Red Hat OpenStack Platform 16.1

Distributed compute node and storage deployment

Deploying Red Hat OpenStack Platform distributed compute node technologies

Last Updated: 2020-09-25

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 with a distributed compute node (DCN) architecture, allowing edge site operability with heat stack separation. Each site is capable of having its own Ceph storage back end for glance multi store. This guide descibes the architecture, set up, and opeartions of DCN with storage.

Table of Contents

CHAPTER 1. UNDERSTANDING DCN 3
1.1. DESIGNING EDGE SITES WITH DCN 3
1.2. ROLES AT THE EDGE 3

CHAPTER 2. HARDWARE 5
2.1. LIMITATIONS TO CONSIDER 5
2.2. NETWORKING 6
2.2.1. Routing between edge sites 6
2.2.1.1. Push complexity to the hardware network infrastructure 6
2.2.1.2. Push complexity to the Red Hat OpenStack Platform cluster 7

CHAPTER 3. CONFIGURING AND INSTALLING THE HUB UNDERCLOUD 9
3.1. CONFIGURING THE SPINE LEAF PROVISIONING NETWORKS 9
3.2. CONFIGURING A DHCP RELAY 10
3.3. PREREQUISITES FOR USING SEPARATE HEAT STACKS 13
3.4. LIMITATIONS OF THE EXAMPLE SEPARATE HEAT STACKS DEPLOYMENT 13
3.5. DESIGNING YOUR SEPARATE HEAT STACKS DEPLOYMENT 13
3.6. REUSING NETWORK RESOURCES IN MULTIPLE STACKS 14
3.7. USING MANAGENETWORKS TO REUSE NETWORK RESOURCES 14
3.8. USING UUIDS TO REUSE NETWORK RESOURCES 15
3.9. MANAGING SEPARATE HEAT STACKS 16
3.10. RETRIEVING THE CONTAINER IMAGES 16
3.11. DEPLOYING THE CENTRAL CONTROLLERS WITHOUT EDGE STORAGE 17
3.12. DEPLOYING EDGE NODES WITHOUT STORAGE 19
3.12.1. Configuring the distributed compute node environment files 19
3.12.2. Deploying the Compute nodes to the DCN site 20

CHAPTER 4. CONFIGURING THE EDGE SITES AND HUB NETWORKS 22

CHAPTER 5. DEPLOYING STORAGE AT THE EDGE 23
5.1. LIMITATIONS OF DISTRIBUTED COMPUTE NODE (DCN) ARCHITECTURE 23
5.1.1. Requirements of storage edge architecture 24
5.2. DEPLOYING THE CENTRAL SITE 24
5.3. DEPLOYING EDGE SITES WITH STORAGE 27
5.4. CREATING ADDITIONAL DISTRIBUTED COMPUTE NODE SITES 31
5.5. UPDATING THE CENTRAL LOCATION 32

CHAPTER 6. VALIDATING EDGE STORAGE 34
6.1. IMPORTING FROM A LOCAL FILE 34
6.2. IMPORTING AN IMAGE FROM A WEB SERVER 34
6.3. COPYING AN IMAGE TO A NEW SITE 35
6.4. CONFIRMING THAT AN INSTANCE AT AN EDGE SITE CAN BOOT WITH IMAGE BASED VOLUMES 36
6.5. CONFIRMING IMAGE SNAPSHOTS CAN BE CREATED AND COPIED BETWEEN SITES 37

APPENDIX A. DEPLOYMENT MIGRATION OPTIONS 38
A.1. MIGRATING TO A SPINE AND LEAF DEPLOYMENT 38
A.2. MIGRATING TO A MULTISTACK DEPLOYMENT 38

CHAPTER 1. UNDERSTANDING DCN

Distributed compute node architecture is a hub and spoke routed network deployment. It 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 primary site with core routers and a datacenter gateway (DC-GW). The spoke is the remote edge, or leaf. The hub site can consist of any role, and at a minimum, requires three controllers.

Compute nodes can exist at the edge, as well as at the primary hub site.

1.1. DESIGNING EDGE SITES WITH DCN

You must deploy multiple stacks as part of a distributed compute node (DCN) architecture. 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 preexisting deloyment. See Section A.2, “Migrating to a multistack deployment” for details.

Image service (glance) multi store is now supported with distributed edge architecture. With this feature, you can now have an image pool at every distributed edge site. This allows you to copy images between hub and edge sites.

1.2. ROLES AT THE EDGE

If block storage is not going to be deployed at the edge, you must follow the section of the document, Section 3.11, “Deploying the central controllers without edge storage” . Without block storage at the edge:

- 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, you must also deploy block storage at the central location. Follow the section of the document Chapter 5, *Deploying storage at the edge* . With block storage at the edge:

- Ceph RBD is used as a Glance backend
- Images may be stored at edge sites
- The Cinder volume service is available at all sites 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**

  DisributedCompute

  This role includes the **GlanceApiEdge** service, so that Image services are consumed at the local edge site as opposed to the central hub location. Start by deploying up to three nodes using the **DistributedCompute** role, before deploying the **DistributedComputeScaleOut** role.

  DistributedComputeScaleOut

  This role includes the **HAproxyEdge** service, to enable instances created on the

DistributedComputeScaleOut role to proxy requests for Image services to nodes that provide that service at the edge site. After you deploy three nodes with a role of **DistributedCompute**, you can use the **DistributedComputeScaleOut** role to scale compute resources. There is no minimum number of hosts required to deploy with the **DistrubutedComputeScaleOut** 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 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.

  DistributedCompute

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

CHAPTER 2. HARDWARE

When you deploy Red Hat OpenStack Platform with distributed compute nodes, your control plane stays at the hub. Compute nodes at the hub site are optional. At edge sites, you can have the following:

- Compute nodes
- Hyperconverged nodes with both Compute services and Ceph storage

2.1. LIMITATIONS TO CONSIDER

- **Network latency**: The edge Compute nodes are integrated with the wider overcloud deployment and must meet certain network latency requirements, with a round-trip time (RTT) that does not exceed 100ms.
- **Network drop outs**: If the edge site temporarily loses its connection, 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 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 block storage is not deployed at the edge, then the Glance images will traverse the WAN during first use. The Glance 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.
    When block storage is deployed at the edge, 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 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 will need to make sure 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 are using networks that are specific to a certain site: When deploying 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 are using VXLAN or Geneve, the result is that every Compute node at every site forms a tunnel with every other Compute node and Controller node whether or not they are actually local or remote. This is not an issue if you are using the same neutron networks everywhere. When using 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 the undercloud to add new network segments and subnets.

- **Autonomy**: There might be specific autonomy requirements for the edge site. This might vary depending on your requirements.

2.2. NETWORKING

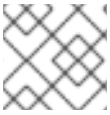
When designing the network for distributed compute node architecture, be aware of the supported technologies and constraints:

The following networking technologies are supported at the edge:

- ML2/OVS
- ML2/SR-IOV
- ML2/OVN as a technology preview

The following fast datapaths for NFV are supported at the edge:

- OVS-DPDK without Neutron DHCP services
- SR-IOV
- TC/Flower offload



NOTE

Fast datapaths at the edge require ML2/OVS.

- If you deploy distributed storage, the maximum storage network latency between central and edge sites for Ceph RBD traffic is 100ms round-trip time (RTT).
- If edge servers are not preprovisioned, 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.

2.2.1. Routing between edge sites

If you need full mesh connectivity between both the central location and edge sites, as well as routing between the edge sites themselves, you must design a solution with inherent complexity, either on the network infrastructure or on the Red Hat OpenStack Platform nodes.

The following approaches satisfy full mesh connectivity between every logical site, both central and edge, for control plane signaling. Tenant (overlay) networks are terminated on site.

There are two ways to create full mesh connectivity between the central and edge sites:

2.2.1.1. Push complexity to the hardware network infrastructure

Allocate a supernet for each network function and allocate a block for each edge and leaf, then use dynamic routing on your network infrastructure to advertise and route each locally connected block.

The benefit of this procedure is that it requires only a single route on each OpenStack node per network function interface to reach the corresponding interfaces at local or remote edge sites.

1. Reserve 16-Bit address blocks for OpenStack endpoints

```
Provisioning: 10.10.0.0/16
Internal API: 10.20.0.0/16
Storage Front-End: 10.40.0.0/16
Storage Back-End: 10.50.0.0/16
```

2. Use smaller blocks from these to allocate addresses for each edge site or leaf. You can summarize a smaller block by using the following:

```
10.10.`pod#`.0/24
```

   For example, the following could be used for a site designated as leaf 2.

```
External:     10.1.2.0/24
Provisioning: 10.10.2.0/24
Internal API: 10.20.2.0/24
Storage:      10.40.2.0/24
Storage Mgmt: 10.50.2.0/24
Provider:     10.60.2.0/24
```

3. Define common static routes for function summaries. Consider the following example:

```
Provisioning: 10.10.0.0/16 > 10.10.[pod#].1
Internal API: 10.20.0.0/16 > 10.20.[pod#].1
Storage Front-End: 10.40.0.0/16 > 10.40.[pod#].1
Storage Back-End: 10.50.0.0/16 > 10.50.[pod#].1
```

2.2.1.2. Push complexity to the Red Hat OpenStack Platform cluster

1. Allocate a route per edge site for each network function (internal api, overlay, storage, and so on) in the **network_data.yaml** file for the cluster:

```
###INTERNAL API NETWORKS
- name: InternalApi
  name_lower: internal_api
  vip: true
  ip_subnet: '10.20.2.0/24'
  allocation_pools: [{'start': '10.20.2.100', 'end': '10.20.2.199'}]
  gateway_ip: '10.20.2.1'
  vlan: 0
  subnets:
    internal_api3
      ip_subnet: '10.20.3.0/24'
      allocation_pools: [{'start': '10.20.3.100', 'end': '10.20.3.199'}]
      vlan: 0
      gateway_ip: '10.20.3.1'
    internal_api4
      ip_subnet: '10.20.4.0/24'
      allocation_pools: [{'start': '10.20.4.100', 'end': '10.20.4.199'}]
```

```
    vlan: 0
    gateway_ip: '10.20.4.1'
...
```

This method allows you to more easily configure summarized static routing on the network infrastructure. Use dynamic routing on the networking infrastructure to further simplify configuration.

CHAPTER 3. CONFIGURING AND INSTALLING THE HUB UNDERCLOUD

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_vip** to the externally facing IP address of the undercloud:

```
undercloud_public_vip = 10.1.1.1
```

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

```
undercloud_admin_vip = 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

The undercloud uses two DHCP servers on the provisioning network:

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

When you configure a DHCP relay, ensure that you 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).

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.

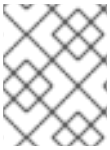
IP network address

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 example, 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:

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

If you require the ability to automatically discover bare metal nodes without the need to create an **instack.json** file, you can use auto-discovery to register overcloud nodes. See Automatically discovering bare metal nodes for more information.

3.3. 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.

3.4. 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.

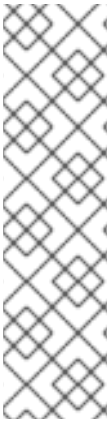
3.5. DESIGNING YOUR SEPARATE HEAT STACKS DEPLOYMENT

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'
```

3.6. 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**.

3.7. 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.

3.8. 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
```

3.9. 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:

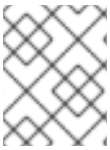
```
$ mkdir dcn-common
$ touch dcn-common/overrides.yaml
$ touch dcn-common/central-export.yaml
```

   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.

3.10. 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 https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/16.1/html/transitioning_to_containerized_services/obtaining-container-images#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**:

```
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.

3.11. DEPLOYING THE CENTRAL CONTROLLERS WITHOUT EDGE STORAGE

You can deploy a distributed compute node cluster without Block storage at edge sites if you use Swift as a backend for glance at the central location.

Important: You cannot use Ceph as a backend for glance at the central location if it will not be used at the edge. 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 the following settings:

```
parameter_defaults:
  NtpServer:
    - clock.redhat.com
    - clock2.redhat.com
  ControllerCount: 3
  ComputeCount: 0
  OvercloudControlFlavor: 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**:

```
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 ~/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.

3.12. 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.

3.12.1. Configuring the distributed compute node environment files

Procedure

1. Generate the config-download files:

```
openstack overcloud config download \
--name central --config-dir ~/central/config-download
```

2. 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
```

IMPORTANT

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.

3.12.2. Deploying the Compute nodes to the DCN site

This procedure uses the **DistributedCompute** role to deploy Compute nodes to an availability zone (AZ) named **dcn0**. This role is used specifically for distributed compute nodes.

Procedure

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

```
parameter_defaults:
  DistributedComputeCount: 3
  DistributedComputeFlavor: baremetal
  DistributedComputeSchedulerHints:
    'capabilities:node': '0-compute-%index%'
  NovaAZAttach: false
```

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

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

3. 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
```

4. Retrieve the container images for the DCN Site:

```
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
```

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

```
#!/bin/bash
STACK=dcn0
source ~/stackrc
if [[ ! -e ~/dcn0/distributed_compute.yaml ]]; then
    openstack overcloud roles generate DistributedCompute -o ~/dcn0/roles_data.yaml
fi
```

```
time openstack overcloud deploy \
    --stack $STACK \
    --templates /usr/share/openstack-tripleo-heat-templates/ \
    -r roles_data.yaml \
    -e /usr/share/openstack-tripleo-heat-templates/environments/cinder-volume-active-
active.yaml \
    -e ~/dcn-common/central-export.yaml \
    -e ~/dcn0/dcn0-images-env.yaml \
    -e site-name.yaml \
    -e overrides.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.

CHAPTER 4. CONFIGURING THE EDGE SITES AND HUB NETWORKS

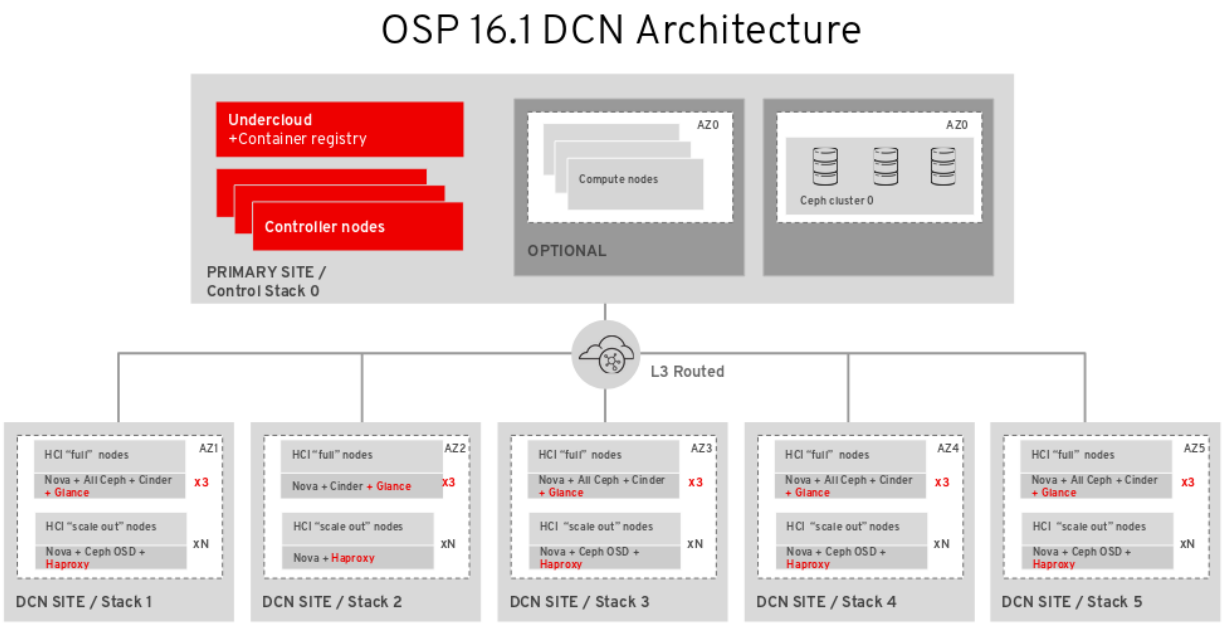
You can automate the configuration of hardware at your hub and edge sites by using Ansible modules. The Ansible networking team and certified partners support a number of configuration modules for the following hardware and software networking solutions:

- ACI
- Big Switch
- Checkpoint
- CloudEngine
- CloudVision
- Cumulus
- EOS
- F5
- Free Range Routing (FRR)
- IOS
- JunOS
- NSO
- Nxos
- Sros
- VyOS

More network modules are available from the community. Additionally, using the platform independent CLI modules, you can push text-based configuration to network devices. For more information, see Ansible Network modules .

CHAPTER 5. 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.



5.1. LIMITATIONS OF DISTRIBUTED COMPUTE NODE (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.
- Key Manager services at edge sites, including support for encrypted volumes or images at edge sites.
- 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. Once the image is copied, you can create a volume from it.
- Migrating or retyping a volume between sites.
- Ceph Dashboard at the edge.
- Ceph Rados Gateway (RGW) at the edge.
- CephFS at the edge.
- Block storage (cinder) backup for edge volumes.
- Instance high availability (HA) at the edge sites.

- Mixed storage environments, wherein only some edge sites are deployed with Ceph as a storage backend.
- Live migration between edge sites or from the central location to edge sites. You can still live migrate instances within a site boundary.
- RBD mirroring between sties.

The following overlay networking technologies are available as a *Technology Preview*, and therefore are not fully supported by Red Hat. These features 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.

- Routed provider networks.
- DPDK at the edge.

5.1.1. Requirements of storage edge architecture

- A copy of each image must exist in the Image service at the central location.
- Prior to creating an instance at an edge site, you must have a local copy of the image at that edge site.
- Source the **centralrc** authentication file to schedule workloads at edge sites as well as at the central location. Authentication files that are automatically generated for edge sites are not needed.
- Images uploaded to an edge site must be copied to the central location before they can be copied to other edge sites.
- Use the Image service RBD driver for all edge sites. Mixed architecture is not supported.
- Multistack must be used with a single stack at each site.
- RBD must be the storage driver for the Image, Compute and Block Storage services.
- For each site, you must assign the same value to the **NovaComputeAvailabilityZone** and **CinderStorageAvailabilityZone** parameters.

5.2. DEPLOYING THE CENTRAL SITE

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, or hub 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. Generate a Ceph key.

```
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())'
```

3. Create the **ceph_keys.yaml** file for the **central** stack so that edge site OpenStack services you deploy later can connect. Use the Ceph key generated in the previous step as the **key** parameter value. This key is sensitive and should be protected.

```
cat > /home/stack/central/ceph_keys.yaml << EOF
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"
EOF
```

4. 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
```

5. 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
```

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

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

7. 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
```

8. 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 ~/central/central-images-env.yaml \
      -e ~/central/role-counts.yaml \
      -e ~/central/nova-az.yaml
      -e ~/central/ceph.yaml \
      -e ~/central/ceph_keys.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, you can do so with the **GlanceImageImportPlugin** parameter:

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

5.3. 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 control-plane \
      --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.

   - The **keys** section should contain the same values that were passed to the **CephExtraKeys** parameter configured for the central location in step 1 of this procedure.
   - The value for **external_cluster_mon_ips** can be obtained from the **tripleo-ansible-inventory.yaml** file in the directory specified by the **--config-download-dir** parameter. Use the IP addresses or hostnames of the nodes that run the **CephMons** service.
   - Additional information such as the FSID can be obtained from the **all.yaml** file in the directory specified by the **--config-download-dir** parameter.
   - You must set the **dashboard_enabled** parameter to **false** when using the **CephExternalMultiConfig** parameter because you cannot deploy the Ceph dashboard when you configure an overcloud as a client of an external Ceph cluster. Relative to the edge site dcn0, **central** is an external Ceph cluster.

```
cat > central_ceph_external.yaml << EOF
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.external"
        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.external.keyring
EOF
```

NOTE

Do not use the **CephExternalMultiConfig** parameter when you configure a single external Ceph cluster. This parameter is only supported when you deploy the following:

   - An *external* Ceph cluster, configured normally, in addition to multiple external Ceph clusters
   - An *internal* Ceph cluster, configured normally, in addition to multiple external Ceph clusters

3. Create a new **ceph_keys.yaml** file for dcn0. Follow the same steps used at the beginning of the procedure to create **central**, but use a new Ceph key. For example:

```
cat > ~/dcn0/ceph_keys.yaml <<EOF
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: "AQBO/mteAAAAABAAc4mVMTpq7OFtrPIRFqN+FQ=="
    mode: "0600"
EOF
```

4. Create the ~/**dcn0**/**glance.yaml** file for glance configuration overrides:

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

5. Configure the **ceph.yaml** file with configuration parameters relative to the available hardware. For more information, see Configuring Ceph with Custom Config Settings:

```
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
```

6. 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** and **DistributedComputeHCIParameters**, see Configure resource allocation.

- For more information about setting the values for the parameters **CephPoolDefaultPgNum**, **CephPoolDefaultSize**, and **DistributedComputeHCIExtraConfig**, see Configuring ceph storage cluster setting.

7. 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
```

8. 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
```

9. 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
```

10. Retrieve the container images for the edge site:

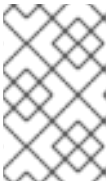
```
openstack tripleo container image prepare \
--environment-directory dcn0 \
-r ~/dcn0/roles_data.yaml \
-e /usr/share/openstack-tripleo-heat-templates/environments/ceph-ansible/ceph-ansible.yaml
\
...
-e /home/stack/dcn-common/central-export.yaml \
-e /home/stack/containers-prepare-parameter.yaml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

11. 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 ~/dnc0/dcn0-images-env.yaml \
  ....
  -e ~/dcn-common/central-export.yaml \
  -e ~/dcn0/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.

5.4. 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 3.9, “Managing separate heat stacks” . This procedure contains example commands.

Procedure

1. As the stack user on the undercloud, create a new directory for **dcn1**:

```
$ cd ~
$ mkdir dcn1
```

2. Copy the existing **dcn0** templates to the new directory and replace the **dcn0** strings with **dcn1**:

```
$ cp dcn0/ceph.yaml dcn1/ceph.yaml
$ sed s/dcn0/dcn1/g -i dcn1/ceph.yaml
$ cp dcn0/overrides.yaml dcn1/overrides.yaml
$ sed s/dcn0/dcn1/g -i dcn1/overrides.yaml
$ sed s/"0-ceph-%index%"/"1-ceph-%index%"/g -i dcn1/overrides.yaml
$ cp dcn0/deploy.sh dcn1/deploy.sh
$ sed s/dcn0/dcn1/g -i dcn1/deploy.sh
```

3. Review the files in the **dcn1** directory to confirm that they suit your requirements.
4. Verify that your nodes are available and in **Provisioning state**:

```
$ openstack baremetal node list
```

5. When your nodes are available, run the **deploy.sh** for the **dcn1** site:

```
$ bash dcn1/deploy.sh
```

5.5. 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.

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

2. Create the **dcn_ceph_external.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**.
   You can find the value for **external_cluster_mon_ips** from the **tripleo-ansible-inventory.yaml** file located in the following directories. Use the IP addresses or hostnames of the nodes that run the **CephMons** service.

   - /var/lib/mistral/dcn0/tripleo-ansible-inventory.yaml
   - /var/lib/mistral/dcn1/tripleo-ansible-inventory.yaml
     Find additional required values for this template, such as the FSID and cluster name, in the following files:
   - /var/lib/mistral/dcn0/ceph-ansible/group_vars/all.yaml
   - /var/lib/mistral/dcn1/ceph-ansible/group_vars/all.yaml

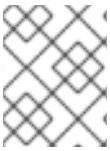
```
parameter_defaults:
  CephExternalMultiConfig:
    - cluster: "dcn0"
      fsid: "539e2b96-316e-4c23-b7df-035a3037ddd1"
      external_cluster_mon_ips: "172.16.11.61, 172.16.11.64, 172.16.11.66"
      keys:
        - name: "client.external"
          caps:
            mgr: "allow *"
            mon: "profile rbd"
            osd: "profile rbd pool=images"
          key: "AQBO/mteAAAAABAAc4mVMTpq7OFtrPlRFqN+FQ=="
          mode: "0600"
      dashboard_enabled: false
      ceph_conf_overrides:
          client:
              keyring: /etc/ceph/dcn0.client.glance.keyring
    - cluster: "dcn1"
      fsid: "7504a91e-5a0f-4408-bb55-33c3ee2c67e9"
      external_cluster_mon_ips: "172.16.11.182, 172.16.11.185, 172.16.11.187"
      keys:
        - name: "client.glance"
          caps:
            mgr: "allow *"
            mon: "profile rbd"
            osd: "profile rbd pool=images"
          key: "AQACCGxeAAAAABAAHocX/cnygrVnLBrKiZHJfw=="
          mode: "0600"
      dashboard_enabled: false
      ceph_conf_overrides:
          client:
              keyring: /etc/ceph/dcn1.client.glance.keyring
```

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

CHAPTER 6. 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"}]                          |
+----------+----------------------------------------------------------------------------------+
```

6.1. 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
```

6.2. 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.

6.3. 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
```

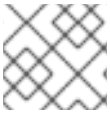
NOTE

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.

6.4. 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
```

6.5. 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
```

APPENDIX A. DEPLOYMENT MIGRATION OPTIONS

This section includes topics related to migrating to a distributed compute node architecture and multi-stack deployments.

A.1. 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.2. 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.