Red Hat OpenStack Platform 16.1

Distributed compute node and storage deployment

Deploying Red Hat OpenStack Platform distributed compute node technologies

Last Updated: 2023-11-22

Red Hat OpenStack Platform 16.1 Distributed compute node and storage deployment

Deploying Red Hat OpenStack Platform distributed compute node technologies

OpenStack Team
rhos-docs@redhat.com

Legal Notice

Abstract

You can deploy Red Hat OpenStack Platform (RHOSP) with a distributed compute node (DCN) architecture for edge site operability with heat stack separation. Each site can have its own Ceph storage back end for Image service (glance) multi store.

Table of Contents

MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see our CTO Chris Wright's message . :leveloffset: +0

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

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

Using the Direct Documentation Feedback (DDF) function

Use the **Add Feedback** DDF function for direct comments on specific sentences, paragraphs, or code blocks.

1. View the documentation in the *Multi-page HTML* format.
2. Ensure that you see the **Feedback** button in the upper right corner of the document.
3. Highlight the part of text that you want to comment on.
4. Click **Add Feedback**.
5. Complete the **Add Feedback** field with your comments.
6. Optional: Add your email address so that the documentation team can contact you for clarification on your issue.
7. Click **Submit**.

CHAPTER 1. UNDERSTANDING DCN

Distributed compute node (DCN) architecture is for edge use cases allowing remote compute and storage nodes to be deployed remotely while sharing a common centralised control plane. DCN architecture allows you to position workloads strategically closer to your operational needs for higher performance.

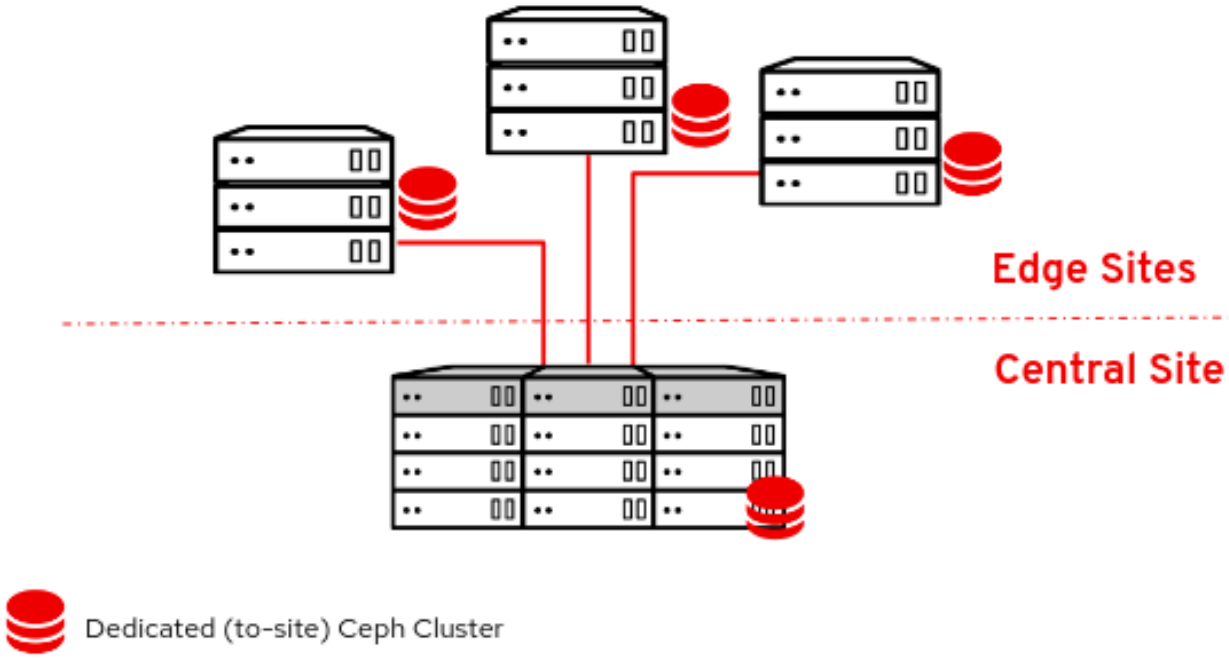
The central location can consist of any role, however at a minimum, requires three controllers. Compute nodes can exist at the edge, as well as at the central location.

DCN architecture is a hub and spoke routed network deployment. DCN is comparable to a spine and leaf deployment for routed provisioning and control plane networking with Red Hat OpenStack Platform director.

- The hub is the central site with core routers and a datacenter gateway (DC-GW).
- The spoke is the remote edge, or leaf.

Edge locations do not have controllers, making them architecturally different from traditional deployments of Red Hat OpenStack Platform:

- Control plane services run remotely, at the central location.
- Pacemaker is not installed.
- The Block Storage service (cinder) runs in active/active mode.
- Etcd is deployed as a distributed lock manager (DLM).



1.1. REQUIRED SOFTWARE FOR DISTRIBUTED COMPUTE NODE ARCHITECTURE

The following table shows the software and minimum versions required to deploy Red Hat OpenStack Platform in a distributed compute node (DCN) architecture:

Platform	Version	Optional
Red Hat Enterprise Linux	8	No
Red Hat OpenStack Platform	16.1	No
Red Hat Ceph Storage	4	Yes

1.2. MULTISTACK DESIGN

When you deploy Red Hat OpenStack Platform (RHOSP) with a DCN design, you use Red Hat director's capabilities for multiple stack deployment and management to deploy each site as a distinct stack.

Managing a DCN architecture as a single stack is unsupported, unless the deployment is an upgrade from Red Hat OpenStack Platform 13. There are no supported methods to split an existing stack, however you can add stacks to a pre-existing deployment. For more information, see Section A.3, "Migrating to a multistack deployment".

The central location is a traditional stack deployment of RHOSP, however you are not required to deploy Compute nodes or Red Hat Ceph storage with the central stack.

With DCN, you deploy each location as a distinct availability zone (AZ).

1.3. DCN STORAGE

You can deploy each edge site, either without storage, or with Ceph on hyperconverged nodes. The storage you deploy is dedicated to the site you deploy it on.

DCN architecture uses Glance multistore. For edge sites deployed without storage, additional tooling is available so that you can cache and store images in the Compute service (nova) cache. Caching glance images in nova provides the faster boot times for instances by avoiding the process of downloading images across a WAN link. For more information, see Chapter 9, *Precaching glance images into nova* .

1.4. DCN EDGE

With Distributed Compute Node architecture, the central location is deployed with the control nodes that manage the edge locations. When you then deploy an edge location, you deploy only compute nodes, making edge sites architecturally different from traditional deployments of Red Hat OpenStack Platform. At edge locations:

- Control plane services run remotely at the central location.
- Pacemaker does not run at DCN sites.
- The Block Storage service (cinder) runs in active/active mode.
- Etcd is deployed as a distributed lock manager (DLM).

CHAPTER 2. PLANNING A DISTRIBUTED COMPUTE NODE (DCN) DEPLOYMENT

When you plan your DCN architecture, check that the technologies that you need are available and supported.

2.1. CONSIDERATIONS FOR STORAGE ON DCN ARCHITECTURE

The following features are not currently supported for DCN architectures:

- Fast forward updates (FFU) on a distributed compute node architecture from Red Hat OpenStack Platform 13 to 16.
- Non-hyperconverged storage nodes at edge sites.
- Copying a volume snapshot between edge sites. You can work around this by creating an image from the volume and using glance to copy the image. After the image is copied, you can create a volume from it.
- Migrating or retyping a volume between sites.
- Ceph Rados Gateway (RGW) at the edge.
- CephFS at the edge.
- Instance high availability (HA) at the edge sites.
- RBD mirroring between sites.
- Instance migration, live or cold, either between edge sites, or from the central location to edge sites. You can still migrate instances within a site boundary. To move an image between sites, you must snapshot the image, and use **glance image-import**. For more information see Confirming image snapshots can be created and copied between sites .

Additionally, you must consider the following:

- You must upload images to the central location before copying them to edge sites; a copy of each image must exist in the Image service (glance) at the central location.
- Before you create an instance at an edge site, you must have a local copy of the image at that edge site.
- You must use the RBD storage driver for the Image, Compute and Block Storage services.
- For each site, assign a unique availability zone, and use the same value for the NovaComputeAvailabilityZone and CinderStorageAvailabilityZone parameters.

2.2. CONSIDERATIONS FOR NETWORKING ON DCN ARCHITECTURE

The following features are not currently supported for DCN architectures:

- Octavia
- DHCP on DPDK nodes

- Conntrack for TC Flower Hardware Offload

The ML2/OVN mechanism driver is available on DCN as a Technology Preview, and therefore using these solutions together is not fully supported by Red Hat. This feature should only be used with DCN for testing, and should not be deployed in a production environment. For more information about Technology Preview features, see Scope of Coverage Details.

NOTE

The ML2/OVN mechanism driver is fully supported outside of DCN environments.

The following networking technologies are supported with ML2/OVS:

- DPDK without DHCP on the DPDK nodes
- SR-IOV
- TC Flower hardware offload, without conntrack
- Neutron availability zones (AZs) with networker nodes at the edge, with on AZ per site
- Routed provider networks

Additionally, you must consider the following:

- Network latency: Balance the latency as measured in round-trip time (RTT), with the expected number of concurrent API operations to maintain acceptable performance. Maximum TCP/IP throughput is inversely proportional to RTT. You can mitigate some issues with high-latency connections with high bandwidth by tuning kernel TCP parameters.Contact Red Hat support if a cross-site communication exceeds 100 ms.
- Network drop outs: If the edge site temporarily loses connection to the central site, then no OpenStack control plane API or CLI operations can be executed at the impacted edge site for the duration of the outage. For example, Compute nodes at the edge site are consequently unable to create a snapshot of an instance, issue an auth token, or delete an image. General OpenStack control plane API and CLI operations remain functional during this outage, and can continue to serve any other edge sites that have a working connection. Image type: You must use raw images when deploying a DCN architecture with Ceph storage.
- Image sizing:
    - Overcloud node images - overcloud node images are downloaded from the central undercloud node. These images are potentially large files that will be transferred across all necessary networks from the central site to the edge site during provisioning.
    - Instance images: If there is no block storage at the edge, then the Image service images traverse the WAN during first use. The images are copied or cached locally to the target edge nodes for all subsequent use. There is no size limit for glance images. Transfer times vary with available bandwidth and network latency.
      If there is block storage at the edge, then the image is copied over the WAN asynchronously for faster boot times at the edge.
- Provider networks: This is the recommended networking approach for DCN deployments. If you use provider networks at remote sites, then you must consider that the Networking service (neutron) does not place any limits or checks on where you can attach available networks. For

example, if you use a provider network only in edge site A, you must ensure that you do not try to attach to the provider network in edge site B. This is because there are no validation checks on the provider network when binding it to a Compute node.

- Site-specific networks: A limitation in DCN networking arises if you use networks that are specific to a certain site: When you deploy centralized neutron controllers with Compute nodes, there are no triggers in neutron to identify a certain Compute node as a remote node. Consequently, the Compute nodes receive a list of other Compute nodes and automatically form tunnels between each other; the tunnels are formed from edge to edge through the central site. If you use VXLAN or Geneve, every Compute node at every site forms a tunnel with every other Compute node and Controller node, whether or not they are local or remote. This is not an issue if you are using the same neutron networks everywhere. When you use VLANs, neutron expects that all Compute nodes have the same bridge mappings, and that all VLANs are available at every site.

- Additional sites: If you need to expand from a central site to additional remote sites, you can use the openstack CLI on Red Hat OpenStack Platform director to add new network segments and subnets.

- If edge servers are not pre-provisioned, you must configure DHCP relay for introspection and provisioning on routed segments.

- Routing must be configured either on the cloud or within the networking infrastructure that connects each edge site to the hub. You should implement a networking design that allocates an L3 subnet for each Red Hat OpenStack Platform cluster network (external, internal API, and so on), unique to each site.

2.3. STORAGE TOPOLOGIES AND ROLES AT THE EDGE

When you deploy Red Hat OpenStack platform with a distributed compute node architecture, you must decide if you need storage at the edge. Based on storage and performance needs, you can deploy each site with one of three configurations. Not all edge sites must have an identical configuration.

If block storage is not going to be deployed at the edge, you must follow the section of the document, Section 6.1, “Deploying edge nodes without storage” . If there is no block storage at the edge site:

- Swift is used as a Glance backend

- Compute nodes at the edge may only cache images.

- Volume services like Cinder are not available at edge sites.

If you plan to deploy storage at the edge at any location, you must also deploy block storage at the central location. Follow the section of the document Section 5.2, “Deploying the central site with storage”. If there is block storage at the edge site:

- Ceph RBD is used as a Glance backend

- Images may be stored at edge sites

- The Cinder volume service is available via the Ceph RBD driver.

The roles required for your deployment will differ based whether or not you deploy Block storage at the edge:

- **No Block storage is required at the edge**

Compute

When you deploy an edge location without block storage, use must use the traditional **compute** role.

- **Block Storage is required at the edge**

  DistributedComputeHCI

  This role includes the following:

  - Default compute services
  - Block Storage (cinder) volume service
  - Ceph Mon
  - Ceph Mgr
  - Ceph OSD
  - GlanceApiEdge
  - Etcd
    This role enables a hyperconverged deployment at the edge. You must use exactly three nodes when using the **DistributedComputeHCI** role.

  DistributedComputeHCIScaleOut

  This role includes the **Ceph OSD** service, which allows storage capacity to be scaled with compute resources when more nodes are added to the edge. This role also includes the **HAproxyEdge** service to redirect image download requests to the **GlanceAPIEdge** nodes at the edge site.

  DistributedComputeScaleOut

  If you want to scale compute resources at the edge without storage, you can use the **DistributedComputeScaleOut** role.

CHAPTER 3. CONFIGURING ROUTED SPINE-LEAF IN THE UNDERCLOUD

This section describes a use case about how to configure the undercloud to accommodate routed spine-leaf with composable networks.

3.1. CONFIGURING THE SPINE LEAF PROVISIONING NETWORKS

To configure the provisioning networks for your spine leaf infrastructure, edit the **undercloud.conf** file and set the relevant parameters included in the following procedure.

Procedure

1. Log in to the undercloud as the **stack** user.
2. If you do not already have an **undercloud.conf** file, copy the sample template file:

```
[stack@director ~]$ cp /usr/share/python-tripleoclient/undercloud.conf.sample
~/undercloud.conf
```

3. Edit the **undercloud.conf** file.
4. Set the following values in the **[DEFAULT]** section:
   a. Set **local_ip** to the undercloud IP on **leaf0**:

```
local_ip = 192.168.10.1/24
```

   b. Set **undercloud_public_host** to the externally facing IP address of the undercloud:

```
undercloud_public_host = 10.1.1.1
```

   c. Set **undercloud_admin_host** to the administration IP address of the undercloud. This IP address is usually on leaf0:

```
undercloud_admin_host = 192.168.10.2
```

   d. Set **local_interface** to the interface to bridge for the local network:

```
local_interface = eth1
```

   e. Set **enable_routed_networks** to **true**:

```
enable_routed_networks = true
```

   f. Define your list of subnets using the **subnets** parameter. Define one subnet for each L2 segment in the routed spine and leaf:

```
subnets = leaf0,leaf1,leaf2
```

   g. Specify the subnet associated with the physical L2 segment local to the undercloud using the **local_subnet** parameter:

```
local_subnet = leaf0
```

h. Set the value of **undercloud_nameservers**.

```
undercloud_nameservers = 10.11.5.19,10.11.5.20
```

TIP

You can find the current IP addresses of the DNS servers that are used for the undercloud nameserver by looking in /etc/resolv.conf.

5. Create a new section for each subnet that you define in the **subnets** parameter:

```
[leaf0]
cidr = 192.168.10.0/24
dhcp_start = 192.168.10.10
dhcp_end = 192.168.10.90
inspection_iprange = 192.168.10.100,192.168.10.190
gateway = 192.168.10.1
masquerade = False

[leaf1]
cidr = 192.168.11.0/24
dhcp_start = 192.168.11.10
dhcp_end = 192.168.11.90
inspection_iprange = 192.168.11.100,192.168.11.190
gateway = 192.168.11.1
masquerade = False

[leaf2]
cidr = 192.168.12.0/24
dhcp_start = 192.168.12.10
dhcp_end = 192.168.12.90
inspection_iprange = 192.168.12.100,192.168.12.190
gateway = 192.168.12.1
masquerade = False
```

6. Save the **undercloud.conf** file.

7. Run the undercloud installation command:

```
[stack@director ~]$ openstack undercloud install
```

This configuration creates three subnets on the provisioning network or control plane. The overcloud uses each network to provision systems within each respective leaf.

To ensure proper relay of DHCP requests to the undercloud, you might need to configure a DHCP relay.

3.2. CONFIGURING A DHCP RELAY

You run the DHCP relay service on a switch, router, or server that is connected to the remote network segment you want to forward the requests from.

NOTE

Do not run the DHCP relay service on the undercloud.

The undercloud uses two DHCP servers on the provisioning network:

- An introspection DHCP server.
- A provisioning DHCP server.

You must configure the DHCP relay to forward DHCP requests to both DHCP servers on the undercloud.

You can use UDP broadcast with devices that support it to relay DHCP requests to the L2 network segment where the undercloud provisioning network is connected. Alternatively, you can use UDP unicast, which relays DHCP requests to specific IP addresses.

NOTE

Configuration of DHCP relay on specific device types is beyond the scope of this document. As a reference, this document provides a DHCP relay configuration example using the implementation in ISC DHCP software. For more information, see manual page dhcrelay(8).

IMPORTANT

DHCP option 79 is required for some relays, particularly relays that serve DHCPv6 addresses, and relays that do not pass on the originating MAC address. For more information, see RFC6939.

Broadcast DHCP relay

This method relays DHCP requests using UDP broadcast traffic onto the L2 network segment where the DHCP server or servers reside. All devices on the network segment receive the broadcast traffic. When using UDP broadcast, both DHCP servers on the undercloud receive the relayed DHCP request. Depending on the implementation, you can configure this by specifying either the interface or IP network address:

Interface

Specify an interface that is connected to the L2 network segment where the DHCP requests are relayed.

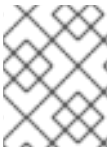
IP network address

Specify the network address of the IP network where the DHCP requests are relayed.

Unicast DHCP relay

This method relays DHCP requests using UDP unicast traffic to specific DHCP servers. When you use UDP unicast, you must configure the device that provides the DHCP relay to relay DHCP requests to both the IP address that is assigned to the interface used for introspection on the undercloud and the IP address of the network namespace that the OpenStack Networking (neutron) service creates to host the DHCP service for the **ctlplane** network.

The interface used for introspection is the one defined as **inspection_interface** in the **undercloud.conf** file. If you have not set this parameter, the default interface for the undercloud is **br-ctlplane**.



NOTE

It is common to use the **br-ctlplane** interface for introspection. The IP address that you define as the **local_ip** in the **undercloud.conf** file is on the **br-ctlplane** interface.

The IP address allocated to the Neutron DHCP namespace is the first address available in the IP range that you configure for the **local_subnet** in the **undercloud.conf** file. The first address in the IP range is the one that you define as **dhcp_start** in the configuration. For example, **192.168.10.10** is the IP address if you use the following configuration:

```
[DEFAULT]
local_subnet = leaf0
subnets = leaf0,leaf1,leaf2

[leaf0]
cidr = 192.168.10.0/24
dhcp_start = 192.168.10.10
dhcp_end = 192.168.10.90
inspection_iprange = 192.168.10.100,192.168.10.190
gateway = 192.168.10.1
masquerade = False
```

WARNING

The IP address for the DHCP namespace is automatically allocated. In most cases, this address is the first address in the IP range. To verify that this is the case, run the following commands on the undercloud:

```
$ openstack port list --device-owner network:dhcp -c "Fixed IP Addresses"
+-----------------------------------------------------------------------------+
| Fixed IP Addresses                                                          |
+-----------------------------------------------------------------------------+
| ip_address='192.168.10.10', subnet_id='7526fbe3-f52a-4b39-a828-
ec59f4ed12b2' |
+-----------------------------------------------------------------------------+
$ openstack subnet show 7526fbe3-f52a-4b39-a828-ec59f4ed12b2 -c name
+-------+--------+
| Field | Value  |
+-------+--------+
| name  | leaf0  |
+-------+--------+
```

Example dhcrelay configuration

In the following examples, the **dhcrelay** command in the **dhcp** package uses the following configuration:

- Interfaces to relay incoming DHCP request: **eth1**, **eth2**, and **eth3**.
- Interface the undercloud DHCP servers on the network segment are connected to: **eth0**.

- The DHCP server used for introspection is listening on IP address: **192.168.10.1**.
- The DHCP server used for provisioning is listening on IP address **192.168.10.10**.

This results in the following **dhcrelay** command:

- **dhcrelay** version 4.2.x:

```
$ sudo dhcrelay -d --no-pid 192.168.10.10 192.168.10.1 \
  -i eth0 -i eth1 -i eth2 -i eth3
```

- **dhcrelay** version 4.3.x and later:

```
$ sudo dhcrelay -d --no-pid 192.168.10.10 192.168.10.1 \
  -iu eth0 -id eth1 -id eth2 -id eth3
```

Example Cisco IOS routing switch configuration

This example uses the following Cisco IOS configuration to perform the following tasks:

- Configure a VLAN to use for the provisioning network.
- Add the IP address of the leaf.
- Forward UDP and BOOTP requests to the introspection DHCP server that listens on IP address: **192.168.10.1**.
- Forward UDP and BOOTP requests to the provisioning DHCP server that listens on IP address **192.168.10.10**.

```
interface vlan 2
ip address 192.168.24.254 255.255.255.0
ip helper-address 192.168.10.1
ip helper-address 192.168.10.10
!
```

Now that you have configured the provisioning network, you can configure the remaining overcloud leaf networks.

3.3. CREATING FLAVORS AND TAGGING NODES FOR LEAF NETWORKS

Each role in each leaf network requires a flavor and role assignment so that you can tag nodes into their respective leaf. Complete the following steps to create and assign each flavor to a role.

Procedure

1. Source the **stackrc** file:

```
[stack@director ~]$ source ~/stackrc
```

2. Create flavors for each custom role:

```
$ ROLES="control compute_leaf0 compute_leaf1 compute_leaf2 ceph-storage_leaf0 ceph-
```

```
storage_leaf1 ceph-storage_leaf2"
$ for ROLE in $ROLES; do openstack flavor create --id auto --ram <ram_size_mb> --disk
<disk_size_gb> --vcpus <no_vcpus> $ROLE ; done
$ for ROLE in $ROLES; do openstack flavor set --property "cpu_arch"="x86_64" --property
"capabilities:boot_option"="local" --property resources:DISK_GB='0' --property
resources:MEMORY_MB='0' --property resources:VCPU='0' $ROLE ; done
```

- Replace **<ram_size_mb>** with the RAM of the bare metal node, in MB.
- Replace **<disk_size_gb>** with the size of the disk on the bare metal node, in GB.
- Replace **<no_vcpus>** with the number of CPUs on the bare metal node.

3. Retrieve a list of your nodes to identify their UUIDs:

```
(undercloud)$ openstack baremetal node list
```

4. Tag each bare metal node to its leaf network and role by using a custom resource class:

```
(undercloud)$ openstack baremetal node set \
 --resource-class baremetal.LEAF-ROLE <node>
```

Replace **<node>** with the ID of the bare metal node.

For example, enter the following command to tag a node with UUID **58c3d07e-24f2-48a7-bbb6-6843f0e8ee13** to the Compute role on Leaf2:

```
(undercloud)$ openstack baremetal node set \
 --resource-class baremetal.COMPUTE-LEAF2 58c3d07e-24f2-48a7-bbb6-6843f0e8ee13
```

5. Associate each leaf network role flavor with the custom resource class:

```
(undercloud)$ openstack flavor set \
 --property resources:CUSTOM_BAREMETAL_LEAF_ROLE=1 \
<custom_role>
```

To determine the name of a custom resource class that corresponds to a resource class of a Bare Metal Provisioning service node, convert the resource class to uppercase, replace each punctuation mark with an underscore, and prefix with **CUSTOM_**.

NOTE

A flavor can request only one instance of a bare metal resource class.

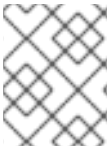
6. In the **node-info.yaml** file, specify the flavor that you want to use for each custom leaf role, and the number of nodes to allocate for each custom leaf role. For example, the following configuration specifies the flavor to use, and the number of nodes to allocate for the custom leaf roles **compute_leaf0**, **compute_leaf1**, **compute_leaf2**, **ceph-storage_leaf0**, **ceph-storage_leaf1**, and **ceph-storage_leaf2**:

```
parameter_defaults:
  OvercloudControllerFlavor: control
  OvercloudComputeLeaf0Flavor: compute_leaf0
  OvercloudComputeLeaf1Flavor: compute_leaf1
```

```
OvercloudComputeLeaf2Flavor: compute_leaf2
OvercloudCephStorageLeaf0Flavor: ceph-storage_leaf0
OvercloudCephStorageLeaf1Flavor: ceph-storage_leaf1
OvercloudCephStorageLeaf2Flavor: ceph-storage_leaf2
ControllerLeaf0Count: 3
ComputeLeaf0Count: 3
ComputeLeaf1Count: 3
ComputeLeaf2Count: 3
CephStorageLeaf0Count: 3
CephStorageLeaf1Count: 3
CephStorageLeaf2Count: 3
```

3.4. MAPPING BARE METAL NODE PORTS TO CONTROL PLANE NETWORK SEGMENTS

To enable deployment on a L3 routed network, you must configure the **physical_network** field on the bare metal ports. Each bare metal port is associated with a bare metal node in the OpenStack Bare Metal (ironic) service. The physical network names are the names that you include in the **subnets** option in the undercloud configuration.



NOTE

The physical network name of the subnet specified as **local_subnet** in the **undercloud.conf** file is always named **ctlplane**.

Procedure

1. Source the **stackrc** file:

```
$ source ~/stackrc
```

2. Check the bare metal nodes:

```
$ openstack baremetal node list
```

3. Ensure that the bare metal nodes are either in **enroll** or **manageable** state. If the bare metal node is not in one of these states, the command that sets the **physical_network** property on the baremetal port fails. To set all nodes to **manageable** state, run the following command:

```
$ for node in $(openstack baremetal node list -f value -c Name); do openstack baremetal
node manage $node --wait; done
```

4. Check which baremetal ports are associated with which baremetal node:

```
$ openstack baremetal port list --node <node-uuid>
```

5. Set the **physical-network** parameter for the ports. In the example below, three subnets are defined in the configuration: **leaf0**, **leaf1**, and **leaf2**. The local_subnet is **leaf0**. Because the physical network for the **local_subnet** is always **ctlplane**, the baremetal port connected to **leaf0** uses ctlplane. The remaining ports use the other leaf names:

```
$ openstack baremetal port set --physical-network ctlplane <port-uuid>
$ openstack baremetal port set --physical-network leaf1 <port-uuid>
$ openstack baremetal port set --physical-network leaf2 <port-uuid>
```

6. Introspect the nodes before you deploy the overcloud. Include the **--all-manageable** and **--provide** options to set the nodes as available for deployment:

```
$ openstack overcloud node introspect --all-manageable --provide
```

3.5. ADDING A NEW LEAF TO A SPINE-LEAF PROVISIONING NETWORK

When increasing network capacity which can include adding new physical sites, you might need to add a new leaf and a corresponding subnet to your Red Hat OpenStack Platform spine-leaf provisioning network. When provisioning a leaf on the overcloud, the corresponding undercloud leaf is used.

Prerequisites

- Your RHOSP deployment uses a spine-leaf network topology.

Procedure

1. Log in to the undercloud host as the stack user.
2. Source the undercloud credentials file:

```
$ source ~/stackrc
```

3. In the **/home/stack/undercloud.conf** file, do the following:
   a. Locate the **subnets** parameter, and add a new subnet for the leaf that you are adding. A subnet represents an L2 segment in the routed spine and leaf:

Example

In this example, a new subnet (**leaf3**) is added for the new leaf (**leaf3**):

```
subnets = leaf0,leaf1,leaf2,leaf3
```

   b. Create a section for the subnet that you added.

Example

In this example, the section **[leaf3]** is added for the new subnet (**leaf3**):

```
[leaf0]
cidr = 192.168.10.0/24
dhcp_start = 192.168.10.10
dhcp_end = 192.168.10.90
inspection_iprange = 192.168.10.100,192.168.10.190
gateway = 192.168.10.1
masquerade = False
```

```
[leaf1]
cidr = 192.168.11.0/24
dhcp_start = 192.168.11.10
dhcp_end = 192.168.11.90
inspection_iprange = 192.168.11.100,192.168.11.190
gateway = 192.168.11.1
masquerade = False

[leaf2]
cidr = 192.168.12.0/24
dhcp_start = 192.168.12.10
dhcp_end = 192.168.12.90
inspection_iprange = 192.168.12.100,192.168.12.190
gateway = 192.168.12.1
masquerade = False

[leaf3]
cidr = 192.168.13.0/24
dhcp_start = 192.168.13.10
dhcp_end = 192.168.13.90
inspection_iprange = 192.168.13.100,192.168.13.190
gateway = 192.168.13.1
masquerade = False
```

4. Save the **undercloud.conf** file.
5. Reinstall your undercloud:

```
$ openstack undercloud install
```

Additional resources

- Adding a new leaf to a spine-leaf deployment

CHAPTER 4. PREPARING OVERCLOUD TEMPLATES FOR DCN DEPLOYMENT

4.1. PREREQUISITES FOR USING SEPARATE HEAT STACKS

Your environment must meet the following prerequisites before you create a deployment using separate heat stacks:

- A working Red Hat OpenStack Platform 16 undercloud.
- For Ceph Storage users: access to Red Hat Ceph Storage 4.
- For the central location: three nodes that are capable of serving as central Controller nodes. All three Controller nodes must be in the same heat stack. You cannot split Controller nodes, or any of the control plane services, across separate heat stacks.
- Ceph storage is a requirement at the central location if you plan to deploy Ceph storage at the edge.
- For each additional DCN site: three HCI compute nodes.
- All nodes must be pre-provisioned or able to PXE boot from the central deployment network. You can use a DHCP relay to enable this connectivity for DCNs.
- All nodes have been introspected by ironic.
- Red Hat recommends leaving the <role>HostnameFormat parameter as the default value: %stackname%-<role>-%index%. If you do not include the %stackname% prefix, your overcloud uses the same hostnames for distributed compute nodes in different stacks. Ensure that your distributed compute nodes use the %stackname% prefix to distinguish nodes from different edge sites. For example, if you deploy two edge sites named **dcn0** and **dcn1**, the stack name prefix helps you to distinguish between dcn0-distributedcompute-0 and dcn1-distributedcompute-0 when you run the **openstack server list** command on the undercloud.
- Source the **centralrc** authentication file to schedule workloads at edge sites as well as at the central location. You do not require authentication files that are automatically generated for edge sites.

4.2. LIMITATIONS OF THE EXAMPLE SEPARATE HEAT STACKS DEPLOYMENT

This document provides an example deployment that uses separate heat stacks on Red Hat OpenStack Platform. This example environment has the following limitations:

- Spine/Leaf networking - The example in this guide does not demonstrate routing requirements, which are required in distributed compute node (DCN) deployments.
- Ironic DHCP Relay - This guide does not include how to configure Ironic with a DHCP relay.

4.3. DESIGNING YOUR SEPARATE HEAT STACKS DEPLOYMENT

To segment your deployment within separate heat stacks, you must first deploy a single overcloud with the control plane. You can then create separate stacks for the distributed compute node (DCN) sites. The following example shows separate stacks for different node types:

- Controller nodes: A separate heat stack named **central**, for example, deploys the controllers. When you create new heat stacks for the DCN sites, you must create them with data from the **central** stack. The Controller nodes must be available for any instance management tasks.
- DCN sites: You can have separate, uniquely named heat stacks, such as **dcn0**, **dcn1**, and so on. Use a DHCP relay to extend the provisioning network to the remote site.

NOTE

You must create a separate availability zone (AZ) for each stack.

NOTE

If you use spine/leaf networking, you must use a specific format to define the **Storage** and **StorageMgmt** networks so that ceph-ansible correctly configures Ceph to use those networks. Define the **Storage** and **StorageMgmt** networks as override values and enclose the values in single quotes. In the following example the storage network (referred to as the **public_network**) spans two subnets, is separated by a comma, and is enclosed in single quotes:

```
CephAnsibleExtraConfig:
  public_network: '172.23.1.0/24,172.23.2.0/24'
```

4.4. REUSING NETWORK RESOURCES IN MULTIPLE STACKS

You can configure multiple stacks to use the same network resources, such as VIPs and subnets. You can duplicate network resources between stacks by using either the **ManageNetworks** setting or the **external_resource_*** fields.

NOTE

Do not use the **ManageNetworks** setting if you are using the **external_resource_*** fields.

If you are not reusing networks between stacks, each network that is defined in **network_data.yaml** must have a unique name across all deployed stacks. For example, the network name **internal_api** cannot be reused between stacks, unless you intend to share the network between the stacks. Give the network a different name and **name_lower** property, such as **InternalApiCompute0** and **internal_api_compute_0**.

4.5. USING MANAGENETWORKS TO REUSE NETWORK RESOURCES

With the **ManageNetworks** setting, multiple stacks can use the same **network_data.yaml** file and the setting is applied globally to all network resources. The **network_data.yaml** file defines the network resources that the stack uses:

```
- name: StorageBackup
  vip: true
  name_lower: storage_backup
  ip_subnet: '172.21.1.0/24'
  allocation_pools: [{'start': '171.21.1.4', 'end': '172.21.1.250'}]
  gateway_ip: '172.21.1.1'
```

When you set ManageNetworks to false, the nodes will use the existing networks that were already created in the **central** stack.

Use the following sequence so that the new stack does not manage the existing network resources.

Procedure

1. Deploy the central stack with **ManageNetworks: true** or leave unset.
2. Deploy the additional stack with **ManageNetworks: false**.

When you add new network resources, for example when you add new leaves in a spine/leaf deployment, you must update the central stack with the new **network_data.yaml**. This is because the central stack still owns and manages the network resources. After the network resources are available in the central stack, you can deploy the additional stack to use them.

4.6. USING UUIDS TO REUSE NETWORK RESOURCES

If you need more control over which networks are reused between stacks, you can use the **external_resource_*** field for resources in the **network_data.yaml** file, including networks, subnets, segments, or VIPs. These resources are marked as being externally managed, and heat does not perform any create, update, or delete operations on them.

Add an entry for each required network definition in the **network_data.yaml** file. The resource is then available for deployment on the separate stack:

```
external_resource_network_id: Existing Network UUID
external_resource_subnet_id: Existing Subnet UUID
external_resource_segment_id: Existing Segment UUID
external_resource_vip_id: Existing VIP UUID
```

This example reuses the **internal_api** network from the control plane stack in a separate stack.

Procedure

1. Identify the UUIDs of the related network resources:

```
$ openstack network show internal_api -c id -f value
$ openstack subnet show internal_api_subnet -c id -f value
$ openstack port show internal_api_virtual_ip -c id -f value
```

2. Save the values that are shown in the output of the above commands and add them to the network definition for the **internal_api** network in the **network_data.yaml** file for the separate stack:

```
- name: InternalApi
  external_resource_network_id: 93861871-7814-4dbc-9e6c-7f51496b43af
  external_resource_subnet_id: c85c8670-51c1-4b17-a580-1cfb4344de27
  external_resource_vip_id: 8bb9d96f-72bf-4964-a05c-5d3fed203eb7
  name_lower: internal_api
  vip: true
  ip_subnet: '172.16.2.0/24'
  allocation_pools: [{'start': '172.16.2.4', 'end': '172.16.2.250'}]
  ipv6_subnet: 'fd00:fd00:fd00:2000::/64'
```

```
  ipv6_allocation_pools: [{'start': 'fd00:fd00:fd00:2000::10', 'end':
'fd00:fd00:fd00:2000:ffff:ffff:ffff:fffe'}]
  mtu: 1400
```

4.7. MANAGING SEPARATE HEAT STACKS

The procedures in this guide show how to organize the environment files for three heat stacks: **central**, **dcn0**, and **dcn1**. Red Hat recommends that you store the templates for each heat stack in a separate directory to keep the information about each deployment isolated.

Procedure

1. Define the **central** heat stack:

```
$ mkdir central
$ touch central/overrides.yaml
```

2. Extract data from the **central** heat stack into a common directory for all DCN sites:

```
$ mkdir dcn-common
$ touch dcn-common/overrides.yaml
$ touch dcn-common/central-export.yaml
```

   The **central-export.yaml** file is created later by the **openstack overcloud export** command. It is in the **dcn-common** directory because all DCN deployments in this guide must use this file.

3. Define the **dcn0** site.

```
$ mkdir dcn0
$ touch dcn0/overrides.yaml
```

To deploy more DCN sites, create additional **dcn** directories by number.

NOTE

The touch is used to provide an example of file organization. Each file must contain the appropriate content for successful deployments.

4.8. RETRIEVING THE CONTAINER IMAGES

Use the following procedure, and its example file contents, to retrieve the container images you need for deployments with separate heat stacks. You must ensure the container images for optional or edge-specific services are included by running the **openstack container image prepare** command with edge site's environment files.

For more information, see Preparing container images.

Procedure

1. Add your Registry Service Account credentials to **containers.yaml**.

```
parameter_defaults:
  ContainerImagePrepare:
```

```
  - push_destination: true
    set:
      ceph_namespace: registry.redhat.io/rhceph
      ceph_image: rhceph-4-rhel8
      ceph_tag: latest
      name_prefix: openstack-
      namespace: registry.redhat.io/rhosp16-rhel8
      tag: latest
  ContainerImageRegistryCredentials:
    # https://access.redhat.com/RegistryAuthentication
    registry.redhat.io:
      registry-service-account-username: registry-service-account-password
```

2. Generate the environment file as **images-env.yaml**:

```
sudo openstack tripleo container image prepare \
-e containers.yaml \
--output-env-file images-env.yaml
```

The resulting **images-env.yaml** file is included as part of the overcloud deployment procedure for the stack for which it is generated.

4.9. CREATING FAST DATAPATH ROLES FOR THE EDGE

To use fast datapath services at the edge, you must create a custom role that defines both fast datapath and edge services. When you create the roles file for deployment, you can include the newly created role that defines services needed for both distributed compute node architecture and fast datapath services such as DPDK or SR-IOV.

For example, create a custom role for distributedCompute with DPDK:

Prerequisites

A successful undercloud installation. For more information, see Installing the undercloud.

Procedure

1. Log in to the undercloud host as the **stack** user.
2. Copy the default **roles** directory:

```
cp -r /usr/share/openstack-tripleo-heat-templates/roles ~/.
```

3. Create a new file named **DistributedComputeDpdk.yaml** from the **DistributedCompute.yaml** file:

```
cp roles/DistributedCompute.yaml roles/DistributedComputeDpdk.yaml
```

4. Add DPDK services to the new **DistributedComputeDpdk.yaml** file. You can identify the parameters that you need to add by identifying the parameters in the **ComputeOvsDpdk.yaml** file that are not present in the **DistributedComputeDpdk.yaml** file.

```
diff -u roles/DistributedComputeDpdk.yaml roles/ComputeOvsDpdk.yaml
```

In the output, the parameters that are preceded by **+** are present in the ComputeOvsDpdk.yaml file but are not present in the DistributedComputeDpdk.yaml file. Include these parameters in the new **DistributedComputeDpdk.yaml** file.

5. Use the **DistributedComputeDpdk.yaml** to create a **DistributedComputeDpdk** roles file :

```
openstack overcloud roles generate --roles-path ~/roles/ -o ~/roles/roles-custom.yaml
DistributedComputeDpdk
```

You can use this same method to create fast datapath roles for SR-IOV, or a combination of SR-IOV and DPDK for the edge to meet your requirements.

Additional Resources

- Creating a custom role
- Supported custom roles

If you are planning to deploy edge sites without block storage, see the following:

- Chapter 5, *Installing the central location*
- Section 6.1, "Deploying edge nodes without storage"

If you are planning to deploy edge sites with Ceph storage, see the following:

- Chapter 5, *Installing the central location*
- Section 7.1, "Deploying edge sites with storage"

CHAPTER 5. INSTALLING THE CENTRAL LOCATION

When you deploy the central location for distributed compute node (DCN) architecture, you can deploy the cluster:

- With or without Compute nodes
- With or without Red Hat Ceph Storage

If you deploy Red Hat OpenStack Platform without Red Hat Ceph Storage at the central location, you cannot deploy any of your edge sites with Red Hat Ceph storage. Additionally, you do not have the option of adding Red Hat Ceph Storage to the central location later by redeploying.

5.1. DEPLOYING THE CENTRAL CONTROLLERS WITHOUT EDGE STORAGE

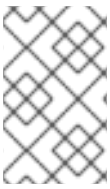
You can deploy a distributed compute node cluster without Block storage at edge sites if you use the Object Storage service (swift) as a back end for the Image service (glance) at the central location. A site deployed without block storage cannot be updated later to have block storage due to the differing role and networking profiles for each architecture.

Important: The following procedure uses lvm as the backend for Cinder which is not supported for production. You must deploy a certified block storage solution as a backend for Cinder.

Deploy the central controller cluster in a similar way to a typical overcloud deployment. This cluster does not require any Compute nodes, so you can set the Compute count to **0** to override the default of **1**. The central controller has particular storage and Oslo configuration requirements. Use the following procedure to address these requirements.

Procedure

The following procedure outlines the steps for the initial deployment of the central location.



NOTE

The following steps detail the deployment commands and environment files associated with an example DCN deployment without glance multistore. These steps do not include unrelated, but necessary, aspects of configuration, such as networking.

1. In the home directory, create directories for each stack that you plan to deploy.

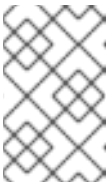
```
mkdir /home/stack/central
mkdir /home/stack/dcn0
mkdir /home/stack/dcn1
```

2. Create a file called **central/overrides.yaml** with settings similar to the following:

```
parameter_defaults:
  NtpServer:
    - 0.pool.ntp.org
    - 1.pool.ntp.org
  ControllerCount: 3
  ComputeCount: 0
  OvercloudControllerFlavor: baremetal
```

```
OvercloudComputeFlavor: baremetal
ControllerSchedulerHints:
  'capabilities:node': '0-controller-%index%'
GlanceBackend: swift
```

- **ControllerCount: 3** specifies that three nodes will be deployed. These will use swift for glance, lvm for cinder, and host the control-plane services for edge compute nodes.
- **ComputeCount: 0** is an optional parameter to prevent Compute nodes from being deployed with the central Controller nodes.
- **GlanceBackend: swift** uses Object Storage (swift) as the Image Service (glance) back end.
  The resulting configuration interacts with the distributed compute nodes (DCNs) in the following ways:
- The Image service on the DCN creates a cached copy of the image it receives from the central Object Storage back end. The Image service uses HTTP to copy the image from Object Storage to the local disk cache.



NOTE

The central Controller node must be able to connect to the distributed compute node (DCN) site. The central Controller node can use a routed layer 3 connection.

3. Generate roles for the central location using roles appropriate for your environment:

```
openstack overcloud roles generate Controller \
-o ~/central/control_plane_roles.yaml
```

4. Generate an environment file ~/**central**/**central-images-env.yaml**:

```
sudo openstack tripleo container image prepare \
-e containers.yaml \
--output-env-file ~/central/central-images-env.yaml
```

5. Configure the naming conventions for your site in the **site-name.yaml** environment file. The Nova availability zone, Cinder storage availability zone must match:

```
cat > /home/stack/central/site-name.yaml << EOF
parameter_defaults:
   NovaComputeAvailabilityZone: central
   ControllerExtraConfig:
      nova::availability_zone::default_schedule_zone: central
   NovaCrossAZAttach: false
   CinderStorageAvailabilityZone: central
EOF
```

6. Deploy the central Controller node. For example, you can use a **deploy.sh** file with the following contents:

```
#!/bin/bash
```

```
source ~/stackrc
time openstack overcloud deploy \
--stack central \
--templates /usr/share/openstack-tripleo-heat-templates/ \
-e /usr/share/openstack-tripleo-heat-templates/environments/nova-az-config.yaml \
-e ~/central/containers-env-file.yaml \
-e ~/central/overrides.yaml \
-e ~/central/site-name.yaml
```

NOTE

You must include heat templates for the configuration of networking in your **openstack overcloud deploy** command. Designing for edge architecture requires spine and leaf networking. See Spine Leaf Networking for more details.

5.2. DEPLOYING THE CENTRAL SITE WITH STORAGE

To deploy the Image service with multiple stores and Ceph Storage as the back end, complete the following steps:

Prerequisites

- Hardware for a Ceph cluster at the hub and in each availability zone, or in each geographic location where storage services are required.
- You must deploy edge sites in a hyper converged architecture.
- Hardware for three Image Service servers at the hub and in each availability zone, or in each geographic location where storage services are required.

The following is an example deployment of two or more stacks:

- One stack at the central location called **central**
- One stack at an edge site called **dcn0**.
- Additional stacks deployed similarly to **dcn0**, such as **dcn1**, **dcn2**, and so on.

Procedure

The following procedure outlines the steps for the initial deployment of the central location.

NOTE

The following steps detail the deployment commands and environment files associated with an example DCN deployment that uses the Image service with multiple stores. These steps do not include unrelated, but necessary, aspects of configuration, such as networking.

1. In the home directory, create directories for each stack that you plan to deploy.

```
mkdir /home/stack/central
mkdir /home/stack/dcn0
mkdir /home/stack/dcn1
```

2. Set the name of the Ceph cluster, as well as configuration parameters relative to the available hardware. For more information, see Configuring Ceph with Custom Config Settings:

```
cat > /home/stack/central/ceph.yaml << EOF
parameter_defaults:
  CephClusterName: central
  CephAnsibleDisksConfig:
    osd_scenario: lvm
    osd_objectstore: bluestore
    devices:
      - /dev/sda
      - /dev/sdb
  CephPoolDefaultSize: 3
  CephPoolDefaultPgNum: 128

EOF
```

3. Generate roles for the central location using roles appropriate for your environment:

```
openstack overcloud roles generate Compute Controller CephStorage \
-o ~/central/central_roles.yaml

cat > /home/stack/central/role-counts.yaml << EOF
parameter_defaults:
  ControllerCount: 3
  ComputeCount: 2
  CephStorage: 3
EOF
```

4. Generate an environment file ~/**central**/**central-images-env.yaml**

```
sudo openstack tripleo container image prepare \
-e containers.yaml \
--output-env-file ~/central/central-images-env.yaml
```

5. Configure the naming conventions for your site in the **site-name.yaml** environment file. The Nova availability zone and the Cinder storage availability zone must match:

```
cat > /home/stack/central/site-name.yaml << EOF
parameter_defaults:
    NovaComputeAvailabilityZone: central
    ControllerExtraConfig:
        nova::availability_zone::default_schedule_zone: central
    NovaCrossAZAttach: false
    CinderStorageAvailabilityZone: central
    GlanceBackendID: central
EOF
```

6. Configure a **glance.yaml** template with contents similar to the following:

```
parameter_defaults:
    GlanceEnabledImportMethods: web-download,copy-image
    GlanceBackend: rbd
```

```
    GlanceStoreDescription: 'central rbd glance store'
    GlanceBackendID: central
    CephClusterName: central
```

7. After you prepare all of the other templates, deploy the **central** stack:

```
openstack overcloud deploy \
    --stack central \
    --templates /usr/share/openstack-tripleo-heat-templates/ \
    -r ~/central/central_roles.yaml \
  ...
    -e /usr/share/openstack-tripleo-heat-templates/environments/ceph-ansible/ceph-ansible.yaml \
    -e /usr/share/openstack-tripleo-heat-templates/environments/nova-az-config.yaml \
    -e ~/central/central-images-env.yaml \
    -e ~/central/role-counts.yaml \
    -e ~/central/site-name.yaml \
    -e ~/central/ceph.yaml \
    -e ~/central/glance.yaml
```

NOTE

You must include heat templates for the configuration of networking in your **openstack overcloud deploy** command. Designing for edge architecture requires spine and leaf networking. See Spine Leaf Networking for more details.

The **ceph-ansible.yaml** file is configured with the following parameters:

- NovaEnableRbdBackend: true
- GlanceBackend: rbd

When you use these settings together, the glance.conf parameter **image_import_plugins** is configured by heat to have a value **image_conversion**, automating the conversion of QCOW2 images with commands such as **glance image-create-via-import --disk-format qcow2**.

This is optimal for the Ceph RBD. If you want to disable image conversion, use the **GlanceImageImportPlugin** parameter:

```
parameter_defaults:
  GlanceImageImportPlugin: []
```

5.3. INTEGRATING EXTERNAL CEPH

You can deploy the central location of a distributed compute node (DCN) architecture and integrate a pre-deployed Red Hat Ceph Storage solution.

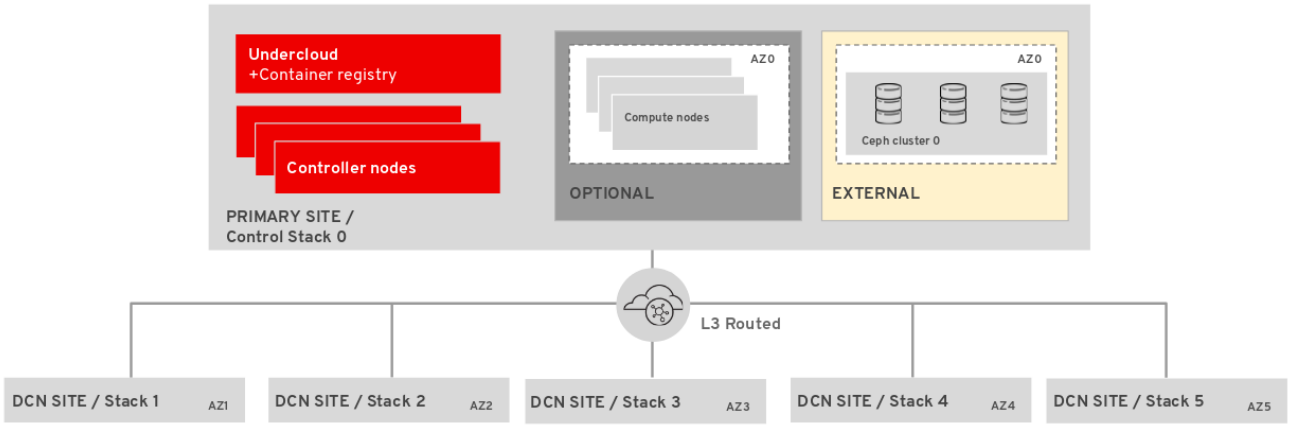
Prerequisites

- Hardware for a Ceph cluster at the central location and in each availability zone, or in each geographic location where storage services are required.
- You must deploy edge sites in a hyper converged architecture.

- Hardware for three Image Service servers at the central location and in each availability zone, or in each geographic location where storage services are required.

The following is an example deployment of two or more stacks:

- One stack at the central location called **central**
- One stack at an edge site called **dcn0**.
- Additional stacks deployed similarly to **dcn0**, such as **dcn1**, **dcn2**, and so on.



You can install the central location so that it is integrated with a pre-existing Red Hat Ceph Storage solution by following the process documented in Integrating an Overcloud with an Existing Red Hat Ceph Cluster. There are no special requirements for integrating Red Hat Ceph Storage with the central site of a DCN deployment, however you must still complete DCN specific steps before deploying the overcloud:

1. In the home directory, create directories for each stack that you plan to deploy. Use this to separate templates designed for their respective sites.

```
mkdir /home/stack/central
mkdir /home/stack/dcn0
mkdir /home/stack/dcn1
```

2. Generate roles for the central location using roles that Red Hat OpenStack Platform director manages. When integrating with external Ceph, do not use Ceph roles:

```
cat > /home/stack/central/role-counts.yaml << EOF
parameter_defaults:
  ControllerCount: 3
  ComputeCount: 2
EOF
```

3. Generate an environment file ~/central/central-images-env.yaml

```
sudo openstack tripleo container image prepare \
-e containers.yaml \
--output-env-file ~/central/central-images-env.yaml
```

4. Configure the naming conventions for your site in the site-name.yaml environment file. The Nova availability zone and the Cinder storage availability zone must match:

```
cat > /home/stack/central/site-name.yaml << EOF
parameter_defaults:
  NovaComputeAvailabilityZone: central
  ControllerExtraConfig:
    nova::availability_zone::default_schedule_zone: central
  NovaCrossAZAttach: false
  CinderStorageAvailabilityZone: central
  GlanceBackendID: central
EOF
```

5. Configure a glance.yaml template with contents similar to the following:

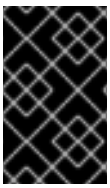
```
parameter_defaults:
  GlanceEnabledImportMethods: web-download,copy-image
  GlanceBackend: rbd
  GlanceStoreDescription: 'central rbd glance store'
  GlanceBackendID: central
  CephClusterName: central
```

6. When Ceph is deployed without Red Hat OpenStack Platform director, do not use the **ceph-ansible.yaml** environment file. Use the **ceph-ansible-external.yaml** environment file instead.

```
openstack overcloud deploy \
    --stack central \
    --templates /usr/share/openstack-tripleo-heat-templates/ \
    -r ~/central/central_roles.yaml \
  ...
    -e /usr/share/openstack-tripleo-heat-templates/environments/ceph-ansible/ceph-ansible-external.yaml \
    -e /usr/share/openstack-tripleo-heat-templates/environments/nova-az-config.yaml \
    -e ~/central/central-images-env.yaml \
    -e ~/central/role-counts.yaml \
    -e ~/central/site-name.yaml \
    -e ~/central/ceph.yaml \
    -e ~/central/glance.yaml
```

CHAPTER 6. DEPLOY THE EDGE WITHOUT STORAGE

You can deploy a distributed compute node cluster without Block storage at edge sites if you use the Object Storage service (swift) as a back end for the Image service (glance) at the central location. A site deployed without block storage cannot be updated later to have block storage due to the differing role and networking profiles for each architecture.



IMPORTANT

The following procedure uses lvm as the back end for the Block Storage service (cinder), which is not supported for production. You must deploy a certified block storage solution as a back end for the Block Storage service.

6.1. DEPLOYING EDGE NODES WITHOUT STORAGE

You can deploy edge compute nodes that will use the central location as the control plane. This procedure shows how to add a new DCN stack to your deployment and reuse the configuration from the existing heat stack to create new environment files. The first heat stack deploys an overcloud within a centralized datacenter. Create additional heat stacks to deploy Compute nodes to a remote location.

6.1.1. Configuring the distributed compute node environment files

This procedure creates a new **central-export.yaml** environment file and uses the passwords in the **plan-environment.yaml** file from the overcloud. The **central-export.yaml** file contains sensitive security data. To improve security, you can remove the file when you no longer require it.

When you specify the directory for the **--config-download-dir** option, use the central hub Ansible configuration that director creates in **/var/lib/mistral** during deployment. Do not use Ansible configuration that you manually generate with the **openstack overcloud config download** command. The manually generated configuration lacks certain files that are created only during a deployment operation.

You must upload images to the central location before copying them to edge sites; a copy of each image must exist in the Image service (glance) at the central location.

You must use the RBD storage driver for the Image, Compute, and Block Storage services.

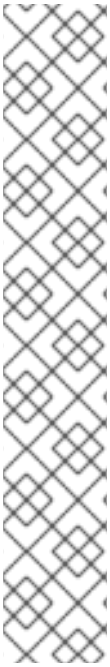
Procedure

1. Generate the configuration files that the DCN sites require:

```
openstack overcloud export \
--config-download-dir /var/lib/mistral/central \
--stack central --output-file ~/dcn-common/central-export.yaml
```

2. Generate roles for the edge location using roles appropriate for your environment:

```
openstack overcloud roles generate Compute -o ~/dcn0/dcn0_roles.yaml
```



NOTE

If you are using ML2/OVS for networking overlay, you must edit the roles file that you created to include the **NeutronDhcpAgent** and **NeutronMetadataAgent** roles:

```
...
   - OS::TripleO::Services::MySQLClient
   - OS::TripleO::Services::NeutronBgpVpnBagpipe
+   - OS::TripleO::Services::NeutronDhcpAgent
+   - OS::TripleO::Services::NeutronMetadataAgent
   - OS::TripleO::Services::NeutronLinuxbridgeAgent
   - OS::TripleO::Services::NeutronVppAgent
   - OS::TripleO::Services::NovaAZConfig
   - OS::TripleO::Services::NovaCompute

...
```

For more information, see Preparing for a routed provider network .

6.1.2. Deploying the Compute nodes to the DCN site

This procedure uses the **Compute** role to deploy Compute nodes to an availability zone (AZ) named **dcn0**. In a distributed compute node (DC) context, this role is used for sites without storage.

Procedure

1. Review the overrides for the distributed compute (DCN) site in dcn0/overrides.yaml

```
parameter_defaults:
  ComputeCount: 3
  ComputeFlavor: baremetal
  ComputeSchedulerHints:
    'capabilities:node': '0-compute-%index%'
  NovaAZAttach: false
```

2. Create a new file called **site-name.yaml** in the ~/**dcn0** directory with the following contents:

```
resource_registry:
  OS::TripleO::Services::NovaAZConfig: /usr/share/openstack-tripleo-heat-
templates/deployment/nova/nova-az-config.yaml
parameter_defaults:
  NovaComputeAvailabilityZone: dcn0
  RootStackName: dcn0
```

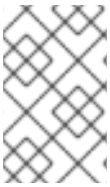
3. Retrieve the container images for the DCN Site:

```
sudo openstack tripleo container image prepare \
--environment-directory dcn0 \
-r ~/dcn0/roles_data.yaml \
-e ~/dcn-common/central-export.yaml \
-e ~/containers-prepare-parameter.yaml \
--output-env-file ~/dcn0/dcn0-images-env.yaml
```

4. Run the deploy.sh deployment script for dcn0:

```
#!/bin/bash
STACK=dcn0
source ~/stackrc
time openstack overcloud deploy \
    --stack $STACK \
    --templates /usr/share/openstack-tripleo-heat-templates/ \
    -e /usr/share/openstack-tripleo-heat-templates/environments/nova-az-config.yaml \
    -e ~/dcn-common/central-export.yaml \
    -e ~/dcn0/dcn0-images-env.yaml \
    -e ~/dcn0/site-name.yaml \
    -e ~/dcn0/overrides.yaml
```

If you deploy additional edge sites that require edits to the **network_data.yaml** file, you must execute a stack update at the central location.

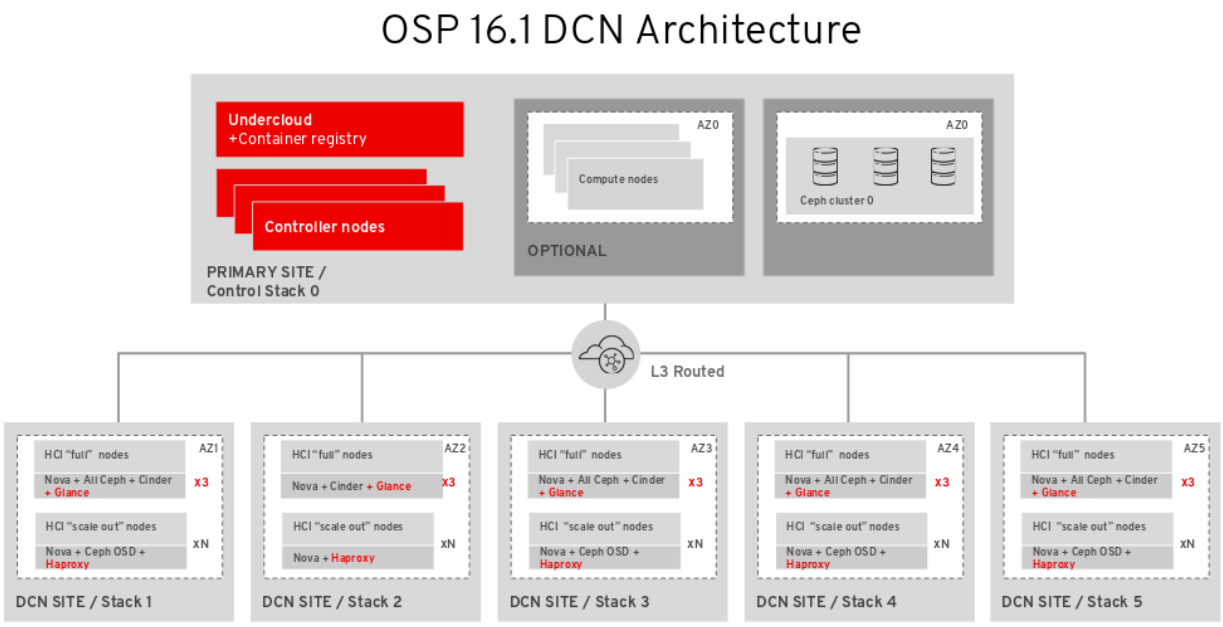


NOTE

You must include heat templates for the configuration of networking in your **openstack overcloud deploy** command. Designing for edge architecture requires spine and leaf networking. See Spine Leaf Networking for more details.

CHAPTER 7. DEPLOYING STORAGE AT THE EDGE

You can leverage Red Hat OpenStack Platform director to extend distributed compute node deployments to include distributed image management and persistent storage at the edge with the benefits of using Red Hat OpenStack Platform and Ceph Storage.



7.1. DEPLOYING EDGE SITES WITH STORAGE

After you deploy the central site, build out the edge sites and ensure that each edge location connects primarily to its own storage backend, as well as to the storage back end at the central location.

A spine and leaf networking configuration should be included with this configuration, with the addition of the **storage** and **storage_mgmt** networks that ceph needs. For more information see Spine leaf networking.

You must have connectivity between the storage network at the central location and the storage network at each edge site so that you can move glance images between sites.

Ensure that the central location can communicate with the **mons** and **osds** at each of the edge sites. However, you should terminate the storage management network at site location boundaries, because the storage management network is used for OSD rebalancing.

Procedure

1. Export stack information from the **central** stack. You must deploy the **central** stack before running this command:

```
openstack overcloud export \
    --config-download-dir /var/lib/mistral/central/ \
    --stack central \
    --output-file ~/dcn-common/central-export.yaml
```

NOTE

The **config-download-dir** value defaults to **/var/lib/mistral/<stack>/**.

2. Create the **central_ceph_external.yaml** file. This environment file connects DCN sites to the central hub Ceph cluster, so the information is specific to the Ceph cluster deployed in the previous steps.

```
sudo -E openstack overcloud export ceph \
--stack central \
--config-download-dir /var/lib/mistral \
--output-file ~/dcn-common/central_ceph_external.yaml
```

When Ceph is deployed without Red Hat OpenStack Platform director, you cannot run the **openstack overcloud export ceph** command. Manually create the **central_ceph_external.yaml** file:

```
parameter_defaults:
  CephExternalMultiConfig:
    - cluster: "central"
      fsid: "3161a3b4-e5ff-42a0-9f53-860403b29a33"
      external_cluster_mon_ips: "172.16.11.84, 172.16.11.87, 172.16.11.92"
      keys:
        - name: "client.openstack"
          caps:
            mgr: "allow *"
            mon: "profile rbd"
            osd: "profile rbd pool=vms, profile rbd pool=volumes, profile rbd pool=images"
          key: "AQD29WteAAAAABAAphgOjFD7nyjdYe8Lz0mQ5Q=="
          mode: "0600"
      dashboard_enabled: false
      ceph_conf_overrides:
        client:
          keyring: /etc/ceph/central.client.openstack.keyring
```

- The **fsid** parameter is the file system ID of your Ceph Storage cluster: This value is specified in the cluster configuration file in the **[global]** section:

```
[global]
fsid = 4b5c8c0a-ff60-454b-a1b4-9747aa737d19
...
```

- The **key** parameter is the ceph client key for the openstack account:

```
[root@ceph ~]# ceph auth list
...
[client.openstack]
    key = AQC+vYNXgDAgAhAAc8UoYt+OTz5uhV7ItLdwUw==
    caps mgr = "allow *"
    caps mon = "profile rbd"
    caps osd = "profile rbd pool=volumes, profile rbd pool=vms, profile rbd pool=images,
profile rbd pool=backups, profile rbd pool=metrics"
...
```

For more information on the parameters shown in the sample **central_ceph_external.yaml** file, see Creating a custom environment file .

3. Create the **~/dcn0/glance.yaml** file for Image service configuration overrides:

```
parameter_defaults:
  GlanceEnabledImportMethods: web-download,copy-image
  GlanceBackend: rbd
  GlanceStoreDescription: 'dcn0 rbd glance store'
  GlanceBackendID: dcn0
  GlanceMultistoreConfig:
    central:
      GlanceBackend: rbd
      GlanceStoreDescription: 'central rbd glance store'
      CephClientUserName: 'openstack'
      CephClusterName: central
```

4. Configure the **ceph.yaml** file with configuration parameters relative to the available hardware.

```
cat > /home/stack/dcn0/ceph.yaml << EOF
parameter_defaults:
  CephClusterName: dcn0
  CephAnsibleDisksConfig:
    osd_scenario: lvm
    osd_objectstore: bluestore
    devices:
      - /dev/sda
      - /dev/sdb
  CephPoolDefaultSize: 3
  CephPoolDefaultPgNum: 128
EOF
```

For more information, see Mapping the Ceph Storage node disk layout .

5. Implement system tuning by using a file that contains the following parameters tuned to the requirements of you environment:

```
cat > /home/stack/dcn0/tuning.yaml << EOF
parameter_defaults:
  CephAnsibleExtraConfig:
    is_hci: true
  CephConfigOverrides:
    osd_recovery_op_priority: 3
    osd_recovery_max_active: 3
    osd_max_backfills: 1
 ## Set relative to your hardware:
  # DistributedComputeHCIParameters:
  #   NovaReservedHostMemory: 181000
  # DistributedComputeHCIExtraConfig:
  #   nova::cpu_allocation_ratio: 8.2
EOF
```

- For more information about setting the values for the parameters **CephAnsibleExtraConfig**, see Setting ceph-ansible group variables.

- For more information about setting the values for the parameters **CephConfigOverrides**, see Customizing the Ceph Storage cluster .

6. Configure the naming conventions for your site in the **site-name.yaml** environment file. The Nova availability zone and the Cinder storage availability zone must match. The **CinderVolumeCluster** parameter is included when deploying an edge site with storage. This parameter is used when cinder-volume is deployed as active/active, which is required at edge sites. As a best practice, set the Cinder cluster name to match the availability zone:

```
cat > /home/stack/central/site-name.yaml << EOF
parameter_defaults:
  ...
  NovaComputeAvailabilityZone: dcn0
  NovaCrossAZAttach: false
  CinderStorageAvailabilityZone: dcn0
  CinderVolumeCluster: dcn0
```

7. Generate the **roles.yaml** file to be used for the dcn0 deployment, for example:

```
openstack overcloud roles generate DistributedComputeHCI
DistributedComputeHCIScaleOut -o ~/dcn0/roles_data.yaml
```

8. Set the number systems in each role by creating the **~/dcn0/roles-counts.yaml** file with the desired values for each role.
   When using hyperconverged infrastructure (HCI), you must allocate three nodes to the DistributedComputeHCICount role to satisfy requirements for Ceph Mon and **GlanceApiEdge** services.

```
parameter_defaults:
  ControllerCount: 0
  ComputeCount: 0
  DistributedComputeHCICount: 3
  DistributedComputeHCIScaleOutCount: 1  # Optional
  DistributedComputeScaleOutCount: 1     # Optional
```

9. Retrieve the container images for the edge site:

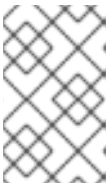
```
sudo openstack tripleo container image prepare \
--environment-directory dcn0 \
-r ~/dcn0/roles_data.yaml \
-e /usr/share/openstack-tripleo-heat-templates/environments/ceph-ansible/ceph-ansible.yaml
\
...
-e /home/stack/dcn-common/central-export.yaml \
-e /home/stack/containers-prepare-parameter.yaml \
--output-env-file ~/dcn0/dcn0-images-env.yaml
```

NOTE

You must include all environment files to be used for the deployment in the **openstack tripleo container image prepare** command.

10. Deploy the edge site:

```
openstack overcloud deploy \
  --stack dcn0 \
  --templates /usr/share/openstack-tripleo-heat-templates/ \
  -r ~/dcn0/roles_data.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/ceph-ansible/ceph-
ansible.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/dcn-hci.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/nova-az-config.yaml \
  -e ~/dnc0/dcn0-images-env.yaml \
  ....
  -e ~/dcn-common/central-export.yaml \
  -e ~/dcn-common/central_ceph_external.yaml \
  -e ~/dcn0/dcn_ceph_keys.yaml \
  -e ~/dcn0/role-counts.yaml \
  -e ~/dcn0/ceph.yaml \
  -e ~/dcn0/site-name.yaml \
  -e ~/dcn0/tuning.yaml \
  -e ~/dcn0/glance.yaml
```



NOTE

You must include heat templates for the configuration of networking in your **openstack overcloud deploy** command. Designing for edge architecture requires spine and leaf networking. See Spine Leaf Networking for more details.

7.2. CREATING ADDITIONAL DISTRIBUTED COMPUTE NODE SITES

A new distributed compute node (DCN) site has its own directory of YAML files on the undercloud. For more information, see Section 4.7, “Managing separate heat stacks” . This procedure contains example commands.

Procedure

1. As the stack user on the undercloud, create a new directory for **dcn9**:

```
$ cd ~
$ mkdir dcn9
```

2. Copy the existing **dcn0** templates to the new directory and replace the **dcn0** strings with **dcn9**:

```
$ cp dcn0/ceph.yaml dcn9/ceph.yaml
$ sed s/dcn0/dcn9/g -i dcn9/ceph.yaml
$ cp dcn0/overrides.yaml dcn9/overrides.yaml
$ sed s/dcn0/dcn9/g -i dcn9/overrides.yaml
$ sed s/"0-ceph-%index%"/"9-ceph-%index%"/g -i dcn9/overrides.yaml
$ cp dcn0/deploy.sh dcn9/deploy.sh
$ sed s/dcn0/dcn9/g -i dcn9/deploy.sh
```

3. Review the files in the **dcn9** directory to confirm that they suit your requirements.

4. Edit undercloud.conf to add a new leaf. In the following example, leaf9 is added to undercloud.conf:

```
[leaf0]
```

```
cidr = 192.168.10.0/24
dhcp_start = 192.168.10.10
dhcp_end = 192.168.10.90
inspection_iprange = 192.168.10.100,192.168.10.190
gateway = 192.168.10.1
masquerade = False

…
[leaf9]
cidr = 192.168.19.0/24
dhcp_start = 192.168.19.10
dhcp_end = 192.168.19.90
inspection_iprange = 192.168.19.100,192.168.19.190
gateway = 192.168.10.1
masquerade = False
```

5. Rerun the *openstack undercloud install* command to update the environment configuration.

6. In your overcloud templates, update the value of the **NetworkDeploymentActions** parameter from a value of **["CREATE"]**, to a value of **["CREATE", "UPDATE"]**. If this parameter is not currently included in your templates, add it to one of your environment files, or create a new environment file.

```
cat > /home/stack/central/network-environment.yaml << EOF
parameter_defaults:
  NetworkDeploymentActions: ["CREATE", "UPDATE"]
EOF
```

7. Run the deploy script for the central location. Include all templates that you used when you first deployed the central location, as well as the newly created or edited network-environment.yaml file:

```
openstack overcloud deploy \
    --stack central \
    --templates /usr/share/openstack-tripleo-heat-templates/ \
    -r ~/central/roles_data.yaml \
    -e /usr/share/openstack-tripleo-heat-templates/environments/ceph-ansible/ceph-
ansible.yaml \
    -e /usr/share/openstack-tripleo-heat-templates/environments/dcn-hci.yaml \
    -e /usr/share/openstack-tripleo-heat-templates/environments/nova-az-config.yaml \
    -e ~/central/dcn9-images-env.yaml \
    ....
    -e ~/dcn-common/central-export.yaml \
    -e ~/dcn-common/central_ceph_external.yaml \
    -e ~/central/dcn_ceph_keys.yaml \
    -e ~/central/role-counts.yaml \
    -e ~/central/ceph.yaml \
    -e ~/central/site-name.yaml \
    -e ~/central/tuning.yaml \
    -e ~/central/glance.yaml
```

8. Verify that your nodes are available and in **Provisioning state**:

```
$ openstack baremetal node list
```

9. When your nodes are available, deploy the new edge site with all appropriate templates:

```
openstack overcloud deploy \
  --stack dcn9 \
  --templates /usr/share/openstack-tripleo-heat-templates/ \
  -r ~/dcn9/roles_data.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/ceph-ansible/ceph-
ansible.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/dcn-hci.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/nova-az-config.yaml \
  -e ~/dnc9/dcn9-images-env.yaml \
  ....
  -e ~/dcn-common/central-export.yaml \
  -e ~/dcn-common/central_ceph_external.yaml \
  -e ~/dcn9/dcn_ceph_keys.yaml \
  -e ~/dcn9/role-counts.yaml \
  -e ~/dcn9/ceph.yaml \
  -e ~/dcn9/site-name.yaml \
  -e ~/dcn9/tuning.yaml \
  -e ~/dcn9/glance.yaml
```

10. If you've deployed the locations with direct edge-to-edge communication, you must redeploy each edge site to update routes and establish communication with the new location.

7.3. UPDATING THE CENTRAL LOCATION

After you configure and deploy all of the edge sites using the sample procedure, update the configuration at the central location so that the central Image service can push images to the edge sites.

WARNING

This procedure restarts the Image service (glance) and interrupts any long running Image service process. For example, if an image is being copied from the central Image service server to a DCN Image service server, that image copy is interrupted and you must restart it. For more information, see Clearing residual data after interrupted Image service processes.

Procedure

1. Create a **~/central/glance_update.yaml** file similar to the following. This example includes a configuration for two edge sites, dcn0 and dcn1:

```
parameter_defaults:
  GlanceEnabledImportMethods: web-download,copy-image
  GlanceBackend: rbd
  GlanceStoreDescription: 'central rbd glance store'
  CephClusterName: central
  GlanceBackendID: central
  GlanceMultistoreConfig:
```

```
    dcn0:
      GlanceBackend: rbd
      GlanceStoreDescription: 'dcn0 rbd glance store'
      CephClientUserName: 'openstack'
      CephClusterName: dcn0
      GlanceBackendID: dcn0
    dcn1:
      GlanceBackend: rbd
      GlanceStoreDescription: 'dcn1 rbd glance store'
      CephClientUserName: 'openstack'
      CephClusterName: dcn1
      GlanceBackendID: dcn1
```

2. Create the **dcn_ceph.yaml** file. In the following example, this file configures the glance service at the central site as a client of the Ceph clusters of the edge sites, **dcn0** and **dcn1**.

```
sudo -E openstack overcloud export ceph \
--stack dcn0,dcn1 \
--config-download-dir /var/lib/mistral \
--output-file ~/central/dcn_ceph.yaml
```

3. Redeploy the central site using the original templates and include the newly created **dcn_ceph.yaml** and **glance_update.yaml** files.

```
openstack overcloud deploy \
      --stack central \
      --templates /usr/share/openstack-tripleo-heat-templates/ \
      -r ~/central/central_roles.yaml \
   ...
      -e /usr/share/openstack-tripleo-heat-templates/environments/ceph-ansible/ceph-
ansible.yaml \
      -e /usr/share/openstack-tripleo-heat-templates/environments/nova-az-config.yaml \
      -e ~/central/central-images-env.yaml \
      -e ~/central/role-counts.yaml \
      -e ~/central/site-name.yaml
      -e ~/central/ceph.yaml \
      -e ~/central/ceph_keys.yaml \
      -e ~/central/glance.yaml \
      -e ~/central/dcn_ceph_external.yaml
```

4. On a controller at the central location, restart the **cinder-volume** service. If you deployed the central location with the **cinder-backup** service, then restart the **cinder-backup** service too:

```
ssh heat-admin@controller-0 sudo pcs resource restart openstack-cinder-volume
ssh heat-admin@controller-0 sudo pcs resource restart openstack-cinder-backup
```

7.3.1. Clearing residual data after interrupted Image service processes

When you restart the central location, any long-running Image service (glance) processes are interrupted. Before you can restart these processes, you must first clean up residual data on the Controller node that you rebooted, and in the Ceph and Image service databases.

Procedure

1. Check and clear residual data in the Controller node that was rebooted. Compare the files in the **glance-api.conf** file for staging store with the corresponding images in the Image service database, for example **<image_ID>.raw**.
   - If these corresponding images show importing status, you must recreate the image.
   - If the images show active status, you must delete the data from staging and restart the copy import.
2. Check and clear residual data in Ceph stores. The images that you cleaned from the staging area must have matching records in their **stores** property in the Ceph stores that contain the image. The image name in Ceph is the image id in the Image service database.
3. Clear the Image service database. Clear any images that are in importing status from the import jobs there were interrupted:

```
$ glance image-delete <image_id>
```

7.4. DEPLOYING RED HAT CEPH STORAGE DASHBOARD ON DCN

Procedure

To deploy the Red Hat Ceph Storage Dashboard to the central location, see Adding the Red Hat Ceph Storage Dashboard to an overcloud deployment. These steps should be completed prior to deploying the central location.

To deploy Red Hat Ceph Storage Dashboard to edge locations, complete the same steps that you completed for central, however you must complete the following following:

- Ensure that the **ManageNetworks** parameter has a value of **false** in your templates for deploying the edge site. When you set **ManageNetworks** to **false**, Edge sites will use the existing networks that were already created in the central stack:

```
parameter_defaults:
  ManageNetworks: false
```

- You must deploy your own solution for load balancing in order to create a high availability virtual IP. Edge sites do not deploy haproxy, nor pacemaker. When you deploy Red Hat Ceph Storage Dashboard to edge locations, the deployment is exposed on the storage network. The dashboard is installed on each of the three DistributedComputeHCI nodes with distinct IP addresses without a load balancing solution.

7.4.1. Creating a composable network for a Virtual IP

You can create an additional network to host virtual IP where the Ceph dashboard can be exposed. You must not be reusing network resources for multiple stacks. For more information on reusing network resources, see Reusing network resources in multiple stacks .

To create this additional network resource, use the provided **network_data_dashboard.yaml** heat template. The name of the created network is **StorageDashboard**.

Procedure

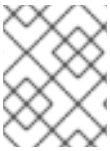
1. Log in to Red Hat OpenStack Platform Director as **stack**.

2. Generate the **DistributedComputeHCIDashboard** role and any other roles appropriate for your environment:

```
openstack overcloud roles generate DistributedComputeHCIDashboard -o ~/dnc0/roles.yaml
```

3. Include the **roles.yaml** and the **network_data_dashboard.yaml** in the overcloud deploy command:

```
$ openstack overcloud deploy --templates \
-r ~/<dcn>/<dcn_site_roles>.yaml \
-n /usr/share/openstack-tripleo-heat-templates/network_data_dashboard.yaml \
-e <overcloud_environment_files> \
...
-e /usr/share/openstack-tripleo-heat-templates/environments/ceph-ansible/ceph-ansible.yaml
\
-e /usr/share/openstack-tripleo-heat-templates/environments/ceph-ansible/ceph-
dashboard.yaml
```



NOTE

The deployment provides the three ip addresses where the dashboard is enabled on the storage network.

Verification

To confirm the dashboard is operational at the central location and that the data it displays from the Ceph cluster is correct, see Accessing Ceph Dashboard.

You can confirm that the dashboard is operating at an edge location through similar steps, however, there are exceptions as there is no load balancer at edge locations.

1. Retrieve dashboard admin login credentials specific to the selected stack from **/var/lib/mistral/<stackname>/ceph-ansible/group_vars/all.yml**

2. Within the inventory specific to the selected stack, **/var/lib/mistral/<stackname>/ceph-ansible/inventory.yml**, locate the DistributedComputeHCI role hosts list and save all three of the **storage_ip** values. In the example below the first two dashboard IPs are 172.16.11.84 and 172.16.11.87:

```
DistributedComputeHCI:
  hosts:
    dcn1-distributed-compute-hci-0:
      ansible_host: 192.168.24.16
...
storage_hostname: dcn1-distributed-compute-hci-0.storage.localdomain
storage_ip: 172.16.11.84
...
    dcn1-distributed-compute-hci-1:
ansible_host: 192.168.24.22
...
storage_hostname: dcn1-distributed-compute-hci-1.storage.localdomain
storage_ip: 172.16.11.87
```

3. You can check that the Ceph Dashboard is active at one of these IP addresses if they are accessible to you. These IP addresses are on the storage network and are not routed. If these IP

addresses are not available, you must configure a load balancer for the three IP addresses that you get from the inventory to obtain a virtual IP address for verification.

CHAPTER 8. DEPLOYING WITH KEY MANAGER

If you have deployed edge sites previous to the release of Red Hat OpenStack Platform 16.1.2, you will need to regenerate roles.yaml to implement this feature: To implement the feature, regenerate the **roles.yaml** file used for the DCN site's deployment.

```
$ openstack overcloud roles generate DistributedComputeHCI DistributedComputeHCIScaleOut -o
~/dcn0/roles_data.yaml
```

8.1. DEPLOYING EDGE SITES WITH KEY MANAGER

If you want to include access to the Key Manager (barbican) service at edge sites, you must configure barbican at the central location. For information on installing and configuring barbican, see Deploying Barbican.

- You can configure access to barbican from DCN sites by including the **/usr/share/openstack-tripleo-heat-templates/environments/services/barbican-edge.yaml**.

```
openstack overcloud deploy \
  --stack dcn0 \
  --templates /usr/share/openstack-tripleo-heat-templates/ \
  -r ~/dcn0/roles_data.yaml \
  ....
  -e /usr/share/openstack-tripleo-heat-templates/environments/services/barbican-edge.yaml
```

CHAPTER 9. PRECACHING GLANCE IMAGES INTO NOVA

When you configure OpenStack Compute to use local ephemeral storage, glance images are cached to quicken the deployment of instances. If an image that is necessary for an instance is not already cached, it is downloaded to the local disk of the Compute node when you create the instance.

The process of downloading a glance image takes a variable amount of time, depending on the image size and network characteristics such as bandwidth and latency.

If you attempt to start an instance, and the image is not available on the on the Ceph cluster that is local, launching an instance will fail with the following message:

> Build of instance 3c04e982-c1d1-4364-b6bd-f876e399325b aborted: Image 20c5ff9d-5f54-4b74-830f-88e78b9999ed is unacceptable: No image locations are accessible

You see the following in the Compute service log:

> 'Image %s is not on my ceph and [workarounds]/ never_download_image_if_on_rbd=True; refusing to fetch and upload.',

The instance fails to start due to a parameter in the **nova.conf** configuration file called **never_download_image_if_on_rbd**, which is set to **true** by default for DCN deployments. You can control this value using the heat parameter **NovaDisableImageDownloadToRbd** which you can find in the **dcn-hci.yaml** file.

If you set the value of **NovaDisableImageDownloadToRbd** to **false** prior to deploying the overcloud, the following occurs:

- The Compute service (nova) will automatically stream images available at the **central** location if they are not available locally.
- You will not be using a COW copy from glance images.
- The Compute (nova) storage will potentially contain multiple copies of the same image, depending on the number of instances using it.
- You may saturate both the WAN link to the **central** location as well as the nova storage pool.

Red Hat recommends leaving this value set to true, and ensuring required images are available locally prior to launching an instance. For more information on making images available to the edge, see Section A.1.3, “Copying an image to a new site” .

For images that are local, you can speed up the creation of VMs by using the **tripleo_nova_image_cache.yml** ansible playbook to pre-cache commonly used images or images that are likely to be deployed in the near future.

9.1. RUNNING THE TRIPLEO_NOVA_IMAGE_CACHE.YML ANSIBLE PLAYBOOK

Prerequisites

- Authentication credentials to the correct API in the shell environment.

Before the command provided in each step, you must ensure that the correct authentication file is sourced.

Procedure

1. Create an ansible inventory file for the stack. You can specify multiple stacks in a comma delimited list to cache images at more than one site:

```
$ source stackrc

$ tripleo-ansible-inventory --plan central,dcn0,dcn1 \
--static-yaml-inventory inventory.yaml
```

2. Create a list of image IDs that you want to pre-cache:

   a. Retrieve a comprehensive list of available images:

```
$ source centralrc

$ openstack image list
+--------------------------------------+---------+--------+
| ID                                   | Name    | Status |
+--------------------------------------+---------+--------+
| 07bc2424-753b-4f65-9da5-5a99d8383fe6 | image_0 | active |
| d5187afa-c821-4f22-aa4b-4e76382bef86 | image_1 | active |
+--------------------------------------+---------+--------+
```

   b. Create an ansible playbook argument file called **nova_cache_args.yml**, and add the IDs of the images that you want to pre-cache:

```
---
tripleo_nova_image_cache_images:
  - id: 07bc2424-753b-4f65-9da5-5a99d8383fe6
  - id: d5187afa-c821-4f22-aa4b-4e76382bef86
```

3. Run the **tripleo_nova_image_cache.yml** ansible playbook:

```
$ source centralrc

$ ansible-playbook -i inventory.yaml \
--extra-vars "@nova_cache_args.yml" \
/usr/share/ansible/tripleo-playbooks/tripleo_nova_image_cache.yml
```

9.2. PERFORMANCE CONSIDERATIONS

You can specify the number of images that you want to download concurrently with the ansible **forks** parameter, which defaults to a value of **5**. You can reduce the time to distribute this image by increasing the value of the **forks** parameter, however you must balance this with the increase in network and glance-api load.

Use the **--forks** parameter to adjust concurrency as shown:

```
ansible-playbook -i inventory.yaml \
--forks 10 \
--extra-vars "@nova_cache_args.yml" \
/usr/share/ansible/tripleo-playbooks/tripleo_nova_image_cache.yml
```

9.3. OPTIMIZING THE IMAGE DISTRIBUTION TO DCN SITES

You can reduce WAN traffic by using a proxy for glance image distribution. When you configure a proxy:

- Glance images are downloaded to a single Compute node that acts as the proxy.
- The proxy redistributes the glance image to other Compute nodes in the inventory.

You can place the following parameters in the **nova_cache_args.yml** ansible argument file to configure a proxy node.

Set the **tripleo_nova_image_cache_use_proxy** parameter to **true** to enable the image cache proxy.

The image proxy uses secure copy **scp** to distribute images to other nodes in the inventory. SCP is inefficient over networks with high latency, such as a WAN between DCN sites. Red Hat recommends that you limit the playbook target to a single DCN location, which correlates to a single stack.

Use the **tripleo_nova_image_cache_proxy_hostname** parameter to select the image cache proxy. The default proxy is the first compute node in the ansible inventory file. Use the **tripleo_nova_image_cache_plan** parameter to limit the playbook inventory to a single site:

```
tripleo_nova_image_cache_use_proxy: true
tripleo_nova_image_cache_proxy_hostname: dcn0-novacompute-1
tripleo_nova_image_cache_plan: dcn0
```

9.4. CONFIGURING THE NOVA-CACHE CLEANUP

A background process runs periodically to remove images from the nova cache when both of the following conditions are true:

- The image is not in use by an instance.
- The age of the image is greater than the value for the nova parameter **remove_unused_original_minimum_age_seconds**.

The default value for the **remove_unused_original_minimum_age_seconds** parameter is **86400**. The value is expressed in seconds and is equal to 24 hours. You can control this value with the **NovaImageCachTTL** tripleo-heat-templates parameter during the initial deployment, or during a stack update of your cloud:

```
parameter_defaults:
  NovaImageCacheTTL: 604800 # Default to 7 days for all compute roles
  Compute2Parameters:
    NovaImageCacheTTL: 1209600 # Override to 14 days for the Compute2 compute role
```

When you instruct the playbook to pre-cache an image that already exists on a Compute node, ansible does not report a change, but the age of the image is reset to 0. Run the ansible play more frequently than the value of the **NovaImageCacheTTL** parameter to maintain a cache of images.

CHAPTER 10. TLS-E FOR DCN

You can enable TLS (transport layer security) on clouds designed for distributed compute node infrastructure. You have the option of either enabling TLS for public access only, or enabling TLS on every network with TLS-e, which allows for encryption on all internal and external dataflows.

You cannot enable public access on edge stacks as edge sites do not have public endpoints. For more information on TLS for public access, see Enabling SSL/TLS on Overcloud Public Endpoints.

10.1. DEPLOYING DISTRIBUTED COMPUTE NODE ARCHITECTURE WITH TLS-E

Prerequisites

When you configure TLS-e on Red Hat OpenStack Platform (RHOSP) distributed compute node architecture with Red Hat Identity Manager (IdM), take the following actions based on the version of Red Hat Enterprise Linux deployed for Red Hat Identity Manager.

Red Hat Enterprise Linux 8.4

1. On the Red Hat Identity Management node, allowed trusted subnets to an ACL In the **ipa-ext.conf** file:

```
acl "trusted_network" {
  localnets;
  localhost;
  192.168.24.0/24;
  192.168.25.0/24;
};
```

1. In the **/etc/named/ipa-options-ext.conf** file, allow recursion, and query cache:

```
allow-recursion { trusted_network; };
allow-query-cache { trusted_network; };
```

2. Restart the `named-pkcs11 service:

```
systemctl restart named-pkcs11
```

Red Hat Enterprise Linux 8.2

If you have Red Hat Identity Manager (IdM) on Red Hat Enterprise Linux (RHEL) 8.2, you must upgrade Red Hat Enterprise Linux and then follow the directions for RHEL 8.4

Red Hat Enterprise Linux 7.x

If you have Red Hat Identity Manager (IdM) on Red Hat Enterprise Linux (RHEL) 7.x, you must add an access control instruction (ACI) for your domain name manually. For example, if the domain name is **redhat.local**, run the following commands on Red Hat Identity Manager to configure the ACI:

```
ADMIN_PASSWORD=redhat_01
DOMAIN_LEVEL_1=local
DOMAIN_LEVEL_2=redhat

cat << EOF | ldapmodify -x -D "cn=Directory Manager" -w ${ADMIN_PASSWORD}
```

```
dn: cn=dns,dc=${DOMAIN_LEVEL_2},dc=${DOMAIN_LEVEL_1}
changetype: modify
add: aci
aci: (targetattr = "aaaarecord || arecord || cnamerecord || idnsname || objectclass || ptrrecord")
(targetfilter = "(&(objectclass=idnsrecord)(|(aaaarecord=)(arecord=)(cnamerecord=)(ptrrecord=)
(idnsZoneActive=TRUE)))")(version 3.0; acl "Allow hosts to read DNS A/AAA/CNAME/PTR records";
allow (read,search,compare) userdn =
"ldap:///fqdn=*,cn=computers,cn=accounts,dc=${DOMAIN_LEVEL_2},dc=${DOMAIN_LEVEL_1}";)
EOF
```

Procedure

For distributed compute node (DCN) architectures, it is required to use the ansible-based **tripleo-ipa** method of implementing TLS-e as opposed to the previous **novajoin** method. For more information on deploying TLS-e with **tripleo-ipa** see Implementing TLS-e with Ansible.

To deploy TLS-e with **tripleo-ipa** for DCN architectures, you will need to also complete the following steps:

1. If you are deploying storage at the edge, include the following parameters in your modified tripleo heat templates for edge stacks:

```
TEMPLATES=/usr/share/openstack-tripleo-heat-templates

resource_registry:
  OS::TripleO::Services::IpaClient:
    ${TEMPLATES}/deployment/ipa/ipaservices-baremetal-ansible.yaml

parameter_defaults:
  EnableEtcdInternalTLS: true
```

Due to differences in design between the central and edge locations, do not include the following files in edge stacks:

tls-everywhere-endpoints-dns.yaml

This file is ignored at edge sites, the endpoints that it sets are overridden by the endpoints exported from the central stack.

haproxy-public-tls-certmonger.yaml

This file causes a failed deployment as there are no public endpoints at the edge.

CHAPTER 11. CREATING A CEPH KEY FOR EXTERNAL ACCESS

External access to Ceph storage is access to Ceph from any site that is not local. Ceph storage at the cental location is external for edge (DCN) sites, just as Ceph storage at the edge is external for the central location.

When you deploy the central or DCN sites with Ceph storage, you have the option of using the default **openstack** keyring for both local and external access. Altenatively, you can create a separate key for access by non-local sites.

If you decide to use additional Ceph keys for access to your external sites, each key must have the same name. The key name is **external** in the examples that follow.

If you use a separate key for access by non-local sites, you have the additional security benefit of being able to revoke and re-issue the external key in response to a security event without interrupting local access. However, using a separate key for external access will result in the loss of acces to some features, such as cross availability zone backups and offline volume migration. You must balance the needs of your security posture against the desired feature set.

By default, the keys for the central and all DCN sites will be shared.

11.1. CREATING A CEPH KEY FOR EXTERNAL ACCESS

Complete the following steps to create an **external** key for non-local access.

Process

1. Create a Ceph key for external access. This key is sensitive. You can generate the key using the following:

```
python3 -c 'import os,struct,time,base64; key = os.urandom(16) ; \
header = struct.pack("<hiih", 1, int(time.time()), 0, len(key)) ; \
print(base64.b64encode(header + key).decode())'
```

2. In the directory of the stack you are deploying, create a **ceph_keys.yaml** environment file with contents like the following, using the output from the previous command for the key:

```
parameter_defaults:
  CephExtraKeys:
    - name: "client.external"
      caps:
        mgr: "allow *"
        mon: "profile rbd"
        osd: "profile rbd pool=vms, profile rbd pool=volumes, profile rbd pool=images"
      key: "AQD29WteAAAAABAAphgOjFD7nyjdYe8Lz0mQ5Q=="
      mode: "0600"
```

3. Include the **ceph_keys.yaml** environment file in the deployment of the site. For example, to deploy the central site with with the **ceph_keys.yaml** environment file, run a command like the following:

```
overcloud deploy \
        --stack central \
        --templates /usr/share/openstack-tripleo-heat-templates/ \
```

```
....
-e ~/central/ceph_keys.yaml
```

11.2. USING EXTERNAL CEPH KEYS

You can only use keys that have already been deployed. For information on deploying a site with an **external** key, see Section 11.1, “Creating a Ceph key for external access” . This should be done for both central and edge sites.

- When you deploy an edge site that will use an **external** key provided by central, complete the following:

  1. Create **dcn_ceph_external.yaml** environment file for the edge site. You must include the **cephx-key-client-name** option to specify the deployed key to include.

     ```
     sudo -E openstack overcloud export ceph \
     --stack central \
     --config-download-dir /var/lib/mistral \
     --cephx-key-client-name external \
     --output-file ~/dcn-common/dcn_ceph_external.yaml
     ```

  2. Include the **dcn_ceph_external.yaml** file so that the edge site can access the Ceph cluster at the central site. Include the **ceph_keys.yaml** file to deploy an external key for the Ceph cluster at the edge site.

- When you update the central location after deploying your edge sites, ensure the central location to use the dcn **external** keys:

  1. Ensure that the **CephClientUserName** parameter matches the key being exported. If you are using the name **external** as shown in these examples, create **glance_update.yaml** to be similar to the following:

     ```
     parameter_defaults:
       GlanceEnabledImportMethods: web-download,copy-image
       GlanceBackend: rbd
       GlanceStoreDescription: 'central rbd glance store'
       CephClusterName: central
       GlanceBackendID: central
       GlanceMultistoreConfig:
       dcn0:
         GlanceBackend: rbd
         GlanceStoreDescription: 'dcn0 rbd glance store'
         CephClientUserName: 'external'
         CephClusterName: dcn0
         GlanceBackendID: dcn0
       dcn1:
         GlanceBackend: rbd
         GlanceStoreDescription: 'dcn1 rbd glance store'
         CephClientUserName: 'external'
         CephClusterName: dcn1
         GlanceBackendID: dcn1
     ```

  2. Use the **openstack overcloud export ceph** command to include the **external** keys for DCN edge access from the central location. To do this you must provide a a comma-delimited list of stacks for the **--stack** argument, and include the **cephx-key-client-name**

option:

```
sudo -E openstack overcloud export ceph \
--stack dcn0,dcn1,dcn2 \
--config-download-dir /var/lib/mistral \
--cephx-key-client-name external \
--output-file ~/central/dcn_ceph_external.yaml
```

3. Redeploy the central site using the original templates and include the newly created **dcn_ceph_external.yaml** and **glance_update.yaml** files.

```
openstack overcloud deploy \
      --stack central \
      --templates /usr/share/openstack-tripleo-heat-templates/ \
      -r ~/central/central_roles.yaml \
   ...
      -e /usr/share/openstack-tripleo-heat-templates/environments/ceph-ansible/ceph-
ansible.yaml \
      -e /usr/share/openstack-tripleo-heat-templates/environments/nova-az-config.yaml \
      -e ~/central/central-images-env.yaml \
      -e ~/central/role-counts.yaml \
      -e ~/central/site-name.yaml
      -e ~/central/ceph.yaml \
      -e ~/central/ceph_keys.yaml \
      -e ~/central/glance.yaml \
      -e ~/central/dcn_ceph_external.yaml
```

APPENDIX A. DEPLOYMENT MIGRATION OPTIONS

This section includes topics related validation of DCN storage, as well as migrating or changing architectures.

A.1. VALIDATING EDGE STORAGE

Ensure that the deployment of central and edge sites are working by testing glance multi-store and instance creation.

You can import images into glance that are available on the local filesystem or available on a web server.

NOTE

Always store an image copy in the central site, even if there are no instances using the image at the central location.

Prerequisites

1. Check the stores that are available through the Image service by using the **glance stores-info** command. In the following example, three stores are available: central, dcn1, and dcn2. These correspond to glance stores at the central location and edge sites, respectively:

```
$ glance stores-info
+----------+----------------------------------------------------------------------------------+
| Property | Value                                                                            |
+----------+----------------------------------------------------------------------------------+
| stores   | [{"default": "true", "id": "central", "description": "central rbd glance         |
|          | store"}, {"id": "dcn0", "description": "dcn0 rbd glance store"},                  |
|          | {"id": "dcn1", "description": "dcn1 rbd glance store"}]                          |
+----------+----------------------------------------------------------------------------------+
```

A.1.1. Importing from a local file

You must upload the image to the central location's store first, then copy the image to remote sites.

1. Ensure that your image file is in RAW format. If the image is not in raw format, you must convert the image before importing it into the Image service:

```
file cirros-0.5.1-x86_64-disk.img
cirros-0.5.1-x86_64-disk.img: QEMU QCOW2 Image (v3), 117440512 bytes

qemu-img convert -f qcow2 -O raw cirros-0.5.1-x86_64-disk.img cirros-0.5.1-x86_64-
disk.raw
```

Import the image into the default back end at the central site:

```
glance image-create \
--disk-format raw --container-format bare \
--name cirros --file cirros-0.5.1-x86_64-disk.raw \
--store central
```

A.1.2. Importing an image from a web server

If the image is hosted on a web server, you can use the **GlanceImageImportPlugins** parameter to upload the image to multiple stores.

This procedure assumes that the default image conversion plugin is enabled in glance. This feature automatically converts QCOW2 file formats into RAW images, which are optimal for Ceph RBD. You can confirm that a glance image is in RAW format by running the **glance image-show ID | grep disk_format**.

Procedure

1. Use the **image-create-via-import** parameter of the **glance** command to import an image from a web server. Use the **--stores** parameter.

```
# glance image-create-via-import \
--disk-format qcow2 \
--container-format bare \
--name cirros \
--uri http://download.cirros-cloud.net/0.4.0/cirros-0.4.0-x86_64-disk.img \
--import-method web-download \
--stores central,dcn1
```

   In this example, the qcow2 cirros image is downloaded from the official Cirros site, converted to RAW by glance, and imported into the central site and edge site 1 as specified by the **--stores** parameter.

Alternatively you can replace **--stores** with **--all-stores True** to upload the image to all of the stores.

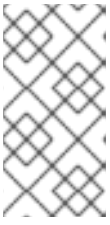
A.1.3. Copying an image to a new site

You can copy existing images from the central location to edge sites, which gives you access to previously created images at newly established locations.

1. Use the UUID of the glance image for the copy operation:

```
ID=$(openstack image show cirros -c id -f value)

glance image-import $ID --stores dcn0,dcn1 --import-method copy-image
```



NOTE

In this example, the **--stores** option specifies that the **cirros** image will be copied from the central site to edge sites dcn1 and dcn2. Alternatively, you can use the **--all-stores True** option, which uploads the image to all the stores that don't currently have the image.

2. Confirm a copy of the image is in each store. Note that the **stores** key, which is the last item in the properties map, is set to **central,dcn0,dcn1**.:

```
 $ openstack image show $ID | grep properties
 | properties       | direct_url=rbd://d25504ce-459f-432d-b6fa-
79854d786f2b/images/8083c7e7-32d8-4f7a-b1da-0ed7884f1076/snap, locations=[{u'url':
u'rbd://d25504ce-459f-432d-b6fa-79854d786f2b/images/8083c7e7-32d8-4f7a-b1da-
```

```
0ed7884f1076/snap', u'metadata': {u'store': u'central'}}, {u'url': u'rbd://0c10d6b5-a455-4c4d-
bd53-8f2b9357c3c7/images/8083c7e7-32d8-4f7a-b1da-0ed7884f1076/snap', u'metadata':
{u'store': u'dcn0'}}, {u'url': u'rbd://8649d6c3-dcb3-4aae-8c19-8c2fe5a853ac/images/8083c7e7-
32d8-4f7a-b1da-0ed7884f1076/snap', u'metadata': {u'store': u'dcn1'}}],
os_glance_failed_import=', os_glance_importing_to_stores=', os_hash_algo='sha512,
os_hash_value=b795f047a1b10ba0b7c95b43b2a481a59289dc4cf2e49845e60b194a911819d
3ada03767bbba4143b44c93fd7f66c96c5a621e28dff51d1196dae64974ce240e,
os_hidden=False, stores=central,dcn0,dcn1 |
```

NOTE

Always store an image copy in the central site even if there is no VM using it on that site.

A.1.4. Confirming that an instance at an edge site can boot with image based volumes

You can use an image at the edge site to create a persistent root volume.

Procedure

1. Identify the ID of the image to create as a volume, and pass that ID to the **openstack volume create** command:

```
IMG_ID=$(openstack image show cirros -c id -f value)
openstack volume create --size 8 --availability-zone dcn0 pet-volume-dcn0 --image $IMG_ID
```

2. Identify the volume ID of the newly created volume and pass it to the **openstack server create** command:

```
VOL_ID=$(openstack volume show -f value -c id pet-volume-dcn0)
openstack server create --flavor tiny --key-name dcn0-key --network dcn0-network --security-
group basic --availability-zone dcn0 --volume $VOL_ID pet-server-dcn0
```

3. You can verify that the volume is based on the image by running the rbd command within a ceph-mon container at the dcn0 edge site to list the volumes pool.

```
$ sudo podman exec ceph-mon-$HOSTNAME rbd --cluster dcn0 -p volumes ls -l
NAME                                        SIZE  PARENT                                    FMT PROT LOCK
volume-28c6fc32-047b-4306-ad2d-de2be02716b7 8 GiB images/8083c7e7-32d8-4f7a-b1da-
0ed7884f1076@snap   2      excl
$
```

4. Confirm that you can create a cinder snapshot of the root volume of the instance. Ensure that the server is stopped to quiesce data to create a clean snapshot. Use the *--force* option, because the volume status remains **in-use** when the instance is off.

```
openstack server stop pet-server-dcn0
openstack volume snapshot create pet-volume-dcn0-snap --volume $VOL_ID --force
openstack server start pet-server-dcn0
```

5. List the contents of the volumes pool on the dcn0 Ceph cluster to show the newly created snapshot.

```
$ sudo podman exec ceph-mon-$HOSTNAME rbd --cluster dcn0 -p volumes ls -l
NAME                                                SIZE  PARENT
FMT PROT LOCK
volume-28c6fc32-047b-4306-ad2d-de2be02716b7                                  8 GiB
images/8083c7e7-32d8-4f7a-b1da-0ed7884f1076@snap   2      excl
volume-28c6fc32-047b-4306-ad2d-de2be02716b7@snapshot-a1ca8602-6819-45b4-a228-
b4cd3e5adf60 8 GiB images/8083c7e7-32d8-4f7a-b1da-0ed7884f1076@snap   2 yes
```

A.1.5. Confirming image snapshots can be created and copied between sites

1. Verify that you can create a new image at the dcn0 site. Ensure that the server is stopped to quiesce data to create a clean snapshot:

```
NOVA_ID=$(openstack server show pet-server-dcn0 -f value -c id)
openstack server stop $NOVA_ID
openstack server image create --name cirros-snapshot $NOVA_ID
openstack server start $NOVA_ID
```

2. Copy the image from the **dcn0** edge site back to the hub location, which is the default back end for glance:

```
IMAGE_ID=$(openstack image show cirros-snapshot -f value -c id)
glance image-import $IMAGE_ID --stores central --import-method copy-image
```

For more information on glance multistore operations, see Image service with multiple stores.

A.2. MIGRATING TO A SPINE AND LEAF DEPLOYMENT

It is possible to migrate an existing cloud with a pre-existing network configuration to one with a spine leaf architecture. For this, the following conditions are needed:

- All bare metal ports must have their **physical-network** property value set to **ctlplane**.
- The parameter **enable_routed_networks** is added and set to **true** in undercloud.conf, followed by a re-run of the undercloud installation command, **openstack undercloud install**.

Once the undercloud is re-deployed, the overcloud is considered a spine leaf, with a single leaf **leaf0**. You can add additional provisioning leaves to the deployment through the following steps.

1. Add the desired subnets to undercloud.conf as shown in Configuring routed spine-leaf in the undercloud.
2. Re-run the undercloud installation command, **openstack undercloud install**.
3. Add the desired additional networks and roles to the overcloud templates, **network_data.yaml** and **roles_data.yaml** respectively.

NOTE

If you are using the **{{network.name}}InterfaceRoutes** parameter in the network configuration file, then you'll need to ensure that the **NetworkDeploymentActions** parameter includes the value *UPDATE*.

```
NetworkDeploymentActions: ['CREATE','UPDATE'])
```

4. Finally, re-run the overcloud installation script that includes all relevant heat templates for your cloud deployment.

A.3. MIGRATING TO A MULTISTACK DEPLOYMENT

You can migrate from a single stack deployment to a multistack deployment by treating the existing deployment as the central site, and adding additional edge sites.

The ability to migrate from single to multistack in this release is a *Technology Preview*, and therefore is not fully supported by Red Hat. It should only be used for testing, and should not be deployed in a production environment. For more information about Technology Preview features, see Scope of Coverage Details.

You cannot split the existing stack. You can scale down the existing stack to remove compute nodes if needed. These compute nodes can then be added to edge sites.

NOTE

This action creates workload interruptions if all compute nodes are removed.

A.4. BACKING UP AND RESTORING ACROSS EDGE SITES

You can back up and restore Block Storage service (cinder) volumes across distributed compute node (DCN) architectures in edge site and availability zones. The **cinder-backup** service runs in the central availability zone (AZ), and backups are stored in the central AZ. The Block Storage service does not store backups at DCN sites.

Prerequisites

- The central site is deployed with the **cinder-backup.yaml** environment file located in **/usr/share/openstack-tripleo-heat-templates/environments**. For more information, see Block Storage backup service deployment.
- The Block Storage service (cinder) CLI is available.
- All sites must use a common **openstack** cephx client name. For more information, see Creating a Ceph key for external access.

Procedure

1. Create a backup of a volume in the first DCN site:

```
$ cinder --os-volume-api-version 3.51 backup-create --name <volume_backup> --availability-zone <az_central> <edge_volume>
```

   - Replace **<volume_backup>** with a name for the volume backup.
   - Replace **<az_central>** with the name of the central availability zone that hosts the **cinder-backup** service.
   - Replace **<edge_volume>** with the name of the volume that you want to back up.

NOTE

If you experience issues with Ceph keyrings, you might need to restart the **cinder-backup** container so that the keyrings copy from the host to the container successfully.

2. Restore the backup to a new volume in the second DCN site:

```
$ cinder --os-volume-api-version 3.51 create --availability-zone <az_2> --name
<new_volume> --backup-id <volume_backup> <volume_size>
```

- Replace **<az_2>** with the name of the availability zone where you want to restore the backup.
- Replace **<new_volume>** with a name for the new volume.
- Replace **<volume_backup>** with the name of the volume backup that you created in the previous step.
- Replace **<volume_size>** with a value in GB equal to or greater than the size of the original volume.