4', 'end': '172.17.0.250'}]
```

You should make a note of the VLAN tag used (**vlan** key) and the IP range (**ip_subnet** key) for each isolated network. The IP range will later be split into separate pools for control plane services and load balancer IP addresses.

You should also determine the list of IP addresses already consumed in the adopted environment. Consult **tripleo-ansible-inventory.yaml** file to find this information. In the file, for each listed host, note IP and VIP addresses consumed by the node.

For example,

```
Standalone:
  hosts:
    standalone:
      ...
      internal_api_ip: 172.17.0.100
    ...
  ...
standalone:
  children:
    Standalone: {}
  vars:
    ...
    internal_api_vip: 172.17.0.2
    ...
```

In the example above, note that the **172.17.0.2** and **172.17.0.100** are consumed and won't be available for the new control plane services, at least until the adoption is complete.

Repeat the process for each isolated network and each host in the configuration.

At the end of this process, you should have the following information:

- A list of isolated networks used in the existing deployment.
- For each of the isolated networks, the VLAN tag and IP ranges used for dynamic address allocation.
- A list of existing IP address allocations used in the environment. You will later exclude these addresses from allocation pools available for the new control plane services.

1.6.2. Planning your IPAM configuration

The new deployment model puts additional burden on the size of IP allocation pools available for Red Hat OpenStack Platform (RHOSP) services. This is because each service deployed on Red Hat OpenShift Container Platform (RHOCP) worker nodes will now require an IP address from the IPAM pool (in the previous deployment model, all services hosted on a controller node shared the same IP address.)

Since the new control plane deployment has different requirements as to the number of IP addresses available for services, it may even be impossible to reuse the existing IP ranges used in adopted environment, depending on its size. Prudent planning is required to determine which options are available in your particular case.

The total number of IP addresses required for the new control plane services, in each isolated network, is calculated as a sum of the following:

- The number of RHOCP worker nodes. (Each node will require 1 IP address in **NodeNetworkConfigurationPolicy** custom resources (CRs).)
- The number of IP addresses required for the data plane nodes. (Each node will require an IP address from **NetConfig** CRs.)
- The number of IP addresses required for control plane services. (Each service will require an IP address from **NetworkAttachmentDefinition** CRs.) This number depends on the number of replicas for each service.

- The number of IP addresses required for load balancer IP addresses. (Each service will require a VIP address from **IPAddressPool** CRs.)

As of the time of writing, the simplest single worker node RHOCP deployment (CRC) has the following IP ranges defined (for the **internalapi** network):

- 1 IP address for the single worker node;
- 1 IP address for the data plane node;
- **NetworkAttachmentDefinition** CRs for control plane services: **X.X.X.30-X.X.X.70** (41 addresses);
- **IPAllocationPool** CRs for load balancer IPs: **X.X.X.80-X.X.X.90** (11 addresses).

Which comes to a total of 54 IP addresses allocated to the **internalapi** allocation pools.

The exact requirements may differ depending on the list of RHOSP services to be deployed, their replica numbers, as well as the number of RHOCP worker nodes and data plane nodes.

Additional IP addresses may be required in future RHOSP releases, so it is advised to plan for some extra capacity, for each of the allocation pools used in the new environment.

Once you know the required IP pool size for the new deployment, you can choose one of the following scenarios to handle IPAM allocation in the new environment.

The first listed scenario is more general and implies using new IP ranges, while the second scenario implies reusing the existing ranges. The end state of the former scenario is using the new subnet ranges for control plane services, but keeping the old ranges, with their node IP address allocations intact, for data plane nodes.

Regardless of the IPAM scenario, the VLAN tags used in the existing deployment will be reused in the new deployment. Depending on the scenario, the IP address ranges to be used for control plane services will be either reused from the old deployment or defined anew. Adjust the configuration as described in Configuring isolated networks.

1.6.2.1. Scenario 1: Using new subnet ranges

This scenario is compatible with any existing subnet configuration, and can be used even when the existing cluster subnet ranges don't have enough free IP addresses for the new control plane services.

The general idea here is to define new IP ranges for control plane services that belong to a different subnet that was not used in the existing cluster. Then, configure link local IP routing between the old and new subnets to allow old and new service deployments to communicate. This involves using director mechanism on pre-adopted cluster to configure additional link local routes there. This will allow EDP deployment to reach out to adopted nodes using their old subnet addresses.

The new subnet should be sized appropriately to accommodate the new control plane services, but otherwise doesn't have any specific requirements as to the existing deployment allocation pools already consumed. Actually, the requirements as to the size of the new subnet are lower than in the second scenario, as the old subnet ranges are kept for the adopted nodes, which means they don't consume any IP addresses from the new range.

In this scenario, you will configure **NetworkAttachmentDefinition** custom resources (CRs) to use a different subnet from what will be configured in **NetConfig** CR for the same networks. The former range will be used for control plane services, while the latter will be used to manage IPAM for data plane nodes.

During the process, you will need to make sure that adopted node IP addresses don't change during the adoption process. This is achieved by listing the addresses in **fixedIP** fields in **OpenstackDataplaneNodeSet** per-node section.

Before proceeding, configure host routes on the adopted nodes for the control plane subnets.

To achieve this, you will need to re-run **tripleo deploy** with additional **routes** entries added to **network_config**. (This change should be applied for every adopted node configuration.) For example, you may add the following to **net_config.yaml**:

```
network_config:
  - type: ovs_bridge
    name: br-ctlplane
    routes:
    - ip_netmask: 0.0.0.0/0
      next_hop: 192.168.1.1
    - ip_netmask: 172.31.0.0/24  # <- new ctlplane subnet
      next_hop: 192.168.1.100    # <- adopted node ctlplane IP address
```

Do the same for other networks that will need to use different subnets for the new and old parts of the deployment.

Once done, run **tripleo deploy** to apply the new configuration.

Note that network configuration changes are not applied by default to avoid risk of network disruption. You will have to enforce the changes by setting the **StandaloneNetworkConfigUpdate: true** in the director configuration files.

Once **tripleo deploy** is complete, you should see new link local routes to the new subnet on each node. For example,

```
# ip route | grep 172
172.31.0.0/24 via 192.168.122.100 dev br-ctlplane
```

The next step is to configure similar routes for the old subnet for control plane services attached to the networks. This is done by adding **routes** entries to **NodeNetworkConfigurationPolicy** CRs for each network. For example,

```
    - destination: 192.168.122.0/24
      next-hop-interface: ospbr
```

Once applied, you should eventually see the following route added to your Red Hat OpenShift Container Platform (RHOCP) nodes.

```
# ip route | grep 192
192.168.122.0/24 dev ospbr proto static scope link
```

At this point, you should be able to ping the adopted nodes from RHOCP nodes using their old subnet addresses; and vice versa.

Finally, during the data plane adoption, you will have to take care of several aspects:

- in network_config, add link local routes to the new subnets, for example:

```
nodeTemplate:
  ansible:
    ansibleUser: root
    ansibleVars:
      additional_ctlplane_host_routes:
      - ip_netmask: 172.31.0.0/24
        next_hop: '{{ ctlplane_ip }}'
      edpm_network_config_template: |
        network_config:
        - type: ovs_bridge
          routes: {{ ctlplane_host_routes + additional_ctlplane_host_routes }}
          ...
```

- list the old IP addresses as **ansibleHost** and **fixedIP**, for example:

```
nodes:
  standalone:
    ansible:
      ansibleHost: 192.168.122.100
      ansibleUser: ""
    hostName: standalone
    networks:
    - defaultRoute: true
      fixedIP: 192.168.122.100
      name: ctlplane
      subnetName: subnet1
```

- expand SSH range for the firewall configuration to include both subnets:

```
    edpm_sshd_allowed_ranges:
    - 192.168.122.0/24
    - 172.31.0.0/24
```

This is to allow SSH access from the new subnet to the adopted nodes as well as the old one.

Since you are applying new network configuration to the nodes, consider also setting **edpm_network_config_update: true** to enforce the changes.

Note that the examples above are incomplete and should be incorporated into your general configuration.

1.6.2.2. Scenario 2: Reusing existing subnet ranges

This scenario is only applicable when the existing subnet ranges have enough IP addresses for the new control plane services. On the other hand, it allows to avoid additional routing configuration between the old and new subnets, as in Scenario 1: Using new subnet ranges .

The general idea here is to instruct the new control plane services to use the same subnet as in the adopted environment, but define allocation pools used by the new services in a way that would exclude IP addresses that were already allocated to existing cluster nodes.

This scenario implies that the remaining IP addresses in the existing subnet is enough for the new control plane services. If not, Scenario 1: Using new subnet ranges should be used instead. For more information, see Planning your IPAM configuration .

No special routing configuration is required in this scenario; the only thing to pay attention to is to make sure that already consumed IP addresses don't overlap with the new allocation pools configured for Red Hat OpenStack Platform control plane services.

If you are especially constrained by the size of the existing subnet, you may have to apply elaborate exclusion rules when defining allocation pools for the new control plane services. For more information, see

1.6.3. Configuring isolated networks

At this point, you should have a good idea about VLAN and IPAM configuration you would like to replicate in the new environment.

Before proceeding, you should have a list of the following IP address allocations to be used for the new control plane services:

- 1 IP address, per isolated network, per Red Hat OpenShift Container Platform worker node. (These addresses configure openshift worker nodes to **NodeNetworkConfigurationPolicy** custom resources (CRs).) For more information, see Configuring Red Hat OpenShift Container Platform worker nodes.
- IP range, per isolated network, for the data plane nodes. (These ranges will configure data plane nodes to **NetConfig** CRs.) For more information, see Configuring data plane nodes.
- IP range, per isolated network, for control plane services. (These ranges will enable pod connectivity to isolated networks to **NetworkAttachmentDefinition** CRs.) For more information, see Configuring the networking for control plane services .
- IP range, per isolated network, for load balancer IP addresses. (These ranges will define load balancer IP addresses to **IPAddressPool** CRs for MetalLB.) For more information, see Configuring the networking for control plane services .

IMPORTANT

Make sure you have the information listed above before proceeding with the next steps.

NOTE

The exact list and configuration of isolated networks in the examples listed below should reflect the actual adopted environment. The number of isolated networks may differ from the example below. IPAM scheme may differ. Only relevant parts of the configuration are shown. Examples are incomplete and should be incorporated into the general configuration for the new deployment, as described in the general Red Hat OpenStack Platform documentation.

1.6.3.1. Configuring Red Hat OpenShift Container Platform worker nodes

Red Hat OpenShift Container Platform worker nodes that run Red Hat OpenStack Platform services need a way to connect the service pods to isolated networks. This requires physical network configuration on the hypervisor.

This configuration is managed by the NMState operator, which uses the custom resources (CRs) to define the desired network configuration for the nodes.

For each node, define a **NodeNetworkConfigurationPolicy** CR that describes the desired network configuration. See the example below.

```
apiVersion: v1
items:
- apiVersion: nmstate.io/v1
  kind: NodeNetworkConfigurationPolicy
  spec:
    interfaces:
    - description: internalapi vlan interface
      ipv4:
        address:
        - ip: 172.17.0.10
          prefix-length: 24
        dhcp: false
        enabled: true
      ipv6:
        enabled: false
      name: enp6s0.20
      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
        dhcp: false
        enabled: true
      ipv6:
        enabled: false
      name: enp6s0.21
      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
        dhcp: false
        enabled: true
      ipv6:
        enabled: false
      name: enp6s0.22
      state: up
      type: vlan
      vlan:
        base-iface: enp6s0
        id: 22
```

```
        reorder-headers: true
    nodeSelector:
      kubernetes.io/hostname: ocp-worker-0
      node-role.kubernetes.io/worker: ""
```

1.6.3.2. Configuring the networking for control plane services

Once NMState operator created the desired hypervisor network configuration for isolated networks, we need to configure Red Hat OpenStack Platform (RHOSP) services to use configured interfaces. This is achieved by defining **NetworkAttachmentDefinition** custom resources (CRs) for each isolated network. (In some clusters, these CRs are managed by the Cluster Network Operator in which case **Network** CRs should be used instead. For more information, see Cluster Network Operator in *OpenShift Container Platform 4.15 Documentation*.)

For example,

```
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
spec:
  config: |
    {
      "cniVersion": "0.3.1",
      "name": "internalapi",
      "type": "macvlan",
      "master": "enp6s0.20",
      "ipam": {
        "type": "whereabouts",
        "range": "172.17.0.0/24",
        "range_start": "172.17.0.20",
        "range_end": "172.17.0.50"
      }
    }
```

Make sure that the interface name and IPAM range match the configuration used in **NodeNetworkConfigurationPolicy** CRs.

When reusing existing IP ranges, you may exclude part of the range defined by **range_start** and **range_end** that was already consumed in the existing deployment. Please use **exclude** as follows.

```
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
spec:
  config: |
    {
      "cniVersion": "0.3.1",
      "name": "internalapi",
      "type": "macvlan",
      "master": "enp6s0.20",
      "ipam": {
        "type": "whereabouts",
        "range": "172.17.0.0/24",
        "range_start": "172.17.0.20",
        "range_end": "172.17.0.50",
        "exclude": [
          "172.17.0.24/32",
```

```
        "172.17.0.44/31"
      ]
    }
  }
```

The example above would exclude addresses **172.17.0.24** as well as **172.17.0.44** and **172.17.0.45** from the allocation pool.

Some RHOSP services require load balancer IP addresses. These IP addresses belong to the same IP range as the control plane services, and are managed by MetalLB. The IP address pool is defined by **IPAllocationPool** CRs. This pool should also be aligned with the adopted configuration.

For example,

```
- apiVersion: metallb.io/v1beta1
  kind: IPAddressPool
  spec:
    addresses:
    - 172.17.0.60-172.17.0.70
```

Define **IPAddressPool** CRs for each isolated network that requires load balancer IP addresses.

When reusing existing IP ranges, you may exclude part of the range by listing multiple **addresses** entries.

For example,

```
- apiVersion: metallb.io/v1beta1
  kind: IPAddressPool
  spec:
    addresses:
    - 172.17.0.60-172.17.0.64
    - 172.17.0.66-172.17.0.70
```

The example above would exclude the **172.17.0.65** address from the allocation pool.

1.6.3.3. Configuring data plane nodes

A complete Red Hat OpenStack Platform (RHOSP) cluster consists of Red Hat OpenShift Container Platform (RHOCP) nodes and data plane nodes. The former use **NodeNetworkConfigurationPolicy** custom resource (CR) to configure physical interfaces. Since data plane nodes are not RHOCP nodes, a different approach to configure their network connectivity is used.

Instead, data plane nodes are configured by **dataplane-operator** and its CRs. The CRs define desired network configuration for the nodes.

In case of adoption, the configuration should reflect the existing network setup. You should be able to pull **net_config.yaml** files from each node and reuse it when defining **OpenstackDataplaneNodeSet**. The format of the configuration hasn't changed (**os-net-config** is still being used under the hood), so you should be able to put network templates under **edpm_network_config_template** variables (either common for all nodes, or per-node).

To make sure the latest network configuration is used during the data plane adoption, you should also set **edpm_network_config_update: true** in the **nodeTemplate**.

You will proceed with the data plane adoption once the RHOCP control plane is deployed in the RHOCP cluster. When doing so, you will configure **NetConfig** and **OpenstackDataplaneNodeSet** CRs, using the same VLAN tags and IPAM configuration as determined in the previous steps.

For example,

```
apiVersion: network.openstack.org/v1beta1
kind: NetConfig
metadata:
  name: netconfig
spec:
  networks:
  - name: internalapi
    dnsDomain: internalapi.example.com
    subnets:
    - name: subnet1
      allocationRanges:
      - end: 172.17.0.250
        start: 172.17.0.100
      cidr: 172.17.0.0/24
      vlan: 20
  - 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
```

List multiple **allocationRanges** entries to exclude some of the IP addresses, e.g. to accommodate for addresses already consumed by the adopted environment.

```
apiVersion: network.openstack.org/v1beta1
kind: NetConfig
metadata:
  name: netconfig
spec:
  networks:
  - name: internalapi
    dnsDomain: internalapi.example.com
    subnets:
    - name: subnet1
      allocationRanges:
      - end: 172.17.0.199
        start: 172.17.0.100
```

```
    - end: 172.17.0.250
      start: 172.17.0.201
    cidr: 172.17.0.0/24
    vlan: 20
```

The example above would exclude the **172.17.0.200** address from the pool.

1.7. STORAGE REQUIREMENTS

When looking into the storage in an Red Hat OpenStack Platform (RHOSP) deployment you can differentiate two kinds, the storage requirements of the services themselves and the storage used for the RHOSP users that the services will manage.

These requirements may drive your Red Hat OpenShift Container Platform (RHOCP) node selection, as mentioned above, and may require you to do some preparations on the RHOCP nodes before you can deploy the services.

1.7.1. Storage driver certification

Before you adopt your Red Hat OpenStack Platform 17.1 deployment to a Red Hat OpenStack Services on OpenShift (RHOSO) 18.0 deployment, confirm that your deployed storage drivers are certified for use with RHOSO 18.0.

1.7.2. Block Storage service requirements

The Block Storage service (cinder) has both local storage used by the service and Red Hat OpenStack Platform (RHOSP) user requirements.

Local storage is used for example when downloading a Image Service (glance) image for the create volume from image operation, which can become considerable when having concurrent operations and not using the Block Storage service volume cache.

In the Operator deployed RHOSP, there is a way to configure the location of the conversion directory to be an NFS share (using the extra volumes feature), something that needed to be done manually before.

Even if it's an adoption and it may seem that there's nothing to consider regarding the Block Storage service backends, because you are using the same ones that you are using in your current deployment, you should still evaluate it, because it may not be so straightforward.

First you need to check the transport protocol the Block Storage service backends are using: RBD, iSCSI, FC, NFS, NVMe-oF, etc.

Once you know all the transport protocols that you are using, you can make sure that you are taking them into consideration when placing the Block Storage services (as mentioned above in the Node Roles section) and the right storage transport related binaries are running on the Red Hat OpenShift Container Platform nodes.

Detailed information about the specifics for each storage transport protocol can be found in the Red Hat OpenShift Container Platform preparation for Block Storage service adoption.

1.8. COMPARING CONFIGURATION FILES BETWEEN DEPLOYMENTS

In order to help users to handle the configuration for the director and Red Hat OpenStack Platform services the tool: https://github.com/openstack-k8s-operators/os-diff has been develop to compare the configuration files between the director deployment and the Red Hat OpenStack Services on

OpenShift (RHOSO) cloud. Make sure Golang is installed and configured on your environment:

```
dnf install -y golang-github-openstack-k8s-operators-os-diff
```

Then configure the **/etc/os-diff/os-diff.cfg** file and the **/etc/os-diff/ssh.config** file according to your environment. To allow os-diff to connect to your clouds and pull files from the services that you describe in the **config.yaml** file you need to properly set the option in the **os-diff.cfg** file:

```
[Default]

local_config_dir=/tmp/
service_config_file=config.yaml

[Tripleo]

ssh_cmd=ssh -F ssh.config
director_host=standalone
container_engine=podman
connection=ssh
remote_config_path=/tmp/tripleo
local_config_path=/tmp/

[Openshift]

ocp_local_config_path=/tmp/ocp
connection=local
ssh_cmd=""
```

Os-diff uses **ssh_cmd** to access your director host via SSH, or the host where your cloud is accessible and the podman/docker binary is installed and allowed to interact with the running containers. This option could have a different form:

```
ssh_cmd=ssh -F ssh.config standalone
director_host=
```

```
ssh_cmd=ssh -F ssh.config
director_host=standalone
```

or without an SSH config file:

```
ssh_cmd=ssh -i /home/user/.ssh/id_rsa stack@my.undercloud.local
director_host=
```

or

```
ssh_cmd=ssh -i /home/user/.ssh/id_rsa stack@
director_host=my.undercloud.local
```

Note that the result of using **ssh_cmd** and **director_host** should be a "successful ssh access".

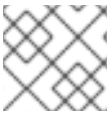
Configure or generate the **ssh.config** file from inventory or hosts file, for example:

```
Host *
```

```
    IdentitiesOnly yes

Host virthost
    Hostname virthost
    IdentityFile ~/.ssh/id_rsa
    User root
    StrictHostKeyChecking no
    UserKnownHostsFile=/dev/null


Host standalone
    Hostname standalone
    IdentityFile <path to SSH key>
    User root
    StrictHostKeyChecking no
    UserKnownHostsFile=/dev/null

Host crc
    Hostname crc
    IdentityFile ~/.ssh/id_rsa
    User stack
    StrictHostKeyChecking no
    UserKnownHostsFile=/dev/null
```

Os-diff can use an **ssh.config** file for getting access to your Red Hat OpenStack Platform environment. The following command can help you generate this SSH config file from your Ansible inventory, for example, **tripleo-ansible-inventory.yaml** file:

```
os-diff configure -i tripleo-ansible-inventory.yaml -o ssh.config --yaml
```



NOTE

You must set the **IdentityFile** key in the file to get full working access:

```
Host standalone
  HostName standalone
  User root
  IdentityFile ~/.ssh/id_rsa
  StrictHostKeyChecking no
  UserKnownHostsFile /dev/null

Host undercloud
  HostName undercloud
  User root
  IdentityFile ~/.ssh/id_rsa
  StrictHostKeyChecking no
  UserKnownHostsFile /dev/null
```

And test your connection:

```
ssh -F ssh.config standalone
```

CHAPTER 2. MIGRATING TLS-E TO THE RHOSO DEPLOYMENT

The Red Hat OpenStack Services on OpenShift (RHOSO) deployment adopts the settings from the Red Hat OpenStack Platform (RHOSP) 17.1 deployment. If TLS everywhere (TLS-e) is disabled in the RHOSP deployment, it is not enabled in the RHOSO deployment.

If the director deployment was deployed with TLS-e, FreeIPA (IdM) is used to issue certificates for the RHOSP services. Certmonger, a client process which is installed on all hosts, interacts with FreeIPA (IdM) to request, install, track and renew these certificates.

The RHOSO Operator-based deployment uses the cert-manager operator to issue, track, and renew the certificates.

Because the same root certificate authority (CA) is used to generate new certificates, you do not have to modify the currently used chain of trust. *Disclaimer: the below steps were reproduced on a FreeIPA 4.10.1 server. The location of files and directories may slightly change on different versions.*

These instructions explain how to extract the CA signing certificate from the FreeIPA instance that is used to provide the certificates in the source environment and import it into certmanager for use in the target environment. In this way, disruption on the Compute nodes can be minimized because a new chain of trust need not be installed.

It is expected that the old FreeIPA node is then decommissioned and no longer used to issue certificates. This might not be possible if the IPA server is used to issue certificates for non-RHOSP systems.

This procedure will also need to be modified if the signing keys are stored in an hardware security module (HSM) instead of an NSS shared database (NSSDB). In that case, if the key is retrievable, special HSM utilities might be required.

Prerequisites

- Your RHOSP deployment is using TLS-e.
- Make sure the previous Adoption steps (if any) have been performed successfully.
- Make sure the backend services on the new deployment are not started yet.
- Define the following shell variables. The values that are used are examples and refer to a single node standalone director deployment. Replace these example values with values that are correct for your environment:

```
IPA_SSH="ssh -i <path_to_ssh_key> root@<freeipa-server-ip-address>"
```

Procedure

1. To locate the CA certificate and key, list all the certificates inside your NSSDB:

```
$IPA_SSH certutil -L -d /etc/pki/pki-tomcat/alias
```

   The **-L** option lists all certificates, and **-d** specifies where they are stored. This will produce some output like this:

```
Certificate Nickname                                         Trust Attributes
                                                             SSL,S/MIME,JAR/XPI

caSigningCert cert-pki-ca                                    CTu,Cu,Cu
ocspSigningCert cert-pki-ca                                  u,u,u
Server-Cert cert-pki-ca                                      u,u,u
subsystemCert cert-pki-ca                                    u,u,u
auditSigningCert cert-pki-ca                                 u,u,Pu
```

The item you need to consider is the first one: **caSigningCert cert-pki-ca**.

2. Export the certificate and key from the **/etc/pki/pki-tomcat/alias** directory:

```
$IPA_SSH pk12util -o /tmp/freeipa.p12 -n 'caSigningCert\ cert-pki-ca' -d /etc/pki/pki-tomcat/alias -k /etc/pki/pki-tomcat/alias/pwdfile.txt -w /etc/pki/pki-tomcat/alias/pwdfile.txt
```

The command generates a P12 file with both the certificate and the key. The **/etc/pki/pki-tomcat/alias/pwdfile.txt** file contains the password that protects the key. You can use it to both extract the key and generate the new file, **/tmp/freeipa.p12**. You can also choose another password. If you choose to apply a different password for the new file, replace the parameter of the **-w** option, or use the **-W** (capital W) option followed by the password (in clear text).

With that file, you can also separately get the certificate and the key by using the **openssl pkcs12** command.

3. Create the secret that contains the root CA:

```
oc create secret generic rootca-internal -n openstack
```

4. Import the certificate and the key from FreeIPA:

```
oc patch secret rootca-internal -n openstack -p="{\"data\":{\"ca.crt\": \"`$IPA_SSH openssl pkcs12 -in /tmp/freeipa.p12 -passin file:/etc/pki/pki-tomcat/alias/pwdfile.txt -nokeys | openssl x509 | base64 -w 0`\"}}"

oc patch secret rootca-internal -n openstack -p="{\"data\":{\"tls.crt\": \"`$IPA_SSH openssl pkcs12 -in /tmp/freeipa.p12 -passin file:/etc/pki/pki-tomcat/alias/pwdfile.txt -nokeys | openssl x509 | base64 -w 0`\"}}"

oc patch secret rootca-internal -n openstack -p="{\"data\":{\"tls.key\": \"`$IPA_SSH openssl pkcs12 -in /tmp/freeipa.p12 -passin file:/etc/pki/pki-tomcat/alias/pwdfile.txt -nocerts -noenc | openssl rsa | base64 -w 0`\"}}"
```

5. Create the cert-manager Issuer and reference the created secret:

```
oc apply -f - <<EOF
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: rootca-internal
  namespace: openstack
  labels:
    osp-rootca-issuer-public: ""
    osp-rootca-issuer-internal: ""
    osp-rootca-issuer-libvirt: ""
```

```
    osp-rootca-issuer-ovn: ""
spec:
  ca:
    secretName: rootca-internal
EOF
```

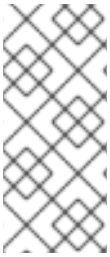
6. Delete the previously created p12 files:

```
$IPA_SSH rm /tmp/freeipa.p12
```

7. Verify that the necessary resources were created by using the following commands:

```
oc get issuers -n openstack
```

```
oc get secret rootca-internal -n openstack -o yaml
```



NOTE

After the adoption procedure is finished, the cert-manager operator is responsible for issuing and refreshing new certificates when they expire. However, since Compute services are not restarted during adoption, you need to restart the data plane (Compute) nodes before the certificates expire. Check the expiration dates of all certificates and plan accordingly.

CHAPTER 3. MIGRATING DATABASES TO THE CONTROL PLANE

To begin creating the control plane, enable backend services and import the databases from your original Red Hat OpenStack Platform 17.1 deployment.

3.1. RETRIEVING TOPOLOGY-SPECIFIC SERVICE CONFIGURATION

Prerequisites

- Define the following shell variables. The values that are used are examples. Replace these example values with values that are correct for your environment:

```
CONTROLLER_SSH="ssh -F ~/director_standalone/vagrant_ssh_config
vagrant@standalone"
MARIADB_IMAGE=registry.redhat.io/rhosp-dev-preview/openstack-mariadb-rhel9:18.0
SOURCE_MARIADB_IP=172.17.0.2
SOURCE_DB_ROOT_PASSWORD=$(cat ~/tripleo-standalone-passwords.yaml | grep '
MysqlRootPassword:' | awk -F ': ' '{ print $2; }')
MARIADB_CLIENT_ANNOTATIONS='--annotations=k8s.v1.cni.cncf.io/networks=internalapi'
```

Procedure

1. Export shell variables for the following outputs to compare it with post-adoption values later on: Test connection to the original database:

```
export PULL_OPENSTACK_CONFIGURATION_DATABASES=$(oc run mariadb-client
${MARIADB_CLIENT_ANNOTATIONS} -q --image ${MARIADB_IMAGE} -i --rm --
restart=Never -- \
    mysql -rsh "$SOURCE_MARIADB_IP" -uroot -p"$SOURCE_DB_ROOT_PASSWORD" -e
'SHOW databases;')
echo "$PULL_OPENSTACK_CONFIGURATION_DATABASES"
```

   Note that the **nova**, **nova_api**, **nova_cell0** databases reside in the same DB host.

2. Run **mysqlcheck** on the original database to look for inaccuracies:

```
export PULL_OPENSTACK_CONFIGURATION_MYSQLCHECK_NOK=$(oc run mariadb-
client ${MARIADB_CLIENT_ANNOTATIONS} -q --image ${MARIADB_IMAGE} -i --rm --
restart=Never -- \
    mysqlcheck --all-databases -h $SOURCE_MARIADB_IP -u root -
p"$SOURCE_DB_ROOT_PASSWORD" | grep -v OK)
echo "$PULL_OPENSTACK_CONFIGURATION_MYSQLCHECK_NOK"
```

3. Get the Compute service (nova) cells mappings from the database:

```
export PULL_OPENSTACK_CONFIGURATION_NOVADB_MAPPED_CELLS=$(oc run
mariadb-client ${MARIADB_CLIENT_ANNOTATIONS} -q --image ${MARIADB_IMAGE} -i --
rm --restart=Never -- \
    mysql -rsh "${SOURCE_MARIADB_IP}" -uroot -p"${SOURCE_DB_ROOT_PASSWORD}"
nova_api -e \
    'select uuid,name,transport_url,database_connection,disabled from cell_mappings;')
echo "$PULL_OPENSTACK_CONFIGURATION_NOVADB_MAPPED_CELLS"
```

4. Get the host names of the registered Compute services:

```
export PULL_OPENSTACK_CONFIGURATION_NOVA_COMPUTE_HOSTNAMES=$(oc
run mariadb-client ${MARIADB_CLIENT_ANNOTATIONS} -q --image ${MARIADB_IMAGE}
-i --rm --restart=Never -- \
    mysql -rsh "$SOURCE_MARIADB_IP" -uroot -p"$SOURCE_DB_ROOT_PASSWORD"
nova_api -e \
    "select host from nova.services where services.binary='nova-compute';")
echo "$PULL_OPENSTACK_CONFIGURATION_NOVA_COMPUTE_HOSTNAMES"
```

5. Get the list of mapped the Compute service cells:

```
export
PULL_OPENSTACK_CONFIGURATION_NOVAMANAGE_CELL_MAPPINGS=$($CONTRO
LLER_SSH sudo podman exec -it nova_api nova-manage cell_v2 list_cells)
echo "$PULL_OPENSTACK_CONFIGURATION_NOVAMANAGE_CELL_MAPPINGS"
```

After the source control plane services shutdown, if either of the exported values lost, it could be no longer evaluated again. Preserving the exported values in an environment file should protect you from such a situation.

6. Store exported variables for future use:

```
cat > ~/.source_cloud_exported_variables << EOF
PULL_OPENSTACK_CONFIGURATION_DATABASES="$(oc run mariadb-client
${MARIADB_CLIENT_ANNOTATIONS} -q --image ${MARIADB_IMAGE} -i --rm --
restart=Never -- \
    mysql -rsh $SOURCE_MARIADB_IP -uroot -p$SOURCE_DB_ROOT_PASSWORD -e
'SHOW databases;')"
PULL_OPENSTACK_CONFIGURATION_MYSQLCHECK_NOK="$(oc run mariadb-client
${MARIADB_CLIENT_ANNOTATIONS} -q --image ${MARIADB_IMAGE} -i --rm --
restart=Never -- \
    mysqlcheck --all-databases -h $SOURCE_MARIADB_IP -u root -
p$SOURCE_DB_ROOT_PASSWORD | grep -v OK)"
PULL_OPENSTACK_CONFIGURATION_NOVADB_MAPPED_CELLS="$(oc run mariadb-
client ${MARIADB_CLIENT_ANNOTATIONS} -q --image ${MARIADB_IMAGE} -i --rm --
restart=Never -- \
    mysql -rsh $SOURCE_MARIADB_IP -uroot -p$SOURCE_DB_ROOT_PASSWORD
nova_api -e \
    'select uuid,name,transport_url,database_connection,disabled from cell_mappings;')"
PULL_OPENSTACK_CONFIGURATION_NOVA_COMPUTE_HOSTNAMES="$(oc run
mariadb-client ${MARIADB_CLIENT_ANNOTATIONS} -q --image ${MARIADB_IMAGE} -i --
rm --restart=Never -- \
    mysql -rsh $SOURCE_MARIADB_IP -uroot -p$SOURCE_DB_ROOT_PASSWORD
nova_api -e \
    "select host from nova.services where services.binary='nova-compute';")"
PULL_OPENSTACK_CONFIGURATION_NOVAMANAGE_CELL_MAPPINGS="$($CONTR
OLLER_SSH sudo podman exec -it nova_api nova-manage cell_v2 list_cells)"
EOF
chmod 0600 ~/.source_cloud_exported_variables
```

7. Optional: If there are neutron-sriov-nic-agent agents running in the deployment, get its configuration:

```
podman run -i --rm --userns=keep-id -u $UID $MARIADB_IMAGE mysql \
    -rsh "$SOURCE_MARIADB_IP" -uroot -p"$SOURCE_DB_ROOT_PASSWORD"
ovs_neutron -e \
    "select host, configurations from agents where agents.binary='neutron-sriov-nic-agent';"
```

This configuration will be required later, during the data plane adoption.

3.2. DEPLOYING BACKEND SERVICES

Create the **OpenStackControlPlane** custom resource (CR) with basic backend services deployed, and all the Red Hat OpenStack Platform (RHOSP) services disabled. This will be the foundation of the control plane.

In subsequent steps, you import the original databases and then add RHOSP control plane services.

Prerequisites

- The cloud that you want to adopt is up and running, and it is on the RHOSP 17.1 release.
- All control plane and data plane hosts of the source cloud are up and running, and continue to run throughout the adoption procedure.
- The **openstack-operator** is deployed, but **OpenStackControlPlane** is not deployed. For production environments, the deployment method will likely be different.
- If TLS Everywhere is enabled on the source environment, the **tls** root CA from the source environment must be copied over to the rootca-internal issuer.
- There are free PVs available to be claimed (for MariaDB and RabbitMQ).
- Set the desired admin password for the control plane deployment. This can be the original deployment's admin password or something else.

  ```
  ADMIN_PASSWORD=SomePassword
  ```

  To use the existing RHOSP deployment password:

  ```
  ADMIN_PASSWORD=$(cat ~/tripleo-standalone-passwords.yaml | grep ' AdminPassword:' |
  awk -F ': ' '{ print $2; }')
  ```

- Set service password variables to match the original deployment. Database passwords can differ in the control plane environment, but synchronizing the service account passwords is a required step.
  For example, in developer environments with director Standalone, the passwords can be extracted like this:

  ```
  AODH_PASSWORD=$(cat ~/tripleo-standalone-passwords.yaml | grep ' AodhPassword:' |
  awk -F ': ' '{ print $2; }')
  BARBICAN_PASSWORD=$(cat ~/tripleo-standalone-passwords.yaml | grep '
  BarbicanPassword:' | awk -F ': ' '{ print $2; }')
  CEILOMETER_METERING_SECRET=$(cat ~/tripleo-standalone-passwords.yaml | grep '
  CeilometerMeteringSecret:' | awk -F ': ' '{ print $2; }')
  CEILOMETER_PASSWORD=$(cat ~/tripleo-standalone-passwords.yaml | grep '
  CeilometerPassword:' | awk -F ': ' '{ print $2; }')
  ```

```
CINDER_PASSWORD=$(cat ~/tripleo-standalone-passwords.yaml | grep ' CinderPassword:' | awk -F ': ' '{ print $2; }')
GLANCE_PASSWORD=$(cat ~/tripleo-standalone-passwords.yaml | grep ' GlancePassword:' | awk -F ': ' '{ print $2; }')
HEAT_AUTH_ENCRYPTION_KEY=$(cat ~/tripleo-standalone-passwords.yaml | grep ' HeatAuthEncryptionKey:' | awk -F ': ' '{ print $2; }')
HEAT_PASSWORD=$(cat ~/tripleo-standalone-passwords.yaml | grep ' HeatPassword:' | awk -F ': ' '{ print $2; }')
IRONIC_PASSWORD=$(cat ~/tripleo-standalone-passwords.yaml | grep ' IronicPassword:' | awk -F ': ' '{ print $2; }')
MANILA_PASSWORD=$(cat ~/tripleo-standalone-passwords.yaml | grep ' ManilaPassword:' | awk -F ': ' '{ print $2; }')
NEUTRON_PASSWORD=$(cat ~/tripleo-standalone-passwords.yaml | grep ' NeutronPassword:' | awk -F ': ' '{ print $2; }')
NOVA_PASSWORD=$(cat ~/tripleo-standalone-passwords.yaml | grep ' NovaPassword:' | awk -F ': ' '{ print $2; }')
OCTAVIA_PASSWORD=$(cat ~/tripleo-standalone-passwords.yaml | grep ' OctaviaPassword:' | awk -F ': ' '{ print $2; }')
PLACEMENT_PASSWORD=$(cat ~/tripleo-standalone-passwords.yaml | grep ' PlacementPassword:' | awk -F ': ' '{ print $2; }')
SWIFT_PASSWORD=$(cat ~/tripleo-standalone-passwords.yaml | grep ' SwiftPassword:' | awk -F ': ' '{ print $2; }')
```

Procedure

1. Make sure you are using the Red Hat OpenShift Container Platform namespace where you want the control plane deployed:

```
oc project openstack
```

2. Create OSP secret.

3. If the **$ADMIN_PASSWORD** is different than the already set password in **osp-secret**, amend the **AdminPassword** key in the **osp-secret** correspondingly:

```
oc set data secret/osp-secret "AdminPassword=$ADMIN_PASSWORD"
```

4. Set service account passwords in **osp-secret** to match the service account passwords from the original deployment:

```
oc set data secret/osp-secret "AodhPassword=$AODH_PASSWORD"
oc set data secret/osp-secret "BarbicanPassword=$BARBICAN_PASSWORD"
oc set data secret/osp-secret "CeilometerMeteringSecret=$CEILOMETER_METERING_SECRET"
oc set data secret/osp-secret "CeilometerPassword=$CEILOMETER_PASSWORD"
oc set data secret/osp-secret "CinderPassword=$CINDER_PASSWORD"
oc set data secret/osp-secret "GlancePassword=$GLANCE_PASSWORD"
oc set data secret/osp-secret "HeatAuthEncryptionKey=$HEAT_AUTH_ENCRYPTION_KEY"
oc set data secret/osp-secret "HeatPassword=$HEAT_PASSWORD"
oc set data secret/osp-secret "IronicPassword=$IRONIC_PASSWORD"
oc set data secret/osp-secret "IronicInspectorPassword=$IRONIC_PASSWORD"
oc set data secret/osp-secret "ManilaPassword=$MANILA_PASSWORD"
oc set data secret/osp-secret "NeutronPassword=$NEUTRON_PASSWORD"
oc set data secret/osp-secret "NovaPassword=$NOVA_PASSWORD"
```

```
oc set data secret/osp-secret "OctaviaPassword=$OCTAVIA_PASSWORD"
oc set data secret/osp-secret "PlacementPassword=$PLACEMENT_PASSWORD"
oc set data secret/osp-secret "SwiftPassword=$SWIFT_PASSWORD"
```

5. Deploy **OpenStackControlPlane**. Make sure to only enable DNS, MariaDB, Memcached, and RabbitMQ services. All other services must be disabled.

6. If the source environment enables TLS Everywhere, modify spec:tls section with the following override before applying it:

```
apiVersion: core.openstack.org/v1beta1
kind: OpenStackControlPlane
metadata:
  name: openstack
spec:
  tls:
    podLevel:
      enabled: true
      internal:
        ca:
          customIssuer: rootca-internal
      libvirt:
        ca:
          customIssuer: rootca-internal
      ovn:
        ca:
          customIssuer: rootca-internal
    ingress:
      ca:
        customIssuer: rootca-internal
      enabled: true
```

7. If the source environment does not enable TLS Everywhere, modify spec:tls section with the following override before applying it:

```
apiVersion: core.openstack.org/v1beta1
kind: OpenStackControlPlane
metadata:
  name: openstack
spec:
  tls:
    podLevel:
      enabled: false
```

```
oc apply -f - <<EOF
apiVersion: core.openstack.org/v1beta1
kind: OpenStackControlPlane
metadata:
  name: openstack
spec:
  secret: osp-secret
  storageClass: local-storage

  barbican:
    enabled: false
```

```
  template:
    barbicanAPI: {}
    barbicanWorker: {}
    barbicanKeystoneListener: {}

cinder:
  enabled: false
  template:
    cinderAPI: {}
    cinderScheduler: {}
    cinderBackup: {}
    cinderVolumes: {}

dns:
  template:
    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
    options:
    - key: server
      values:
      - 192.168.122.1
    replicas: 1

glance:
  enabled: false
  template:
    glanceAPIs: {}

heat:
  enabled: false
  template: {}

horizon:
  enabled: false
  template: {}

ironic:
  enabled: false
  template:
    ironicConductors: []

keystone:
  enabled: false
  template: {}

manila:
  enabled: false
  template:
    manilaAPI: {}
```

```
    manilaScheduler: {}
    manilaShares: {}

mariadb:
  enabled: false
  templates: {}

galera:
  enabled: true
  templates:
    openstack:
      secret: osp-secret
      replicas: 1
      storageRequest: 500M
    openstack-cell1:
      secret: osp-secret
      replicas: 1
      storageRequest: 500M

memcached:
  enabled: true
  templates:
    memcached:
      replicas: 1

neutron:
  enabled: false
  template: {}

nova:
  enabled: false
  template: {}

ovn:
  enabled: false
  template:
    ovnController:
      networkAttachment: tenant
      nodeSelector:
        node: non-existing-node-name
    ovnNorthd:
      replicas: 0
    ovnDBCluster:
      ovndbcluster-nb:
        dbType: NB
        networkAttachment: internalapi
      ovndbcluster-sb:
        dbType: SB
        networkAttachment: internalapi

placement:
  enabled: false
  template: {}

rabbitmq:
  templates:
```

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

  telemetry:
    enabled: false
    template: {}

  swift:
    enabled: false
    template:
      swiftRing:
        ringReplicas: 1
      swiftStorage:
        replicas: 0
      swiftProxy:
        replicas: 1
EOF
```

Verification

- Check that MariaDB is running.

```
oc get pod openstack-galera-0 -o jsonpath='{.status.phase}{"\n"}'
oc get pod openstack-cell1-galera-0 -o jsonpath='{.status.phase}{"\n"}'
```

3.3. CONFIGURING A CEPH BACKEND

If the original deployment uses a Ceph storage backend for any service (e.g. Image Service (glance), Block Storage service (cinder), Compute service (nova), Shared File Systems service (manila)), the same backend must be used in the adopted deployment and custom resources (CRs) must be configured accordingly.

If you use Shared File Systems service (manila), on director environments, the CephFS driver in Shared File Systems service is configured to use its own keypair. For convenience, modify the **openstack** user so that you can use it across all Red Hat OpenStack Platform services.

Using the same user across the services serves two purposes:

- The capabilities of the user required to interact with Shared File Systems service became far simpler and hence, more became more secure with RHOSO 18.0.
- It is simpler to create a common ceph secret (keyring and ceph config file) and propagate the secret to all services that need it.

TIP

To run **ceph** commands, you must use SSH to connect to a Ceph storage node and run **sudo cephadm shell**. This brings up a ceph orchestrator container that allows you to run administrative commands against the ceph cluster. If you deployed the ceph cluster by using director, you may launch the **cephadm** shell from an RHOSP controller node.

```
ceph auth caps client.openstack \
  mgr 'allow *' \
  mon 'allow r, profile rbd' \
  osd 'profile rbd pool=vms, profile rbd pool=volumes, profile rbd pool=images, allow rw pool
manila_data'
```

Prerequisites

- The **OpenStackControlPlane** custom resource (CR) must already exist.
- Define the following shell variables. The values that are used are examples. Replace these example values with values that are correct for your environment:

```
CEPH_SSH="ssh -i <path to SSH key> root@<node IP>"
CEPH_KEY=$($CEPH_SSH "cat /etc/ceph/ceph.client.openstack.keyring | base64 -w 0")
CEPH_CONF=$($CEPH_SSH "cat /etc/ceph/ceph.conf | base64 -w 0")
```

Procedure

1. Create the **ceph-conf-files** secret, containing Ceph configuration:

```
oc apply -f - <<EOF
apiVersion: v1
data:
  ceph.client.openstack.keyring: $CEPH_KEY
  ceph.conf: $CEPH_CONF
kind: Secret
metadata:
  name: ceph-conf-files
  namespace: openstack
type: Opaque
EOF
```

The content of the file should look something like this:

```
apiVersion: v1
kind: Secret
metadata:
  name: ceph-conf-files
  namespace: openstack
```

```
stringData:
  ceph.client.openstack.keyring: |
    [client.openstack]
        key = <secret key>
        caps mgr = "allow *"
        caps mon = "allow r, profile rbd"
        caps osd = "pool=vms, profile rbd pool=volumes, profile rbd pool=images, allow rw pool
manila_data'
  ceph.conf: |
    [global]
    fsid = 7a1719e8-9c59-49e2-ae2b-d7eb08c695d4
    mon_host = 10.1.1.2,10.1.1.3,10.1.1.4
```

2. Configure **extraMounts** within the **OpenStackControlPlane** CR:

```
oc patch openstackcontrolplane openstack --type=merge --patch '
spec:
  extraMounts:
    - name: v1
      region: r1
      extraVol:
        - propagation:
          - CinderVolume
          - CinderBackup
          - GlanceAPI
          - ManilaShare
          extraVolType: Ceph
          volumes:
          - name: ceph
            projected:
              sources:
              - secret:
                  name: ceph-conf-files
          mounts:
          - name: ceph
            mountPath: "/etc/ceph"
            readOnly: true
'
```

3.4. CREATING A NFS GANESHA CLUSTER

If you use the Ceph via NFS backend with Shared File Systems service (manila), prior to adoption, you must create a new clustered NFS service on the Ceph cluster. This service will replace the standalone, pacemaker-controlled **ceph-nfs** service that was used on Red Hat OpenStack Platform 17.1.

Procedure

1. You must identify the ceph nodes to deploy the new clustered NFS service.This service must be deployed on the **StorageNFS** isolated network so that it is easier for clients to mount their existing shares through the new NFS export locations.

2. You must propagate the **StorageNFS** network to the target nodes where the **ceph-nfs** service will be deployed. The following steps will be relevant if the Ceph Storage nodes were deployed via director.

a. Identify the node definition file used in the environment. This is the input file associated with the **openstack overcloud node provision** command. For example, this file may be called **overcloud-baremetal-deploy.yaml**

b. Edit the networks associated with the Red Hat Ceph Storage nodes to include the **StorageNFS** network:

```
- name: CephStorage
  count: 3
  hostname_format: cephstorage-%index%
  instances:
  - hostname: cephstorage-0
    name: ceph-0
  - hostname: cephstorage-1
    name: ceph-1
  - hostname: cephstorage-2
    name: ceph-2
  defaults:
    profile: ceph-storage
    network_config:
      template: /home/stack/network/nic-configs/ceph-storage.j2
      network_config_update: true
    networks:
    - network: ctlplane
      vif: true
    - network: storage
    - network: storage_mgmt
    - network: storage_nfs
```

c. Edit the network configuration template file for the Red Hat Ceph Storage nodes to include an interface connecting to the **StorageNFS** network. In the example above, the path to the network configuration template file is **/home/stack/network/nic-configs/ceph-storage.j2**. This file is modified to include the following NIC template:

```
- type: vlan
  device: nic2
  vlan_id: {{ storage_nfs_vlan_id }}
  addresses:
  - ip_netmask: {{ storage_nfs_ip }}/{{ storage_nfs_cidr }}
  routes: {{ storage_nfs_host_routes }}
```

d. Re-run the **openstack overcloud node provision** command to update the Red Hat Ceph Storage nodes.

```
openstack overcloud node provision \
    --stack overcloud   \
    --network-config -y  \
    -o overcloud-baremetal-deployed-storage_nfs.yaml \
    --concurrency 2 \
    /home/stack/network/baremetal_deployment.yaml
```

When the update is complete, ensure that the Red Hat Ceph Storage nodes have a new interface created and tagged with the appropriate VLAN associated with **StorageNFS**.

3. Identify an IP address from the **StorageNFS** network to use as the Virtual IP address for the Ceph NFS service. This IP address must be provided in place of the **{{ VIP }}** in the example below. You can query used IP addresses with:

```
openstack port list -c "Fixed IP Addresses" --network storage_nfs
```

- Pick an appropriate size for the NFS cluster. The NFS service provides active/active high availability when the cluster size is more than one node. It is recommended that the ``{{ cluster_size }}`` is at least one less than the number of hosts identified. This solution has been well tested with a 3-node NFS cluster.

- The **ingress-mode** argument must be set to ``haproxy-protocol``. No other ingress-mode will be supported. This ingress mode will allow enforcing client restrictions through Shared File Systems service. For more information on deploying the clustered Ceph NFS service, see the Management of NFS-Ganesha gateway using the Ceph Orchestrator (Limited Availability) in *Red Hat Ceph Storage 7 Operations Guide* .

- The following commands are run inside a **cephadm shell** to create a clustered Ceph NFS service.

```
# wait for shell to come up, then execute:
ceph orch host ls

# Identify the hosts that can host the NFS service.
# Repeat the following command to label each host identified:
ceph orch host label add <HOST> nfs

# Set the appropriate {{ cluster_size }} and {{ VIP }}:
ceph nfs cluster create cephfs \
   "{{ cluster_size }} label:nfs" \
   --ingress \
   --virtual-ip={{ VIP }}
   --ingress-mode=haproxy-protocol
}}

# Check the status of the nfs cluster with these commands
ceph nfs cluster ls
ceph nfs cluster info cephfs
```

3.5. STOPPING RED HAT OPENSTACK PLATFORM SERVICES

Before you start the adoption, you must stop the Red Hat OpenStack Platform (RHOSP) services.

This is an important step to avoid inconsistencies in the data migrated for the data-plane adoption procedure caused by resource changes after the DB has been copied to the new deployment.

Some services are easy to stop because they only perform short asynchronous operations, but other services are a bit more complex to gracefully stop because they perform synchronous or long running operations that you might want to complete instead of aborting them.

Since gracefully stopping all services is non-trivial and beyond the scope of this guide, the following procedure uses the force method and presents recommendations on how to check some things in the services.

Note that you should not stop the infrastructure management services yet, such as:

- database
- RabbitMQ
- HAProxy Load Balancer
- ceph-nfs
- Compute service
- containerized modular libvirt daemons
- Object Storage service (swift) backend services

Prerequisites

- Confirm that there no long-running operations that require the services you plan to stop.
- Ensure that there are no ongoing instance live migrations, volume migrations (online or offline), volume creation, backup restore, attaching, detaching, and so on.

```
openstack server list --all-projects -c ID -c Status |grep -E '\| .+ing \|'
openstack volume list --all-projects -c ID -c Status |grep -E '\| .+ing \|'| grep -vi error
openstack volume backup list --all-projects -c ID -c Status |grep -E '\| .+ing \|' | grep -vi error
openstack share list --all-projects -c ID -c Status |grep -E '\| .+ing \|'| grep -vi error
openstack image list -c ID -c Status |grep -E '\| .+ing \|'
```

- Collect the services topology-specific configuration before stopping services required to gather it live. The topology-specific configuration is necessary for migrating the databases. For more information, see Retrieving topology-specific service configuration .
- Define the following shell variables. The values that are used are examples and refer to a single node standalone director deployment. Replace these example values with values that are correct for your environment:

```
CONTROLLER1_SSH="ssh -i <path to SSH key> root@<node IP>"
CONTROLLER2_SSH=""
CONTROLLER3_SSH=""
```

Procedure

You can stop RHOSP services at any moment, but you might leave your environment in an undesired state. You should confirm that there are no ongoing operations.

1. Connect to all the controller nodes.
2. Remove any constraints between infrastructure and RHOSP control plane services.
3. Stop the control plane services.
4. Verify the control plane services are stopped.

The cinder-backup service on RHOSP 17.1 could be running as Active-Passive under pacemaker or as Active-Active, so you must check how it is running and stop it.

If the deployment enables CephFS through NFS as a backend for Shared File Systems service (manila),

there are pacemaker ordering and co-location constraints that govern the Virtual IP address assigned to the **ceph-nfs** service, the **ceph-nfs** service itself and **manila-share** service. These constraints must be removed:

```
# check the co-location and ordering constraints concerning "manila-share"
sudo pcs constraint list --full

# remove these constraints
sudo pcs constraint remove colocation-openstack-manila-share-ceph-nfs-INFINITY
sudo pcs constraint remove order-ceph-nfs-openstack-manila-share-Optional
```

The following steps to disable RHOSP control plane services can be automated with a simple script that relies on the previously defined environmental variables and function:

```
# Update the services list to be stopped
ServicesToStop=("tripleo_horizon.service"
                "tripleo_keystone.service"
                "tripleo_barbican_api.service"
                "tripleo_barbican_worker.service"
                "tripleo_barbican_keystone_listener.service"
                "tripleo_cinder_api.service"
                "tripleo_cinder_api_cron.service"
                "tripleo_cinder_scheduler.service"
                "tripleo_cinder_volume.service"
                "tripleo_cinder_backup.service"
                "tripleo_glance_api.service"
                "tripleo_manila_api.service"
                "tripleo_manila_api_cron.service"
                "tripleo_manila_scheduler.service"
                "tripleo_neutron_api.service"
                "tripleo_placement_api.service"
                "tripleo_nova_api_cron.service"
                "tripleo_nova_api.service"
                "tripleo_nova_conductor.service"
                "tripleo_nova_metadata.service"
                "tripleo_nova_scheduler.service"
                "tripleo_nova_vnc_proxy.service"
                "tripleo_aodh_api.service"
                "tripleo_aodh_api_cron.service"
                "tripleo_aodh_evaluator.service"
                "tripleo_aodh_listener.service"
                "tripleo_aodh_notifier.service"
                "tripleo_ceilometer_agent_central.service"
                "tripleo_ceilometer_agent_compute.service"
                "tripleo_ceilometer_agent_ipmi.service"
                "tripleo_ceilometer_agent_notification.service"
                "tripleo_ovn_cluster_northd.service"
                "tripleo_ironic_neutron_agent.service"
                "tripleo_ironic_api.service"
                "tripleo_ironic_inspector.service"
                "tripleo_ironic_conductor.service")

PacemakerResourcesToStop=("openstack-cinder-volume"
                          "openstack-cinder-backup"
                          "openstack-manila-share")
```

```
echo "Stopping systemd OpenStack services"
for service in ${ServicesToStop[*]}; do
    for i in {1..3}; do
        SSH_CMD=CONTROLLER${i}_SSH
        if [ ! -z "${!SSH_CMD}" ]; then
            echo "Stopping the $service in controller $i"
            if ${!SSH_CMD} sudo systemctl is-active $service; then
                ${!SSH_CMD} sudo systemctl stop $service
            fi
        fi
    done
done

echo "Checking systemd OpenStack services"
for service in ${ServicesToStop[*]}; do
    for i in {1..3}; do
        SSH_CMD=CONTROLLER${i}_SSH
        if [ ! -z "${!SSH_CMD}" ]; then
            if ! ${!SSH_CMD} systemctl show $service | grep ActiveState=inactive >/dev/null; then
                echo "ERROR: Service $service still running on controller $i"
            else
                echo "OK: Service $service is not running on controller $i"
            fi
        fi
    done
done

echo "Stopping pacemaker OpenStack services"
for i in {1..3}; do
    SSH_CMD=CONTROLLER${i}_SSH
    if [ ! -z "${!SSH_CMD}" ]; then
        echo "Using controller $i to run pacemaker commands"
        for resource in ${PacemakerResourcesToStop[*]}; do
            if ${!SSH_CMD} sudo pcs resource config $resource &>/dev/null; then
                echo "Stopping $resource"
                ${!SSH_CMD} sudo pcs resource disable $resource
            else
                echo "Service $resource not present"
            fi
        done
        break
    fi
done

echo "Checking pacemaker OpenStack services"
for i in {1..3}; do
    SSH_CMD=CONTROLLER${i}_SSH
    if [ ! -z "${!SSH_CMD}" ]; then
        echo "Using controller $i to run pacemaker commands"
        for resource in ${PacemakerResourcesToStop[*]}; do
            if ${!SSH_CMD} sudo pcs resource config $resource &>/dev/null; then
                if ! ${!SSH_CMD} sudo pcs resource status $resource | grep Started; then
                    echo "OK: Service $resource is stopped"
                else
                    echo "ERROR: Service $resource is started"
```

```
            fi
        fi
    done
    break
  fi
done
```

3.6. MIGRATING DATABASES TO MARIADB INSTANCES

This document describes how to move the databases from the original Red Hat OpenStack Platform (RHOSP) deployment to the MariaDB instances in the Red Hat OpenShift Container Platform cluster.

NOTE

This example scenario describes a simple single-cell setup. Real multi-stack topology recommended for production use results in different cells DBs layout, and should be using different naming schemes (not covered here this time).

Prerequisites

- Make sure the previous Adoption steps have been performed successfully.
  - The **OpenStackControlPlane** resource must be already created.
  - The control plane MariaDB and RabbitMQ are running. No other control plane services are running.
  - Required services specific topology. For more information, see Retrieving topology-specific service configuration.
  - RHOSP services have been stopped. For more information, see Stopping Red Hat OpenStack Platform services.
  - There must be network routability between the original MariaDB and the MariaDB for the control plane.
- Define the following shell variables. The values that are used are examples. Replace these example values with values that are correct for your environment:

```
PODIFIED_MARIADB_IP=$(oc get svc --selector "mariadb/name=openstack" -
ojsonpath='{.items[0].spec.clusterIP}')
PODIFIED_CELL1_MARIADB_IP=$(oc get svc --selector "mariadb/name=openstack-cell1" -
ojsonpath='{.items[0].spec.clusterIP}')
PODIFIED_DB_ROOT_PASSWORD=$(oc get -o json secret/osp-secret | jq -r
.data.DbRootPassword | base64 -d)

# The CHARACTER_SET and collation should match the source DB
# if the do not then it will break foreign key relationships
# for any tables that are created in the future as part of db sync
CHARACTER_SET=utf8
COLLATION=utf8_general_ci

STORAGE_CLASS=local-storage
MARIADB_IMAGE=registry.redhat.io/rhosp-dev-preview/openstack-mariadb-rhel9:18.0
# Replace with your environment's MariaDB Galera cluster VIP and backend IPs:
```

```
SOURCE_MARIADB_IP=172.17.0.2
declare -A SOURCE_GALERA_MEMBERS
SOURCE_GALERA_MEMBERS=(
  ["standalone.localdomain"]=172.17.0.100
  # ...
)
SOURCE_DB_ROOT_PASSWORD=$(cat ~/tripleo-standalone-passwords.yaml | grep '
MysqlRootPassword:' | awk -F ': ' '{ print $2; }')
```

- Prepare MariaDB copy directory and the adoption helper pod
- Create a temporary folder to store adoption helper pod (pick storage requests to fit MySQL database size):

```
mkdir ~/adoption-db
cd ~/adoption-db
```

```
oc apply -f - <<EOF
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mariadb-data
spec:
  storageClassName: $STORAGE_CLASS
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
---
apiVersion: v1
kind: Pod
metadata:
  name: mariadb-copy-data
  annotations:
    openshift.io/scc: anyuid
    k8s.v1.cni.cncf.io/networks: internalapi
  labels:
    app: adoption
spec:
  containers:
  - image: $MARIADB_IMAGE
    command: [ "sh", "-c", "sleep infinity"]
    name: adoption
    volumeMounts:
    - mountPath: /backup
      name: mariadb-data
  securityContext:
    allowPrivilegeEscalation: false
    capabilities:
      drop: ALL
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
```

```
  volumes:
  - name: mariadb-data
    persistentVolumeClaim:
      claimName: mariadb-data
EOF
```

- Wait for the pod to come up

```
oc wait --for condition=Ready pod/mariadb-copy-data --timeout=30s
```

Procedure

1. Check that the source Galera database cluster members are online and synced:

```
for i in "${!SOURCE_GALERA_MEMBERS[@]}"; do
  echo "Checking for the database node $i WSREP status Synced"
  oc rsh mariadb-copy-data mysql \
    -h "${SOURCE_GALERA_MEMBERS[$i]}" -uroot -
p"$SOURCE_DB_ROOT_PASSWORD" \
    -e "show global status like 'wsrep_local_state_comment'" | \
    grep -qE "\bSynced\b"
done
```

2. Get the count of not-OK source databases:

```
oc rsh mariadb-copy-data mysql -h "${SOURCE_MARIADB_IP}" -uroot -
p"${SOURCE_DB_ROOT_PASSWORD}" -e "SHOW databases;"
```

3. Run mysqlcheck on the original DB to look for things that are not OK:

```
. ~/.source_cloud_exported_variables
test -z "$PULL_OPENSTACK_CONFIGURATION_MYSQLCHECK_NOK"  || [
"$PULL_OPENSTACK_CONFIGURATION_MYSQLCHECK_NOK" = " " ]
```

4. Test connection to control plane DBs (show databases):

```
oc run mariadb-client --image $MARIADB_IMAGE -i --rm --restart=Never -- \
    mysql -rsh "$PODIFIED_MARIADB_IP" -uroot -p"$PODIFIED_DB_ROOT_PASSWORD" -
e 'SHOW databases;'
oc run mariadb-client --image $MARIADB_IMAGE -i --rm --restart=Never -- \
    mysql -rsh "$PODIFIED_CELL1_MARIADB_IP" -uroot -
p"$PODIFIED_DB_ROOT_PASSWORD" -e 'SHOW databases;'
```

NOTE

You need to transition Compute service (nova) services imported later on into a superconductor architecture. For that, delete the old service records in cells DBs, starting from the cell1. New records will be registered with different hostnames provided by the Compute service service operator. All Compute service services, except the compute agent, have no internal state, and its service records can be safely deleted. You also need to rename the former **default** cell to **cell1**.

5. Create a dump of the original databases:

```
oc rsh mariadb-copy-data << EOF
  mysql -h"${SOURCE_MARIADB_IP}" -uroot -p"${SOURCE_DB_ROOT_PASSWORD}" \
  -N -e "show databases" | grep -E -v "schema|mysql|gnocchi" | \
  while read dbname; do
    echo "Dumping \${dbname}";
    mysqldump -h"${SOURCE_MARIADB_IP}" -uroot -
p"${SOURCE_DB_ROOT_PASSWORD}" \
      --single-transaction --complete-insert --skip-lock-tables --lock-tables=0 \
      "\${dbname}" > /backup/"\${dbname}".sql;
   done
EOF
```

6. Restore the databases from .sql files into the control plane MariaDB:

```
oc rsh mariadb-copy-data << EOF
  # db schemas to rename on import
  declare -A db_name_map
  db_name_map['nova']='nova_cell1'
  db_name_map['ovs_neutron']='neutron'
  db_name_map['ironic-inspector']='ironic_inspector'

  # db servers to import into
  declare -A db_server_map
  db_server_map['default']=${PODIFIED_MARIADB_IP}
  db_server_map['nova_cell1']=${PODIFIED_CELL1_MARIADB_IP}

  # db server root password map
  declare -A db_server_password_map
  db_server_password_map['default']=${PODIFIED_DB_ROOT_PASSWORD}
  db_server_password_map['nova_cell1']=${PODIFIED_DB_ROOT_PASSWORD}

  cd /backup
  for db_file in \$(ls *.sql); do
    db_name=\$(echo \${db_file} | awk -F'.' '{ print \$1; }')
    if [[ -v "db_name_map[\${db_name}]" ]]; then
      echo "renaming \${db_name} to \${db_name_map[\${db_name}]}"
      db_name=\${db_name_map[\${db_name}]}
    fi
    db_server=\${db_server_map["default"]}
    if [[ -v "db_server_map[\${db_name}]" ]]; then
      db_server=\${db_server_map[\${db_name}]}
    fi
    db_password=\${db_server_password_map['default']}
    if [[ -v "db_server_password_map[\${db_name}]" ]]; then
      db_password=\${db_server_password_map[\${db_name}]}
    fi
    echo "creating \${db_name} in \${db_server}"
    mysql -h"\${db_server}" -uroot "-p\${db_password}" -e \
      "CREATE DATABASE IF NOT EXISTS \${db_name} DEFAULT \
      CHARACTER SET ${CHARACTER_SET} DEFAULT COLLATE ${COLLATION};"
    echo "importing \${db_name} into \${db_server}"
    mysql -h "\${db_server}" -uroot "-p\${db_password}" "\${db_name}" < "\${db_file}"
  done

  mysql -h "\${db_server_map['default']}" -uroot -p"\${db_server_password_map['default']}" -e
```

```
\
    "update nova_api.cell_mappings set name='cell1' where name='default';"
  mysql -h "\${db_server_map['nova_cell1']}" -uroot -
p"\${db_server_password_map['nova_cell1']}" -e \
    "delete from nova_cell1.services where host not like '%nova-cell1-%' and services.binary
!= 'nova-compute';"
EOF
```

Verification

Compare the following outputs with the topology specific configuration. For more information, see Retrieving topology-specific service configuration .

1. Check that the databases were imported correctly:

```
. ~/.source_cloud_exported_variables

# use 'oc exec' and 'mysql -rs' to maintain formatting
dbs=$(oc exec openstack-galera-0 -c galera -- mysql -rs -uroot "-
p$PODIFIED_DB_ROOT_PASSWORD" -e 'SHOW databases;')
echo $dbs | grep -Eq '\bkeystone\b'

# ensure neutron db is renamed from ovs_neutron
echo $dbs | grep -Eq '\bneutron\b'
echo $PULL_OPENSTACK_CONFIGURATION_DATABASES | grep -Eq '\bovs_neutron\b'

# ensure nova cell1 db is extracted to a separate db server and renamed from nova to
nova_cell1
c1dbs=$(oc exec openstack-cell1-galera-0 -c galera -- mysql -rs -uroot "-
p$PODIFIED_DB_ROOT_PASSWORD" -e 'SHOW databases;')
echo $c1dbs | grep -Eq '\bnova_cell1\b'

# ensure default cell renamed to cell1, and the cell UUIDs retained intact
novadb_mapped_cells=$(oc exec openstack-galera-0 -c galera -- mysql -rs -uroot "-
p$PODIFIED_DB_ROOT_PASSWORD" \
  nova_api -e 'select uuid,name,transport_url,database_connection,disabled from
cell_mappings;')
uuidf='\S{8,}-\S{4,}-\S{4,}-\S{4,}-\S{12,}'
left_behind=$(comm -23 \
  <(echo $PULL_OPENSTACK_CONFIGURATION_NOVADB_MAPPED_CELLS | grep -oE "
$uuidf \S+") \
  <(echo $novadb_mapped_cells | tr -s "| " " " | grep -oE " $uuidf \S+"))
changed=$(comm -13 \
  <(echo $PULL_OPENSTACK_CONFIGURATION_NOVADB_MAPPED_CELLS | grep -oE "
$uuidf \S+") \
  <(echo $novadb_mapped_cells | tr -s "| " " " | grep -oE " $uuidf \S+"))
test $(grep -Ec ' \S+$' <<<$left_behind) -eq 1
default=$(grep -E ' default$' <<<$left_behind)
test $(grep -Ec ' \S+$' <<<$changed) -eq 1
grep -qE " $(awk '{print $1}' <<<$default) cell1$" <<<$changed

# ensure the registered Compute service name has not changed
novadb_svc_records=$(oc exec openstack-cell1-galera-0 -c galera -- mysql -rs -uroot "-
p$PODIFIED_DB_ROOT_PASSWORD" \
  nova_cell1 -e "select host from services where services.binary='nova-compute' order by
```

```
host asc;")
diff -Z <(echo $novadb_svc_records) <(echo
$PULL_OPENSTACK_CONFIGURATION_NOVA_COMPUTE_HOSTNAMES)
```

2. During the pre/post checks the pod **mariadb-client** might have returned a pod security warning related to the **restricted:latest** security context constraint. This is due to default security context constraints and will not prevent pod creation by the admission controller. You'll see a warning for the short-lived pod but it will not interfere with functionality.

3. Delete the **mariadb-data** pod and **mariadb-copy-data** persistent volume claim with databases backup (consider making a snapshot of it, before deleting)

```
oc delete pod mariadb-copy-data
oc delete pvc mariadb-data
```

For more information, see About pod security standards and warnings.

3.7. MIGRATING OVN DATA

This document describes how to move OVN northbound and southbound databases from the original Red Hat OpenStack Platform deployment to **ovsdb-server** instances running in the Red Hat OpenShift Container Platform cluster. While it may be argued that the control plane Networking service (neutron) ML2/OVN driver and OVN northd service will reconstruct the databases on startup, the reconstruction may be time consuming on large existing clusters. The procedure below allows to speed up data migration and avoid unnecessary data plane disruptions due to incomplete OpenFlow table contents.

Prerequisites

- Make sure the previous Adoption steps have been performed successfully.
  - The **OpenStackControlPlane** resource must be already created at this point.
  - NetworkAttachmentDefinition custom resource definitions (CRDs) for the original cluster are already defined. Specifically, **openstack/internalapi** network is defined.
  - Control plane MariaDB and RabbitMQ may already run. The Networking service and OVN are not running yet.
  - Original OVN is older or equal to the control plane version.
  - Original Neutron Server and OVN northd services are stopped.
  - There must be network routability between:
    - The adoption host and the original OVN.
    - The adoption host and the control plane OVN.
- Define the following shell variables. The values that are used are examples. Replace these example values with values that are correct for your environment:

```
STORAGE_CLASS=local-storage
OVSDB_IMAGE=registry.redhat.io/rhosp-dev-preview/openstack-ovn-base-rhel9:18.0
SOURCE_OVSDB_IP=172.17.1.49
```

You can get the value to set **SOURCE_OVSDB_IP** by querying the puppet-generated configurations:

```
grep -rI 'ovn_[ns]b_conn' /var/lib/config-data/puppet-generated/
```

Procedure

1. Prepare the OVN DBs copy dir and the adoption helper pod (pick the storage requests to fit the OVN databases sizes)

```
oc apply -f - <<EOF
---
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: ovn-data-cert
  namespace: openstack
spec:
  commonName: ovn-data-cert
  secretName: ovn-data-cert
  issuerRef:
    name: rootca-internal
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: ovn-data
spec:
  storageClassName: $STORAGE_CLASS_NAME
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
---
apiVersion: v1
kind: Pod
metadata:
  name: ovn-copy-data
  annotations:
    openshift.io/scc: anyuid
    k8s.v1.cni.cncf.io/networks: internalapi
  labels:
    app: adoption
spec:
  containers:
  - image: $OVSDB_IMAGE
    command: [ "sh", "-c", "sleep infinity"]
    name: adoption
    volumeMounts:
    - mountPath: /backup
      name: ovn-data
    - mountPath: /etc/pki/tls/misc
      name: ovn-data-cert
      readOnly: true
```

```
  securityContext:
    allowPrivilegeEscalation: false
    capabilities:
      drop: ALL
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  volumes:
  - name: ovn-data
    persistentVolumeClaim:
      claimName: ovn-data
  - name: ovn-data-cert
    secret:
      secretName: ovn-data-cert
EOF
```

2. Wait for the pod to come up

```
oc wait --for=condition=Ready pod/ovn-copy-data --timeout=30s
```

3. Backup OVN databases on an environment without TLS everywhere.

```
oc exec ovn-copy-data -- bash -c "ovsdb-client backup tcp:$SOURCE_OVSDB_IP:6641 >
/backup/ovs-nb.db"
oc exec ovn-copy-data -- bash -c "ovsdb-client backup tcp:$SOURCE_OVSDB_IP:6642 >
/backup/ovs-sb.db"
```

4. Backup OVN databases on a TLS everywhere environment.

```
oc exec ovn-copy-data -- bash -c "ovsdb-client backup --ca-cert=/etc/pki/tls/misc/ca.crt --
private-key=/etc/pki/tls/misc/tls.key --certificate=/etc/pki/tls/misc/tls.crt
ssl:$SOURCE_OVSDB_IP:6641 > /backup/ovs-nb.db"
oc exec ovn-copy-data -- bash -c "ovsdb-client backup --ca-cert=/etc/pki/tls/misc/ca.crt --
private-key=/etc/pki/tls/misc/tls.key --certificate=/etc/pki/tls/misc/tls.crt
ssl:$SOURCE_OVSDB_IP:6642 > /backup/ovs-sb.db"
```

5. Start the control plane OVN database services prior to import, keeping **northd**/**ovn-controller** stopped.

```
oc patch openstackcontrolplane openstack --type=merge --patch '
spec:
  ovn:
    enabled: true
    template:
      ovnDBCluster:
        ovndbcluster-nb:
          dbType: NB
          storageRequest: 10G
          networkAttachment: internalapi
        ovndbcluster-sb:
          dbType: SB
          storageRequest: 10G
          networkAttachment: internalapi
      ovnNorthd:
```

```
      replicas: 0
    ovnController:
      networkAttachment: tenant
      nodeSelector:
        node: non-existing-node-name
'
```

6. Wait for the OVN DB pods reaching the running phase.

```
oc wait --for=jsonpath='{.status.phase}'=Running pod --selector=service=ovsdbserver-nb
oc wait --for=jsonpath='{.status.phase}'=Running pod --selector=service=ovsdbserver-sb
```

7. Fetch the control plane OVN IP addresses on the clusterIP service network.

```
PODIFIED_OVSDB_NB_IP=$(oc get svc --selector "statefulset.kubernetes.io/pod-
name=ovsdbserver-nb-0" -ojsonpath='{.items[0].spec.clusterIP}')
PODIFIED_OVSDB_SB_IP=$(oc get svc --selector "statefulset.kubernetes.io/pod-
name=ovsdbserver-sb-0" -ojsonpath='{.items[0].spec.clusterIP}')
```

8. Upgrade database schema for the backup files on an environment without TLS everywhere.

```
oc exec ovn-copy-data -- bash -c "ovsdb-client get-schema
tcp:$PODIFIED_OVSDB_NB_IP:6641 > /backup/ovs-nb.ovsschema && ovsdb-tool convert
/backup/ovs-nb.db /backup/ovs-nb.ovsschema"
oc exec ovn-copy-data -- bash -c "ovsdb-client get-schema
tcp:$PODIFIED_OVSDB_SB_IP:6642 > /backup/ovs-sb.ovsschema && ovsdb-tool convert
/backup/ovs-sb.db /backup/ovs-sb.ovsschema"
```

9. Upgrade database schema for the backup files on a TLS everywhere environment.

```
oc exec ovn-copy-data -- bash -c "ovsdb-client get-schema --ca-cert=/etc/pki/tls/misc/ca.crt --
private-key=/etc/pki/tls/misc/tls.key --certificate=/etc/pki/tls/misc/tls.crt
ssl:$PODIFIED_OVSDB_NB_IP:6641 > /backup/ovs-nb.ovsschema && ovsdb-tool convert
/backup/ovs-nb.db /backup/ovs-nb.ovsschema"
oc exec ovn-copy-data -- bash -c "ovsdb-client get-schema --ca-cert=/etc/pki/tls/misc/ca.crt --
private-key=/etc/pki/tls/misc/tls.key --certificate=/etc/pki/tls/misc/tls.crt
ssl:$PODIFIED_OVSDB_SB_IP:6642 > /backup/ovs-sb.ovsschema && ovsdb-tool convert
/backup/ovs-sb.db /backup/ovs-sb.ovsschema"
```

10. Restore database backup to the control plane OVN database servers on an environment without TLS everywhere.

```
oc exec ovn-copy-data -- bash -c "ovsdb-client restore tcp:$PODIFIED_OVSDB_NB_IP:6641
< /backup/ovs-nb.db"
oc exec ovn-copy-data -- bash -c "ovsdb-client restore tcp:$PODIFIED_OVSDB_SB_IP:6642
< /backup/ovs-sb.db"
```

11. Restore database backup to control plane OVN database servers on a TLS everywhere environment.

```
oc exec ovn-copy-data -- bash -c "ovsdb-client restore --ca-cert=/etc/pki/tls/misc/ca.crt --
private-key=/etc/pki/tls/misc/tls.key --certificate=/etc/pki/tls/misc/tls.crt
ssl:$PODIFIED_OVSDB_NB_IP:6641 < /backup/ovs-nb.db"
```

```
oc exec ovn-copy-data -- bash -c "ovsdb-client restore --ca-cert=/etc/pki/tls/misc/ca.crt --
private-key=/etc/pki/tls/misc/tls.key --certificate=/etc/pki/tls/misc/tls.crt
ssl:$PODIFIED_OVSDB_SB_IP:6642 < /backup/ovs-sb.db"
```

12. Check that the control plane OVN databases contain objects from backup, for example:

```
oc exec -it ovsdbserver-nb-0 -- ovn-nbctl show
oc exec -it ovsdbserver-sb-0 -- ovn-sbctl list Chassis
```

13. Finally, you can start **ovn-northd** service that will keep OVN northbound and southbound databases in sync.

```
oc patch openstackcontrolplane openstack --type=merge --patch '
spec:
  ovn:
    enabled: true
    template:
      ovnNorthd:
        replicas: 1
'
```

14. Also enable **ovn-controller**:

```
oc patch openstackcontrolplane openstack --type=json -p="[{'op': 'remove', 'path':
'/spec/ovn/template/ovnController/nodeSelector'}]"
```

15. Delete the **ovn-data** pod and persistent volume claim with OVN databases backup (consider making a snapshot of it, before deleting):

```
oc delete pod ovn-copy-data
oc delete pvc ovn-data
```

16. Stop old OVN database servers.

```
ServicesToStop=("tripleo_ovn_cluster_north_db_server.service"
                "tripleo_ovn_cluster_south_db_server.service")

echo "Stopping systemd OpenStack services"
for service in ${ServicesToStop[*]}; do
    for i in {1..3}; do
        SSH_CMD=CONTROLLER${i}_SSH
        if [ ! -z "${!SSH_CMD}" ]; then
            echo "Stopping the $service in controller $i"
            if ${!SSH_CMD} sudo systemctl is-active $service; then
                ${!SSH_CMD} sudo systemctl stop $service
            fi
        fi
    done
done

echo "Checking systemd OpenStack services"
for service in ${ServicesToStop[*]}; do
    for i in {1..3}; do
        SSH_CMD=CONTROLLER${i}_SSH
```

```
        if [ ! -z "${!SSH_CMD}" ]; then
            if ! ${!SSH_CMD} systemctl show $service | grep ActiveState=inactive >/dev/null; then
                echo "ERROR: Service $service still running on controller $i"
            else
                echo "OK: Service $service is not running on controller $i"
            fi
        fi
    done
done
```

CHAPTER 4. ADOPTING RED HAT OPENSTACK PLATFORM CONTROL PLANE SERVICES

Adopt your Red Hat OpenStack Platform 17.1 control plane services to deploy them in the Red Hat OpenStack Services on OpenShift (RHOSO) 18.0 control plane.

4.1. ADOPTING THE IDENTITY SERVICE

Prerequisites

- Previous Adoption steps completed. Notably, the Migrating databases to MariaDB instances must already be imported into the control plane MariaDB.
- Ensure that you copy the fernet keys. Create the **keystone** secret, containing fernet keys:

```
oc apply -f - <<EOF
apiVersion: v1
data:
  CredentialKeys0: $($CONTROLLER1_SSH sudo cat /var/lib/config-data/puppet-generated/keystone/etc/keystone/credential-keys/0 | base64 -w 0)
  CredentialKeys1: $($CONTROLLER1_SSH sudo cat /var/lib/config-data/puppet-generated/keystone/etc/keystone/credential-keys/1 | base64 -w 0)
  FernetKeys0: $($CONTROLLER1_SSH sudo cat /var/lib/config-data/puppet-generated/keystone/etc/keystone/fernet-keys/0 | base64 -w 0)
  FernetKeys1: $($CONTROLLER1_SSH sudo cat /var/lib/config-data/puppet-generated/keystone/etc/keystone/fernet-keys/1 | base64 -w 0)
kind: Secret
metadata:
  name: keystone
  namespace: openstack
type: Opaque
EOF
```

Procedure

1. Patch **OpenStackControlPlane** to deploy Identity service:

```
oc patch openstackcontrolplane openstack --type=merge --patch '
spec:
  keystone:
    enabled: true
    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
'
```

2. Create an alias to use **openstack** command in the adopted deployment:

```
$ alias openstack="oc exec -t openstackclient -- openstack"
```

3. Clean up old services and endpoints that still point to the old control plane, excluding the Identity service and its endpoints:

```
$ openstack endpoint list | grep keystone | awk '/admin/{ print $2; }' | xargs
${BASH_ALIASES[openstack]} endpoint delete || true

for service in aodh heat heat-cfn barbican cinderv3 glance manila manilav2 neutron nova
placement swift ironic-inspector ironic; do
  openstack service list | awk "/ $service /{ print \$2; }" | xargs ${BASH_ALIASES[openstack]}
service delete || true
done
```

Verification

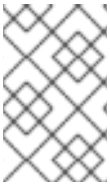
- See that Identity service endpoints are defined and pointing to the control plane FQDNs:

```
$ openstack endpoint list | grep keystone
```

4.2. ADOPTING THE KEY MANAGER SERVICE

Adopting Key Manager service (barbican) means that an existing **OpenStackControlPlane** custom resource (CR), where Key Manager service is initialy disabled, should be patched to start the service with the configuration parameters provided by the source environment.

When the procedure is over, the expectation is to see the **BarbicanAPI**, **BarbicanWorker**, **BarbicanKeystoneListener** services are up and running. **Keystone endpoints** should also be updated and the same crypto plugin of the source Cloud will be available. If the conditions above are met, the adoption is considered concluded.



NOTE

This procedure configures the Key Manager service to use the **simple_crypto** backend. Additional backends are available, such as PKCS11 and DogTag, however they are not supported in this release.

Prerequisites

- Previous Adoption steps completed. Notably, MariaDB, RabbitMQ, and Identity service (keystone). should be already adopted.

Procedure

1. Add the kek secret. In this case we are updating and using **osp-secret**, which contains other service passwords:

```
oc set data secret/osp-secret "BarbicanSimpleCryptoKEK=$($CONTROLLER1_SSH
"python3 -c \"import configparser; c = configparser.ConfigParser(); c.read('/var/lib/config-
data/puppet-generated/barbican/etc/barbican/barbican.conf'); print(c['simple_crypto_plugin']
['kek'])\"" | base64 -w 0)"
```

2. Patch **OpenStackControlPlane** to deploy the Key Manager service:

```
oc patch openstackcontrolplane openstack --type=merge --patch '
spec:
  barbican:
    enabled: true
    apiOverride:
      route: {}
    template:
      databaseInstance: openstack
      databaseAccount: barbican
      databaseUser: barbican
      rabbitMqClusterName: rabbitmq
      secret: osp-secret
      simpleCryptoBackendSecret: osp-secret
      serviceAccount: barbican
      serviceUser: barbican
      passwordSelectors:
        database: BarbicanDatabasePassword
        service: BarbicanPassword
        simplecryptokek: BarbicanSimpleCryptoKEK
      barbicanAPI:
        replicas: 1
        override:
          service:
            internal:
              metadata:
                annotations:
                  metallb.universe.tf/address-pool: internalapi
                  metallb.universe.tf/allow-shared-ip: internalapi
                  metallb.universe.tf/loadBalancerIPs: 172.17.0.80
              spec:
                type: LoadBalancer
      barbicanWorker:
        replicas: 1
      barbicanKeystoneListener:
        replicas: 1
'
```

Verification

- Check that the Identity service (keystone) endpoints are defined and pointing to the control plane FQDNs:

```
$ openstack endpoint list | grep key-manager
```

- Check that Barbican API service is registered in Identity service:

```
$ openstack service list | grep key-manager
```

```
$ openstack endpoint list | grep key-manager
```

- List secrets:

```
$ openstack secret list
```

4.3. ADOPTING THE NETWORKING SERVICE

Adopting Networking service (neutron) means that an existing **OpenStackControlPlane** custom resource (CR), where the Networking service is supposed to be disabled, should be patched to start the service with the configuration parameters provided by the source environment.

When the procedure is over, the expectation is to see the **NeutronAPI** service is running: the Identity service (keystone) endpoints should be updated and the same backend of the source Cloud will be available. If the conditions above are met, the adoption is considered concluded.

This guide also assumes that:

1. A director environment (the source Cloud) is running on one side;
2. A **SNO** / **CodeReadyContainers** is running on the other side.

Prerequisites

- Previous Adoption steps completed. Notably, MariaDB,Identity service (keystone), and Migrating OVN data should be already adopted.

Procedure

- Patch **OpenStackControlPlane** to deploy Networking service:

```
oc patch openstackcontrolplane openstack --type=merge --patch '
spec:
  neutron:
    enabled: true
    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
      databaseAccount: neutron
      secret: osp-secret
      networkAttachments:
      - internalapi
'
```

Verification

- Inspect the resulting Networking service pods:

```
NEUTRON_API_POD=`oc get pods -l service=neutron | tail -n 1 | cut -f 1 -d' '`
oc exec -t $NEUTRON_API_POD -c neutron-api -- cat /etc/neutron/neutron.conf
```

- Check that the **Neutron API** service is registered in the Identity service:

```
$ openstack service list | grep network
```

```
$ openstack endpoint list | grep network

| 6a805bd6c9f54658ad2f24e5a0ae0ab6 | regionOne | neutron      | network      | True    |
public    | http://neutron-public-openstack.apps-crc.testing  |
| b943243e596847a9a317c8ce1800fa98 | regionOne | neutron      | network      | True    |
internal  | http://neutron-internal.openstack.svc:9696        |
| f97f2b8f7559476bb7a5eafe3d33cee7 | regionOne | neutron      | network      | True    | admin
| http://192.168.122.99:9696                        |
```

- Create sample resources. You can test whether the user can create networks, subnets, ports, or routers.

```
$ openstack network create net
$ openstack subnet create --network net --subnet-range 10.0.0.0/24 subnet
$ openstack router create router
```

4.4. ADOPTING THE OBJECT STORAGE SERVICE

This section only applies if you are using OpenStack Swift as Object Storage service (swift). If you are using the Object Storage API of Ceph RGW this section can be skipped.

Prerequisites

- Previous adoption steps completed.
- The Object Storage service storage backend services must still be running.
- Storage network has been properly configured on the Red Hat OpenShift Container Platform cluster.

Procedure

1. Create the **swift-conf** secret, containing the Object Storage service hash path suffix and prefix:

```
oc apply -f - <<EOF
apiVersion: v1
kind: Secret
metadata:
  name: swift-conf
  namespace: openstack
type: Opaque
data:
```

```
  swift.conf: $($CONTROLLER1_SSH sudo cat /var/lib/config-data/puppet-
generated/swift/etc/swift/swift.conf | base64 -w0)
EOF
```

2. Create the **swift-ring-files** configmap, containing the Object Storage service ring files:

```
oc apply -f - <<EOF
apiVersion: v1
kind: ConfigMap
metadata:
  name: swift-ring-files
binaryData:
  swiftrings.tar.gz: $($CONTROLLER1_SSH "cd /var/lib/config-data/puppet-
generated/swift/etc/swift && tar cz *.builder *.ring.gz backups/ | base64 -w0")
  account.ring.gz: $($CONTROLLER1_SSH "base64 -w0 /var/lib/config-data/puppet-
generated/swift/etc/swift/account.ring.gz")
  container.ring.gz: $($CONTROLLER1_SSH "base64 -w0 /var/lib/config-data/puppet-
generated/swift/etc/swift/container.ring.gz")
  object.ring.gz: $($CONTROLLER1_SSH "base64 -w0 /var/lib/config-data/puppet-
generated/swift/etc/swift/object.ring.gz")
EOF
```

3. Patch **OpenStackControlPlane** to deploy the Object Storage service:

```
oc patch openstackcontrolplane openstack --type=merge --patch '
spec:
  swift:
    enabled: true
    template:
      memcachedInstance: memcached
      swiftRing:
        ringReplicas: 1
      swiftStorage:
        replicas: 0
        networkAttachments:
        - storage
        storageClass: local-storage
        storageRequest: 10Gi
      swiftProxy:
        secret: osp-secret
        replicas: 1
        passwordSelectors:
          service: SwiftPassword
        serviceUser: swift
        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
'
```

Verification

- Inspect the resulting Object Storage service service pods:

```
$ oc get pods -l component=swift-proxy
```

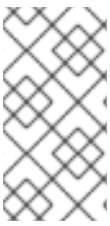
- Check that the Object Storage service proxy service is registered in the Identity service:

```
$ openstack service list | grep swift
| b5b9b1d3c79241aa867fa2d05f2bbd52 | swift    | object-store |
```

```
$ openstack endpoint list | grep swift
| 32ee4bd555414ab48f2dc90a19e1bcd5 | regionOne | swift        | object-store | True    |
public    | https://swift-public-openstack.apps-crc.testing/v1/AUTH_%(tenant_id)s |
| db4b8547d3ae4e7999154b203c6a5bed | regionOne | swift        | object-store | True    |
internal  | http://swift-internal.openstack.svc:8080/v1/AUTH_%(tenant_id)s        |
```

- Check that you are able to up- and download objects:

```
echo "Hello World!" > obj
openstack container create test
+---------------------------------------+-----------+------------------------------------+
| account                               | container | x-trans-id                         |
+---------------------------------------+-----------+------------------------------------+
| AUTH_4d9be0a9193e4577820d187acdd2714a | test      | txe5f9a10ce21e4cddad473-
0065ce41b9 |
+---------------------------------------+-----------+------------------------------------+

openstack object create test obj
+--------+-----------+----------------------------------+
| object | container | etag                             |
+--------+-----------+----------------------------------+
| obj    | test      | d41d8cd98f00b204e9800998ecf8427e |
+--------+-----------+----------------------------------+

openstack object save test obj --file -
Hello World!
```



NOTE

At this point data is still stored on the previously existing nodes. For more information about migrating the actual data from the old to the new deployment, see Migrating the Object Storage service (swift) data from RHOSP to Red Hat OpenStack Services on OpenShift (RHOSO) nodes.

4.5. ADOPTING THE IMAGE SERVICE

Adopting Image Service (glance) means that an existing **OpenStackControlPlane** custom resource (CR), where Image service is supposed to be disabled, should be patched to start the service with the configuration parameters provided by the source environment.

When the procedure is over, the expectation is to see the **GlanceAPI** service up and running: the Identity service endpoints are updated and the same backend of the source Cloud is available. If the conditions above are met, the adoption is considered concluded.

This guide also assumes that:

- A director environment (the source Cloud) is running on one side.
- A **SNO** / **CodeReadyContainers** is running on the other side.
- (optional) An internal/external **Ceph** cluster is reachable by both **crc** and director.

4.5.1. Adopting the Image service that is deployed with a Object Storage service backend

Adopt the Image Service (glance) that you deployed with an Object Storage service (swift) backend. When Image service is deployed with Object Storage service (swift) as a backend in the Red Hat OpenStack Platform environment based on director, the control plane **glanceAPI** instance is deployed with the following configuration:

```
..
spec
  glance:
    ...
      customServiceConfig: |
        [DEFAULT]
        enabled_backends = default_backend:swift
        [glance_store]
        default_backend = default_backend
        [default_backend]
        swift_store_create_container_on_put = True
        swift_store_auth_version = 3
        swift_store_auth_address = {{ .KeystoneInternalURL }}
        swift_store_endpoint_type = internalURL
        swift_store_user = service:glance
        swift_store_key = {{ .ServicePassword }}
```

Prerequisites

- Previous Adoption steps completed. Notably, MariaDB, Identity service (keystone) and Key Manager service (barbican) should be already adopted.

Procedure

1. Write the patch manifest into a file, for example **glance_swift.patch**. For example:

```
spec:
  glance:
    enabled: true
    apiOverride:
      route: {}
```

```
  template:
    databaseInstance: openstack
    storageClass: "local-storage"
    storageRequest: 10G
    customServiceConfig: |
      [DEFAULT]
      enabled_backends = default_backend:swift
      [glance_store]
      default_backend = default_backend
      [default_backend]
      swift_store_create_container_on_put = True
      swift_store_auth_version = 3
      swift_store_auth_address = {{ .KeystoneInternalURL }}
      swift_store_endpoint_type = internalURL
      swift_store_user = service:glance
      swift_store_key = {{ .ServicePassword }}
    glanceAPIs:
      default:
        replicas: 1
        override:
          service:
            internal:
              metadata:
                annotations:
                  metallb.universe.tf/address-pool: internalapi
                  metallb.universe.tf/allow-shared-ip: internalapi
                  metallb.universe.tf/loadBalancerIPs: 172.17.0.80
            spec:
              type: LoadBalancer
        networkAttachments:
          - storage
```

Having Object Storage service as a backend establishes a dependency between the two services, and any deployed **GlanceAPI** instance would not work if Image service is configured with Object Storage service that is still not available in the **OpenStackControlPlane**. Once Object Storage service, and in particular **SwiftProxy**, has been adopted, it is possible to proceed with the **GlanceAPI** adoption. For more information, see Adopting the Object Storage service.

2. Verify that **SwiftProxy** is available:

```
$ oc get pod -l component=swift-proxy | grep Running
swift-proxy-75cb47f65-92rxq   3/3     Running   0
```

3. Patch the **GlanceAPI** service deployed in the control plane context:

```
$ oc patch openstackcontrolplane openstack --type=merge --patch-file=glance_swift.patch
```

4.5.2. Adopting the Image service that is deployed with a Block Storage service backend

Adopt the Image Service (glance) that you deployed with an Block Storage service (cinder) backend. When Image service is deployed with Block Storage service as a backend in the Red Hat OpenStack Platform environment based on director, the control plane **glanceAPI** instance is deployed with the following configuration:

```
..
spec
  glance:
    ...
      customServiceConfig: |
          [DEFAULT]
          enabled_backends = default_backend:cinder
          [glance_store]
          default_backend = default_backend
          [default_backend]
          rootwrap_config = /etc/glance/rootwrap.conf
          description = Default cinder backend
          cinder_store_auth_address = {{ .KeystoneInternalURL }}
          cinder_store_user_name = {{ .ServiceUser }}
          cinder_store_password = {{ .ServicePassword }}
          cinder_store_project_name = service
          cinder_catalog_info = volumev3::internalURL
          cinder_use_multipath = true
```

Prerequisites

- Previous Adoption steps completed. Notably, MariaDB, Identity service (keystone) and Key Manager service (barbican) should be already adopted.

Procedure

1. Write the patch manifest into a file, for example **glance_cinder.patch**. For example:

```
spec:
  glance:
    enabled: true
    apiOverride:
      route: {}
    template:
      databaseInstance: openstack
      storageClass: "local-storage"
      storageRequest: 10G
      customServiceConfig: |
          [DEFAULT]
          enabled_backends = default_backend:cinder
          [glance_store]
          default_backend = default_backend
          [default_backend]
          rootwrap_config = /etc/glance/rootwrap.conf
          description = Default cinder backend
          cinder_store_auth_address = {{ .KeystoneInternalURL }}
          cinder_store_user_name = {{ .ServiceUser }}
          cinder_store_password = {{ .ServicePassword }}
          cinder_store_project_name = service
          cinder_catalog_info = volumev3::internalURL
          cinder_use_multipath = true
      glanceAPIs:
        default:
          replicas: 1
          override:
```

```
    service:
      internal:
        metadata:
          annotations:
            metallb.universe.tf/address-pool: internalapi
            metallb.universe.tf/allow-shared-ip: internalapi
            metallb.universe.tf/loadBalancerIPs: 172.17.0.80
      spec:
        type: LoadBalancer
  networkAttachments:
    - storage
```

Having Block Storage service as a backend establishes a dependency between the two services, and any deployed **GlanceAPI** instance would not work if the Image service is configured with Block Storage service that is still not available in the **OpenStackControlPlane**. Once Block Storage service, and in particular **CinderVolume**, has been adopted, it is possible to proceed with the **GlanceAPI** adoption.

2. Verify that **CinderVolume** is available:

```
$ oc get pod -l component=cinder-volume | grep Running
cinder-volume-75cb47f65-92rxq   3/3     Running   0
```

3. Patch the **GlanceAPI** service deployed in the control plane context:

```
oc patch openstackcontrolplane openstack --type=merge --patch-file=glance_cinder.patch
```

4.5.3. Adopting the Image service that is deployed with an NFS Ganesha backend

Adopt the Image Service (glance) that you deployed with an NFS Ganesha backend. The following steps assume that:

1. The Storage network has been propagated to the RHOSP control plane.
2. The Image service is able to reach the Storage network and connect to the nfs-server through the port **2049**.

Prerequisites

- Previous Adoption steps completed. Notably, MariaDB, Identity service (keystone) and Key Manager service (barbican) should be already adopted.
- In the source cloud, verify the NFS Ganesha parameters used by the overcloud to configure the Image service backend. In particular, find among the director heat templates the following variables that are usually an override of the default content provided by **/usr/share/openstack-tripleo-heat-templates/environments/storage/glance-nfs.yaml**[glance-nfs.yaml]:

```
**GlanceBackend**: file

**GlanceNfsEnabled**: true

**GlanceNfsShare**: 192.168.24.1:/var/nfs
```

In the example above, as the first variable shows, the Image service has no notion of NFS Ganesha backend: the **File** driver is used in this scenario, and behind the scenes, the

filesystem_store_datadir which usually points to **/var/lib/glance/images/** is mapped to the export value provided by the **GlanceNfsShare** variable. If the **GlanceNfsShare** is not exported through a network that is supposed to be propagated to the adopted Red Hat OpenStack Platform control plane, an extra action is required by the human administrator, who must stop the **nfs-server** and remap the export to the **storage** network. This action usually happens when the Image service is stopped in the source Controller nodes. In the control plane, the Image service is attached to the Storage network, propagated via the associated **NetworkAttachmentsDefinition** custom resource, and the resulting Pods have already the right permissions to handle the Image service traffic through this network. In a deployed RHOSP control plane, you can verify that the network mapping matches with what has been deployed in the director-based environment by checking both the **NodeNetworkConfigPolicy** (**nncp**) and the **NetworkAttachmentDefinition** (**net-attach-def**):

```
$ oc get nncp
NAME                        STATUS      REASON
enp6s0-crc-8cf2w-master-0   Available   SuccessfullyConfigured

$ oc get net-attach-def
NAME
ctlplane
internalapi
storage
tenant

$ oc get ipaddresspool -n metallb-system
NAME          AUTO ASSIGN   AVOID BUGGY IPS   ADDRESSES
ctlplane      true          false             ["192.168.122.80-192.168.122.90"]
internalapi   true          false             ["172.17.0.80-172.17.0.90"]
storage       true          false             ["172.18.0.80-172.18.0.90"]
tenant        true          false             ["172.19.0.80-172.19.0.90"]
```

The above represents an example of the output that should be checked in the Red Hat OpenShift Container Platform environment to make sure there are no issues with the propagated networks.

Procedure

1. Adopt the Image service and create a new **default GlanceAPI** instance connected with the existing NFS Ganesha share.

```
cat << EOF > glance_nfs_patch.yaml

spec:
  extraMounts:
  - extraVol:
    - extraVolType: Nfs
      mounts:
      - mountPath: /var/lib/glance/images
        name: nfs
      propagation:
      - Glance
      volumes:
      - name: nfs
        nfs:
          path: /var/nfs
```

```
        server: 172.17.3.20
    name: r1
    region: r1
  glance:
    enabled: true
    template:
      databaseInstance: openstack
      customServiceConfig: |
        [DEFAULT]
        enabled_backends = default_backend:file
        [glance_store]
        default_backend = default_backend
        [default_backend]
        filesystem_store_datadir = /var/lib/glance/images/
      storageClass: "local-storage"
      storageRequest: 10G
      glanceAPIs:
        default:
          replicas: 1
          type: single
          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
EOF
```

NOTE

Replace in **glance_nfs_patch.yaml** the **nfs/server** IP address with the IP used to reach the **nfs-server** and make sure the **nfs/path** points to the exported path in the **nfs-server**.

2. Patch **OpenStackControlPlane** to deploy Image service with a NFS Ganesha backend:

```
$ oc patch openstackcontrolplane openstack --type=merge --patch-file
glance_nfs_patch.yaml
```

Verification

- When GlanceAPI is active, you can see a single API instance:

```
$ oc get pods -l service=glance
NAME                    READY   STATUS    RESTARTS
glance-default-single-0   3/3     Running   0
```

and the description of the pod must report:

```
Mounts:
...
  nfs:
    Type:      NFS (an NFS mount that lasts the lifetime of a pod)
    Server:    {{ server ip address }}
    Path:      {{ nfs export path }}
    ReadOnly:  false
...
```

- Check the mountpoint:

```
oc rsh -c glance-api glance-default-single-0

sh-5.1# mount
...
...
{{ ip address }}:/var/nfs on /var/lib/glance/images type nfs4
(rw,relatime,vers=4.2,rsize=1048576,wsize=1048576,namlen=255,hard,proto=tcp,timeo=600,re
trans=2,sec=sys,clientaddr=172.18.0.5,local_lock=none,addr=172.18.0.5)
...
...
```

- Confirm that the UUID has been created in the exported directory on the NFS Ganesha node. For example:

```
$ oc rsh openstackclient
$ openstack image list

sh-5.1$  curl -L -o /tmp/cirros-0.5.2-x86_64-disk.img http://download.cirros-
cloud.net/0.5.2/cirros-0.5.2-x86_64-disk.img
...
...

sh-5.1$ openstack image create --container-format bare --disk-format raw --file /tmp/cirros-
0.5.2-x86_64-disk.img cirros
...
...

sh-5.1$ openstack image list
+--------------------------------------+--------+--------+
| ID                                   | Name   | Status |
+--------------------------------------+--------+--------+
| 634482ca-4002-4a6d-b1d5-64502ad02630 | cirros | active |
+--------------------------------------+--------+--------+
```

- On the nfs-server node, the same **uuid** is in the exported **/var/nfs**:

```
$ ls /var/nfs/
634482ca-4002-4a6d-b1d5-64502ad02630
```

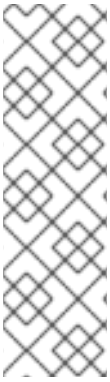
4.5.4. Adopting the Image service that is deployed with a Red Hat Ceph Storage backend

Adopt the Image Service (glance) that you deployed with a Red Hat Ceph Storage backend. Use the **customServiceConfig** parameter to inject the right configuration to the **GlanceAPI** instance.

Prerequisites

- Previous Adoption steps completed. Notably, MariaDB, Identity service (keystone) and Key Manager service (barbican) should be already adopted.
- Make sure the Ceph-related secret (**ceph-conf-files**) was created in the **openstack** namespace and that the **extraMounts** property of the **OpenStackControlPlane** custom resource (CR) has been configured properly. These tasks are described in an earlier Adoption step Configuring a Ceph backend.

```
cat << EOF > glance_patch.yaml
spec:
  glance:
    enabled: true
    template:
      databaseInstance: openstack
      customServiceConfig: |
        [DEFAULT]
        enabled_backends=default_backend:rbd
        [glance_store]
        default_backend=default_backend
        [default_backend]
        rbd_store_ceph_conf=/etc/ceph/ceph.conf
        rbd_store_user=openstack
        rbd_store_pool=images
        store_description=Ceph glance store backend.
      storageClass: "local-storage"
      storageRequest: 10G
      glanceAPIs:
        default:
          replicas: 1
          override:
            service:
              internal:
                metadata:
                  annotations:
                    metallb.universe.tf/address-pool: internalapi
                    metallb.universe.tf/allow-shared-ip: internalapi
                    metallb.universe.tf/loadBalancerIPs: 172.17.0.80
              spec:
                type: LoadBalancer
          networkAttachments:
          - storage
EOF
```



NOTE

If you have previously backed up your RHOSP services configuration file from the old environment, you can use os-diff to compare and make sure the configuration is correct. For more information, see Pulling the configuration from a director deployment .

```
os-diff diff /tmp/collect_tripleo_configs/glance/etc/glance/glance-api.conf
glance_patch.yaml --crd
```

This produces the difference between both ini configuration files.

Procedure

- Patch **OpenStackControlPlane** CR to deploy Image service with a Red Hat Ceph Storage backend:

```
$ oc patch openstackcontrolplane openstack --type=merge --patch-file glance_patch.yaml
```

4.5.5. Verifying the Image service adoption

Verify that you successfully adopted your Image Service (glance) to the Red Hat OpenStack Services on OpenShift (RHOSO) 18.0 deployment.

Procedure

1. Test the Image Service (glance) from the Red Hat OpenStack Platform CLI. You can compare and make sure the configuration has been correctly applied to the Image service pods:

```
os-diff diff /etc/glance/glance.conf.d/02-config.conf glance_patch.yaml --frompod -p glance-
api
```

   If no line appears, then the configuration is correctly done.

2. Inspect the resulting glance pods:

```
GLANCE_POD=`oc get pod |grep glance-default-external-0 | cut -f 1 -d' '`
oc exec -t $GLANCE_POD -c glance-api -- cat /etc/glance/glance.conf.d/02-config.conf

[DEFAULT]
enabled_backends=default_backend:rbd
[glance_store]
default_backend=default_backend
[default_backend]
rbd_store_ceph_conf=/etc/ceph/ceph.conf
rbd_store_user=openstack
rbd_store_pool=images
store_description=Ceph glance store backend.
```

3. If you use a Ceph backend, ensure that the Ceph secrets are properly mounted:

```
oc exec -t $GLANCE_POD -c glance-api -- ls /etc/ceph
ceph.client.openstack.keyring
ceph.conf
```

4. Check that the service is active and the endpoints are properly updated in the RHOSP CLI:

```
(openstack)$ service list | grep image

| fc52dbffef36434d906eeb99adfc6186 | glance    | image        |

(openstack)$ endpoint list | grep image

| 569ed81064f84d4a91e0d2d807e4c1f1 | regionOne | glance       | image        | True    |
internal  | http://glance-internal-openstack.apps-crc.testing   |
| 5843fae70cba4e73b29d4aff3e8b616c | regionOne | glance       | image        | True    | public
| http://glance-public-openstack.apps-crc.testing     |
| 709859219bc24ab9ac548eab74ad4dd5 | regionOne | glance       | image        | True    |
admin     | http://glance-admin-openstack.apps-crc.testing      |
```

5. Check that the images that you previously listed in the source Cloud are available in the adopted service:

```
(openstack)$ image list
+--------------------------------------+--------+--------+
| ID                                   | Name   | Status |
+--------------------------------------+--------+--------+
| c3158cad-d50b-452f-bec1-f250562f5c1f | cirros | active |
+--------------------------------------+--------+--------+
```

4.6. ADOPTING THE PLACEMENT SERVICE

Prerequisites

- Previous Adoption steps completed. Notably,
  - the Migrating databases to MariaDB instances must already be imported into the control plane MariaDB.
  - the Adopting the Identity service needs to be imported.
  - the Memcached operator needs to be deployed (nothing to import for it from the source environment).

Procedure

- Patch **OpenStackControlPlane** to deploy the Placement service:

```
oc patch openstackcontrolplane openstack --type=merge --patch '
spec:
  placement:
    enabled: true
    apiOverride:
      route: {}
    template:
      databaseInstance: openstack
      databaseAccount: placement
      secret: osp-secret
      override:
```

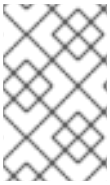
```
      service:
        internal:
          metadata:
            annotations:
              metallb.universe.tf/address-pool: internalapi
              metallb.universe.tf/allow-shared-ip: internalapi
              metallb.universe.tf/loadBalancerIPs: 172.17.0.80
          spec:
            type: LoadBalancer
'
```

Verification

- Check that Placement endpoints are defined and pointing to the control plane FQDNs and that Placement API responds:

```
alias openstack="oc exec -t openstackclient -- openstack"

openstack endpoint list | grep placement


# Without OpenStack CLI placement plugin installed:
PLACEMENT_PUBLIC_URL=$(openstack endpoint list -c 'Service Name' -c 'Service Type' -c URL | grep placement | grep public | awk '{ print $6; }')
oc exec -t openstackclient -- curl "$PLACEMENT_PUBLIC_URL"

# With OpenStack CLI placement plugin installed:
openstack resource class list
```

4.7. ADOPTING THE COMPUTE SERVICE



NOTE

This example scenario describes a simple single-cell setup. Real multi-stack topology recommended for production use results in different cells DBs layout, and should be using different naming schemes (not covered here this time).

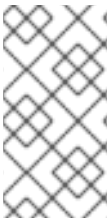
Prerequisites

- Previous Adoption steps completed. Notably,
  - the Migrating databases to MariaDB instances must already be imported into the control plane MariaDB;
  - the Adopting the Identity service needs to be imported;
  - the Adopting the Key Manager service needs to be imported;
  - the Adopting the Placement service needs to be imported;
  - the Adopting the Image service needs to be imported;
  - the Migrating OVN data need to be imported;

- the Adopting the Networking service needs to be imported;
- the Bare Metal Provisioning service needs to be imported;
- Required topology-specific service configuration. For more information, see Retrieving topology-specific service configuration.
- Red Hat OpenStack Platform services have been stopped. For more information, see Stopping Red Hat OpenStack Platform services.

- Define the following shell variables. The values that are used are examples. Replace these example values with values that are correct for your environment:

```
alias openstack="oc exec -t openstackclient -- openstack"
```



PROCEDURE

This procedure assumes that Compute service Metadata is deployed on the top level and not on each cell level, so this example imports it the same way. If the source deployment has a per cell metadata deployment, adjust the given below patch as needed. Metadata service cannot be run in **cell0**.

1. Patch **OpenStackControlPlane** to deploy the Compute service:

```
oc patch openstackcontrolplane openstack -n openstack --type=merge --patch '
spec:
  nova:
    enabled: true
    apiOverride:
      route: {}
    template:
      secret: osp-secret
      apiServiceTemplate:
        override:
          service:
            internal:
              metadata:
                annotations:
                  metallb.universe.tf/address-pool: internalapi
                  metallb.universe.tf/allow-shared-ip: internalapi
                  metallb.universe.tf/loadBalancerIPs: 172.17.0.80
              spec:
                type: LoadBalancer
        customServiceConfig: |
          [workarounds]
          disable_compute_service_check_for_ffu=true
      metadataServiceTemplate:
        enabled: true # deploy single nova metadata on the top level
        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
        customServiceConfig: |
          [workarounds]
          disable_compute_service_check_for_ffu=true
      schedulerServiceTemplate:
        customServiceConfig: |
          [workarounds]
          disable_compute_service_check_for_ffu=true
      cellTemplates:
        cell0:
          conductorServiceTemplate:
            customServiceConfig: |
              [workarounds]
              disable_compute_service_check_for_ffu=true
        cell1:
          metadataServiceTemplate:
            enabled: false # enable here to run it in a cell instead
            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
            customServiceConfig: |
              [workarounds]
              disable_compute_service_check_for_ffu=true
          conductorServiceTemplate:
            customServiceConfig: |
              [workarounds]
              disable_compute_service_check_for_ffu=true
'
```

- If adopting Compute service (nova) with the Baremetal service (**ironic**), append the following **novaComputeTemplates** in the **cell1** section of the Compute service CR patch: **NOTE**: Set the **[DEFAULT]host** configuration option to match the hostname of the node running the **ironic** compute driver in the source cloud.

```
        cell1:
          novaComputeTemplates:
            standalone:
              customServiceConfig: |
                [DEFAULT]
                host = standalone.localdomain
                [workarounds]
                disable_compute_service_check_for_ffu=true
```

2. Wait for Compute service control plane services' custom resources (CRs) to become ready:

```
oc wait --for condition=Ready --timeout=300s Nova/nova
```

The local Conductor services will be started for each cell, while the superconductor runs in **cell0**.

Note that **disable_compute_service_check_for_ffu** is mandatory for all imported Nova services, until the external data plane is imported, and until Nova Compute services fast-forward upgraded. For more information, see Adopting Compute services to the RHOSO data plane and Performing a fast-forward upgrade on Compute services .

Verification

- Check that Compute service endpoints are defined and pointing to the control plane FQDNs and that Nova API responds.

```
$ openstack endpoint list | grep nova
$ openstack server list
```

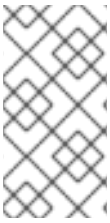
Compare the following outputs with the topology specific configuration in Retrieving topology-specific service configuration.

- Query the superconductor for cell1 existance and compare it to pre-adoption values:

```
. ~/.source_cloud_exported_variables
echo $PULL_OPENSTACK_CONFIGURATION_NOVAMANAGE_CELL_MAPPINGS
oc rsh nova-cell0-conductor-0 nova-manage cell_v2 list_cells | grep -F '| cell1 |'
```

  The expected changes to happen:

  - cell1's **nova** DB and user name become **nova_cell1**.
  - Default cell is renamed to **cell1** (in a multi-cell setup, it should become indexed as the last cell instead).
  - RabbitMQ transport URL no longer uses **guest**.



NOTE

At this point, the Compute service control plane services do not control the existing Compute service Compute workloads. The control plane manages the data plane only after the data adoption process is successfully completed. For more information, see Adopting Compute services to the RHOSO data plane .

4.8. ADOPTING THE BLOCK STORAGE SERVICE

Adopting a director-deployed Block Storage service (cinder) service into Red Hat OpenStack Platform usually entails:

- Checking existing limitations.
- Considering the placement of the Block Storage service services.
- Preparing the Red Hat OpenShift Container Platform nodes where volume and backup services will run.
- Crafting the manifest based on the existing **cinder.conf** file.
- Deploying Block Storage service.
- Validating the new deployment.

This guide provides necessary knowledge to complete these steps in most situations, but it still requires knowledge on how RHOSP services work and the structure of a Block Storage service configuration file.

4.8.1. Limitations for adopting the Block Storage service

There are currently limitations that are worth highlighting; some are related to this guideline while some to the operator:

- There is no global **nodeSelector** for all Block Storage service (cinder) volumes, so it needs to be specified per backend.
- There is no global **customServiceConfig** or **customServiceConfigSecrets** for all Block Storage service volumes, so it needs to be specified per backend.
- Adoption of LVM backends, where the volume data is stored in the compute nodes, is not currently being documented in this process.
- Support for Block Storage service backends that require kernel modules not included in RHEL has not been tested in Operator deployed Red Hat OpenStack Platform.
- Adoption of DCN/Edge deployment is not currently described in this guide.

4.8.2. Red Hat OpenShift Container Platform preparation for Block Storage service adoption

Before deploying Red Hat OpenStack Platform (RHOSP) in Red Hat OpenShift Container Platform, you must ensure that the networks are ready, that you have decided the node selection, and also make sure any necessary changes to the RHOCP nodes have been made. For Block Storage service (cinder) volume and backup services all these 3 must be carefully considered.

Node Selection

You might need, or want, to restrict the RHOCP nodes where Block Storage service volume and backup services can run.
The best example of when you need to do node selection for a specific Block Storage service is when you deploy the Block Storage service with the LVM driver. In that scenario, the LVM data where the volumes are stored only exists in a specific host, so you need to pin the Block Storage-volume service to that specific RHOCP node. Running the service on any other RHOCP node would not work. Since **nodeSelector** only works on labels, you cannot use the RHOCP host node name to restrict the LVM backend and you need to identify it using a unique label, an existing label, or new label:

```
$ oc label nodes worker0 lvm=cinder-volumes
```

```
apiVersion: core.openstack.org/v1beta1
kind: OpenStackControlPlane
metadata:
  name: openstack
spec:
  secret: osp-secret
  storageClass: local-storage
  cinder:
    enabled: true
    template:
      cinderVolumes:
        lvm-iscsi:
```

```
        nodeSelector:
          lvm: cinder-volumes
< . . . >
```

As mentioned in the About node selector, an example where you need to use labels is when using FC storage and you do not have HBA cards in all your RHOCP nodes. In this scenario you need to restrict all the Block Storage service volume backends (not only the FC one) as well as the backup services.

Depending on the Block Storage service backends, their configuration, and the usage of Block Storage service, you can have network intensive Block Storage service volume services with lots of I/O as well as Block Storage service backup services that are not only network intensive but also memory and CPU intensive. This may be a concern for the RHOCP human operators, and they may want to use the **nodeSelector** to prevent these service from interfering with their other RHOCP workloads. For more information about node selection, see About node selector.

When selecting the nodes where the Block Storage service volume is going to run remember that Block Storage service-volume may also use local storage when downloading a Image Service (glance) image for the create volume from image operation, and it can require a considerable amount of space when having concurrent operations and not using Block Storage service volume cache.

If you do not have nodes with enough local disk space for the temporary images, you can use a remote NFS location for the images. You had to manually set this up in director deployments, but with operators, you can do it automatically using the extra volumes feature ()**extraMounts**.

Transport protocols

Due to the specifics of the storage transport protocols some changes may be required on the RHOCP side, and although this is something that must be documented by the Vendor here wer are going to provide some generic instructions that can serve as a guide for the different transport protocols.
Check the backend sections in your **cinder.conf** file that are listed in the **enabled_backends** configuration option to figure out the transport storage protocol used by the backend.

Depending on the backend, you can find the transport protocol:

- Looking at the **volume_driver** configuration option, as it may contain the protocol itself: RBD, iSCSI, FC...
- Looking at the **target_protocol** configuration option

WARNING

Any time a **MachineConfig** is used to make changes to RHOCP nodes the node will reboot!! Act accordingly.

NFS

There is nothing to do for NFS. RHOCP can connect to NFS backends without any additional changes.

RBD/Ceph

There is nothing to do for RBD/Ceph in terms of preparing the nodes, RHOCP can connect to Ceph backends without any additional changes. Credentials and configuration files will need to be provided to the services though.

iSCSI

Connecting to iSCSI volumes requires that the iSCSI initiator is running on the RHOCP hosts where volume and backup services are going to run, because the Linux Open iSCSI initiator does not currently support network namespaces, so you must only run 1 instance of the service for the normal RHOCP usage, plus the RHOCP CSI plugins, plus the RHOSP services.

If you are not already running **iscsid** on the RHOCP nodes, then you need to apply a **MachineConfig** similar to this one:

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
    service: cinder
  name: 99-master-cinder-enable-iscsid
spec:
  config:
    ignition:
      version: 3.2.0
    systemd:
      units:
      - enabled: true
        name: iscsid.service
```

If you are using labels to restrict the nodes where the Block Storage services are running you need to use a **MachineConfigPool** as described in the About node selector to limit the effects of the **MachineConfig** to only the nodes where your services may run.

If you are using a single node deployment to test the process, replace **worker** with **master** in the **MachineConfig**.

FC

There is nothing to do for FC volumes to work, but the Block Storage service volume and Block Storage service backup services need to run in an RHOCP host that has HBAs, so if there are nodes that do not have HBAs then you need to use labels to restrict where these services can run, as mentioned in About node selector.

This also means that for virtualized RHOCP clusters using FC you need to expose the host's HBAs inside the VM.

NVMe-oF

Connecting to NVMe-oF volumes requires that the nvme kernel modules are loaded on the RHOCP hosts.

If you are not already loading the **nvme-fabrics** module on the RHOCP nodes where volume and backup services are going to run then you need to apply a **MachineConfig** similar to this one:

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
```

```
    machineconfiguration.openshift.io/role: worker
    service: cinder
  name: 99-master-cinder-load-nvme-fabrics
spec:
  config:
    ignition:
      version: 3.2.0
    storage:
      files:
        - path: /etc/modules-load.d/nvme_fabrics.conf
          overwrite: false
          # Mode must be decimal, this is 0644
          mode: 420
          user:
            name: root
          group:
            name: root
          contents:
            # Source can be a http, https, tftp, s3, gs, or data as defined in rfc2397.
            # This is the rfc2397 text/plain string format
            source: data:,nvme-fabrics
```

If you are using labels to restrict the nodes where Block Storage services are running, you need to use a **MachineConfigPool** as described in the About node selector to limit the effects of the **MachineConfig** to only the nodes where your services may run.

If you are using a single node deployment to test the process,replace **worker** with **master** in the **MachineConfig**.

You are only loading the **nvme-fabrics** module because it takes care of loading the transport specific modules (tcp, rdma, fc) as needed.

For production deployments using NVMe-oF volumes it is recommended that you use multipathing. For NVMe-oF volumes RHOSP uses native multipathing, called ANA.

Once the RHOCP nodes have rebooted and are loading the **nvme-fabrics** module you can confirm that the Operating System is configured and supports ANA by checking on the host:

```
cat /sys/module/nvme_core/parameters/multipath
```

IMPORTANT

ANA does not use the Linux Multipathing Device Mapper, but the current RHOSP code requires **multipathd** on Compute nodes to be running for Compute service (nova) to be able to use multipathing.

Multipathing

For iSCSI and FC protocols, using multipathing is recommended, which has 4 parts:

- Prepare the RHOCP hosts
- Configure the Block Storage services
- Prepare the Compute service computes

- Configure the Compute service service
  To prepare the RHOCP hosts, you need to ensure that the Linux Multipath Device Mapper is configured and running on the RHOCP hosts, and you do that using **MachineConfig** like this one:

```
# Includes the /etc/multipathd.conf contents and the systemd unit changes
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
    service: cinder
  name: 99-master-cinder-enable-multipathd
spec:
  config:
    ignition:
      version: 3.2.0
    storage:
      files:
        - path: /etc/multipath.conf
          overwrite: false
          # Mode must be decimal, this is 0600
          mode: 384
          user:
            name: root
          group:
            name: root
          contents:
            # Source can be a http, https, tftp, s3, gs, or data as defined in rfc2397.
            # This is the rfc2397 text/plain string format
            source:
data:,defaults%20%7B%0A%20%20user_friendly_names%20no%0A%20%20recheck_w
wid%20yes%0A%20%20skip_kpartx%20yes%0A%20%20find_multipaths%20yes%0A%7
D%0A%0Ablacklist%20%7B%0A%7D
    systemd:
      units:
      - enabled: true
        name: multipathd.service
```

  If you are using labels to restrict the nodes where Block Storage services are running you need to use a **MachineConfigPool** as described in the About node selector to limit the effects of the **MachineConfig** to only the nodes where your services may run.

  If you are using a single node deployment to test the process, replace **worker** with **master** in the **MachineConfig**.

  To configure the Block Storage services to use multipathing, enable the **use_multipath_for_image_xfer** configuration option in all the backend sections and in the **[DEFAULT]** section for the backup service. This is the default in control plane deployments. Multipathing works as long as the service is running on the RHOCP host. Do not override this option by setting **use_multipath_for_image_xfer = false**.

4.8.3. Preparing the Block Storage service configurations for adoption

The Block Storage service (cinder) is configured using configuration snippets instead of using configuration parameters defined by the installer. For more information, see Service configurations.

The recommended way to deploy Block Storage service volume backends has changed to remove old limitations, add flexibility, and improve operations.

When deploying with director you used to run a single Block Storage service volume service with all your backends (each backend would run on its own process), and even though that way of deploying is still supported, it is not recommended. It is recommended to use a volume service per backend since it is a superior deployment model.

With an LVM and a Ceph backend you have 2 entries in **cinderVolume** and, as mentioned in the limitations section, you cannot set global defaults for all volume services, so you have to define it for each of them, like this:

```
apiVersion: core.openstack.org/v1beta1
kind: OpenStackControlPlane
metadata:
  name: openstack
spec:
  cinder:
    enabled: true
    template:
      cinderVolume:
        lvm:
          customServiceConfig: |
            [DEFAULT]
            debug = True
            [lvm]
< . . . >
        ceph:
          customServiceConfig: |
            [DEFAULT]
            debug = True
            [ceph]
< . . . >
```

Reminder that for volume backends that have sensitive information using **Secret** and the **customServiceConfigSecrets** key is the recommended way to go.

For adoption instead of using a whole deployment manifest you use a targeted patch, like you did with other services, and in this patch you will enable the different Block Storage services with their specific configurations.

WARNING

Check that all configuration options are still valid for the new Red Hat OpenStack Platform version. Configuration options may have been deprecated, removed, or added. This applies to both backend driver specific configuration options and other generic options.

4.8.3.1. Preparing the Block Storage service configuration

Creating the Cinder configuration entails:

Procedure

1. Determine what part of the configuration is generic for all the Block Storage service (cinder) services and remove anything that would change when deployed in Red Hat OpenShift Container Platform, like the **connection** in the **[dabase]** section, the **transport_url** and **log_dir** in **[DEFAULT]**, the whole **[coordination]** and **[barbican]** sections. This configuration goes into the **customServiceConfig** (or a **Secret** and then used in **customServiceConfigSecrets**) at the **cinder: template:** level.

2. Determine if there's any scheduler specific configuration and add it to the **customServiceConfig** section in **cinder: template: cinderScheduler**.

3. Determine if there's any API specific configuration and add it to the **customServiceConfig** section in **cinder: template: cinderAPI**.

4. If you have Block Storage service backup deployed, then you get the Block Storage service backup relevant configuration options and add them to **customServiceConfig** (or a **Secret** and then used in **customServiceConfigSecrets**) at the **cinder: template: cinderBackup:** level. You should remove the **host** configuration in the **[DEFAULT]** section to facilitate supporting multiple replicas in the future.

5. Determine the individual volume backend configuration for each of the drivers. The configuration will not only be the specific driver section, it should also include the **[backend_defaults]** section and FC zoning sections is they are being used, because the Block Storage service operator doesn't support a **customServiceConfig** section global for all volume services. Each backend would have its own section under **cinder: template: cinderVolumes** and the configuration would go in **customServiceConfig** (or a **Secret** and then used in **customServiceConfigSecrets**).

6. Check if any of the Block Storage service volume drivers being used requires a custom vendor image. If they do, find the location of the image in the vendor's instruction available in the Red Hat OpenStack Platform Block Storage service ecosystem page and add it under the specific's driver section using the **containerImage** key. The following example shows a CRD for a Pure Storage array with a certified driver:

```
spec:
  cinder:
    enabled: true
    template:
      cinderVolume:
        pure:
          containerImage: registry.connect.redhat.com/purestorage/openstack-cinder-volume-
pure-rhosp-18-0'
          customServiceConfigSecrets:
            - openstack-cinder-pure-cfg
< . . . >
```

7. External files: Block Storage services sometimes use external files, for example for a custom policy, or to store credentials, or SSL CA bundles to connect to a storage array, and you need to make those files available to the right containers. To achieve this, you use **Secrets** or

ConfigMap to store the information in RHOCP and then the **extraMounts** key. For example, for the Ceph credentials stored in a **Secret** called **ceph-conf-files** you patch the top level **extraMounts** in **OpenstackControlPlane**:

```
spec:
  extraMounts:
  - extraVol:
    - extraVolType: Ceph
      mounts:
      - mountPath: /etc/ceph
        name: ceph
        readOnly: true
      propagation:
      - CinderVolume
      - CinderBackup
      - Glance
      volumes:
      - name: ceph
        projected:
          sources:
          - secret:
              name: ceph-conf-files
```

But for a service specific one, like the API policy, you do it directly on the service itself. In this example, you include the Block Storage API configuration that references the policy you are adding from a **ConfigMap** called **my-cinder-conf** that has a key **policy** with the contents of the policy:

```
spec:
  cinder:
    enabled: true
    template:
      cinderAPI:
        customServiceConfig: |
          [oslo_policy]
          policy_file=/etc/cinder/api/policy.yaml
      extraMounts:
      - extraVol:
        - extraVolType: Ceph
          mounts:
          - mountPath: /etc/cinder/api
            name: policy
            readOnly: true
          propagation:
          - CinderAPI
          volumes:
          - name: policy
            projected:
              sources:
              - configMap:
                  name: my-cinder-conf
                  items:
                    - key: policy
                      path: policy.yaml
```

4.8.4. Deploying the Block Storage services

Assuming you have already stopped Block Storage service (cinder) services, prepared the Red Hat OpenShift Container Platform nodes, deployed the Red Hat OpenStack Platform (RHOSP) operators and a bare RHOSP manifest, and migrated the database, and prepared the patch manifest with the Block Storage service configuration, you must apply the patch and wait for the operator to apply the changes and deploy the Block Storage services.

Prerequisites

- Previous Adoption steps completed. Notably, Block Storage service must have been stopped and the service databases must already be imported into the control plane MariaDB.
- Identity service (keystone) and Key Manager service (barbican) should be already adopted.
- Storage network has been properly configured on the RHOCP cluster.
- You need the contents of **cinder.conf** file. Download the file so that you can access it locally:

```
$CONTROLLER1_SSH cat /var/lib/config-data/puppet-generated/cinder/etc/cinder/cinder.conf > cinder.conf
```

Procedure

1. It is recommended to write the patch manifest into a file, for example **cinder.patch** and then apply it with something like:

```
oc patch openstackcontrolplane openstack --type=merge --patch-file=cinder.patch
```

For example, for the RBD deployment from the Development Guide the **cinder.patch** would look like this:

```
spec:
  extraMounts:
  - extraVol:
    - extraVolType: Ceph
      mounts:
      - mountPath: /etc/ceph
        name: ceph
        readOnly: true
      propagation:
      - CinderVolume
      - CinderBackup
      - Glance
      volumes:
      - name: ceph
        projected:
          sources:
          - secret:
              name: ceph-conf-files
  cinder:
    enabled: true
    apiOverride:
      route: {}
    template:
```

```
    databaseInstance: openstack
    databaseAccount: cinder
    secret: osp-secret
    cinderAPI:
      override:
        service:
          internal:
            metadata:
              annotations:
                metallb.universe.tf/address-pool: internalapi
                metallb.universe.tf/allow-shared-ip: internalapi
                metallb.universe.tf/loadBalancerIPs: 172.17.0.80
            spec:
              type: LoadBalancer
      replicas: 1
      customServiceConfig: |
        [DEFAULT]
        default_volume_type=tripleo
    cinderScheduler:
      replicas: 1
    cinderBackup:
      networkAttachments:
      - storage
      replicas: 1
      customServiceConfig: |
        [DEFAULT]
        backup_driver=cinder.backup.drivers.ceph.CephBackupDriver
        backup_ceph_conf=/etc/ceph/ceph.conf
        backup_ceph_user=openstack
        backup_ceph_pool=backups
    cinderVolumes:
      ceph:
        networkAttachments:
        - storage
        replicas: 1
        customServiceConfig: |
          [tripleo_ceph]
          backend_host=hostgroup
          volume_backend_name=tripleo_ceph
          volume_driver=cinder.volume.drivers.rbd.RBDDriver
          rbd_ceph_conf=/etc/ceph/ceph.conf
          rbd_user=openstack
          rbd_pool=volumes
          rbd_flatten_volume_from_snapshot=False
          report_discard_supported=True
```

2. Once the services have been deployed you need to clean up the old scheduler and backup services which will appear as being down while you have others that appear as being up:

```
openstack volume service list

+------------------+------------------------+------+---------+-------+----------------------------+
| Binary           | Host                   | Zone | Status  | State | Updated At                 |
+------------------+------------------------+------+---------+-------+----------------------------+
| cinder-backup    | standalone.localdomain | nova | enabled | down  | 2023-06-
28T11:00:59.000000 |
```

```
| cinder-scheduler | standalone.localdomain | nova | enabled | down  | 2023-06-
28T11:00:29.000000 |
| cinder-volume    | hostgroup@tripleo_ceph | nova | enabled | up    | 2023-06-
28T17:00:03.000000 |
| cinder-scheduler | cinder-scheduler-0     | nova | enabled | up    | 2023-06-
28T17:00:02.000000 |
| cinder-backup    | cinder-backup-0        | nova | enabled | up    | 2023-06-
28T17:00:01.000000 |
+------------------+------------------------+------+---------+-------+----------------------------+
```

3. In this case you need to remove services for hosts **standalone.localdomain**

```
oc exec -it cinder-scheduler-0 -- cinder-manage service remove cinder-backup
standalone.localdomain
oc exec -it cinder-scheduler-0 -- cinder-manage service remove cinder-scheduler
standalone.localdomain
```

   The reason why we haven't preserved the name of the backup service is because we have taken the opportunity to change its configuration to support Active-Active, even though we are not doing so right now because we have 1 replica.

4. Now that the Block Storage services are running, the DB schema migration has been completed and you can proceed to apply the DB data migrations. While it is not necessary to run these data migrations at this precise moment, because you can run them right before the next upgrade, for adoption it is best to run them now to make sure there are no issues before running production workloads on the deployment.
   The command to run the DB data migrations is:

```
oc exec -it cinder-scheduler-0 -- cinder-manage db online_data_migrations
```

Verification

Before you can run any checks you need to set the right cloud configuration for the **openstack** command to be able to connect to your RHOCP control plane.

1. Ensure that the **openstack** alias is defined:

```
alias openstack="oc exec -t openstackclient -- openstack"
```

Now you can run a set of tests to confirm that the deployment is using your old database contents:

1. See that Block Storage service endpoints are defined and pointing to the control plane FQDNs:

```
openstack endpoint list --service cinderv3
```

2. Check that the Block Storage services are running and up. The API won't show but if you get a response you know it's up as well:

```
openstack volume service list
```

3. Check that your old volume types, volumes, snapshots, and backups are there:

```
openstack volume type list
openstack volume list
```

```
openstack volume snapshot list
openstack volume backup list
```

To confirm that the configuration is working, the following basic operations are recommended:

1. Create a volume from an image to check that the connection to Image Service (glance) is working.

```
openstack volume create --image cirros --bootable --size 1 disk_new
```

2. Backup the old attached volume to a new backup. Example:

```
openstack --os-volume-api-version 3.47 volume create --backup backup restored
```

NOTE

You do not boot a Compute service (nova) instance using the new volume from image or try to detach the old volume because Compute service and the Block Storage service are still not connected.

4.9. ADOPTING THE DASHBOARD SERVICE

Prerequisites

- Previous Adoption steps completed. Notably, Memcached and Identity service (keystone) should be already adopted.

Procedure

- Patch **OpenStackControlPlane** to deploy the Dashboard service:

```
oc patch openstackcontrolplane openstack --type=merge --patch '
spec:
  horizon:
    enabled: true
    apiOverride:
      route: {}
    template:
      memcachedInstance: memcached
      secret: osp-secret
'
```

Verification

1. See that the Dashboard service instance is successfully deployed and ready

```
oc get horizon
```

2. Check that the Dashboard service is reachable and returns status code **200**

```
PUBLIC_URL=$(oc get horizon horizon -o jsonpath='{.status.endpoint}')
curl --silent --output /dev/stderr --head --write-out "%{http_code}"
"$PUBLIC_URL/dashboard/auth/login/?next=/dashboard/" -k | grep 200
```

4.10. ADOPTING THE SHARED FILE SYSTEMS SERVICE

The Shared File Systems service (manila) provides Red Hat OpenStack Platform (RHOSP) users with a self-service API to create and manage file shares. File shares (or simply, "shares"), are built for concurrent read/write access by any number of clients. This, coupled with the inherent elasticity of the underlying storage makes the Shared File Systems service essential in cloud environments with require RWX ("read write many") persistent storage.

File shares in RHOSP are accessed directly over a network. Hence, it is essential to plan the networking of the cloud to create a successful and sustainable orchestration layer for shared file systems.

The Shared File Systems service supports two levels of storage networking abstractions - one where users can directly control the networking for their respective file shares; and another where the storage networking is configured by the RHOSP administrator. It is important to ensure that the networking in the RHOSP 17.1 environment matches the network plans for your new cloud after adoption. This ensures that tenant workloads remain connected to storage through the adoption process, even as the control plane suffers a minor interruption. The Shared File Systems service control plane services are not in the data path; and shutting down the API, scheduler and share manager services will not impact access to existing shared file systems.

Typically, storage and storage device management networks are separate. Shared File Systems services only need access to the storage device management network. For example, if a Ceph cluster was used in the deployment, the "storage" network refers to the Ceph cluster's public network, and the Shared File Systems service's share manager service needs to be able to reach it.

4.10.1. Changes to CephFS through NFS

If the Red Hat OpenStack Platform (RHOSP) 17.1 deployment uses CephFS through NFS as a backend for Shared File Systems service (manila), there's a **ceph-nfs** service on the RHOSP controller nodes deployed and managed by director. This service cannot be directly imported into Red Hat OpenStack Services on OpenShift (RHOSO) 18.0. On RHOSO 18.0, the Shared File Systems service only supports using a "clustered" NFS service that is directly managed on the Red Hat Ceph Storage cluster. So, adoption with this service will involve a data path disruption to existing NFS clients. The timing of this disruption can be controlled by the deployer independent of this adoption procedure.

On RHOSP 17.1, pacemaker controls the high availability of the **ceph-nfs** service. This service is assigned a Virtual IP (VIP) address that is also managed by pacemaker. The VIP is typically created on an isolated **StorageNFS** network. There are ordering and collocation constraints established between this VIP, **ceph-nfs** and the Shared File Systems service's share manager service on the controller nodes. Prior to adopting Shared File Systems service, pacemaker's ordering and collocation constraints must be adjusted to separate the share manager service. This establishes **ceph-nfs** with its VIP as an isolated, standalone NFS service that can be decommissioned at will after completing the RHOSP adoption.

Red Hat Ceph Storage 7.0 introduced a native **clustered Ceph NFS service**. This service has to be deployed on the Red Hat Ceph Storage cluster using the Ceph orchestrator prior to adopting the Shared File Systems service. This NFS service will eventually replace the standalone NFS service from RHOSP 17.1 in your deployment. When the Shared File Systems service is adopted into the RHOSO 18.0 environment, it will establish all the existing exports and client restrictions on the new clustered Ceph NFS service. Clients can continue to read and write data on their existing NFS shares, and are not affected until the old standalone NFS service is decommissioned. This switchover window allows clients to re-mount the same share from the new clustered Ceph NFS service during a scheduled downtime.

In order to ensure that existing clients can easily switchover to the new NFS service, it is necessary that the clustered Ceph NFS service is assigned an IP address from the same isolated **StorageNFS** network. Doing this will ensure that NFS users aren't expected to make any networking changes to their existing workloads. These users only need to discover and re-mount their shares using new export paths. When the adoption procedure is complete, RHOSP users can query the Shared File Systems service API to list the export locations on existing shares to identify the **preferred** paths to mount these shares. These **preferred** paths will correspond to the new clustered Ceph NFS service in contrast to other non-preferred export paths that continue to be displayed until the old isolated, standalone NFS service is decommissioned.

See Creating a NFS Ganesha cluster for instructions on setting up a clustered NFS service.

4.10.2. Deploying the Shared File Systems service control plane

Copy the Shared File Systems service (manila) configuration from the Red Hat OpenStack Platform 17.1 deployment, and then deploy the Shared File Systems service on the control plane.

Prerequisites

- Ensure that Shared File Systems service systemd services (**api**, **cron**, **scheduler**) are stopped. For more information, see Stopping Red Hat OpenStack Platform services.
- If the deployment uses CephFS through NFS as a storage backend, ensure that pacemaker ordering and collocation constraints are adjusted. For more information, see Stopping Red Hat OpenStack Platform services.
- Ensure that the Shared File Systems service pacemaker service (**openstack-manila-share**) is stopped. For more information, see Stopping Red Hat OpenStack Platform services.
- Ensure that the database migration has completed. For more information, see Migrating databases to MariaDB instances.
- Ensure that Red Hat OpenShift Container Platform nodes where **manila-share** service will be deployed can reach the management network that the storage system is in.
- If the deployment uses CephFS through NFS as a storage backend, ensure that a new clustered Ceph NFS service is deployed on the Ceph cluster with the help of Ceph orchestrator. For more information, see Creating a Ceph NFS cluster .
- Ensure that services such as Identity service (keystone) and memcached are available prior to adopting the Shared File Systems services.
- If tenant-driven networking was enabled (**driver_handles_share_servers=True**), ensure that Networking service (neutron) has been deployed prior to adopting Shared File Systems services.

Procedure

1. Define the **CONTROLLER1_SSH** environment variable, if it hasn't been defined already. Then copy the configuration file from RHOSP 17.1 for reference.

```
$CONTROLLER1_SSH cat /var/lib/config-data/puppet-generated/manila/etc/manila/manila.conf | awk '!/^ *#/ && NF' > ~/manila.conf
```

2. Review this configuration alongside any configuration changes that were noted since RHOSP 17.1. Not all of it makes sense to bring into the new cloud environment:

   - The Shared File Systems service operator is capable of setting up database related configuration (**[database]**), service authentication (**auth_strategy**, **[keystone_authtoken]**), message bus configuration (**transport_url**, **control_exchange**), the default paste config (**api_paste_config**) and inter-service communication configuration (**[neutron]**, **[nova]**, **[cinder]**, **[glance] [oslo_messaging_*]**). So all of these can be ignored.

   - Ignore the **osapi_share_listen** configuration. In Red Hat OpenStack Services on OpenShift (RHOSO) 18.0, you rely on Red Hat OpenShift Container Platform routes and ingress.

   - Pay attention to policy overrides. In RHOSO 18.0, the Shared File Systems service ships with a secure default RBAC, and overrides may not be necessary. If a custom policy is necessary, you must provide it as a **ConfigMap**. The following sample spec illustrates how a **ConfigMap** called **manila-policy** can be set up with the contents of a file called **policy.yaml**.

     ```
     spec:
       manila:
         enabled: true
         template:
           manilaAPI:
             customServiceConfig: |
               [oslo_policy]
               policy_file=/etc/manila/policy.yaml
           extraMounts:
           - extraVol:
             - extraVolType: Undefined
               mounts:
               - mountPath: /etc/manila/
                 name: policy
                 readOnly: true
               propagation:
               - ManilaAPI
               volumes:
               - name: policy
                 projected:
                   sources:
                   - configMap:
                       name: manila-policy
                       items:
                         - key: policy
                           path: policy.yaml
     ```

   - You must preserve the value of the **host** option under the **[DEFAULT]** section as **hostgroup**.

   - The Shared File Systems service API service needs the **enabled_share_protocols** option to be added in the **customServiceConfig** section in **manila: template: manilaAPI**.

   - If you had scheduler overrides, add them to the **customServiceConfig** section in **manila: template: manilaScheduler**.

- If you had multiple storage backend drivers configured with RHOSP 17.1, you will need to split them up when deploying RHOSO 18.0. Each storage backend driver needs to use its own instance of the **manila-share** service.

- If a storage backend driver needs a custom container image, find it on the RHOSP Ecosystem Catalog and set **manila: template: manilaShares: <custom name> : containerImage** value. The following example illustrates multiple storage backend drivers, using custom container images.

```
spec:
  manila:
    enabled: true
    template:
      manilaAPI:
        customServiceConfig: |
          [DEFAULT]
          enabled_share_protocols = nfs
        replicas: 3
      manilaScheduler:
        replicas: 3
      manilaShares:
       netapp:
         customServiceConfig: |
           [DEFAULT]
           debug = true
           enabled_share_backends = netapp
           host = hostgroup
           [netapp]
           driver_handles_share_servers = False
           share_backend_name = netapp
           share_driver = manila.share.drivers.netapp.common.NetAppDriver
           netapp_storage_family = ontap_cluster
           netapp_transport_type = http
         replicas: 1
       pure:
         customServiceConfig: |
           [DEFAULT]
           debug = true
           enabled_share_backends=pure-1
           host = hostgroup
           [pure-1]
           driver_handles_share_servers = False
           share_backend_name = pure-1
           share_driver =
manila.share.drivers.purestorage.flashblade.FlashBladeShareDriver
           flashblade_mgmt_vip = 203.0.113.15
           flashblade_data_vip = 203.0.10.14
         containerImage: registry.connect.redhat.com/purestorage/openstack-manila-
share-pure-rhosp-18-0
           replicas: 1
```

3. If providing sensitive information, such as passwords, hostnames and usernames, it is recommended to use Red Hat OpenShift Container Platform secrets, and the **customServiceConfigSecrets** key. An example:

```
cat << __EOF__ > ~/netapp_secrets.conf
```

```
[netapp]
netapp_server_hostname = 203.0.113.10
netapp_login = fancy_netapp_user
netapp_password = secret_netapp_password
netapp_vserver = mydatavserver
__EOF__
```

```
oc create secret generic osp-secret-manila-netapp --from-file=~/netapp_secrets.conf -n
openstack
```

- **customConfigSecrets** can be used in any service, the following is a config example using the secret you created above.

```
spec:
  manila:
    enabled: true
    template:
      < . . . >
      manilaShares:
       netapp:
         customServiceConfig: |
           [DEFAULT]
           debug = true
           enabled_share_backends = netapp
           host = hostgroup
           [netapp]
           driver_handles_share_servers = False
           share_backend_name = netapp
           share_driver = manila.share.drivers.netapp.common.NetAppDriver
           netapp_storage_family = ontap_cluster
           netapp_transport_type = http
         customServiceConfigSecrets:
           - osp-secret-manila-netapp
         replicas: 1
  < . . . >
```

- If you need to present extra files to any of the services, you can use **extraMounts**. For example, when using ceph, you'd need the Shared File Systems service ceph user's keyring file as well as the **ceph.conf** configuration file available. These are mounted via **extraMounts** as done with the example below.

- Ensure that the names of the backends (**share_backend_name**) remain as they did on RHOSP 17.1.

- It is recommended to set the replica count of the **manilaAPI** service and the **manilaScheduler** service to 3. You should ensure to set the replica count of the **manilaShares** service/s to 1.

- Ensure that the appropriate storage management network is specified in the **manilaShares** section. The example below connects the **manilaShares** instance with the CephFS backend driver to the **storage** network.

- Prior to adopting the **manilaShares** service for CephFS through NFS, ensure that you have a clustered Ceph NFS service created. You will need to provide the name of the service as ``cephfs_nfs_cluster_id``.

4. Patch **OpenStackControlPlane** to deploy the Shared File Systems service; here's an example that uses Native CephFS:

```
cat << __EOF__ > ~/manila.patch
spec:
  manila:
    enabled: true
    apiOverride:
      route: {}
    template:
      databaseInstance: openstack
      databaseAccount: manila
      secret: osp-secret
      manilaAPI:
        replicas: 3
        customServiceConfig: |
          [DEFAULT]
          enabled_share_protocols = cephfs
        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
      manilaScheduler:
        replicas: 3
      manilaShares:
        cephfs:
          replicas: 1
          customServiceConfig: |
            [DEFAULT]
            enabled_share_backends = tripleo_ceph
            host = hostgroup
            [cephfs]
            driver_handles_share_servers=False
            share_backend_name=cephfs
            share_driver=manila.share.drivers.cephfs.driver.CephFSDriver
            cephfs_conf_path=/etc/ceph/ceph.conf
            cephfs_auth_id=openstack
            cephfs_cluster_name=ceph
            cephfs_volume_mode=0755
            cephfs_protocol_helper_type=CEPHFS
          networkAttachments:
              - storage
__EOF__
```

Below is an example that uses CephFS through NFS. In this example:

- The **cephfs_ganesha_server_ip** option is preserved from the configuration on the old RHOSP 17.1 environment.
- The **cephfs_nfs_cluster_id** option is set with the name of the NFS cluster created on Ceph.

```
cat << __EOF__ > ~/manila.patch
spec:
  manila:
    enabled: true
    apiOverride:
      route: {}
    template:
      databaseInstance: openstack
      secret: osp-secret
      manilaAPI:
        replicas: 3
        customServiceConfig: |
          [DEFAULT]
          enabled_share_protocols = cephfs
        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
      manilaScheduler:
        replicas: 3
      manilaShares:
        cephfs:
          replicas: 1
          customServiceConfig: |
            [DEFAULT]
            enabled_share_backends = cephfs
            host = hostgroup
            [cephfs]
            driver_handles_share_servers=False
            share_backend_name=tripleo_ceph
            share_driver=manila.share.drivers.cephfs.driver.CephFSDriver
            cephfs_conf_path=/etc/ceph/ceph.conf
            cephfs_auth_id=openstack
            cephfs_cluster_name=ceph
            cephfs_protocol_helper_type=NFS
            cephfs_nfs_cluster_id=cephfs
            cephfs_ganesha_server_ip=172.17.5.47
          networkAttachments:
              - storage
__EOF__
```

```
oc patch openstackcontrolplane openstack --type=merge --patch-file=~/manila.patch
```

Verification

1. Inspect the resulting Shared File Systems service pods:

```
oc get pods -l service=manila
```

2. Check that the Shared File Systems service API service is registered in Identity service (keystone):

```
openstack service list | grep manila
```

```
openstack endpoint list | grep manila

| 1164c70045d34b959e889846f9959c0e | regionOne | manila       | share        | True    |
internal  | http://manila-internal.openstack.svc:8786/v1/%(project_id)s        |
| 63e89296522d4b28a9af56586641590c | regionOne | manilav2     | sharev2      | True    |
public    | https://manila-public-openstack.apps-crc.testing/v2                |
| af36c57adcdf4d50b10f484b616764cc | regionOne | manila       | share        | True    | public
| https://manila-public-openstack.apps-crc.testing/v1/%(project_id)s |
| d655b4390d7544a29ce4ea356cc2b547 | regionOne | manilav2     | sharev2      | True    |
internal  | http://manila-internal.openstack.svc:8786/v2                       |
```

3. Test the health of the service:

```
openstack share service list
openstack share pool list --detail
```

4. Check on existing workloads:

```
openstack share list
openstack share snapshot list
```

5. You can create further resources:

```
openstack share create cephfs 10 --snapshot mysharesnap --name myshareclone
openstack share create nfs 10 --name mynfsshare
openstack share export location list mynfsshare
```

4.10.3. Decommissioning the Red Hat OpenStack Platform standalone Ceph NFS service

If the deployment uses CephFS through NFS, you must inform your Red Hat OpenStack Platform(RHOSP) users that the old, standalone NFS service will be decommissioned. Users can discover the new export locations for their pre-existing shares by querying the Shared File Systems service API. To stop using the old NFS server, they need to unmount and remount their shared file systems on each client. If users are consuming the Shared File Systems service shares via the Shared File Systems service CSI plugin for Red Hat OpenShift Container Platform, this migration can be done by scaling down the application pods and scaling them back up. Clients spawning new workloads must be discouraged from using share exports via the old NFS service. The Shared File Systems service will no longer communicate with the old NFS service, and so it cannot apply or alter any export rules on the old NFS service.

Since the old NFS service will no longer be supported by future software upgrades, it is recommended that the decommissioning period is short.

Procedure

1. Once the old NFS service is no longer used, you can adjust the configuration for the **manila-share** service to remove the **cephfs_ganesha_server_ip** option. Doing this will restart the **manila-share** process and remove the export locations that pertained to the old NFS service from all the shares.

```
cat << __EOF__ > ~/manila.patch
spec:
  manila:
    enabled: true
    apiOverride:
      route: {}
    template:
      manilaShares:
        cephfs:
          replicas: 1
          customServiceConfig: |
            [DEFAULT]
            enabled_share_backends = cephfs
            host = hostgroup
            [cephfs]
            driver_handles_share_servers=False
            share_backend_name=cephfs
            share_driver=manila.share.drivers.cephfs.driver.CephFSDriver
            cephfs_conf_path=/etc/ceph/ceph.conf
            cephfs_auth_id=openstack
            cephfs_cluster_name=ceph
            cephfs_protocol_helper_type=NFS
            cephfs_nfs_cluster_id=cephfs
          networkAttachments:
              - storage
__EOF__
```

```
oc patch openstackcontrolplane openstack --type=merge --patch-file=~/manila.patch
```

2. To cleanup the standalone ceph nfs service from the RHOSP control plane nodes, you can disable and delete the pacemaker resources associated with the service. Replace **<VIP>** in the following commands with the IP address assigned to the ceph-nfs service in your environment.

```
sudo pcs resource disable ceph-nfs
sudo pcs resource disable ip-<VIP>
sudo pcs resource unmanage ceph-nfs
sudo pcs resource unmanage ip-<VIP>
```

4.11. ADOPTING THE BARE METAL PROVISIONING SERVICE

Review information about your Bare Metal Provisioning service (ironic) configuration and then adopt your Bare Metal Provisioning service to the Red Hat OpenStack Services on OpenShift (RHOSO) control plane.

4.11.1. Bare Metal Provisioning service configurations

The Bare Metal Provisioning service (ironic) is configured by using configuration snippets. For more information about the configuration snippets, see Service configurations.

director generally took care to not override the defaults of the Bare Metal Provisioning service, however as with any system of descreet configuration management attempting to provide a cross-version compatability layer, some configuration was certainly defaulted in particular ways. For example, PXE Loader file names were often overridden at intermediate layers, and you will thus want to pay particular attention to the settings you choose to apply in your adopted deployment. The operator attempts to apply reasonable working default configuration, but if you override them with prior configuration, your experience may not be ideal or your new Bare Metal Provisioning service will fail to operate. Similarly, additional configuration may be necessary, for example if your **ironic.conf** has additional hardware types enabled and in use.

Furthermore, the model of reasonable defaults includes commonly used hardware-types and driver interfaces. For example, if you previously needed to enable the **redfish-virtual-media** boot interface and the **ramdisk** deploy interface, the good news is you don't need to, they are enabled by default. One aspect to be on the watch for after completing adoption is when adding new bare metal nodes, the driver interface selection occurs based upon order of presidence in the configuration if not explicitly set on the node creation request or as an established default in **ironic.conf**.

That being said, some configuration parameters are provided as either a convenience to the operator so they don't need to be set on an individual node level while also needing to know precise values, for example, network UUID values, or it is centrally configured in **ironic.conf** as the setting controls behaivor a security control.

The settings, if configured, and formatted as [section] and parameter name, are critical to be maintained from the prior deployment to the new deployment as it will govern quite a bit of the underlying behavior and values in the previous configuration, would have used specific values if set.

- [neutron]cleaning_network
- [neutron]provisioning_network
- [neutron]rescuing_network
- [neutron]inspection_network
- [conductor]automated_clean
- [deploy]erase_devices_priority
- [deploy]erase_devices_metadata_priority
- [conductor]force_power_state_during_sync

The following parameters **can** be set individually on a node, however, some operators choose to use embedded configuration options to avoid the need to set it individually when creating/managing bare metal nodes. We recommend you check your prior ironic.conf file for these parameters, and if set apply as specific override configuration.

- [conductor]bootloader
- [conductor]rescue_ramdisk
- [conductor]rescue_kernel

- [conductor]deploy_kernel
- [conductor]deploy_ramdisk

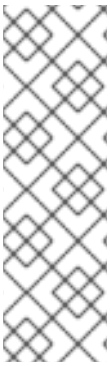
Finally, a parameter which may be important based upon your configuration and experience, are the instances of **kernel_append_params**, formerly **pxe_append_params** in the **[pxe]** and **[redfish]** configuration sections. Largely this parameter is used to appy boot time options like "console" for the deployment ramdisk and as such often seeks to be changed.

As a warning, hardware types set via the **ironic.conf enabled_hardware_types** parameter and hardware type driver interfaces starting with **staging-** are not available to be migrated into an adopted configuration.

Furthermore, director-based deployments made architectural decisions based upon self-management of services. When adopting deployments, you don't necessarilly need multiple replicas of secondary services such as the Introspection service. Should the host the container is running upon fail, Red Hat OpenShift Container Platform will restart the container on another host. The short-term transitory loss

4.11.2. Deploying the Bare Metal Provisioning service

Applying the configuration to deploy the Bare Metal Provisioning service (ironic).



NOTE

By default, newer versions of the Bare Metal Provisioning service contain a more restrictive access control model while also becoming multi-tenant aware. As a result, bare metal nodes might be missing from a **openstack baremetal node list** command after upgrade. Your nodes have not been deleted, but you must set the **owner** field on each bare metal node due to the increased access restrictions in the role-based access control (RBAC) model. Because this involves access controls and the model of use which can be site specific, it is highly recommended that you identify the "project" to "own" the bare metal nodes.

Prerequisites

- Previous Adoption steps completed. Notably, the service databases must already be imported into the control plane MariaDB, Identity service (keystone), Networking service (neutron), Image Service (glance), and Block Storage service (cinder) should be in an operational state. Ideally, Compute service (nova) has not been adopted yet if Bare Metal Provisioning service is leveraged in a Bare Metal as a Service configuration.
- Before deploying Red Hat OpenStack Platform in Red Hat OpenStack Services on OpenShift (RHOSO), you must ensure that the networks are ready, that you have decided the node selection, and also make sure any necessary changes to the RHOSO nodes have been made. For Bare Metal Provisioning service conductor services, it is necessary that the services be able to reach Baseboard Management Controllers of hardware which is configured to be managed by Bare Metal Provisioning service. If this hardware is unreachable, the nodes may enter "maintenance" state and be unable to be acted upon until connectivity is restored at a later point in time.
- You need the contents of **ironic.conf** file. Download the file so that you can access it locally:

```
$CONTROLLER1_SSH cat /var/lib/config-data/puppet-generated/ironic/etc/ironic/ironic.conf
> ironic.conf
```

NOTE

It is critical that this configuration file comes from one of the controllers and not a director undercloud node. The director undercloud node specifically operated with different configuration which would not be appropriate or applicable to apply when adopting the Overcloud Ironic deployment.

- If adopting the Ironic Inspector service you need the value of the **IronicInspectorSubnets** director parameter. Use the same values to populate the **dhcpRanges** parameter in the target environment.
- Define the following shell variables. The values that are used are examples. Replace these example values with values that are correct for your environment:

```
alias openstack="oc exec -t openstackclient -- openstack"
```

Procedure

1. Patch the **OpenStackControlPlane** to deploy the Bare Metal Provisioning service:

```
oc patch openstackcontrolplane openstack -n openstack --type=merge --patch '
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
        networkAttachments:
          - baremetal
        provisionNetwork: baremetal
        storageRequest: 10G
        customServiceConfig: |
          [neutron]
          cleaning_network=<cleaning network uuid>
          provisioning_network=<provisioning network uuid>
          rescuing_network=<rescuing network uuid>
          inspection_network=<introspection network uuid>
          [conductor]
          automated_clean=true
      ironicInspector:
        replicas: 1
```

```
      inspectionNetwork: baremetal
      networkAttachments:
        - baremetal
      dhcpRanges:
        - name: inspector-0
          cidr: 172.20.1.0/24
          start: 172.20.1.190
          end: 172.20.1.199
          gateway: 172.20.1.1
      serviceUser: ironic-inspector
      databaseAccount: ironic-inspector
      passwordSelectors:
        database: IronicInspectorDatabasePassword
        service: IronicInspectorPassword
    ironicNeutronAgent:
      replicas: 1
      rabbitMqClusterName: rabbitmq
    secret: osp-secret
'
```

The operator begins to apply the configuration and start the necessary Bare Metal Provisioning services. Once the services have reached a running state, the Bare Metal Provisioning service automatically begins polling the power state of bare metal nodes for which it is configured to manage.

2. Wait for Bare Metal Provisioning control plane services' custom resources to become ready:

```
oc wait --for condition=Ready --timeout=300s ironics.ironic.openstack.org ironic
//kgilliga: Is "optionally verify the individual service" part of the code block, or is it a separate step?
# Optionally verify the individual services
oc wait --for condition=Ready --timeout=300s ironicapis.ironic.openstack.org ironic-api
oc wait --for condition=Ready --timeout=300s ironicconductors.ironic.openstack.org ironic-conductor
oc wait --for condition=Ready --timeout=300s ironicinspectors.ironic.openstack.org ironic-inspector
oc wait --for condition=Ready --timeout=300s ironicneutronagents.ironic.openstack.org ironic-ironic-neutron-agent
```

3. Update the DNS Nameservers on the provisoning/cleaning/rescue networks. For name resolution to work for Bare Metal Provisioning service operations the DNS nameserver must be set to use the internal DNS servers in the RHOSO control plane:

```
openstack subnet set --dns-nameserver 192.168.122.80 provisioning-subnet
```

4. Your Bare Metal Provisioning service nodes might be missing from a **openstack baremetal node list** command due to increased access restrictions in the role-based access control model. To see the nodes again, temporarily disable the new role- based access control policy, which you can then re-enable after setting the *owner* field on the nodes.

```
oc patch openstackcontrolplane openstack -n openstack --type=merge --patch '
spec:
  ironic:
    enabled: true
    template:
```

```
      databaseInstance: openstack
      ironicAPI:
        replicas: 1
        customServiceConfig: |
          [oslo_policy]
          enforce_scope=false
          enforce_new_defaults=false
'
```

5. Once this configuration has applied, the operator restarts the Ironic API service disabling the new RBAC policy which is enabled by default. After which, you should be able to view bare metal nodes without an **owner** field:

```
openstack baremetal node list -f uuid,provision_state,owner
```

6. Run the following command to assign all bare metal nodes with no owner to a new project, for example, the "admin" project:

```
ADMIN_PROJECT_ID=$(openstack project show -c id -f value --domain default admin)
for node in $(openstack baremetal node list -f json -c UUID -c Owner | jq -r '.[] | select(.Owner == null) | .UUID'); do openstack baremetal node set --owner $ADMIN_PROJECT_ID $node; done
```

7. Re-apply the default access control policy:

```
oc patch openstackcontrolplane openstack -n openstack --type=merge --patch '
spec:
  ironic:
    enabled: true
    template:
      databaseInstance: openstack
      ironicAPI:
        replicas: 1
        customServiceConfig: |
          [oslo_policy]
          enforce_scope=true
          enforce_new_defaults=true
'
```

Verification

- After applying the configuration update to RHOSO, apply the configuration and start the related services. The Bare Metal Provisioning service begins to poll power state of the bare metal nodes:

```
openstack endpoint list |grep ironic
openstack baremetal node list
```

  The time required for the Bare Metal Provisioning service to review and reconcile the power state of bare metal nodes is dependent upon the number of operating conductors through the **replicas** parameter and which are present in the Bare Metal Provisioning service deployment being adopted.

4.12. ADOPTING THE ORCHESTRATION SERVICE

Adopting the Orchestration service (heat) means that an existing **OpenStackControlPlane** custom resource (CR), where Orchestration service is supposed to be disabled, should be patched to start the service with the configuration parameters provided by the source environment.

After the adoption process has been completed, a user can expect that they will then have CRs for **Heat**, **HeatAPI**, **HeatEngine** and **HeatCFNAPI**. Additionally, a user should have endpoints created within Identity service (keystone) to facilitate the above mentioned servies.

This guide also assumes that:

1. A director environment (the source Cloud) is running on one side;
2. A Red Hat OpenShift Container Platform environment is running on the other side.

Prerequisites

- Previous Adoption steps completed. Notably, MariaDB and Identity service should be already adopted.
- In addition, if your existing Orchestration service stacks contain resources from other services such as Networking service (neutron), Compute service (nova), Object Storage service (swift), etc. Those services should be adopted first before trying to adopt Orchestration service.

Procedure

1. Patch the **osp-secret** to update the **HeatAuthEncryptionKey** and **HeatPassword**. This needs to match what you have configured in the existing director Orchestration service configuration. You can retrieve and verify the existing **auth_encryption_key** and **service** passwords via:

```
[stack@rhosp17 ~]$ grep -E 'HeatPassword|HeatAuth' ~/overcloud-
deploy/overcloud/overcloud-passwords.yaml
  HeatAuthEncryptionKey: Q60Hj8PqbrDNu2dDCbyIQE2dibpQUPg2
  HeatPassword: dU2N0Vr2bdelYH7eQonAwPfI3
```

2. And verifying on one of the Controllers that this is indeed the value in use:

```
[stack@rhosp17 ~]$ ansible -i overcloud-deploy/overcloud/config-download/overcloud/tripleo-
ansible-inventory.yaml overcloud-controller-0 -m shell -a "grep auth_encryption_key
/var/lib/config-data/puppet-generated/heat/etc/heat/heat.conf | grep -Ev '^#|^$'" -b
overcloud-controller-0 | CHANGED | rc=0 >>
auth_encryption_key=Q60Hj8PqbrDNu2dDCbyIQE2dibpQUPg2
```

3. This password needs to be base64 encoded and added to the **osp-secret**

```
❯ echo Q60Hj8PqbrDNu2dDCbyIQE2dibpQUPg2 | base64
UTYwSGo4UHFickROdTJkRENieUlRRTJkaWJwUVVQZzIK

❯ oc patch secret osp-secret --type='json' -p='[{"op" : "replace" ,"path" :
"/data/HeatAuthEncryptionKey" ,"value" :
"UTYwSGo4UHFickROdTJkRENieUlRRTJkaWJwUVVQZzIK"}]'
secret/osp-secret patched
```

4. Patch **OpenStackControlPlane** to deploy the Orchestration service:

```
oc patch openstackcontrolplane openstack --type=merge --patch '
spec:
  heat:
    enabled: true
    apiOverride:
      route: {}
    template:
      databaseInstance: openstack
      databaseAccount: heat
      secret: osp-secret
      memcachedInstance: memcached
      passwordSelectors:
        authEncryptionKey: HeatAuthEncryptionKey
        database: HeatDatabasePassword
        service: HeatPassword
'
```

Verification

1. Ensure all of the CRs reach the "Setup Complete" state:

```
❯ oc get Heat,HeatAPI,HeatEngine,HeatCFNAPI
NAME                          STATUS   MESSAGE
heat.heat.openstack.org/heat   True     Setup complete

NAME                                STATUS   MESSAGE
heatapi.heat.openstack.org/heat-api   True     Setup complete

NAME                                    STATUS   MESSAGE
heatengine.heat.openstack.org/heat-engine   True     Setup complete

NAME                                    STATUS   MESSAGE
heatcfnapi.heat.openstack.org/heat-cfnapi   True     Setup complete
```

2. Check that the Orchestration service is registered in Identity service:

```
 oc exec -it openstackclient -- openstack service list -c Name -c Type
+------------+----------------+
| Name       | Type           |
+------------+----------------+
| heat       | orchestration  |
| glance     | image          |
| heat-cfn   | cloudformation |
| ceilometer | Ceilometer     |
| keystone   | identity       |
| placement  | placement      |
| cinderv3   | volumev3       |
| nova       | compute        |
| neutron    | network        |
+------------+----------------+
```

```
❯ oc exec -it openstackclient -- openstack endpoint list --service=heat -f yaml
- Enabled: true
  ID: 1da7df5b25b94d1cae85e3ad736b25a5
```

```
  Interface: public
  Region: regionOne
  Service Name: heat
  Service Type: orchestration
  URL: http://heat-api-public-openstack-operators.apps.okd.bne-shift.net/v1/%(tenant_id)s
- Enabled: true
  ID: 414dd03d8e9d462988113ea0e3a330b0
  Interface: internal
  Region: regionOne
  Service Name: heat
  Service Type: orchestration
  URL: http://heat-api-internal.openstack-operators.svc:8004/v1/%(tenant_id)s
```

3. Check the Orchestration service engine services are up:

```
 oc exec -it openstackclient -- openstack orchestration service list -f yaml
- Binary: heat-engine
  Engine ID: b16ad899-815a-4b0c-9f2e-e6d9c74aa200
  Host: heat-engine-6d47856868-p7pzz
  Hostname: heat-engine-6d47856868-p7pzz
  Status: up
  Topic: engine
  Updated At: '2023-10-11T21:48:01.000000'
- Binary: heat-engine
  Engine ID: 887ed392-0799-4310-b95c-ac2d3e6f965f
  Host: heat-engine-6d47856868-p7pzz
  Hostname: heat-engine-6d47856868-p7pzz
  Status: up
  Topic: engine
  Updated At: '2023-10-11T21:48:00.000000'
- Binary: heat-engine
  Engine ID: 26ed9668-b3f2-48aa-92e8-2862252485ea
  Host: heat-engine-6d47856868-p7pzz
  Hostname: heat-engine-6d47856868-p7pzz
  Status: up
  Topic: engine
  Updated At: '2023-10-11T21:48:00.000000'
- Binary: heat-engine
  Engine ID: 1011943b-9fea-4f53-b543-d841297245fd
  Host: heat-engine-6d47856868-p7pzz
  Hostname: heat-engine-6d47856868-p7pzz
  Status: up
  Topic: engine
  Updated At: '2023-10-11T21:48:01.000000'
```

4. Verify you can now see your the Orchestration service stacks again. Test whether you can create networks, subnets, ports, or routers:

```
❯ openstack stack list -f yaml
- Creation Time: '2023-10-11T22:03:20Z'
  ID: 20f95925-7443-49cb-9561-a1ab736749ba
  Project: 4eacd0d1cab04427bc315805c28e66c9
  Stack Name: test-networks
  Stack Status: CREATE_COMPLETE
  Updated Time: null
```

4.13. ADOPTING TELEMETRY SERVICES

Adopting Telemetry means that an existing **OpenStackControlPlane** custom resource (CR), where Telemetry services are supposed to be disabled, should be patched to start the service with the configuration parameters provided by the source environment.

This guide also assumes that:

1. A director environment (the source Cloud) is running on one side;
2. A **SNO** / **CodeReadyContainers** is running on the other side.

Prerequisites

- Previous Adoption steps completed. MariaDB, the Identity service (keystone) and the data plane should be already adopted.

Procedure

1. Patch the **OpenStackControlPlane** CR to deploy Ceilometer services:

```
cat << EOF > ceilometer_patch.yaml
spec:
  ceilometer:
    enabled: true
    template:
      centralImage: registry.redhat.io/rhosp-dev-preview/openstack-ceilometer-central-rhel9:18.0
      computeImage: registry.redhat.io/rhosp-dev-preview/openstack-ceilometer-compute-rhel9:18.0
      customServiceConfig: |
        [DEFAULT]
        debug=true
      ipmiImage: registry.redhat.io/rhosp-dev-preview/openstack-ceilometer-ipmi-rhel9:18.0
      nodeExporterImage: quay.io/prometheus/node-exporter:v1.5.0
      notificationImage: registry.redhat.io/rhosp-dev-preview/openstack-ceilometer-notification-rhel9:18.0
      secret: osp-secret
      sgCoreImage: quay.io/infrawatch/sg-core:v5.1.1
EOF
```

2. Optional: If you previously backed up your RHOSP services configuration file from the old environment, you can use os-diff to compare and make sure the configuration is correct. This will produce the difference between both ini configuration files:

```
os-diff diff /tmp/collect_tripleo_configs/ceilometer/etc/ceilometer/ceilometer.conf ceilometer_patch.yaml --crd
```

   For more information, see Reviewing the Red Hat OpenStack Platform control plane configuration.

3. Patch the **OpenStackControlPlane** CR to deploy Ceilometer services:

```
oc patch openstackcontrolplane openstack --type=merge --patch-file ceilometer_patch.yaml
```

Verification

1. Inspect the resulting Ceilometer pods:

```
CEILOMETETR_POD=`oc get pods -l service=ceilometer | tail -n 1 | cut -f 1 -d' '`
oc exec -t $CEILOMETETR_POD -c ceilometer-central-agent -- cat
/etc/ceilometer/ceilometer.conf
```

2. Inspect the resulting Ceilometer IPMI agent pod on data plane nodes:

```
podman ps | grep ceilometer-ipmi
```

3. Inspect enabled pollsters:

```
oc get secret ceilometer-config-data -o jsonpath="{.data['polling\.yaml']}"  | base64 -d
```

4. Enable pollsters according to requirements:

```
cat << EOF > polling.yaml
---
sources:
    - name: pollsters
      interval: 300
      meters:
        - volume.size
        - image.size
        - cpu
        - memory
EOF

oc patch secret ceilometer-config-data  --patch="{\"data\": { \"polling.yaml\": \"$(base64 -w0
polling.yaml)\"}}"
```

4.14. ADOPTING AUTOSCALING

Adopting autoscaling means that an existing **OpenStackControlPlane** custom resource (CR), where Aodh services are supposed to be disabled, should be patched to start the service with the configuration parameters provided by the source environment.

This guide also assumes that:

1. A director environment (the source Cloud) is running on one side;
2. A **SNO** / **CodeReadyContainers** is running on the other side.

Prerequisites

- Previous Adoption steps completed. MariaDB, the Identity service (keystone), the Orchestration service (heat), and Telemetry should be already adopted.

Procedure

1. Patch the **OpenStackControlPlane** CR to deploy autoscaling services:

```
cat << EOF > aodh_patch.yaml
spec:
  autoscaling:
    enabled: true
    prometheus:
      deployPrometheus: false
    aodh:
      customServiceConfig: |
        [DEFAULT]
        debug=true
      secret: osp-secret
      apiImage: "registry.redhat.io/rhosp-dev-preview/openstack-aodh-api-rhel9:18.0"
      evaluatorImage: "registry.redhat.io/rhosp-dev-preview/openstack-aodh-evaluator-
rhel9:18.0"
      notifierImage: "registry.redhat.io/rhosp-dev-preview/openstack-aodh-notifier-rhel9:18.0"
      listenerImage: "registry.redhat.io/rhosp-dev-preview/openstack-aodh-listener-rhel9:18.0"
      passwordSelectors:
      databaseUser: aodh
      databaseInstance: openstack
      memcachedInstance: memcached
EOF
```

2. Optional: If you have previously backed up your RHOSP services configuration file from the old environment, you can use os-diff to compare and make sure the configuration is correct. This will producre the difference between both ini configuration files:

```
os-diff diff /tmp/collect_tripleo_configs/aodh/etc/aodh/aodh.conf aodh_patch.yaml --crd
```

For more information, see Reviewing the Red Hat OpenStack Platform control plane configuration.

3. Patch the **OpenStackControlPlane** CR to deploy Aodh services:

```
oc patch openstackcontrolplane openstack --type=merge --patch-file aodh_patch.yaml
```

Verification

1. If autoscaling services are enabled, inspect Aodh pods:

```
AODH_POD=`oc get pods -l service=aodh | tail -n 1 | cut -f 1 -d' '`
oc exec -t $AODH_POD -c aodh-api -- cat /etc/aodh/aodh.conf
```

2. Check whether Aodh API service is registered in Identity service:

```
openstack endpoint list | grep aodh
| 6a805bd6c9f54658ad2f24e5a0ae0ab6 | regionOne | aodh      | network      | True    | public
| http://aodh-public-openstack.apps-crc.testing  |
| b943243e596847a9a317c8ce1800fa98 | regionOne | aodh      | network      | True    |
internal  | http://aodh-internal.openstack.svc:9696       |
| f97f2b8f7559476bb7a5eafe3d33cee7 | regionOne | aodh      | network      | True    | admin
| http://192.168.122.99:9696                    |
```

3. Create sample resources. You can test whether you can create alarms:

```
openstack alarm create \
--name low_alarm \
--type gnocchi_resources_threshold \
--metric cpu \
--resource-id b7ac84e4-b5ca-4f9e-a15c-ece7aaf68987 \
--threshold 35000000000 \
--comparison-operator lt \
--aggregation-method rate:mean \
--granularity 300 \
--evaluation-periods 3 \
--alarm-action 'log:\\' \
--ok-action 'log:\\' \
--resource-type instance
```

4.15. REVIEWING THE RED HAT OPENSTACK PLATFORM CONTROL PLANE CONFIGURATION

Before starting the adoption workflow, pull the configuration from the Red Hat OpenStack Platform services and director on your file system to back up the configuration files. You can then use the files later, during the configuration of the adopted services, and for the record to compare and make sure nothing has been missed or misconfigured.

Make sure you installed and configured the os-diff tool. For more information, see Comparing configuration files between deployments.

4.15.1. Pulling the configuration from a director deployment

You can pull configuration from your Red Hat OpenStack Platform (RHOSP) services.

All the services are describes in a yaml file:

service config parameters

Procedure

1. Update your ssh parameters according to your environment in the os-diff.cfg. Os-diff uses those parameters to connect to your director node, query and download the configuration files:

```
ssh_cmd=ssh -F ssh.config standalone
container_engine=podman
connection=ssh
remote_config_path=/tmp/tripleo
```

   Make sure the ssh command you provide in **ssh_cmd** parameter is correct and with key authentication.

2. Enable or disable the services that you want in the **/etc/os-diff/config.yaml** file. Make sure that you have the correct rights to edit the file, for example:

```
chown ospng:ospng /etc/os-diff/config.yaml
```

Example with default Identity service (keystone):

```
# service name and file location
services:
  # Service name
  keystone:
    # Bool to enable/disable a service (not implemented yet)
    enable: true
    # Pod name, in both OCP and podman context.
    # It could be strict match or will only just grep the podman_name
    # and work with all the pods which matched with pod_name.
    # To enable/disable use strict_pod_name_match: true/false
    podman_name: keystone
    pod_name: keystone
    container_name: keystone-api
    # pod options
    # strict match for getting pod id in TripleO and podman context
    strict_pod_name_match: false
    # Path of the config files you want to analyze.
    # It could be whatever path you want:
    # /etc/<service_name> or /etc or /usr/share/<something> or even /
    # @TODO: need to implement loop over path to support multiple paths such as:
    # - /etc
    # - /usr/share
    path:
      - /etc/
      - /etc/keystone
      - /etc/keystone/keystone.conf
      - /etc/keystone/logging.conf
```

Repeat this step for each RHOSP service that you want to disable or enable.

1. If you are using non-containerized services, such as the **ovs-external-ids**, os-diff can pull configuration or command output:

```
services:
  ovs_external_ids:
    hosts:
      - standalone
    service_command: "ovs-vsctl list Open_vSwitch . | grep external_ids | awk -F ': ' '{ print $2;
}'"
    cat_output: true
    path:
      - ovs_external_ids.json
    config_mapping:
      ovn-bridge-mappings: edpm_ovn_bridge_mappings
      ovn-bridge: edpm_ovn_bridge
      ovn-encap-type: edpm_ovn_encap_type
      ovn-match-northd-version: ovn_match_northd_version
      ovn-monitor-all: ovn_monitor_all
      ovn-remote-probe-interval: edpm_ovn_remote_probe_interval
      ovn-ofctrl-wait-before-clear: edpm_ovn_ofctrl_wait_before_clear
```

   This service is not an Red Hat OpenStack Platform service executed in a container, so the description and the behavior is different. It is important to correctly configure an SSH config file or equivalent for non-standard services such as OVS. The **ovs_external_ids** does not run in a container, and the ovs data is stored on each host of our cloud: controller_1/controller_2/...

With the **hosts** key, os-diff loops on each host and runs the command in the **service_command** key:

```
    ovs_external_ids:
      path:
        - ovs_external_ids.json
      hosts:
        - standalone
```

The **service_command** provides the required information. It could be a simple cat from a config file. If you want os-diff to get the output of the command and store the output in a file specified by the key path, set **cat_output** to true. Then you can provide a mapping between in this case the EDPM CRD, and the ovs-vsctl output with config_mapping:

```
      service_command: 'ovs-vsctl list Open_vSwitch . | grep external_ids | awk -F ": " "{ print
$2; }'"
      cat_output: true
      config_mapping:
        ovn-bridge: edpm_ovn_bridge
        ovn-bridge-mappings: edpm_ovn_bridge_mappings
        ovn-encap-type: edpm_ovn_encap_type
        ovn-match-northd-version: ovn_match_northd_version
        ovn-monitor-all: ovn_monitor_all
        ovn-ofctrl-wait-before-clear: edpm_ovn_ofctrl_wait_before_clear
        ovn-remote-probe-interval: edpm_ovn_remote_probe_interval
```

Then you can use the following command to compare the values:

```
os-diff diff ovs_external_ids.json edpm.crd --crd --service ovs_external_ids
```

For example, to check the **/etc/yum.conf** on every host, you must put the following statement in the **config.yaml** file. The following example uses a file called **yum_config**:

```
services:
  yum_config:
    hosts:
      - undercloud
      - controller_1
      - compute_1
      - compute_2
    service_command: "cat /etc/yum.conf"
    cat_output: true
    path:
      - yum.conf
```

2. Pull the configuration:
   This command will pull all the configuration files that are described in the **/etc/os-diff/config.yaml** file. Os-diff can update this file automatically according to your running environment with the command **--update** or **--update-only**. This option sets the podman information into the **config.yaml** for all running containers. It can be useful later, when all the Red Hat OpenStack Platform services are turned off.

Note that when the **config.yaml** file is populated automatically you must provide the configuration paths manually for each service.

```
# will only update the /etc/os-diff/config.yaml
os-diff pull --update-only
```

```
# will update the /etc/os-diff/config.yaml and pull configuration
os-diff pull --update
```

```
# will update the /etc/os-diff/config.yaml and pull configuration
os-diff pull
```

The configuration will be pulled and stored by default:

```
/tmp/tripleo/
```

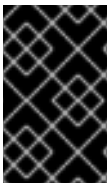
Verification

- You should have into your local path a directory per services such as:

```
▾ tmp/
  ▾ tripleo/
    ▾ glance/
    ▾ keystone/
```

4.16. ROLLING BACK THE CONTROL PLANE ADOPTION

If you encountered a problem during the adoption of the Red Hat OpenStack Platform (RHOSP) control plane services that prevents you from completing the adoption procedure, you can roll back the control plane adoption.



IMPORTANT

The roll back operation is only possible during the control plane parts of the adoption procedure. If you altered the data plane nodes in any way during the procedure, the roll back is not possible.

During the control plane adoption, services on the source cloud's control plane are stopped but not removed. The databases on the source control plane are not edited by the adoption procedure. The destination control plane received a copy of the original control plane databases. The roll back procedure assumes that the data plane has not yet been touched by the adoption procedure and it is still connected to the source control plane.

The rollback procedure consists of the following steps:

- Restoring the functionality of the source control plane.
- Removing the partially or fully deployed destination control plane.

Procedure

1. To restore the source cloud to a working state, start the RHOSP control plane services that you previously stopped during the adoption procedure:

```
ServicesToStart=("tripleo_horizon.service"
```

```
                "tripleo_keystone.service"
                "tripleo_barbican_api.service"
                "tripleo_barbican_worker.service"
                "tripleo_barbican_keystone_listener.service"
                "tripleo_cinder_api.service"
                "tripleo_cinder_api_cron.service"
                "tripleo_cinder_scheduler.service"
                "tripleo_cinder_volume.service"
                "tripleo_cinder_backup.service"
                "tripleo_glance_api.service"
                "tripleo_manila_api.service"
                "tripleo_manila_api_cron.service"
                "tripleo_manila_scheduler.service"
                "tripleo_neutron_api.service"
                "tripleo_placement_api.service"
                "tripleo_nova_api_cron.service"
                "tripleo_nova_api.service"
                "tripleo_nova_conductor.service"
                "tripleo_nova_metadata.service"
                "tripleo_nova_scheduler.service"
                "tripleo_nova_vnc_proxy.service"
                "tripleo_aodh_api.service"
                "tripleo_aodh_api_cron.service"
                "tripleo_aodh_evaluator.service"
                "tripleo_aodh_listener.service"
                "tripleo_aodh_notifier.service"
                "tripleo_ceilometer_agent_central.service"
                "tripleo_ceilometer_agent_compute.service"
                "tripleo_ceilometer_agent_ipmi.service"
                "tripleo_ceilometer_agent_notification.service"
                "tripleo_ovn_cluster_north_db_server.service"
                "tripleo_ovn_cluster_south_db_server.service"
                "tripleo_ovn_cluster_northd.service")

PacemakerResourcesToStart=("galera-bundle"
                           "haproxy-bundle"
                           "rabbitmq-bundle"
                           "openstack-cinder-volume"
                           "openstack-cinder-backup"
                           "openstack-manila-share")

echo "Starting systemd OpenStack services"
for service in ${ServicesToStart[*]}; do
    for i in {1..3}; do
        SSH_CMD=CONTROLLER${i}_SSH
        if [ ! -z "${!SSH_CMD}" ]; then
            if ${!SSH_CMD} sudo systemctl is-enabled $service &> /dev/null; then
                echo "Starting the $service in controller $i"
                ${!SSH_CMD} sudo systemctl start $service
            fi
        fi
    done
done

echo "Checking systemd OpenStack services"
for service in ${ServicesToStart[*]}; do
```

```
    for i in {1..3}; do
        SSH_CMD=CONTROLLER${i}_SSH
        if [ ! -z "${!SSH_CMD}" ]; then
            if ${!SSH_CMD} sudo systemctl is-enabled $service &> /dev/null; then
                if ! ${!SSH_CMD} systemctl show $service | grep ActiveState=active >/dev/null;
then
                    echo "ERROR: Service $service is not running on controller $i"
                else
                    echo "OK: Service $service is running in controller $i"
                fi
            fi
        fi
    done
done

echo "Starting pacemaker OpenStack services"
for i in {1..3}; do
    SSH_CMD=CONTROLLER${i}_SSH
    if [ ! -z "${!SSH_CMD}" ]; then
        echo "Using controller $i to run pacemaker commands"
        for resource in ${PacemakerResourcesToStart[*]}; do
            if ${!SSH_CMD} sudo pcs resource config $resource &>/dev/null; then
                echo "Starting $resource"
                ${!SSH_CMD} sudo pcs resource enable $resource
            else
                echo "Service $resource not present"
            fi
        done
        break
    fi
done

echo "Checking pacemaker OpenStack services"
for i in {1..3}; do
    SSH_CMD=CONTROLLER${i}_SSH
    if [ ! -z "${!SSH_CMD}" ]; then
        echo "Using controller $i to run pacemaker commands"
        for resource in ${PacemakerResourcesToStop[*]}; do
            if ${!SSH_CMD} sudo pcs resource config $resource &>/dev/null; then
                if ${!SSH_CMD} sudo pcs resource status $resource | grep Started >/dev/null; then
                    echo "OK: Service $resource is started"
                else
                    echo "ERROR: Service $resource is stopped"
                fi
            fi
        done
        break
    fi
done
```

2. If the Ceph NFS service is running on the deployment as a Shared File Systems service (manila) backend, you must restore the pacemaker ordering and colocation constraints involving the "openstack-manila-share" service:

```
sudo pcs constraint order start ceph-nfs then openstack-manila-share kind=Optional
id=order-ceph-nfs-openstack-manila-share-Optional
```

```
sudo pcs constraint colocation add openstack-manila-share with ceph-nfs score=INFINITY
id=colocation-openstack-manila-share-ceph-nfs-INFINITY
```

3. Verify that the source cloud is operational again, e.g. by running **openstack** CLI commands or using the Dashboard service (horizon).

4. Remove the partially or fully deployed control plane so that another adoption attempt can be made later:

```
oc delete --ignore-not-found=true --wait=false openstackcontrolplane/openstack
oc patch openstackcontrolplane openstack --type=merge --patch '
metadata:
  finalizers: []
' || true

while oc get pod | grep rabbitmq-server-0; do
    sleep 2
done
while oc get pod | grep openstack-galera-0; do
    sleep 2
done

oc delete --ignore-not-found=true --wait=false pod mariadb-copy-data
oc delete --ignore-not-found=true --wait=false pvc mariadb-data
oc delete --ignore-not-found=true --wait=false pod ovn-copy-data
oc delete --ignore-not-found=true secret osp-secret
```

NOTE

Since restoring the source control plane services, their internal state may have changed. Before retrying the adoption procedure, it is important to verify that the control plane resources have been removed and there are no leftovers which could affect the following adoption procedure attempt. Notably, the previously created copies of the database contents must not be used in another adoption attempt, and new copies must be made according to the adoption procedure documentation.

CHAPTER 5. ADOPTING THE DATA PLANE

Adopting the Red Hat OpenStack Services on OpenShift (RHOSO) data plane involves the following steps:

1. Stopping any remaining services on the Red Hat OpenStack Platform (RHOSP) control plane.
2. Deploying the required custom resources.
3. If applicable, performing a fast-forward upgrade on Compute services from RHOSP 17.1 to RHOSO 18.0.

WARNING

After the RHOSO control plane is managing the newly deployed data plane, you must not re-enable services on the RHOSP 17.1 control plane and data plane. Re-enabling services causes workloads to be managed by two control planes or two data planes, resulting in data corruption, loss of control of existing workloads, inability to start new workloads, or other issues.

5.1. STOPPING INFRASTRUCTURE MANAGEMENT AND COMPUTE SERVICES

The source cloud's control plane can be decomissioned, which is taking down only cloud controllers, database and messaging nodes. Nodes that must remain functional are those running the Compute, storage, or networker roles (in terms of composable roles covered by director Heat Templates).

Prerequisites

- Define the following shell variables. The values that are used are examples and refer to a single node standalone director deployment. Replace these example values with values that are correct for your environment:

```
EDPM_PRIVATEKEY_PATH="<path to SSH key>"
declare -A computes
computes=(
  ["standalone.localdomain"]="192.168.122.100"
  # ...
)
```

  - Replace **["standalone.localdomain"]="192.168.122.100"** with the name of the Compute node and its IP address.
  - These ssh variables with the ssh commands are used instead of ansible to create instructions that are independent of where they are running. But ansible commands could be used to achieve the same result if you are in the right host, for example to stop a service:

```
. stackrc
ansible -i $(which tripleo-ansible-inventory) Compute -m shell -a "sudo systemctl stop
tripleo_virtqemud.service" -b
```

-

Procedure

- Run the following script to remove the conflicting repositories and packages (in case of a devsetup that uses Standalone director) from all Compute hosts. That is required to install libvirt packages, when these hosts become adopted as data plane nodes, where modular libvirt daemons are no longer running in podman containers:

```
PacemakerResourcesToStop=(
                "galera-bundle"
                "haproxy-bundle"
                "rabbitmq-bundle")

echo "Stopping pacemaker services"
for i in {1..3}; do
    SSH_CMD=CONTROLLER${i}_SSH
    if [ ! -z "${!SSH_CMD}" ]; then
        echo "Using controller $i to run pacemaker commands"
        for resource in ${PacemakerResourcesToStop[*]}; do
            if ${!SSH_CMD} sudo pcs resource config $resource; then
                ${!SSH_CMD} sudo pcs resource disable $resource
            fi
        done
        break
    fi
done
```

5.2. ADOPTING COMPUTE SERVICES TO THE RHOSO DATA PLANE

Prerequisites

- Remaining source cloud Stopping infrastructure management and Compute services on Compute hosts.
- Ceph backend for Nova/Libvirt is configured Configuring a Ceph backend .
- Make sure the IPAM is configured

```
oc apply -f - <<EOF
apiVersion: network.openstack.org/v1beta1
kind: NetConfig
metadata:
  name: netconfig
spec:
  networks:
  - name: ctlplane
    dnsDomain: ctlplane.example.com
    subnets:
    - name: subnet1
      allocationRanges:
      - 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
      cidr: 172.17.0.0/24
      vlan: 20
  - name: External
    dnsDomain: external.example.com
    subnets:
    - name: subnet1
      allocationRanges:
      - end: 10.0.0.250
        start: 10.0.0.100
      cidr: 10.0.0.0/24
      gateway: 10.0.0.1
  - name: storage
    dnsDomain: storage.example.com
    subnets:
    - name: subnet1
      allocationRanges:
      - end: 172.18.0.250
        start: 172.18.0.100
      cidr: 172.18.0.0/24
      vlan: 21
  - name: storagemgmt
    dnsDomain: storagemgmt.example.com
    subnets:
    - name: subnet1
      allocationRanges:
      - end: 172.20.0.250
        start: 172.20.0.100
      cidr: 172.20.0.0/24
      vlan: 23
  - name: tenant
    dnsDomain: tenant.example.com
    subnets:
    - name: subnet1
      allocationRanges:
      - end: 172.19.0.250
        start: 172.19.0.100
      cidr: 172.19.0.0/24
      vlan: 22
EOF
```

- When **neutron-sriov-nic-agent** is running on the existing Compute nodes, check the physical device mappings and ensure that they match the values that are defined in the **OpenStackDataPlaneNodeSet** custom resource (CR). For more information, see Pulling the configuration from a director deployment.

- Define the shell variables necessary to run the script that runs the fast-forward upgrade. Omit setting **CEPH_FSID**, if the local storage backend is going to be configured by Nova for Libvirt.

The storage backend cannot be changed during adoption, and must match the one used on the source cloud:

```
PODIFIED_DB_ROOT_PASSWORD=$(oc get -o json secret/osp-secret | jq -r
.data.DbRootPassword | base64 -d)
CEPH_FSID=$(oc get secret ceph-conf-files -o json | jq -r '.data."ceph.conf"' | base64 -d | grep fsid |
sed -e 's/fsid = //'

alias openstack="oc exec -t openstackclient -- openstack"
declare -A computes
export computes=(
  ["standalone.localdomain"]="192.168.122.100"
  # ...
)
```

- Replace **["standalone.localdomain"]="192.168.122.100"** with the name of the Compute node and its IP address.

Procedure

1. Create a ssh authentication secret for the data plane nodes:

```
oc apply -f - <<EOF
apiVersion: v1
kind: Secret
metadata:
    name: dataplane-adoption-secret
    namespace: openstack
data:
    ssh-privatekey: |
$(cat <path to SSH key> | base64 | sed 's/^/        /')
EOF
```

2. Generate an ssh key-pair **nova-migration-ssh-key** secret:

```
cd "$(mktemp -d)"
ssh-keygen -f ./id -t ecdsa-sha2-nistp521 -N ''
oc get secret nova-migration-ssh-key || oc create secret generic nova-migration-ssh-key \
  -n openstack \
  --from-file=ssh-privatekey=id \
  --from-file=ssh-publickey=id.pub \
  --type kubernetes.io/ssh-auth
rm -f id*
cd -
```

3. Create a **nova-compute-extra-config** service (with local storage backend for libivrt):

4. If TLS Everywhere is enabled, append the following to the OpenStackDataPlaneService spec:

```
  tlsCert:
    contents:
      - dnsnames
      - ips
    networks:
      - ctlplane
```

```
    issuer: osp-rootca-issuer-internal
  caCerts: combined-ca-bundle
  edpmServiceType: nova
```

```
oc apply -f - <<EOF
apiVersion: v1
kind: ConfigMap
metadata:
  name: nova-extra-config
  namespace: openstack
data:
  19-nova-compute-cell1-workarounds.conf: |
    [workarounds]
    disable_compute_service_check_for_ffu=true
EOF
```

The secret **nova-cell<X>-compute-config** is auto-generated for each **cell<X>**. You must specify **nova-cell<X>-compute-config** and **nova-migration-ssh-key** for each custom **OpenStackDataPlaneService** related to the Compute service.

That service removes pre-FFU workarounds and configures Compute services for local storage backend.

- Or, create a **nova-compute-extra-config** service (with Ceph backend for libvirt):

```
oc apply -f - <<EOF
apiVersion: v1
kind: ConfigMap
metadata:
  name: nova-extra-config
  namespace: openstack
data:
  19-nova-compute-cell1-workarounds.conf: |
    [workarounds]
    disable_compute_service_check_for_ffu=true
  03-ceph-nova.conf: |
    [libvirt]
    images_type=rbd
    images_rbd_pool=vms
    images_rbd_ceph_conf=/etc/ceph/ceph.conf
    images_rbd_glance_store_name=default_backend
    images_rbd_glance_copy_poll_interval=15
    images_rbd_glance_copy_timeout=600
    rbd_user=openstack
    rbd_secret_uuid=$CEPH_FSID
EOF
```

That service removes pre-FFU workarounds and configures Compute services for Ceph storage backend. Provided above resources should contain a cell-specific configurations. For multi-cell, config maps and Red Hat OpenStack Platform data plane services should be named like **nova-custom-ceph-cellX** and **nova-compute-extraconfig-cellX**.

1. Create a secret for the subscription manager and a secret for the Red Hat registry:

```
oc apply -f - <<EOF
```

```
apiVersion: v1
kind: Secret
metadata:
  name: subscription-manager
data:
  username: <base64 encoded subscription-manager username>
  password: <base64 encoded subscription-manager password>
---
apiVersion: v1
kind: Secret
metadata:
  name: redhat-registry
data:
  username: <base64 encoded registry username>
  password: <base64 encoded registry password>
EOF
```

2. Deploy the **OpenStackDataPlaneNodeSet** CR:

3. If TLS Everywhere is enabled, change spec:tlsEnabled to true

4. If using a custom DNS Domain, modify the spec:nodes:[NODE NAME]:hostName to use fqdn for the node

```
oc apply -f - <<EOF
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneNodeSet
metadata:
  name: openstack
spec:
  tlsEnabled: false
  networkAttachments:
      - ctlplane
  preProvisioned: true
  services:
    - bootstrap
    - download-cache
    - configure-network
    - validate-network
    - install-os
    - configure-os
    - ssh-known-hosts
    - run-os
    - install-certs
    - libvirt
    - nova
    - ovn
    - neutron-metadata
  env:
    - name: ANSIBLE_CALLBACKS_ENABLED
      value: "profile_tasks"
    - name: ANSIBLE_FORCE_COLOR
      value: "True"
  nodes:
    standalone:
      hostName: standalone
```

```
    ansible:
      ansibleHost: ${computes[standalone.localdomain]}
    networks:
    - defaultRoute: true
      fixedIP: ${computes[standalone.localdomain]}
      name: ctlplane
      subnetName: subnet1
    - name: internalapi
      subnetName: subnet1
    - name: storage
      subnetName: subnet1
    - name: tenant
      subnetName: subnet1
nodeTemplate:
  ansibleSSHPrivateKeySecret: dataplane-adoption-secret
  ansible:
    ansibleUser: root
    ansibleVarsFrom:
    - prefix: subscription_manager_
      secretRef:
        name: subscription-manager
    - prefix: registry_
      secretRef:
        name: redhat-registry
    ansibleVars:
      edpm_bootstrap_release_version_package: []
      service_net_map:
        nova_api_network: internalapi
        nova_libvirt_network: internalapi

      # edpm_network_config
      # Default nic config template for a EDPM compute node
      # These vars are edpm_network_config role vars
      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 %}

      edpm_network_config_hide_sensitive_logs: false
      #
      # These vars are for the network config templates themselves and are
      # considered EDPM network defaults.
      neutron_physical_bridge_name: br-ctlplane
      neutron_public_interface_name: eth0

      # edpm_nodes_validation
      edpm_nodes_validation_validate_controllers_icmp: false
      edpm_nodes_validation_validate_gateway_icmp: false

      # edpm ovn-controller configuration
      edpm_ovn_bridge_mappings: <bridge_mappings>
      edpm_ovn_bridge: br-int
      edpm_ovn_encap_type: geneve
      ovn_match_northd_version: false
      ovn_monitor_all: true
      edpm_ovn_remote_probe_interval: 60000
      edpm_ovn_ofctrl_wait_before_clear: 8000

      timesync_ntp_servers:
      - hostname: clock.redhat.com
      - hostname: clock2.redhat.com

      edpm_bootstrap_command: |
        subscription-manager register --username {{ subscription_manager_username }} --
password {{ subscription_manager_password }}
        subscription-manager release --set=9.2
        subscription-manager repos --disable=*
        subscription-manager repos --enable=rhel-9-for-x86_64-baseos-eus-rpms --
enable=rhel-9-for-x86_64-appstream-eus-rpms --enable=rhel-9-for-x86_64-
highavailability-eus-rpms --enable=openstack-17.1-for-rhel-9-x86_64-rpms --enable=fast-
datapath-for-rhel-9-x86_64-rpms --enable=openstack-dev-preview-for-rhel-9-x86_64-
rpms
        # FIXME: perform dnf upgrade for other packages in EDPM ansible
        # here we only ensuring that decontainerized libvirt can start
        dnf -y upgrade openstack-selinux
        rm -f /run/virtlogd.pid
        podman login -u {{ registry_username }} -p {{ registry_password }} registry.redhat.io

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

```
        # SELinux module
        edpm_selinux_mode: enforcing

        # Do not attempt OVS major upgrades here
        edpm_ovs_packages:
        - openvswitch3.1
EOF
```

- Prepare adopted EDPM workloads to use Ceph backend for Block Storage service (cinder), if configured so

```
oc patch osdpns/openstack --type=merge --patch '
spec:
  services:
    - repo-setup
    - download-cache
    - bootstrap
    - configure-network
    - validate-network
    - install-os
    - configure-os
    - run-os
    - install-certs
    - ceph-client
    - libvirt
    - nova
    - ovn
    - neutron-metadata
  nodeTemplate:
    extraMounts:
    - extraVolType: Ceph
      volumes:
      - name: ceph
        secret:
          secretName: ceph-conf-files
      mounts:
      - name: ceph
        mountPath: "/etc/ceph"
        readOnly: true
'
```

- Replace **<bridge_mappings>** with the value of the bridge mappings in your configuration, for example, **"datacentre:br-ctlplane"**.

1. Ensure that the **ovn-controller** settings that are configured in the **OpenStackDataPlaneNodeSet** CR are the same as were set in the Compute nodes before adoption. This configuration is stored in the **external_ids`** column in the **Open_vSwitch** table in the Open vSwitch database:

```
ovs-vsctl list Open .
...
external_ids        : {hostname=standalone.localdomain, ovn-bridge=br-int, ovn-bridge-mappings=<bridge_mappings>, ovn-chassis-mac-mappings="datacentre:1e:0a:bb:e6:7c:ad", ovn-encap-ip="172.19.0.100", ovn-encap-tos="0", ovn-encap-type=geneve, ovn-match-northd-version=False, ovn-monitor-all=True, ovn-ofctrl-wait-before-clear="8000", ovn-openflow-probe-interval="60", ovn-
```

```
remote="tcp:ovsdbserver-sb.openstack.svc:6642", ovn-remote-probe-interval="60000",
rundir="/var/run/openvswitch", system-id="2eec68e6-aa21-4c95-a868-31aeafc11736"}
...
```

Note that you should retain the original **OpenStackDataPlaneNodeSet** services composition, except the inserted **ceph-client** service.

+ * Replace **<bridge_mappings>** with the value of the bridge mappings in your configuration, for example, **"datacentre:br-ctlplane"**.

1. Optional: Enable **neutron-sriov-nic-agent** in the **OpenStackDataPlaneNodeSet** CR:

```
oc patch openstackdataplanenodeset openstack --type='json' --patch='[
 {
  "op": "add",
  "path": "/spec/services/-",
  "value": "neutron-sriov"
 }, {
  "op": "add",
  "path":
"/spec/nodeTemplate/ansible/ansibleVars/edpm_neutron_sriov_agent_SRIOV_NIC_physical_device_mappings",
  "value": "dummy_sriov_net:dummy-dev"
 }, {
  "op": "add",
  "path":
"/spec/nodeTemplate/ansible/ansibleVars/edpm_neutron_sriov_agent_SRIOV_NIC_resource_provider_bandwidths",
  "value": "dummy-dev:40000000:40000000"
 }, {
  "op": "add",
  "path":
"/spec/nodeTemplate/ansible/ansibleVars/edpm_neutron_sriov_agent_SRIOV_NIC_resource_provider_hypervisors",
  "value": "dummy-dev:standalone.localdomain"
 }
]'
```

2. Optional: Enable **neutron-dhcp** in the **OpenStackDataPlaneNodeSet** CR:

```
oc patch openstackdataplanenodeset openstack --type='json' --patch='[
 {
  "op": "add",
  "path": "/spec/services/-",
  "value": "neutron-dhcp"
 }]'
```

3. Run pre-adoption validation:

   a. Create the validation service:

```
oc apply -f - <<EOF
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneService
metadata:
```

```
  name: pre-adoption-validation
spec:
  playbook: osp.edpm.pre_adoption_validation
EOF
```

b. Create a **OpenStackDataPlaneDeployment** CR that runs the validation only:

```
oc apply -f - <<EOF
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneDeployment
metadata:
  name: openstack-pre-adoption
spec:
  nodeSets:
  - openstack
  servicesOverride:
  - pre-adoption-validation
EOF
```

Wait for the validation to finish.

c. Confirm that all the Ansible EE pods reach a **Completed** status:

```
# watching the pods
watch oc get pod -l app=openstackansibleee
```

```
# following the ansible logs with:
oc logs -l app=openstackansibleee -f --max-log-requests 20
```

d. Wait for the deployment to reach the **Ready** status:

```
oc wait --for condition=Ready openstackdataplanedeployment/openstack-pre-adoption --
timeout=10m
```

4. If any openstack-pre-adoption validations fail, you must first determine which ones were unsuccessful based on the ansible logs and then follow the instructions below for each case:

- if the hostname validation failed then check that the hostname of the EDPM node is correctly listed in the **OpenStackDataPlaneNodeSet**
- if the kernel argument check failed then make sure that the **OpenStackDataPlaneNodeSet** has the same kernel argument configuration in **edpm_kernel_args** and **edpm_kernel_hugepages** variables than what is used in the 17 node.
- if the tuned profile check failed then make sure that the **edpm_tuned_profile** variable in the **OpenStackDataPlaneNodeSet** is configured to use the same profile as set on the (source) OSP 17 node.

5. Remove leftover director services

a. Create cleanup data plane service

```
---
oc apply -f - <<EOF
apiVersion: dataplane.openstack.org/v1beta1
```

```
kind: OpenStackDataPlaneService
metadata:
  name: tripleo-cleanup
spec:
  playbook: osp.edpm.tripleo_cleanup
EOF
---
```

b. Create OpenStackDataPlaneDeployment to run cleanup

```
---
oc apply -f - <<EOF
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneDeployment
metadata:
  name: tripleo-cleanup
spec:
  nodeSets:
  - openstack
  servicesOverride:
  - tripleo-cleanup
EOF
---
```

c. Wait for the removal to finish.

6. Deploy the **OpenStackDataPlaneDeployment** CR:

```
oc apply -f - <<EOF
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneDeployment
metadata:
  name: openstack
spec:
  nodeSets:
  - openstack
EOF
```

Verification

1. Confirm that all the Ansible EE pods reach a **Completed** status:

```
# watching the pods
watch oc get pod -l app=openstackansibleee
```

```
# following the ansible logs with:
oc logs -l app=openstackansibleee -f --max-log-requests 20
```

2. Wait for the data plane node set to reach the **Ready** status:

```
oc wait --for condition=Ready osdpns/openstack --timeout=30m
```

3. Verify that Networking service (neutron) agents are alive:

```
oc exec openstackclient -- openstack network agent list
+--------------------------------------+----------------------------+-----------------------+---------------
----+-------+-------+---------------------------+
| ID                                   | Agent Type                 | Host                  | Availability Zone | Alive |
State | Binary                    |
+--------------------------------------+----------------------------+-----------------------+---------------
----+-------+-------+---------------------------+
| 174fc099-5cc9-4348-b8fc-59ed44fcfb0e | DHCP agent                 | standalone.localdomain
| nova              | :-)   | UP    | neutron-dhcp-agent        |
| 10482583-2130-5b0d-958f-3430da21b929 | OVN Metadata agent         |
standalone.localdomain |                   | :-)   | UP    | neutron-ovn-metadata-agent |
| a4f1b584-16f1-4937-b2b0-28102a3f6eaa | OVN Controller agent       |
standalone.localdomain |                   | :-)   | UP    | ovn-controller            |
+--------------------------------------+----------------------------+-----------------------+---------------
----+-------+-------+---------------------------+
```

5.3. PERFORMING A FAST-FORWARD UPGRADE ON COMPUTE SERVICES

Compute services rolling upgrade cannot be done during adoption, there is in a lock-step with Compute control plane services, because those are managed independently by data plane ansible and Kubernetes Operators. Compute service operator and Dataplane Operator ensure upgrading is done independently of each other, by configuring **[upgrade_levels]compute=auto** for Compute services. Compute control plane services apply the change right after custom resource (CR) is patched. Compute data plane services will catch up the same config change with ansible deployment later on.

Procedure

1. Wait for cell1 Compute data plane services version updated (it may take some time):

```
oc exec openstack-cell1-galera-0 -c galera -- mysql -rs -uroot -
p$PODIFIED_DB_ROOT_PASSWORD \
    -e "select a.version from nova_cell1.services a join nova_cell1.services b where
a.version!=b.version and a.binary='nova-compute';"
```

   The above query should return an empty result as a completion criterion.

2. Remove pre-fast-forward upgrade workarounds for Compute control plane services:

```
oc patch openstackcontrolplane openstack -n openstack --type=merge --patch '
spec:
  nova:
    template:
      cellTemplates:
        cell0:
          conductorServiceTemplate:
            customServiceConfig: |
              [workarounds]
              disable_compute_service_check_for_ffu=false
        cell1:
          metadataServiceTemplate:
            customServiceConfig: |
              [workarounds]
              disable_compute_service_check_for_ffu=false
```

```
        conductorServiceTemplate:
          customServiceConfig: |
            [workarounds]
            disable_compute_service_check_for_ffu=false
      apiServiceTemplate:
        customServiceConfig: |
          [workarounds]
          disable_compute_service_check_for_ffu=false
      metadataServiceTemplate:
        customServiceConfig: |
          [workarounds]
          disable_compute_service_check_for_ffu=false
      schedulerServiceTemplate:
        customServiceConfig: |
          [workarounds]
          disable_compute_service_check_for_ffu=false
'
```

3. Wait for Compute control plane services' CRs to be ready:

```
oc wait --for condition=Ready --timeout=300s Nova/nova
```

4. Remove pre-fast-forward upgrade workarounds for Compute data plane services:

```
oc apply -f - <<EOF
apiVersion: v1
kind: ConfigMap
metadata:
  name: nova-extra-config
  namespace: openstack
data:
  20-nova-compute-cell1-workarounds.conf: |
    [workarounds]
    disable_compute_service_check_for_ffu=false
---
apiVersion: dataplane.openstack.org/v1beta1
kind: OpenStackDataPlaneDeployment
metadata:
  name: openstack-nova-compute-ffu
  namespace: openstack
spec:
  nodeSets:
    - openstack
  servicesOverride:
    - nova
EOF
```

5. Wait for Compute data plane service to be ready:

```
oc wait --for condition=Ready openstackdataplanedeployment/openstack-nova-compute-ffu --timeout=5m
```

6. Run Compute database online migrations to complete the fast-forward upgrade:

```
oc exec -it nova-cell0-conductor-0 -- nova-manage db online_data_migrations
oc exec -it nova-cell1-conductor-0 -- nova-manage db online_data_migrations
```

7. Discover Compute hosts in the cell:

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

8. Verify if Compute services can stop the existing test VM instance:

```
${BASH_ALIASES[openstack]} server list | grep -qF '| test | ACTIVE |' &&
${BASH_ALIASES[openstack]} server stop test || echo PASS
${BASH_ALIASES[openstack]} server list | grep -qF '| test | SHUTOFF |' || echo FAIL
${BASH_ALIASES[openstack]} server --os-compute-api-version 2.48 show --diagnostics test
2>&1 || echo PASS
```

9. Verify if Compute services can start the existing test VM instance:

```
${BASH_ALIASES[openstack]} server list | grep -qF '| test | SHUTOFF |' &&
${BASH_ALIASES[openstack]} server start test || echo PASS
${BASH_ALIASES[openstack]} server list | grep -F '| test | ACTIVE |' && \
  ${BASH_ALIASES[openstack]} server --os-compute-api-version 2.48 show --diagnostics
test --fit-width -f json | jq -r '.state' | grep running || echo FAIL
```

NOTE

After the data plane adoption, the hosts continue to run Red Hat Enterprise Linux (RHEL) 9.2. To take advantage of RHEL 9.4, perform a minor update procedure after finishing the adoption procedure.

CHAPTER 6. MIGRATING RED HAT CEPH STORAGE RBD TO EXTERNAL RHEL NODES

For hyperconverged infrastructure (HCI) or dedicated Storage nodes that are running Red Hat Ceph Storage version 6 or later, you must migrate the daemons that are included in the Red Hat OpenStack Platform control plane into the existing external Red Hat Enterprise Linux (RHEL) nodes. The external RHEL nodes typically include the Compute nodes for an HCI environment or dedicated storage nodes.

To migrate Red Hat Ceph Storage Rados Block Device (RBD), your environment must meet the following requirements:

- Red Hat Ceph Storage is running version 6 or later and is managed by cephadm/orchestrator.
- NFS (ganesha) is migrated from a director-based deployment to cephadm. For more information, see Creating a NFS Ganesha cluster .
- Both the Red Hat Ceph Storage public and cluster networks are propagated, with director, to the target nodes.
- Ceph Monitors need to keep their IPs to avoid cold migration.

6.1. MIGRATING CEPH MONITOR AND CEPH MANAGER DAEMONS TO RED HAT CEPH STORAGE NODES

Migrate your Ceph Monitor daemons, Ceph Manager daemons, and object storage daemons (OSDs) from your Red Hat OpenStack Platform Controller nodes to existing Red Hat Ceph Storage nodes. During the migration, ensure that you can do the following actions:

- Keep the mon IP addresses by moving them to the Red Hat Ceph Storage nodes.
- Drain the existing Controller nodes and shut them down.
- Deploy additional monitors to the existing nodes, and promote them as _admin nodes that administrators can use to manage the Red Hat Ceph Storage cluster and perform day 2 operations against it.
- Keep the Red Hat Ceph Storage cluster operational during the migration.

The following procedure shows an example migration from a Controller node (**oc0-controller-1**) and a Red Hat Ceph Storage node (**oc0-ceph-0**). Use the names of the nodes in your environment.

Prerequisites

- Configure the Storage nodes to have both storage and storage_mgmt network to ensure that you can use both Red Hat Ceph Storage public and cluster networks. This step requires you to interact with director. From Red Hat OpenStack Platform 17.1 and later you do not have to run a stack update. However, there are commands that you must perform to run **os-net-config** on the bare metal node and configure additional networks.

  a. Ensure that the network is defined in the **metalsmith.yaml** for the CephStorageNodes:

```
- name: CephStorage
  count: 2
  instances:
    - hostname: oc0-ceph-0
```

```
      name: oc0-ceph-0
    - hostname: oc0-ceph-1
      name: oc0-ceph-1
  defaults:
    networks:
      - network: ctlplane
        vif: true
      - network: storage_cloud_0
          subnet: storage_cloud_0_subnet
      - network: storage_mgmt_cloud_0
          subnet: storage_mgmt_cloud_0_subnet
    network_config:
      template: templates/single_nic_vlans/single_nic_vlans_storage.j2
```

b. Run the following command:

```
openstack overcloud node provision \
  -o overcloud-baremetal-deployed-0.yaml --stack overcloud-0 \
  --network-config -y --concurrency 2 /home/stack/metalsmith-0.yam
```

c. Verify that the storage network is running on the node:

```
(undercloud) [CentOS-9 - stack@undercloud ~]$ ssh heat-admin@192.168.24.14 ip -o -
4 a
Warning: Permanently added '192.168.24.14' (ED25519) to the list of known hosts.
1: lo    inet 127.0.0.1/8 scope host lo\       valid_lft forever preferred_lft forever
5: br-storage    inet 192.168.24.14/24 brd 192.168.24.255 scope global br-storage\
valid_lft forever preferred_lft forever
6: vlan1    inet 192.168.24.14/24 brd 192.168.24.255 scope global vlan1\       valid_lft
forever preferred_lft forever
7: vlan11    inet 172.16.11.172/24 brd 172.16.11.255 scope global vlan11\       valid_lft
forever preferred_lft forever
8: vlan12    inet 172.16.12.46/24 brd 172.16.12.255 scope global vlan12\       valid_lft
forever preferred_lft forever
```

Procedure

1. To migrate mon(s) and mgr(s) on the two existing Red Hat Ceph Storage nodes, create a Red Hat Ceph Storage spec based on the default roles with the mon/mgr on the controller nodes.

```
openstack overcloud ceph spec -o ceph_spec.yaml -y  \
  --stack overcloud-0     overcloud-baremetal-deployed-0.yaml
```

2. Deploy the Red Hat Ceph Storage cluster:

```
openstack overcloud ceph deploy overcloud-baremetal-deployed-0.yaml \
  --stack overcloud-0 -o deployed_ceph.yaml \
  --network-data ~/oc0-network-data.yaml \
  --ceph-spec ~/ceph_spec.yaml
```

NOTE

The **ceph_spec.yaml**, which is the OSP-generated description of the Red Hat Ceph Storage cluster, will be used, later in the process, as the basic template required by cephadm to update the status/info of the daemons.

3. Check the status of the Red Hat Ceph Storage cluster:

```
[ceph: root@oc0-controller-0 /]# ceph -s
  cluster:
    id:     f6ec3ebe-26f7-56c8-985d-eb974e8e08e3
    health: HEALTH_OK

  services:
    mon: 3 daemons, quorum oc0-controller-0,oc0-controller-1,oc0-controller-2 (age 19m)
    mgr: oc0-controller-0.xzgtvo(active, since 32m), standbys: oc0-controller-1.mtxohd, oc0-
controller-2.ahrgsk
    osd: 8 osds: 8 up (since 12m), 8 in (since 18m); 1 remapped pgs

  data:
    pools:   1 pools, 1 pgs
    objects: 0 objects, 0 B
    usage:   43 MiB used, 400 GiB / 400 GiB avail
    pgs:     1 active+clean
```

```
[ceph: root@oc0-controller-0 /]# ceph orch host ls
HOST             ADDR          LABELS         STATUS
oc0-ceph-0       192.168.24.14  osd
oc0-ceph-1       192.168.24.7   osd
oc0-controller-0  192.168.24.15  _admin mgr mon
oc0-controller-1  192.168.24.23  _admin mgr mon
oc0-controller-2  192.168.24.13  _admin mgr mon
```

4. Log in to the **controller-0** node, then

```
cephadm shell -v /home/ceph-admin/specs:/specs
```

5. Log in to the **ceph-0** node, then

```
sudo “watch podman ps”  # watch the new mon/mgr being deployed here
```

6. Optional: If mgr is active in the source node, then:

```
ceph mgr fail <mgr instance>
```

7. From the cephadm shell, remove the labels on **oc0-controller-1**:

```
  for label in mon mgr _admin; do
        ceph orch host rm label oc0-controller-1 $label;
  done
```

8. Add the missing labels to **oc0-ceph-0**:

```
[ceph: root@oc0-controller-0 /]#
> for label in mon mgr _admin; do ceph orch host label add oc0-ceph-0 $label; done
Added label mon to host oc0-ceph-0
Added label mgr to host oc0-ceph-0
Added label _admin to host oc0-ceph-0
```

9. Drain and force-remove the **oc0-controller-1** node:

```
[ceph: root@oc0-controller-0 /]# ceph orch host drain oc0-controller-1
Scheduled to remove the following daemons from host 'oc0-controller-1'
type                 id
-------------------- ---------------
mon                  oc0-controller-1
mgr                  oc0-controller-1.mtxohd
crash                oc0-controller-1
```

```
[ceph: root@oc0-controller-0 /]# ceph orch host rm oc0-controller-1 --force
Removed  host 'oc0-controller-1'

[ceph: root@oc0-controller-0 /]# ceph orch host ls
HOST             ADDR           LABELS          STATUS
oc0-ceph-0       192.168.24.14  osd
oc0-ceph-1       192.168.24.7   osd
oc0-controller-0  192.168.24.15  mgr mon _admin
oc0-controller-2  192.168.24.13  _admin mgr mon
```

10. If you have only 3 mon nodes, and the drain of the node doesn't work as expected (the containers are still there), then log in to controller-1 and force-purge the containers in the node:

```
[root@oc0-controller-1 ~]# sudo podman ps
CONTAINER ID  IMAGE                                                                                COMMAND
CREATED        STATUS            PORTS       NAMES
5c1ad36472bc
registry.redhat.io/ceph/rhceph@sha256:320c364dcc8fc8120e2a42f54eb39ecdba12401a25467
63b7bef15b02ce93bc4  -n mon.oc0-contro...  35 minutes ago  Up 35 minutes ago
ceph-f6ec3ebe-26f7-56c8-985d-eb974e8e08e3-mon-oc0-controller-1
3b14cc7bf4dd
registry.redhat.io/ceph/rhceph@sha256:320c364dcc8fc8120e2a42f54eb39ecdba12401a25467
63b7bef15b02ce93bc4  -n mgr.oc0-contro...  35 minutes ago  Up 35 minutes ago
ceph-f6ec3ebe-26f7-56c8-985d-eb974e8e08e3-mgr-oc0-controller-1-mtxohd

[root@oc0-controller-1 ~]# cephadm rm-cluster --fsid f6ec3ebe-26f7-56c8-985d-
eb974e8e08e3 --force

[root@oc0-controller-1 ~]# sudo podman ps
CONTAINER ID  IMAGE       COMMAND     CREATED     STATUS      PORTS       NAMES
```

NOTE

Cephadm rm-cluster on a node that is not part of the cluster anymore has the effect of removing all the containers and doing some cleanup on the filesystem.

11. Before shutting the oc0-controller-1 down, move the IP address (on the same network) to the oc0-ceph-0 node:

```
mon_host = [v2:172.16.11.54:3300/0,v1:172.16.11.54:6789/0]
[v2:172.16.11.121:3300/0,v1:172.16.11.121:6789/0]
[v2:172.16.11.205:3300/0,v1:172.16.11.205:6789/0]

[root@oc0-controller-1 ~]# ip -o -4 a
1: lo    inet 127.0.0.1/8 scope host lo\       valid_lft forever preferred_lft forever
5: br-ex    inet 192.168.24.23/24 brd 192.168.24.255 scope global br-ex\       valid_lft forever
preferred_lft forever
6: vlan100    inet 192.168.100.96/24 brd 192.168.100.255 scope global vlan100\       valid_lft
forever preferred_lft forever
7: vlan12    inet 172.16.12.154/24 brd 172.16.12.255 scope global vlan12\       valid_lft forever
preferred_lft forever
8: vlan11    inet 172.16.11.121/24 brd 172.16.11.255 scope global vlan11\       valid_lft forever
preferred_lft forever
9: vlan13    inet 172.16.13.178/24 brd 172.16.13.255 scope global vlan13\       valid_lft forever
preferred_lft forever
10: vlan70    inet 172.17.0.23/20 brd 172.17.15.255 scope global vlan70\       valid_lft forever
preferred_lft forever
11: vlan1    inet 192.168.24.23/24 brd 192.168.24.255 scope global vlan1\       valid_lft forever
preferred_lft forever
12: vlan14    inet 172.16.14.223/24 brd 172.16.14.255 scope global vlan14\       valid_lft
forever preferred_lft forever
```

12. On the oc0-ceph-0, add the IP address of the mon that has been deleted from **controller-0**, and verify that the IP address has been assigned and can be reached:

```
$ sudo ip a add 172.16.11.121 dev vlan11
$ ip -o -4 a
```

```
[heat-admin@oc0-ceph-0 ~]$ ip -o -4 a
1: lo    inet 127.0.0.1/8 scope host lo\       valid_lft forever preferred_lft forever
5: br-storage    inet 192.168.24.14/24 brd 192.168.24.255 scope global br-storage\
valid_lft forever preferred_lft forever
6: vlan1    inet 192.168.24.14/24 brd 192.168.24.255 scope global vlan1\       valid_lft forever
preferred_lft forever
7: vlan11    inet 172.16.11.172/24 brd 172.16.11.255 scope global vlan11\       valid_lft forever
preferred_lft forever
8: vlan12    inet 172.16.12.46/24 brd 172.16.12.255 scope global vlan12\       valid_lft forever
preferred_lft forever
[heat-admin@oc0-ceph-0 ~]$ sudo ip a add 172.16.11.121 dev vlan11
[heat-admin@oc0-ceph-0 ~]$ ip -o -4 a
1: lo    inet 127.0.0.1/8 scope host lo\       valid_lft forever preferred_lft forever
5: br-storage    inet 192.168.24.14/24 brd 192.168.24.255 scope global br-storage\
valid_lft forever preferred_lft forever
6: vlan1    inet 192.168.24.14/24 brd 192.168.24.255 scope global vlan1\       valid_lft forever
preferred_lft forever
7: vlan11    inet 172.16.11.172/24 brd 172.16.11.255 scope global vlan11\       valid_lft forever
preferred_lft forever
7: vlan11    inet 172.16.11.121/32 scope global vlan11\       valid_lft forever preferred_lft
forever
8: vlan12    inet 172.16.12.46/24 brd 172.16.12.255 scope global vlan12\       valid_lft forever
preferred_lft forever
```

13. Optional: Power off oc0-controller-1.

14. Add the new mon on oc0-ceph-0 using the old IP address:

```
[ceph: root@oc0-controller-0 /]# ceph orch daemon add mon oc0-ceph-0:172.16.11.121
Deployed mon.oc0-ceph-0 on host 'oc0-ceph-0'
```

15. Check the new container in the oc0-ceph-0 node:

```
b581dc8bbb78
registry.redhat.io/ceph/rhceph@sha256:320c364dcc8fc8120e2a42f54eb39ecdba12401a25467
63b7bef15b02ce93bc4  -n mon.oc0-ceph-0...  24 seconds ago  Up 24 seconds ago
ceph-f6ec3ebe-26f7-56c8-985d-eb974e8e08e3-mon-oc0-ceph-0
```

16. On the cephadm shell, backup the existing ceph_spec.yaml, edit the spec removing any oc0-controller-1 entry, and replacing it with oc0-ceph-0:

```
cp ceph_spec.yaml ceph_spec.yaml.bkp # backup the ceph_spec.yaml file

[ceph: root@oc0-controller-0 specs]# diff -u ceph_spec.yaml.bkp ceph_spec.yaml

--- ceph_spec.yaml.bkp  2022-07-29 15:41:34.516329643 +0000
+++ ceph_spec.yaml      2022-07-29 15:28:26.455329643 +0000
@@ -7,14 +7,6 @@
 - mgr
 service_type: host
 ---
-addr: 192.168.24.12
-hostname: oc0-controller-1
-labels:
-- _admin
-- mon
-- mgr
-service_type: host
----
 addr: 192.168.24.19
 hostname: oc0-controller-2
 labels:
@@ -38,7 +30,7 @@
 placement:
   hosts:
   - oc0-controller-0
-  - oc0-controller-1
+  - oc0-ceph-0
   - oc0-controller-2
 service_id: mon
 service_name: mon
@@ -47,8 +39,8 @@
 placement:
   hosts:
   - oc0-controller-0
-  - oc0-controller-1
   - oc0-controller-2
+  - oc0-ceph-0
```

```
service_id: mgr
service_name: mgr
service_type: mgr
```

17. Apply the resulting spec:

```
ceph orch apply -i ceph_spec.yaml

 The result of 12 is having a new mgr deployed on the oc0-ceph-0 node, and the spec
reconciled within cephadm

[ceph: root@oc0-controller-0 specs]# ceph orch ls
NAME                   PORTS  RUNNING  REFRESHED  AGE  PLACEMENT
crash                           4/4  5m ago     61m  *
mgr                             3/3  5m ago     69s  oc0-controller-0;oc0-ceph-0;oc0-controller-2
mon                             3/3  5m ago     70s  oc0-controller-0;oc0-ceph-0;oc0-controller-2
osd.default_drive_group           8  2m ago     69s  oc0-ceph-0;oc0-ceph-1

[ceph: root@oc0-controller-0 specs]# ceph -s
  cluster:
    id:     f6ec3ebe-26f7-56c8-985d-eb974e8e08e3
    health: HEALTH_WARN
            1 stray host(s) with 1 daemon(s) not managed by cephadm

  services:
    mon: 3 daemons, quorum oc0-controller-0,oc0-controller-2,oc0-ceph-0 (age 5m)
    mgr: oc0-controller-0.xzgtvo(active, since 62m), standbys: oc0-controller-2.ahrgsk, oc0-
ceph-0.hccsbb
    osd: 8 osds: 8 up (since 42m), 8 in (since 49m); 1 remapped pgs

  data:
    pools:   1 pools, 1 pgs
    objects: 0 objects, 0 B
    usage:   43 MiB used, 400 GiB / 400 GiB avail
    pgs:     1 active+clean
```

18. Fix the warning by refreshing the mgr:

```
ceph mgr fail oc0-controller-0.xzgtvo
```

At this point the Red Hat Ceph Storage cluster is clean:

```
[ceph: root@oc0-controller-0 specs]# ceph -s
  cluster:
    id:     f6ec3ebe-26f7-56c8-985d-eb974e8e08e3
    health: HEALTH_OK

  services:
    mon: 3 daemons, quorum oc0-controller-0,oc0-controller-2,oc0-ceph-0 (age 7m)
    mgr: oc0-controller-2.ahrgsk(active, since 25s), standbys: oc0-controller-0.xzgtvo, oc0-
ceph-0.hccsbb
    osd: 8 osds: 8 up (since 44m), 8 in (since 50m); 1 remapped pgs

  data:
    pools:   1 pools, 1 pgs
```

```
    objects: 0 objects, 0 B
    usage:   43 MiB used, 400 GiB / 400 GiB avail
    pgs:     1 active+clean
```

The **oc0-controller-1** is removed and powered off without leaving traces on the Red Hat Ceph Storage cluster.

19. Repeat this procedure for additional Controller nodes in your environment until you have migrated all the Ceph Mon and Ceph Manager daemons to the target nodes.

CHAPTER 7. MIGRATING RED HAT CEPH STORAGE RGW TO EXTERNAL RHEL NODES

For hyperconverged infrastructure (HCI) or dedicated Storage nodes that are running Red Hat Ceph Storage version 6 or later, you must migrate the RGW daemons that are included in the Red Hat OpenStack Platform Controller nodes into the existing external Red Hat Enterprise Linux (RHEL) nodes. The existing external RHEL nodes typically include the Compute nodes for an HCI environment or Red Hat Ceph Storage nodes.

To migrate Ceph Object Gateway (RGW), your environment must meet the following requirements:

- Red Hat Ceph Storage is running version 6 or later and is managed by cephadm/orchestrator.
- An undercloud is still available, and the nodes and networks are managed by director.

7.1. RED HAT CEPH STORAGE DAEMON CARDINALITY

Red Hat Ceph Storage 6 and later applies strict constraints in the way daemons can be colocated within the same node. For more information, see Red Hat Ceph Storage: Supported configurations . The resulting topology depends on the available hardware, as well as the amount of Red Hat Ceph Storage services present in the Controller nodes which are going to be retired. For more information about the procedure that is required to migrate the RGW component and keep an HA model using the Ceph ingress daemon, see High availability for the Ceph Object Gateway in *Object Gateway Guide*. As a general rule, the number of services that can be migrated depends on the number of available nodes in the cluster. The following diagrams cover the distribution of the Red Hat Ceph Storage daemons on the Red Hat Ceph Storage nodes where at least three nodes are required in a scenario that sees only RGW and RBD, without the Dashboard service (horizon):

```
|    |                     |             |
|----|---------------------|-------------|
| osd | mon/mgr/crash      | rgw/ingress |
| osd | mon/mgr/crash      | rgw/ingress |
| osd | mon/mgr/crash      | rgw/ingress |
```

With the Dashboard service, and without Shared File Systems service (manila) at least four nodes are required. The Dashboard service has no failover:

```
|    |                     |             |
|-----|---------------------|-------------|
| osd | mon/mgr/crash | rgw/ingress       |
| osd | mon/mgr/crash | rgw/ingress       |
| osd | mon/mgr/crash | dashboard/grafana |
| osd | rgw/ingress   | (free)            |
```

With the Dashboard service and the Shared File Systems service, 5 nodes minimum are required, and the Dashboard service has no failover:

```
|    |                     |                         |
|-----|---------------------|-------------------------|
| osd | mon/mgr/crash       | rgw/ingress             |
| osd | mon/mgr/crash       | rgw/ingress             |
| osd | mon/mgr/crash       | mds/ganesha/ingress     |
| osd | rgw/ingress         | mds/ganesha/ingress     |
| osd | mds/ganesha/ingress | dashboard/grafana       |
```

7.2. COMPLETING PREREQUISITES FOR MIGRATING RED HAT CEPH STORAGE RGW

You must complete the following prerequisites before you begin the Red Hat Ceph Storage RGW migration.

Procedure

1. Check the current status of the Red Hat Ceph Storage nodes:

```
(undercloud) [stack@undercloud-0 ~]$ metalsmith list


+------------------------+    +----------------+
| IP Addresses           |    | Hostname       |
+------------------------+    +----------------+
| ctlplane=192.168.24.25 |    | cephstorage-0  |
| ctlplane=192.168.24.10 |    | cephstorage-1  |
| ctlplane=192.168.24.32 |    | cephstorage-2  |
| ctlplane=192.168.24.28 |    | compute-0      |
| ctlplane=192.168.24.26 |    | compute-1      |
| ctlplane=192.168.24.43 |    | controller-0   |
| ctlplane=192.168.24.7  |    | controller-1   |
| ctlplane=192.168.24.41 |    | controller-2   |
+------------------------+    +----------------+
```

2. Log in to **controller-0** and check the **pacemaker** status to help you identify the information that you need before you start the RGW migration.

```
Full List of Resources:
  * ip-192.168.24.46 (ocf:heartbeat:IPaddr2):      Started controller-0
  * ip-10.0.0.103    (ocf:heartbeat:IPaddr2):      Started controller-1
  * ip-172.17.1.129  (ocf:heartbeat:IPaddr2):      Started controller-2
  * ip-172.17.3.68   (ocf:heartbeat:IPaddr2):      Started controller-0
  * ip-172.17.4.37   (ocf:heartbeat:IPaddr2):      Started controller-1
  * Container bundle set: haproxy-bundle

[undercloud-0.ctlplane.redhat.local:8787/rh-osbs/rhosp17-openstack-haproxy:pcmklatest]:
    * haproxy-bundle-podman-0   (ocf:heartbeat:podman):  Started controller-2
    * haproxy-bundle-podman-1   (ocf:heartbeat:podman):  Started controller-0
    * haproxy-bundle-podman-2   (ocf:heartbeat:podman):  Started controller-1
```

3. Use the **ip** command to identify the ranges of the storage networks.

```
[heat-admin@controller-0 ~]$ ip -o -4 a

1: lo inet 127.0.0.1/8 scope host lo\    valid_lft forever preferred_lft forever
2: enp1s0 inet 192.168.24.45/24 brd 192.168.24.255 scope global enp1s0\    valid_lft forever
preferred_lft forever
2: enp1s0 inet 192.168.24.46/32 brd 192.168.24.255 scope global enp1s0\    valid_lft forever
preferred_lft forever
7: br-ex inet 10.0.0.122/24 brd 10.0.0.255 scope global br-ex\    valid_lft forever preferred_lft
forever
```

```
8: vlan70 inet 172.17.5.22/24 brd 172.17.5.255 scope global vlan70\    valid_lft forever
preferred_lft forever
8: vlan70 inet 172.17.5.94/32 brd 172.17.5.255 scope global vlan70\    valid_lft forever
preferred_lft forever
9: vlan50 inet 172.17.2.140/24 brd 172.17.2.255 scope global vlan50\    valid_lft forever
preferred_lft forever
10: vlan30 inet 172.17.3.73/24 brd 172.17.3.255 scope global vlan30\    valid_lft forever
preferred_lft forever
10: vlan30 inet 172.17.3.68/32 brd 172.17.3.255 scope global vlan30\    valid_lft forever
preferred_lft forever
11: vlan20 inet 172.17.1.88/24 brd 172.17.1.255 scope global vlan20\    valid_lft forever
preferred_lft forever
12: vlan40 inet 172.17.4.24/24 brd 172.17.4.255 scope global vlan40\    valid_lft forever
preferred_lft forever
```

- vlan30 represents the Storage Network, where the new RGW instances should be started on the Red Hat Ceph Storage nodes.
- br-ex represents the External Network, which is where in the current environment, haproxy has the frontend Virtual IP (VIP) assigned.

4. Identify the network that you previously had in haproxy and propagate it through director to the Red Hat Ceph Storage nodes. This network is used to reserve a new VIP that is owned by Red Hat Ceph Storage and used as the entry point for the RGW service.

   a. Log into **controller-0** and check the current HAProxy configuration until you find **ceph_rgw** section:

```
$ less /var/lib/config-data/puppet-generated/haproxy/etc/haproxy/haproxy.cfg

...
...
listen ceph_rgw
  bind 10.0.0.103:8080 transparent
  bind 172.17.3.68:8080 transparent
  mode http
  balance leastconn
  http-request set-header X-Forwarded-Proto https if { ssl_fc }
  http-request set-header X-Forwarded-Proto http if !{ ssl_fc }
  http-request set-header X-Forwarded-Port %[dst_port]
  option httpchk GET /swift/healthcheck
  option httplog
  option forwardfor
  server controller-0.storage.redhat.local 172.17.3.73:8080 check fall 5 inter 2000 rise 2
  server controller-1.storage.redhat.local 172.17.3.146:8080 check fall 5 inter 2000 rise 2
  server controller-2.storage.redhat.local 172.17.3.156:8080 check fall 5 inter 2000 rise 2
```

   b. Confirm that the network is used as an HAProxy frontend:

```
[controller-0]$ ip -o -4 a

...
7: br-ex inet 10.0.0.106/24 brd 10.0.0.255 scope global br-ex\    valid_lft forever
preferred_lft forever
...
```

This example shows that **controller-0** is exposing the services by using the external network, which is not present in the Red Hat Ceph Storage nodes, and you need to propagate it through director.

5. Propagate the HAProxy frontend network to Red Hat Ceph Storage nodes.

   a. Change the NIC template used to define the **ceph-storage** network interfaces and add the new config section:

```
---
network_config:
- type: interface
  name: nic1
  use_dhcp: false
  dns_servers: {{ ctlplane_dns_nameservers }}
  addresses:
  - ip_netmask: {{ ctlplane_ip }}/{{ ctlplane_cidr }}
  routes: {{ ctlplane_host_routes }}
- type: vlan
  vlan_id: {{ storage_mgmt_vlan_id }}
  device: nic1
  addresses:
  - ip_netmask: {{ storage_mgmt_ip }}/{{ storage_mgmt_cidr }}
  routes: {{ storage_mgmt_host_routes }}
- type: interface
  name: nic2
  use_dhcp: false
  defroute: false
- type: vlan
  vlan_id: {{ storage_vlan_id }}
  device: nic2
  addresses:
  - ip_netmask: {{ storage_ip }}/{{ storage_cidr }}
  routes: {{ storage_host_routes }}
- type: ovs_bridge
  name: {{ neutron_physical_bridge_name }}
  dns_servers: {{ ctlplane_dns_nameservers }}
  domain: {{ dns_search_domains }}
  use_dhcp: false
  addresses:
  - ip_netmask: {{ external_ip }}/{{ external_cidr }}
  routes: {{ external_host_routes }}
  members:
  - type: interface
    name: nic3
    primary: true
```

   b. In addition, add the External Network to the **baremetal.yaml** file used by metalsmith:

```
- name: CephStorage
  count: 3
  hostname_format: cephstorage-%index%
  instances:
  - hostname: cephstorage-0
  name: ceph-0
  - hostname: cephstorage-1
```

```
  name: ceph-1
  - hostname: cephstorage-2
  name: ceph-2
  defaults:
  profile: ceph-storage
  network_config:
     template: /home/stack/composable_roles/network/nic-configs/ceph-storage.j2
  networks:
  - network: ctlplane
     vif: true
  - network: storage
  - network: storage_mgmt
  - network: external
```

c. Run the **overcloud node provision** command passing the **--network-config** option:

```
(undercloud) [stack@undercloud-0]$

openstack overcloud node provision
   -o overcloud-baremetal-deployed-0.yaml
   --stack overcloud
   --network-config -y
   $PWD/network/baremetal_deployment.yaml
```

d. Check the new network on the Red Hat Ceph Storage nodes:

```
[root@cephstorage-0 ~]# ip -o -4 a

1: lo inet 127.0.0.1/8 scope host lo\    valid_lft forever preferred_lft forever
2: enp1s0 inet 192.168.24.54/24 brd 192.168.24.255 scope global enp1s0\    valid_lft
forever preferred_lft forever
11: vlan40 inet 172.17.4.43/24 brd 172.17.4.255 scope global vlan40\    valid_lft forever
preferred_lft forever
12: vlan30 inet 172.17.3.23/24 brd 172.17.3.255 scope global vlan30\    valid_lft forever
preferred_lft forever
14: br-ex inet 10.0.0.133/24 brd 10.0.0.255 scope global br-ex\    valid_lft forever
preferred_lft forever
```

7.3. MIGRATING THE RED HAT CEPH STORAGE RGW BACKENDS

To match the cardinality diagram, you use cephadm labels to refer to a group of nodes where a given daemon type should be deployed. For more information about the cardinality diagram, see Red Hat Ceph Storage daemon cardinality.

Procedure

1. Add the RGW label to the Red Hat Ceph Storage nodes:

```
for i in 0 1 2; {
    ceph orch host label add cephstorage-$i rgw;
}
```

```
[ceph: root@controller-0 /]#
```

```
for i in 0 1 2; {
    ceph orch host label add cephstorage-$i rgw;
}

Added label rgw to host cephstorage-0
Added label rgw to host cephstorage-1
Added label rgw to host cephstorage-2

[ceph: root@controller-0 /]# ceph orch host ls

HOST        ADDR        LABELS       STATUS
cephstorage-0  192.168.24.54  osd rgw
cephstorage-1  192.168.24.44  osd rgw
cephstorage-2  192.168.24.30  osd rgw
controller-0   192.168.24.45  _admin mon mgr
controller-1   192.168.24.11  _admin mon mgr
controller-2   192.168.24.38  _admin mon mgr

6 hosts in cluster
```

2. During the overcloud deployment, RGW is applied at step 2 (external_deployment_steps), and a cephadm compatible spec is generated in **/home/ceph-admin/specs/rgw** from director. Find the RGW spec:

```
[root@controller-0 heat-admin]# cat rgw

networks:
- 172.17.3.0/24
placement:
  hosts:
  - controller-0
  - controller-1
  - controller-2
service_id: rgw
service_name: rgw.rgw
service_type: rgw
spec:
  rgw_frontend_port: 8080
  rgw_realm: default
  rgw_zone: default
```

3. In the **placement** section, replace the following values:

   - Replace the controller nodes with the **label: rgw** label.
   - Change the ` rgw_frontend_port` value to **8090** to avoid conflicts with the Ceph ingress daemon.

```
---
networks:
- 172.17.3.0/24
placement:
  label: rgw
service_id: rgw
service_name: rgw.rgw
service_type: rgw
```

```
spec:
  rgw_frontend_port: 8090
  rgw_realm: default
  rgw_zone: default
```

4. Apply the new RGW spec by using the orchestrator CLI:

```
$ cephadm shell -m /home/ceph-admin/specs/rgw
$ cephadm shell -- ceph orch apply -i /mnt/rgw
```

This command triggers the redeploy:

```
...
osd.9                      cephstorage-2
rgw.rgw.cephstorage-0.wsjlgx  cephstorage-0  172.17.3.23:8090   starting
rgw.rgw.cephstorage-1.qynkan  cephstorage-1  172.17.3.26:8090   starting
rgw.rgw.cephstorage-2.krycit  cephstorage-2  172.17.3.81:8090   starting
rgw.rgw.controller-1.eyvrzw   controller-1   172.17.3.146:8080  running (5h)
rgw.rgw.controller-2.navbxa   controller-2   172.17.3.66:8080   running (5h)

...
osd.9                      cephstorage-2
rgw.rgw.cephstorage-0.wsjlgx  cephstorage-0  172.17.3.23:8090  running (19s)
rgw.rgw.cephstorage-1.qynkan  cephstorage-1  172.17.3.26:8090  running (16s)
rgw.rgw.cephstorage-2.krycit  cephstorage-2  172.17.3.81:8090  running (13s)
```

5. Ensure that the new RGW backends are reachable on the new ports, because you are going to enable an IngressDaemon on port 8080 later in the process. For this reason, log in to each RGW node (the Red Hat Ceph Storage nodes) and add the iptables rule to allow connections to both 8080 and 8090 ports in the Red Hat Ceph Storage nodes.

```
iptables -I INPUT -p tcp -m tcp --dport 8080 -m conntrack --ctstate NEW -m comment --
comment "ceph rgw ingress" -j ACCEPT

iptables -I INPUT -p tcp -m tcp --dport 8090 -m conntrack --ctstate NEW -m comment --
comment "ceph rgw backends" -j ACCEPT

for port in 8080 8090; {
    for i in 25 10 32; {
       ssh heat-admin@192.168.24.$i sudo iptables -I INPUT \
       -p tcp -m tcp --dport $port -m conntrack --ctstate NEW \
       -j ACCEPT;
    }
}
```

6. From a Controller node (e.g. controller-0) try to reach (curl) the RGW backends:

```
for i in 26 23 81; do {
    echo "---"
    curl 172.17.3.$i:8090;
    echo "---"
    echo
done
```

You should observe the following output:

```
---
Query 172.17.3.23
<?xml version="1.0" encoding="UTF-8"?><ListAllMyBucketsResult
xmlns="http://s3.amazonaws.com/doc/2006-03-01/"><Owner><ID>anonymous</ID>
<DisplayName></DisplayName></Owner><Buckets></Buckets></ListAllMyBucketsResult>
---

---
Query 172.17.3.26
<?xml version="1.0" encoding="UTF-8"?><ListAllMyBucketsResult
xmlns="http://s3.amazonaws.com/doc/2006-03-01/"><Owner><ID>anonymous</ID>
<DisplayName></DisplayName></Owner><Buckets></Buckets></ListAllMyBucketsResult>
---

---
Query 172.17.3.81
<?xml version="1.0" encoding="UTF-8"?><ListAllMyBucketsResult
xmlns="http://s3.amazonaws.com/doc/2006-03-01/"><Owner><ID>anonymous</ID>
<DisplayName></DisplayName></Owner><Buckets></Buckets></ListAllMyBucketsResult>
---
```

7. If RGW backends are migrated in the Red Hat Ceph Storage nodes, there is no "**internalAPI**" network(this is not true in the case of HCI). Reconfigure the RGW keystone endpoint, pointing to the external network that has been propagated. For more information about propagating the external network, see Completing prerequisites for migrating Red Hat Ceph Storage RGW .

```
[ceph: root@controller-0 /]# ceph config dump | grep keystone
global   basic rgw_keystone_url  http://172.16.1.111:5000

[ceph: root@controller-0 /]# ceph config set global rgw_keystone_url http://10.0.0.103:5000
```

7.4. DEPLOYING A RED HAT CEPH STORAGE INGRESS DAEMON

To match the cardinality diagram, you use cephadm labels to refer to a group of nodes where a given daemon type should be deployed. For more information about the cardinality diagram, see Red Hat Ceph Storage daemon cardinality. **HAProxy** is managed by director through **Pacemaker**: the three running instances at this point will point to the old RGW backends, resulting in a broken configuration. Since you are going to deploy the Ceph ingress daemon, the first thing to do is remove the existing **ceph_rgw** config, clean up the config created by director and restart the service to make sure other services are not affected by this change. After you complete this procedure, you can reach the RGW backend from the ingress daemon and use RGW through the Object Storage service command line interface (CLI).

Procedure

1. Log in to each Controller node and remove the following configuration from the **/var/lib/config-data/puppet-generated/haproxy/etc/haproxy/haproxy.cfg** file:

```
listen ceph_rgw
  bind 10.0.0.103:8080 transparent
  mode http
  balance leastconn
```

```
  http-request set-header X-Forwarded-Proto https if { ssl_fc }
  http-request set-header X-Forwarded-Proto http if !{ ssl_fc }
  http-request set-header X-Forwarded-Port %[dst_port]
  option httpchk GET /swift/healthcheck
  option httplog
  option forwardfor
   server controller-0.storage.redhat.local 172.17.3.73:8080 check fall 5 inter 2000 rise 2
  server controller-1.storage.redhat.local 172.17.3.146:8080 check fall 5 inter 2000 rise 2
  server controller-2.storage.redhat.local 172.17.3.156:8080 check fall 5 inter 2000 rise 2
```

2. Restart **haproxy-bundle** and ensure it is started:

```
[root@controller-0 ~]# sudo pcs resource restart haproxy-bundle
haproxy-bundle successfully restarted


[root@controller-0 ~]# sudo pcs status | grep haproxy

  * Container bundle set: haproxy-bundle [undercloud-0.ctlplane.redhat.local:8787/rh-
osbs/rhosp17-openstack-haproxy:pcmklatest]:
    * haproxy-bundle-podman-0   (ocf:heartbeat:podman):  Started controller-0
    * haproxy-bundle-podman-1   (ocf:heartbeat:podman):  Started controller-1
    * haproxy-bundle-podman-2   (ocf:heartbeat:podman):  Started controller-2
```

3. Confirm that no process is bound to 8080:

```
[root@controller-0 ~]# ss -antop | grep 8080
[root@controller-0 ~]#
```

The Object Storage service (swift) CLI fails at this point:

```
(overcloud) [root@cephstorage-0 ~]# swift list

HTTPConnectionPool(host='10.0.0.103', port=8080): Max retries exceeded with url:
/swift/v1/AUTH_852f24425bb54fa896476af48cbe35d3?format=json (Caused by
NewConnectionError('<urllib3.connection.HTTPConnection object at 0x7fc41beb0430>:
Failed to establish a new connection: [Errno 111] Connection refused'))
```

4. Set the required images for both HAProxy and Keepalived:

```
[ceph: root@controller-0 /]# ceph config set mgr mgr/cephadm/container_image_haproxy
registry.redhat.io/rhceph/rhceph-haproxy-rhel9:latest
[ceph: root@controller-0 /]# ceph config set mgr mgr/cephadm/container_image_keepalived
registry.redhat.io/rhceph/keepalived-rhel9:latest
```

5. Create a file called **rgw_ingress** in the **/home/ceph-admin/specs/** directory in **controller-0**:

```
$ sudo vim /home/ceph-admin/specs/rgw_ingress
```

6. Paste the following content in to the **rgw_ingress** file:

```
---
service_type: ingress
service_id: rgw.rgw
```

```
placement:
  label: rgw
spec:
  backend_service: rgw.rgw
  virtual_ip: 10.0.0.89/24
  frontend_port: 8080
  monitor_port: 8898
  virtual_interface_networks:
    - <external_network>
```

- Replace **<external_network>** with your external network, for example, **10.0.0.0/24**. For more information, see Completing prerequisites for migrating Red Hat Ceph Storage RGW .

7. Apply the **rgw_ingress** spec by using the Ceph orchestrator CLI:

```
$ cephadm shell -m /home/ceph-admin/specs/rgw_ingress
$ cephadm shell -- ceph orch apply -i /mnt/rgw_ingress
```

8. Wait until the ingress is deployed and query the resulting endpoint:

```
[ceph: root@controller-0 /]# ceph orch ls

NAME                 PORTS            RUNNING  REFRESHED  AGE  PLACEMENT
crash                                  6/6  6m ago  3d   *
ingress.rgw.rgw      10.0.0.89:8080,8898   6/6  37s ago 60s  label:rgw
mds.mds                3/3  6m ago  3d   controller-0;controller-1;controller-2
mgr                    3/3  6m ago  3d   controller-0;controller-1;controller-2
mon                    3/3  6m ago  3d   controller-0;controller-1;controller-2
osd.default_drive_group   15  37s ago 3d   cephstorage-0;cephstorage-1;cephstorage-2
rgw.rgw   ?:8090         3/3  37s ago 4m   label:rgw
```

```
[ceph: root@controller-0 /]# curl  10.0.0.89:8080

---
<?xml version="1.0" encoding="UTF-8"?><ListAllMyBucketsResult
xmlns="http://s3.amazonaws.com/doc/2006-03-01/"><Owner><ID>anonymous</ID>
<DisplayName></DisplayName></Owner><Buckets></Buckets></ListAllMyBucketsResult>
[ceph: root@controller-0 /]#
—
```

7.5. UPDATING THE OBJECT-STORE ENDPOINTS

The object-storage endpoints still point to the original virtual IP address (VIP) that is owned by pacemaker. You must update the object-store endpoints because other services stll use the original VIP, and you reserved a new VIP on the same network.

Procedure

1. List the current endpoints:

```
(overcloud) [stack@undercloud-0 ~]$ openstack endpoint list | grep object

| 1326241fb6b6494282a86768311f48d1 | regionOne | swift     | object-store   | True | internal
| http://172.17.3.68:8080/swift/v1/AUTH_%(project_id)s |
```

```
| 8a34817a9d3443e2af55e108d63bb02b | regionOne | swift     | object-store   | True | public |
http://10.0.0.103:8080/swift/v1/AUTH_%(project_id)s  |
| fa72f8b8b24e448a8d4d1caaeaa7ac58 | regionOne | swift     | object-store   | True | admin  |
http://172.17.3.68:8080/swift/v1/AUTH_%(project_id)s |
```

2. Update the endpoints that are pointing to the Ingress VIP:

```
(overcloud) [stack@undercloud-0 ~]$ openstack endpoint set --url
"http://10.0.0.89:8080/swift/v1/AUTH_%(project_id)s" 95596a2d92c74c15b83325a11a4f07a3

(overcloud) [stack@undercloud-0 ~]$ openstack endpoint list | grep object-store
| 6c7244cc8928448d88ebfad864fdd5ca | regionOne | swift     | object-store   | True | internal
| http://172.17.3.79:8080/swift/v1/AUTH_%(project_id)s |
| 95596a2d92c74c15b83325a11a4f07a3 | regionOne | swift     | object-store   | True | public |
http://10.0.0.89:8080/swift/v1/AUTH_%(project_id)s   |
| e6d0599c5bf24a0fb1ddf6ecac00de2d | regionOne | swift     | object-store   | True | admin  |
http://172.17.3.79:8080/swift/v1/AUTH_%(project_id)s |
```

   Repeat this step for both internal and admin endpoints.

3. Test the migrated service:

```
(overcloud) [stack@undercloud-0 ~]$ swift list --debug

DEBUG:swiftclient:Versionless auth_url - using http://10.0.0.115:5000/v3 as endpoint
DEBUG:keystoneclient.auth.identity.v3.base:Making authentication request to
http://10.0.0.115:5000/v3/auth/tokens
DEBUG:urllib3.connectionpool:Starting new HTTP connection (1): 10.0.0.115:5000
DEBUG:urllib3.connectionpool:http://10.0.0.115:5000 "POST /v3/auth/tokens HTTP/1.1" 201
7795
DEBUG:keystoneclient.auth.identity.v3.base:{"token": {"methods": ["password"], "user":
{"domain": {"id": "default", "name": "Default"}, "id": "6f87c7ffdddf463bbc633980cfd02bb3",
"name": "admin", "password_expires_at": null},



...
...
...

DEBUG:swiftclient:REQ: curl -i
http://10.0.0.89:8080/swift/v1/AUTH_852f24425bb54fa896476af48cbe35d3?format=json -X
GET -H "X-Auth-Token:
gAAAAABj7KHdjZ95syP4c8v5a2zfXckPwxFQZYg0pgWR42JnUs83CcKhYGY6PFNF5Cg5g2W
uiYwMIXHm8xftyWf08zwTycJLLMeEwoxLkcByXPZr7kT92ApT-36wTfpi-
zbYXd1tI5R00xtAzDjO3RH1kmeLXDgIQEVp0jMRAxoVH4zb-DVHUos" -H "Accept-
Encoding: gzip"
DEBUG:swiftclient:RESP STATUS: 200 OK
DEBUG:swiftclient:RESP HEADERS: {'content-length': '2', 'x-timestamp':
'1676452317.72866', 'x-account-container-count': '0', 'x-account-object-count': '0', 'x-account-
bytes-used': '0', 'x-account-bytes-used-actual': '0', 'x-account-storage-policy-default-
placement-container-count': '0', 'x-account-storage-policy-default-placement-object-count': '0',
'x-account-storage-policy-default-placement-bytes-used': '0', 'x-account-storage-policy-
default-placement-bytes-used-actual': '0', 'x-trans-id': 'tx00000765c4b04f1130018-
0063eca1dd-1dcba-default', 'x-openstack-request-id': 'tx00000765c4b04f1130018-
```

```
0063eca1dd-1dcba-default', 'accept-ranges': 'bytes', 'content-type': 'application/json;
charset=utf-8', 'date': 'Wed, 15 Feb 2023 09:11:57 GMT'}
DEBUG:swiftclient:RESP BODY: b'[]'
```

4. Run tempest tests against object-storage:

```
(overcloud) [stack@undercloud-0 tempest-dir]$  tempest run --regex
tempest.api.object_storage
...
...
...
======
Totals
======
Ran: 141 tests in 606.5579 sec.
 - Passed: 128
 - Skipped: 13
 - Expected Fail: 0
 - Unexpected Success: 0
 - Failed: 0
Sum of execute time for each test: 657.5183 sec.

==============
Worker Balance
==============
 - Worker 0 (1 tests) => 0:10:03.400561
 - Worker 1 (2 tests) => 0:00:24.531916
 - Worker 2 (4 tests) => 0:00:10.249889
 - Worker 3 (30 tests) => 0:00:32.730095
 - Worker 4 (51 tests) => 0:00:26.246044
 - Worker 5 (6 tests) => 0:00:20.114803
 - Worker 6 (20 tests) => 0:00:16.290323
 - Worker 7 (27 tests) => 0:00:17.103827
```

CHAPTER 8. MIGRATING RED HAT CEPH STORAGE MDS TO NEW NODES WITHIN THE EXISTING CLUSTER

In the context of data plane adoption, where the Red Hat OpenStack Platform (RHOSP) services are redeployed in Red Hat OpenShift Container Platform, a director-deployed Red Hat Ceph Storage cluster will undergo a migration in a process we are calling “externalizing” the Red Hat Ceph Storage cluster. There are two deployment topologies, broadly, that include an “internal” Red Hat Ceph Storage cluster today: one is where RHOSP includes dedicated Red Hat Ceph Storage nodes to host object storage daemons (OSDs), and the other is Hyperconverged Infrastructure (HCI) where Compute nodes double up as Red Hat Ceph Storage nodes. In either scenario, there are some Red Hat Ceph Storage processes that are deployed on RHOSP Controller nodes: Red Hat Ceph Storage monitors, Ceph Object Gateway (RGW), Rados Block Device (RBD), Ceph Metadata Server (MDS), Ceph Dashboard, and NFS Ganesha. This document describes how to migrate the MDS daemon in case Shared File Systems service (manila) (deployed with either a cephfs-native or ceph-nfs backend) is part of the overcloud deployment. The MDS migration is performed by cephadm, and as done for the other daemons, the general idea is to move the daemons placement from a "hosts" based approach to a "label" based one. This ensures that the human operator can easily visualize the status of the cluster and where daemons are placed using the **ceph orch host** command, and have a general view of how the daemons are co-located within a given host, according to the cardinality matrix.

For this procedure, we assume that we are beginning with a RHOSP based on 17.1 and a Red Hat Ceph Storage 7 deployment managed by director. We assume that:

- Red Hat Ceph Storage is upgraded to Red Hat Ceph Storage 7 and is managed by cephadm/orchestrator.
- Both the Red Hat Ceph Storage public and cluster networks are propagated, throughdirector, to the target nodes.

Prerequisites

- Verify that the Red Hat Ceph Storage cluster is healthy and check the MDS status:

```
[ceph: root@controller-0 /]# ceph fs ls
name: cephfs, metadata pool: manila_metadata, data pools: [manila_data ]

[ceph: root@controller-0 /]# ceph mds stat
cephfs:1 {0=mds.controller-2.oebubl=up:active} 2 up:standby

[ceph: root@controller-0 /]# ceph fs status cephfs

cephfs - 0 clients
======
RANK  STATE         MDS           ACTIVITY  DNS INOS   DIRS   CAPS
 0 active  mds.controller-2.oebubl  Reqs: 0 /s   696 196 173   0
   POOL      TYPE  USED  AVAIL
manila_metadata  metadata   152M   141G
  manila_data    data 3072M   141G
   STANDBY MDS
mds.controller-0.anwiwd
mds.controller-1.cwzhog
MDS version: ceph version 17.2.6-100.el9cp (ea4e3ef8df2cf26540aae06479df031dcfc80343) quincy
(stable)
```

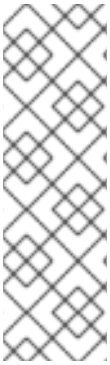
- Retrieve more detailed information on the Ceph File System (CephFS) MDS status:

```
[ceph: root@controller-0 /]# ceph fs dump

e8
enable_multiple, ever_enabled_multiple: 1,1
default compat: compat={},rocompat={},incompat={1=base v0.20,2=client writeable ranges,3=default
file layouts on dirs,4=dir inode in separate object,5=mds uses versioned encoding,6=dirfrag is stored
in omap,8=no anchor table,9=file layout v2,10=snaprealm v2}
legacy client fscid: 1

Filesystem 'cephfs' (1)
fs_name cephfs
epoch   5
flags   12 joinable allow_snaps allow_multimds_snaps
created 2024-01-18T19:04:01.633820+0000
modified    2024-01-18T19:04:05.393046+0000
tableserver  0
root 0
session_timeout 60
session_autoclose    300
max_file_size   1099511627776
required_client_features    {}
last_failure 0
last_failure_osd_epoch  0
compat  compat={},rocompat={},incompat={1=base v0.20,2=client writeable ranges,3=default file
layouts on dirs,4=dir inode in separate object,5=mds uses versioned encoding,6=dirfrag is stored in
omap,7=mds uses inline data,8=no anchor table,9=file layout v2,10=snaprealm v2}
max_mds 1
in   0
up   {0=24553}
failed
damaged
stopped
data_pools   [7]
metadata_pool   9
inline_data  disabled
balancer
standby_count_wanted 1
[mds.mds.controller-2.oebubl{0:24553} state up:active seq 2 addr
[v2:172.17.3.114:6800/680266012,v1:172.17.3.114:6801/680266012] compat {c=[1],r=[1],i=[7ff]}]


Standby daemons:

[mds.mds.controller-0.anwiwd{-1:14715} state up:standby seq 1 addr
[v2:172.17.3.20:6802/3969145800,v1:172.17.3.20:6803/3969145800] compat {c=[1],r=[1],i=[7ff]}]
[mds.mds.controller-1.cwzhog{-1:24566} state up:standby seq 1 addr
[v2:172.17.3.43:6800/2227381308,v1:172.17.3.43:6801/2227381308] compat {c=[1],r=[1],i=[7ff]}]
dumped fsmap epoch 8
```

- Check the OSD blocklist and clean up the client list:

```
[ceph: root@controller-0 /]# ceph osd blocklist ls
..
..
```

```
for item in $(ceph osd blocklist ls | awk '{print $0}'); do
    ceph osd blocklist rm $item;
done
```



NOTE

When a file system client is unresponsive or misbehaving, it may happen that the access to the file system is forcibly terminated. This process is called eviction. Evicting a CephFS client prevents it from communicating further with MDS daemons and OSD daemons. Ordinarily, a blocklisted client may not reconnect to the servers: it must be unmounted and then remounted. However, in some situations it may be useful to permit a client that was evicted to attempt to reconnect. Because CephFS uses the RADOS OSD blocklist to control client eviction, CephFS clients can be permitted to reconnect by removing them from the blocklist.

Procedure

1. Get the hosts that are currently part of the Red Hat Ceph Storage cluster:

```
[ceph: root@controller-0 /]# ceph orch host ls
HOST                        ADDR          LABELS          STATUS
cephstorage-0.redhat.local  192.168.24.25  osd mds
cephstorage-1.redhat.local  192.168.24.50  osd mds
cephstorage-2.redhat.local  192.168.24.47  osd mds
controller-0.redhat.local   192.168.24.24  _admin mgr mon
controller-1.redhat.local   192.168.24.42  mgr _admin mon
controller-2.redhat.local   192.168.24.37  mgr _admin mon
6 hosts in cluster

[ceph: root@controller-0 /]# ceph orch ls --export mds
service_type: mds
service_id: mds
service_name: mds.mds
placement:
  hosts:
  - controller-0.redhat.local
  - controller-1.redhat.local
  - controller-2.redhat.local
```

2. Extend the MDS labels to the target nodes:

```
for item in $(sudo cephadm shell --  ceph orch host ls --format json | jq -r '.[].hostname'); do
    sudo cephadm shell -- ceph orch host label add  $item mds;
done
```

3. Verify all the hosts have the MDS label:

```
[tripleo-admin@controller-0 ~]$ sudo cephadm shell -- ceph orch host ls

HOST                        ADDR          LABELS
cephstorage-0.redhat.local  192.168.24.11  osd mds
cephstorage-1.redhat.local  192.168.24.12  osd mds
cephstorage-2.redhat.local  192.168.24.47  osd mds
```

```
controller-0.redhat.local   192.168.24.35  _admin mon mgr mds
controller-1.redhat.local   192.168.24.53  mon _admin mgr mds
controller-2.redhat.local   192.168.24.10  mon _admin mgr mds
```

4. Dump the current MDS spec:

```
[ceph: root@controller-0 /]# ceph orch ls --export mds > mds.yaml
```

5. Edit the retrieved spec and replace the **placement.hosts** section with **placement.label**:

```
service_type: mds
service_id: mds
service_name: mds.mds
placement:
  label: mds
```

6. Use the **ceph orchestrator** to apply the new MDS spec: it results in an increased number of mds daemons:

```
$ sudo cephadm shell -m mds.yaml -- ceph orch apply -i /mnt/mds.yaml
Scheduling new mds deployment …
```

7. Check the new standby daemons temporarily added to the cephfs fs:

```
$ ceph fs dump

Active

standby_count_wanted    1
[mds.mds.controller-0.awzplm{0:463158} state up:active seq 307 join_fscid=1 addr
[v2:172.17.3.20:6802/51565420,v1:172.17.3.20:6803/51565420] compat {c=[1],r=[1],i=[7ff]}]


Standby daemons:

[mds.mds.cephstorage-1.jkvomp{-1:463800} state up:standby seq 1 join_fscid=1 addr
[v2:172.17.3.135:6820/2075903648,v1:172.17.3.135:6821/2075903648] compat {c=[1],r=
[1],i=[7ff]}]
[mds.mds.controller-2.gfrqvc{-1:475945} state up:standby seq 1 addr
[v2:172.17.3.114:6800/2452517189,v1:172.17.3.114:6801/2452517189] compat {c=[1],r=
[1],i=[7ff]}]
[mds.mds.cephstorage-0.fqcshx{-1:476503} state up:standby seq 1 join_fscid=1 addr
[v2:172.17.3.92:6820/4120523799,v1:172.17.3.92:6821/4120523799] compat {c=[1],r=[1],i=
[7ff]}]
[mds.mds.cephstorage-2.gnfhfe{-1:499067} state up:standby seq 1 addr
[v2:172.17.3.79:6820/2448613348,v1:172.17.3.79:6821/2448613348] compat {c=[1],r=[1],i=
[7ff]}]
[mds.mds.controller-1.tyiziq{-1:499136} state up:standby seq 1 addr
[v2:172.17.3.43:6800/3615018301,v1:172.17.3.43:6801/3615018301] compat {c=[1],r=[1],i=
[7ff]}]
```

8. To migrate MDS to the right nodes, set the MDS affinity that manages the MDS failover:

```
ceph config set mds.mds.cephstorage-0.fqcshx mds_join_fs cephfs
```

9. Remove the labels from Controller nodes and force the MDS failover to the target node:

```
$ for i in 0 1 2; do ceph orch host label rm "controller-$i.redhat.local" mds; done

Removed label mds from host controller-0.redhat.local
Removed label mds from host controller-1.redhat.local
Removed label mds from host controller-2.redhat.local
```

   The switch happens behind the scenes, and the new active MDS is the one that you set through the **mds_join_fs** command.

10. Check the result of the failover and the new deployed daemons:

```
$ ceph fs dump
…
…
standby_count_wanted    1
[mds.mds.cephstorage-0.fqcshx{0:476503} state up:active seq 168 join_fscid=1 addr
[v2:172.17.3.92:6820/4120523799,v1:172.17.3.92:6821/4120523799] compat {c=[1],r=[1],i=
[7ff]}]


Standby daemons:

[mds.mds.cephstorage-2.gnfhfe{-1:499067} state up:standby seq 1 addr
[v2:172.17.3.79:6820/2448613348,v1:172.17.3.79:6821/2448613348] compat {c=[1],r=[1],i=
[7ff]}]
[mds.mds.cephstorage-1.jkvomp{-1:499760} state up:standby seq 1 join_fscid=1 addr
[v2:172.17.3.135:6820/452139733,v1:172.17.3.135:6821/452139733] compat {c=[1],r=[1],i=
[7ff]}]


$ ceph orch ls

NAME                 PORTS   RUNNING  REFRESHED  AGE  PLACEMENT
crash                            6/6  10m ago    10d  *
mds.mds                          3/3  10m ago    32m  label:mds


$ ceph orch ps | grep mds


mds.mds.cephstorage-0.fqcshx  cephstorage-0.redhat.local                running (79m)     3m
ago  79m    27.2M        -  17.2.6-100.el9cp  1af7b794f353  2a2dc5ba6d57
mds.mds.cephstorage-1.jkvomp  cephstorage-1.redhat.local                running (79m)
3m ago  79m    21.5M        -  17.2.6-100.el9cp  1af7b794f353  7198b87104c8
mds.mds.cephstorage-2.gnfhfe  cephstorage-2.redhat.local                running (79m)     3m
ago  79m    24.2M        -  17.2.6-100.el9cp  1af7b794f353  f3cb859e2a15
```

CHAPTER 9. MIGRATING THE MONITORING STACK COMPONENT TO NEW NODES WITHIN AN EXISTING RED HAT CEPH STORAGE CLUSTER

In the context of data plane adoption, where the Red Hat OpenStack Platform (RHOSP) services are redeployed in Red Hat OpenShift Container Platform, a director-deployed Red Hat Ceph Storage cluster will undergo a migration in a process we are calling "externalizing" the Red Hat Ceph Storage cluster. There are two deployment topologies, broadly, that include an "internal" Red Hat Ceph Storage cluster today: one is where RHOSP includes dedicated Red Hat Ceph Storage nodes to host object storage daemons (OSDs), and the other is Hyperconverged Infrastructure (HCI) where Compute nodes double up as Red Hat Ceph Storage nodes. In either scenario, there are some Red Hat Ceph Storage processes that are deployed on RHOSP Controller nodes: Red Hat Ceph Storage monitors, Ceph Object Gateway (RGW), Rados Block Device (RBD), Ceph Metadata Server (MDS), Ceph Dashboard, and NFS Ganesha. The Ceph Dashboard module adds web-based monitoring and administration to the Ceph Manager. With director-deployed Red Hat Ceph Storage this component is enabled as part of the overcloud deploy and it's composed by:

- Ceph Manager module
- Grafana
- Prometheus
- Alertmanager
- Node exporter

The Ceph Dashboard containers are included through **tripleo-container-image-prepare** parameters and the high availability relies on **Haproxy** and **Pacemaker** deployed on the RHOSP front. For an external Red Hat Ceph Storage cluster, high availability is not supported. The goal of this procedure is to migrate and relocate the Ceph Monitoring components to free Controller nodes.

For this procedure, we assume that we are beginning with a RHOSP based on 17.1 and a Red Hat Ceph Storage 7 deployment managed by director. We assume that:

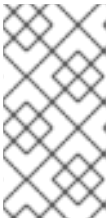
- Red Hat Ceph Storage has been upgraded to 7 and is managed by cephadm/orchestrator
- Both the Red Hat Ceph Storage public and cluster networks are propagated, throughdirector, to the target nodes

9.1. COMPLETING PREREQUISITES FOR A RED HAT CEPH STORAGE CLUSTER WITH MONITORING STACK COMPONENTS

You must complete the following prerequisites before you migrate a Red Hat Ceph Storage cluster with monitoring stack components.

Procedure

1. Gather the current status of the monitoring stack. Verify that the hosts have no **monitoring** label (or **grafana**, **prometheus**, **alertmanager** in case of a per daemons placement evaluation) associated:



NOTE

The entire relocation process is driven by cephadm and relies on labels to be assigned to the target nodes, where the daemons are scheduled. Review the cardinality matrix before assigning labels and choose carefully the nodes where the monitoring stack components should be scheduled on.

```
[tripleo-admin@controller-0 ~]$ sudo cephadm shell -- ceph orch host ls

HOST                  ADDR       LABELS              STATUS
cephstorage-0.redhat.local  192.168.24.11  osd mds
cephstorage-1.redhat.local  192.168.24.12  osd mds
cephstorage-2.redhat.local  192.168.24.47  osd mds
controller-0.redhat.local   192.168.24.35  _admin mon mgr
controller-1.redhat.local   192.168.24.53  mon _admin mgr
controller-2.redhat.local   192.168.24.10  mon _admin mgr
6 hosts in cluster
```

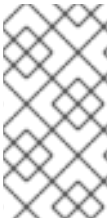
Confirm that the cluster is healthy and both **ceph orch ls** and **ceph orch ps** return the expected number of deployed daemons.

2. Review and update the container image registry. If the Red Hat Ceph Storage externalization procedure is executed after the Red Hat OpenStack Platform control plane has been migrated, it's important to consider updating the container images referenced in the Red Hat Ceph Storage cluster config. The current container images point to the undercloud registry, and it might be no longer available. As the undercloud won't be available in the future, replace the undercloud provided images with an alternative registry.

```
$ ceph config dump
...
...
mgr   advanced  mgr/cephadm/container_image_alertmanager    undercloud-
0.ctlplane.redhat.local:8787/rh-osbs/openshift-ose-prometheus-alertmanager:v4.10
mgr   advanced  mgr/cephadm/container_image_base           undercloud-
0.ctlplane.redhat.local:8787/rh-osbs/rhceph
mgr   advanced  mgr/cephadm/container_image_grafana        undercloud-
0.ctlplane.redhat.local:8787/rh-osbs/grafana:latest
mgr   advanced  mgr/cephadm/container_image_node_exporter   undercloud-
0.ctlplane.redhat.local:8787/rh-osbs/openshift-ose-prometheus-node-exporter:v4.10
mgr   advanced  mgr/cephadm/container_image_prometheus      undercloud-
0.ctlplane.redhat.local:8787/rh-osbs/openshift-ose-prometheus:v4.10
```

3. Remove the undercloud container images:

```
# remove the base image
cephadm shell -- ceph config rm mgr mgr/cephadm/container_image_base
# remove the undercloud images associated to the monitoring
# stack components
for i in prometheus grafana alertmanager node_exporter; do
    cephadm shell -- ceph config rm mgr mgr/cephadm/container_image_$i
done
```



NOTE

In the example above, in addition to the monitoring stack related container images, we update the config entry related to the container_image_base. This has an impact on all the Red Hat Ceph Storage daemons that rely on the undercloud images. New daemons will be deployed using the new/default Red Hat Ceph Storage image.

9.2. MIGRATING THE MONITORING STACK TO THE TARGET NODES

The migration procedure relies on nodes re-labeling: this kind of action, combined with an update in the existing spec, results in the daemons' relocation on the target nodes.

Before start this process, a few considerations are required:

- There's no need to migrate node exporters: these daemons are deployed across the nodes that are part of the Red Hat Ceph Storage cluster (placement is '*'), and we're going to lose metrics as long as the Controller nodes are not part of the Red Hat Ceph Storage cluster anymore
- Each monitoring stack component is bound to specific ports that director is supposed to open beforehand; make sure to double check the firewall rules are in place and the ports are opened for a given monitoring stack service

Depending on the target nodes and the number of deployed/active daemons, it is possible to either relocate the existing containers to the target nodes, or select a subset of nodes that are supposed to host the monitoring stack daemons. As we mentioned in the previous section, HA is not supported, hence reducing the placement with **count: 1** is a reasonable solution and allows to successfully migrate the existing daemons in an HCI (or HW limited) scenario without impacting other services. However, it is still possible to put in place a dedicated HA solution and realize a component that is consistent with the director model to reach HA. Building and deployment such HA model is out of scope for this procedure.

9.2.1. Scenario 1: Migrating the existing daemons to the target nodes

Assuming we have 3 Red Hat Ceph Storage nodes or ComputeHCI, this scenario extends the "monitoring" labels to all the Red Hat Ceph Storage (or ComputeHCI) nodes that are part of the cluster. This means that we keep the count: 3 placements for the target nodes.

Procedure

1. Add the monitoring label to all the Red Hat Ceph Storage (or ComputeHCI) nodes in the cluster:

```
for item in $(sudo cephadm shell --  ceph orch host ls --format json | jq -r '.[].hostname'); do
    sudo cephadm shell -- ceph orch host label add  $item monitoring;
done
```

2. Verify that all the (three) hosts have the monitoring label:

```
[tripleo-admin@controller-0 ~]$ sudo cephadm shell -- ceph orch host ls

HOST                     ADDR          LABELS
cephstorage-0.redhat.local  192.168.24.11  osd monitoring
cephstorage-1.redhat.local  192.168.24.12  osd monitoring
cephstorage-2.redhat.local  192.168.24.47  osd monitoring
```

```
controller-0.redhat.local   192.168.24.35  _admin mon mgr monitoring
controller-1.redhat.local   192.168.24.53  mon _admin mgr monitoring
controller-2.redhat.local   192.168.24.10  mon _admin mgr monitoring
```

3. Remove the labels from the Controller nodes:

```
$ for i in 0 1 2; do ceph orch host label rm "controller-$i.redhat.local" monitoring; done

Removed label monitoring from host controller-0.redhat.local
Removed label monitoring from host controller-1.redhat.local
Removed label monitoring from host controller-2.redhat.local
```

4. Dump the current monitoring stack spec:

```
function export_spec {
    local component="$1"
    local target_dir="$2"
    sudo cephadm shell -- ceph orch ls --export "$component" > "$target_dir/$component"
}

SPEC_DIR=${SPEC_DIR:-"$PWD/ceph_specs"}
for m in grafana prometheus alertmanager; do
    export_spec "$m" "$SPEC_DIR"
done
```

5. For each daemon, edit the current spec and replace the placement/hosts section with the placement/label section, for example:

```
service_type: grafana
service_name: grafana
placement:
  label: monitoring
networks:
- 172.17.3.0/24
spec:
  port: 3100
```

   The same procedure applies to Prometheus and Alertmanager specs.

6. Apply the new monitoring spec to relocate the monitoring stack daemons:

```
SPEC_DIR=${SPEC_DIR:-"$PWD/ceph_specs"}
function migrate_daemon {
    local component="$1"
    local target_dir="$2"
    sudo cephadm shell -m "$target_dir" -- ceph orch apply -i /mnt/ceph_specs/$component
}
for m in grafana prometheus alertmanager; do
    migrate_daemon  "$m" "$SPEC_DIR"
done
```

7. Verify that the daemons are deployed on the expected nodes:

```
[ceph: root@controller-0 /]# ceph orch ps | grep -iE "(prome|alert|grafa)"
```

```
alertmanager.cephstorage-2  cephstorage-2.redhat.local  172.17.3.144:9093,9094
grafana.cephstorage-0       cephstorage-0.redhat.local  172.17.3.83:3100
prometheus.cephstorage-1    cephstorage-1.redhat.local  172.17.3.53:9092
```

NOTE

After you migrate the monitoring stack, you lose High Availability: the monitoring stack daemons have no VIP and HAproxy anymore; Node exporters are still running on all the nodes: instead of using labels we keep the current approach as we want to not reduce the monitoring space covered.

8. You must review the Red Hat Ceph Storageconfiguration to ensure that it is aligned with the relocation you just made. In particular, focus on the following configuration entries:

```
[ceph: root@controller-0 /]# ceph config dump
...
mgr  advanced  mgr/dashboard/ALERTMANAGER_API_HOST  http://172.17.3.83:9093
mgr  advanced  mgr/dashboard/GRAFANA_API_URL        https://172.17.3.144:3100
mgr  advanced  mgr/dashboard/PROMETHEUS_API_HOST    http://172.17.3.83:9092
mgr  advanced  mgr/dashboard/controller-0.ycokob/server_addr  172.17.3.33
mgr  advanced  mgr/dashboard/controller-1.lmzpuc/server_addr  172.17.3.147
mgr  advanced  mgr/dashboard/controller-2.xpdgfl/server_addr  172.17.3.138
```

9. Verify that **grafana**, **alertmanager** and **prometheus API_HOST/URL** point to the IP addresses (on the storage network) of the node where each daemon has been relocated. This should be automatically addressed by cephadm and it shouldn't require any manual action.

```
[ceph: root@controller-0 /]# ceph orch ps | grep -iE "(prome|alert|grafa)"
alertmanager.cephstorage-0  cephstorage-0.redhat.local  172.17.3.83:9093,9094
alertmanager.cephstorage-1  cephstorage-1.redhat.local  172.17.3.53:9093,9094
alertmanager.cephstorage-2  cephstorage-2.redhat.local  172.17.3.144:9093,9094
grafana.cephstorage-0       cephstorage-0.redhat.local  172.17.3.83:3100
grafana.cephstorage-1       cephstorage-1.redhat.local  172.17.3.53:3100
grafana.cephstorage-2       cephstorage-2.redhat.local  172.17.3.144:3100
prometheus.cephstorage-0    cephstorage-0.redhat.local  172.17.3.83:9092
prometheus.cephstorage-1    cephstorage-1.redhat.local  172.17.3.53:9092
prometheus.cephstorage-2    cephstorage-2.redhat.local  172.17.3.144:9092
```

```
[ceph: root@controller-0 /]# ceph config dump
...
...
mgr  advanced  mgr/dashboard/ALERTMANAGER_API_HOST   http://172.17.3.83:9093
mgr  advanced  mgr/dashboard/PROMETHEUS_API_HOST     http://172.17.3.83:9092
mgr  advanced  mgr/dashboard/GRAFANA_API_URL         https://172.17.3.144:3100
```

10. The Ceph Dashboard (mgr module plugin) has not been impacted at all by this relocation. The service is provided by the Ceph Manager daemon, hence we might experience an impact when the active mgr is migrated or is force-failed. However, having three replicas definition allows to redirect requests to a different instance (it's still an A/P model), hence the impact should be limited.

    a. When the RBD migration is over, the following Red Hat Ceph Storage config keys must be regenerated to point to the right mgr container:

```
mgr    advanced  mgr/dashboard/controller-0.ycokob/server_addr  172.17.3.33
mgr    advanced  mgr/dashboard/controller-1.lmzpuc/server_addr  172.17.3.147
mgr    advanced  mgr/dashboard/controller-2.xpdgfl/server_addr  172.17.3.138
```

```
$ sudo cephadm shell
$ ceph orch ps | awk '/mgr./ {print $1}'
```

b. For each retrieved mgr, update the entry in the Red Hat Ceph Storage configuration:

```
$ ceph config set mgr mgr/dashboard/<>/server_addr/<ip addr>
```

CHAPTER 10. MIGRATING THE OBJECT STORAGE SERVICE (SWIFT) TO RED HAT OPENSTACK SERVICES ON OPENSHIFT (RHOSO) NODES

This section only applies if you are using Red Hat OpenStack Platform Object Storage service (swift) as Object Storage service. If you are using the Object Storage **API** of Ceph Object Gateway (RGW), you can skip this section.

Data migration to the new deployment might be a long running process that runs mostly in the background. The Object Storage service replicators will take care of moving data from old to new nodes, but depending on the amount of used storage this might take a very long time. You can still use the old nodes as long as they are running and continue with adopting other services in the meantime, reducing the amount of downtime. Note that performance might be decreased to the amount of replication traffic in the network.

Migration of the data happens replica by replica. Assuming you start with 3 replicas, only 1 one them is being moved at any time, ensuring the remaining 2 replicas are still available and the Object Storage service is usable during the migration.

10.1. MIGRATING THE OBJECT STORAGE SERVICE (SWIFT) DATA FROM RHOSP TO RED HAT OPENSTACK SERVICES ON OPENSHIFT (RHOSO) NODES

To ensure availability during the Object Storage service (swift) migration, you perform the following steps:

1. Add new nodes to the Object Storage service rings
2. Set weights of existing nodes to 0
3. Rebalance rings, moving one replica
4. Copy rings to old nodes and restart services
5. Check replication status and repeat previous two steps until old nodes are drained
6. Remove the old nodes from the rings

Prerequisites

- Previous Object Storage service adoption steps are completed.
- No new environmental variables need to be defined, though you use the **CONTROLLER1_SSH** alias that was defined in a previous step.
- For DNS servers, all existing nodes must be able to resolve host names of the Red Hat OpenShift Container Platform pods, for example by using the external IP of the DNSMasq service as name server in **/etc/resolv.conf**:

```
oc get service dnsmasq-dns -o jsonpath="{.status.loadBalancer.ingress[0].ip}" |
CONTROLLER1_SSH tee /etc/resolv.conf
```

- To track the current status of the replication a tool called **swift-dispersion** is used. It consists of

two parts, a population tool to be run before changing the Object Storage service rings and a report tool to run afterwards to gather the current status. Run the **swift-dispersion-populate** command:

```
oc debug --keep-labels=true job/swift-ring-rebalance -- /bin/sh -c 'swift-ring-tool get && swift-dispersion-populate'
```

The command might need a few minutes to complete. It creates 0-byte objects distributed across the Object Storage service deployment, and its counter-part **swift-dispersion-report** can be used afterwards to show the current replication status.

The output of the **swift-dispersion-report** command should look like the following:

```
oc debug --keep-labels=true job/swift-ring-rebalance -- /bin/sh -c 'swift-ring-tool get && swift-dispersion-report'
```

```
Queried 1024 containers for dispersion reporting, 5s, 0 retries
100.00% of container copies found (3072 of 3072)
Sample represents 100.00% of the container partition space
Queried 1024 objects for dispersion reporting, 4s, 0 retries
There were 1024 partitions missing 0 copies.
100.00% of object copies found (3072 of 3072)
Sample represents 100.00% of the object partition space
```

Procedure

1. Add new nodes by scaling up the SwiftStorage resource from 0 to 3. In that case 3 storage instances using PVCs are created, running on the Red Hat OpenShift Container Platform cluster.

```
oc patch openstackcontrolplane openstack --type=merge -p='{"spec":{"swift":{"template":{"swiftStorage":{"replicas": 3}}}}}'
```

2. Wait until all three pods are running:

```
oc wait pods --for condition=Ready -l component=swift-storage
```

3. Drain the existing nodes. Get the storage management IP addresses of the nodes to drain from the current rings:

```
oc debug --keep-labels=true job/swift-ring-rebalance -- /bin/sh -c 'swift-ring-tool get && swift-ring-builder object.builder' | tail -n +7 | awk '{print $4}' | sort -u
```

The output will look similar to the following:

```
172.20.0.100:6200
swift-storage-0.swift-storage.openstack.svc:6200
swift-storage-1.swift-storage.openstack.svc:6200
swift-storage-2.swift-storage.openstack.svc:6200
```

In this case the old node 172.20.0.100 is drained. Your nodes might be different, and depending on the deployment there are likely more nodes to be included in the following commands.

```
oc debug --keep-labels=true job/swift-ring-rebalance -- /bin/sh -c '
swift-ring-tool get
swift-ring-tool drain 172.20.0.100
swift-ring-tool rebalance
swift-ring-tool push'
```

4. Copy and apply the updated rings need to the original nodes. Run the ssh commands for your existing nodes storing Object Storage service data.

```
oc extract --confirm cm/swift-ring-files
CONTROLLER1_SSH "tar -C /var/lib/config-data/puppet-generated/swift/etc/swift/ -xzf -" <
swiftrings.tar.gz
CONTROLLER1_SSH "systemctl restart tripleo_swift_*"
```

5. Track the replication progress by using the **swift-dispersion-report** tool:

```
oc debug --keep-labels=true job/swift-ring-rebalance -- /bin/sh -c "swift-ring-tool get && swift-
dispersion-report"
```

The output shows less than 100% of copies found. Repeat the above command until both the container and all container and object copies are found:

```
Queried 1024 containers for dispersion reporting, 6s, 0 retries
There were 5 partitions missing 1 copy.
99.84% of container copies found (3067 of 3072)
Sample represents 100.00% of the container partition space
Queried 1024 objects for dispersion reporting, 7s, 0 retries
There were 739 partitions missing 1 copy.
There were 285 partitions missing 0 copies.
75.94% of object copies found (2333 of 3072)
Sample represents 100.00% of the object partition space
```

6. Move the next replica to the new nodes. To do so, rebalance and distribute the rings again:

```
oc debug --keep-labels=true job/swift-ring-rebalance -- /bin/sh -c '
swift-ring-tool get
swift-ring-tool rebalance
swift-ring-tool push'

oc extract --confirm cm/swift-ring-files
CONTROLLER1_SSH "tar -C /var/lib/config-data/puppet-generated/swift/etc/swift/ -xzf -" <
swiftrings.tar.gz
CONTROLLER1_SSH "systemctl restart tripleo_swift_*"
```

Monitor the **swift-dispersion-report** output again, wait until all copies are found again and repeat this step until all your replicas are moved to the new nodes.

7. After the nodes are drained, remove the nodes from the rings:

```
oc debug --keep-labels=true job/swift-ring-rebalance -- /bin/sh -c '
swift-ring-tool get
swift-ring-tool remove 172.20.0.100
swift-ring-tool rebalance
swift-ring-tool push'
```

Verification

- Even if all replicas are already on the the new nodes and the **swift-dispersion-report** command reports 100% of the copies found, there might still be data on old nodes. This data is removed by the replicators, but it might take some more time.
  You can check the disk usage of all disks in the cluster:

```
oc debug --keep-labels=true job/swift-ring-rebalance -- /bin/sh -c 'swift-ring-tool get && swift-recon -d'
```

- Confirm that there are no more ***.db** or ***.data** files in the directory **/srv/node** on these nodes:

```
CONTROLLER1_SSH "find /srv/node/ -type f -name '*.db' -o -name '*.data' | wc -l"
```

10.2. TROUBLESHOOTING THE OBJECT STORAGE SERVICE (SWIFT) MIGRATION

You can troubleshoot issues with the Object Storage service (swift) migration.

- The following command might be helpful to debug if the replication is not working and the **swift-dispersion-report** is not back to 100% availability.

```
CONTROLLER1_SSH tail /var/log/containers/swift/swift.log | grep object-server
```

  This should show the replicator progress, for example:

```
Mar 14 06:05:30 standalone object-server[652216]: <f+++++++++ 4e2/9cbea55c47e243994b0b10d8957184e2/1710395823.58025.data
Mar 14 06:05:30 standalone object-server[652216]: Successful rsync of /srv/node/vdd/objects/626/4e2 to swift-storage-1.swift-storage.openstack.svc::object/d1/objects/626 (0.094)
Mar 14 06:05:30 standalone object-server[652216]: Removing partition: /srv/node/vdd/objects/626
Mar 14 06:05:31 standalone object-server[652216]: <f+++++++++ 85f/cf53b5a048e5b19049e05a548cde185f/1710395796.70868.data
Mar 14 06:05:31 standalone object-server[652216]: Successful rsync of /srv/node/vdb/objects/829/85f to swift-storage-2.swift-storage.openstack.svc::object/d1/objects/829 (0.095)
Mar 14 06:05:31 standalone object-server[652216]: Removing partition: /srv/node/vdb/objects/829
```

- You can also check the ring consistency and replicator status:

```
oc debug --keep-labels=true job/swift-ring-rebalance -- /bin/sh -c 'swift-ring-tool get && swift-recon -r --md5'
```

  Note that the output might show a md5 mismatch until approx. 2 minutes after pushing new rings. Eventually it looks similar to the following example:

```
[...]
Oldest completion was 2024-03-14 16:53:27 (3 minutes ago) by 172.20.0.100:6000.
Most recent completion was 2024-03-14 16:56:38 (12 seconds ago) by swift-storage-0.swift-storage.openstack.svc:6200.
```

```
================================================================================
====
[2024-03-14 16:56:50] Checking ring md5sums
4/4 hosts matched, 0 error[s] while checking hosts.
[...]
```