Configure routed spine-leaf networks using Red Hat OpenStack Platform director
Configure routed spine-leaf networks using Red Hat OpenStack Platform director

OpenStack Team
rhos-docs@redhat.com
Abstract

This guide provides a basic scenario on how to configure a routed spine-leaf network on the overcloud. This includes setting up the undercloud, writing the main configuration files, and creating roles for your nodes.
# Table of Contents

**CHAPTER 1. INTRODUCTION**  ................................................................. 3  
1.1. SPINE-LEAF NETWORKING ......................................................... 3  
1.2. NETWORK TOPOLOGY ............................................................. 3  
1.3. SPINE-LEAF REQUIREMENTS ................................................... 5  
1.4. SPINE-LEAF LIMITATIONS ...................................................... 5  

**CHAPTER 2. CONFIGURING THE UNDERCLOUD**  .................................. 7  
2.1. CONFIGURING THE SPINE LEAF PROVISIONING NETWORKS .............. 7  
2.2. CONFIGURING A DHCP RELAY ................................................... 8  
2.3. CREATING FLAVORS AND TAGGING NODES FOR LEAF NETWORKS ... 11  
2.4. MAPPING BARE METAL NODE PORTS TO CONTROL PLANE NETWORK SEGMENTS .................................................. 12  

**CHAPTER 3. ALTERNATIVE PROVISIONING NETWORK METHODS** ............ 14  
3.1. VLAN PROVISIONING NETWORK ............................................. 14  
3.2. VXLAN PROVISIONING NETWORK ........................................... 14  

**CHAPTER 4. CONFIGURING THE OVERCLOUD**  .................................. 16  
4.1. CREATING A NETWORK DATA FILE ........................................... 16  
4.2. CREATING A ROLES DATA FILE ............................................. 17  
4.3. CREATING A CUSTOM NIC CONFIGURATION ................................. 18  
4.4. SETTING CONTROL PLANE PARAMETERS .................................. 21  
4.5. SETTING THE SUBNET FOR VIRTUAL IP ADDRESSES .................... 21  
4.6. MAPPING SEPARATE NETWORKS ............................................. 22  
4.7. DEPLOYING A SPINE-LEAF ENABLED OVERCLOUD ....................... 23
CHAPTER 1. INTRODUCTION

This guide provides information of how to construct a spine-leaf network topology for your Red Hat OpenStack Platform environment. This includes a full end-to-end scenario and example files to help replicate a more extensive network topology within your own environment.

1.1. SPINE-LEAF NETWORKING

Red Hat OpenStack Platform’s composable network architecture allows you to adapt your networking to the popular routed spine-leaf data center topology. In a practical application of routed spine-leaf, a leaf is represented as a composable Compute or Storage role usually in a data center rack, as shown in Figure 1.1, “Routed spine-leaf example”. The Leaf 0 rack has an undercloud node, controllers, and compute nodes. The composable networks are presented to the nodes, which have been assigned to composable roles. In this diagram:

- The **StorageLeaf** networks are presented to the Ceph storage and Compute nodes.
- The **NetworkLeaf** represents an example of any network you might want to compose.

![Figure 1.1. Routed spine-leaf example](image)

1.2. NETWORK TOPOLOGY

This scenario takes advantage of OpenStack Networking (neutron) functionality to define multiple subnets within segments of a single network. Each network uses a base network which acts as Leaf 0. The director creates Leaf 1 and Leaf 2 subnets as segments from the main network.
This scenario uses the following networks:

**Table 1.1. Leaf 0 Networks (base networks)**

<table>
<thead>
<tr>
<th>Network</th>
<th>Roles attached</th>
<th>Subnet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Provisioning / Ctlplane / Leaf0</td>
<td>Controller, ComputeLeaf0, CephStorageLeaf0</td>
<td>192.168.10.0/24</td>
</tr>
<tr>
<td>Storage</td>
<td>Controller, ComputeLeaf0, CephStorageLeaf0</td>
<td>172.16.0.0/24</td>
</tr>
<tr>
<td>StorageMgmt</td>
<td>Controller, CephStorageLeaf0</td>
<td>172.17.0.0/24</td>
</tr>
<tr>
<td>InternalApi</td>
<td>Controller, ComputeLeaf0</td>
<td>172.18.0.0/24</td>
</tr>
<tr>
<td>Tenant</td>
<td>Controller, ComputeLeaf0</td>
<td>172.19.0.0/24</td>
</tr>
<tr>
<td>External</td>
<td>Controller, ComputeLeaf0</td>
<td>10.1.1.0/24</td>
</tr>
</tbody>
</table>

**Table 1.2. Leaf 1 Networks**

<table>
<thead>
<tr>
<th>Network</th>
<th>Roles attached</th>
<th>Subnet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Provisioning / Ctlplane / Leaf1</td>
<td>ComputeLeaf1, CephStorageLeaf1</td>
<td>192.168.11.0/24</td>
</tr>
<tr>
<td>StorageLeaf1</td>
<td>ComputeLeaf1, CephStorageLeaf1</td>
<td>172.16.1.0/24</td>
</tr>
<tr>
<td>StorageMgmtLeaf1</td>
<td>CephStorageLeaf1</td>
<td>172.17.1.0/24</td>
</tr>
<tr>
<td>InternalApiLeaf1</td>
<td>ComputeLeaf1</td>
<td>172.18.1.0/24</td>
</tr>
<tr>
<td>TenantLeaf1</td>
<td>ComputeLeaf1</td>
<td>172.19.1.0/24</td>
</tr>
</tbody>
</table>

**Table 1.3. Leaf 2 Networks**

<table>
<thead>
<tr>
<th>Network</th>
<th>Roles attached</th>
<th>Subnet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Provisioning / Ctlplane / Leaf2</td>
<td>ComputeLeaf2, CephStorageLeaf2</td>
<td>192.168.12.0/24</td>
</tr>
<tr>
<td>StorageLeaf2</td>
<td>ComputeLeaf2, CephStorageLeaf2</td>
<td>172.16.2.0/24</td>
</tr>
<tr>
<td>StorageMgmtLeaf2</td>
<td>CephStorageLeaf2</td>
<td>172.17.2.0/24</td>
</tr>
</tbody>
</table>
**1.3. SPINE-LEAF REQUIREMENTS**

To deploy the overcloud on a network with a layer-3 routed architecture, you must meet the following requirements:

**Layer-3 routing**

The network infrastructure must have routing configured to enable traffic between the different layer-2 segments. This can be statically or dynamically configured.

**DHCP-Relay**

Each layer-2 segment not local to the undercloud must provide `dhcp-relay`. You must forward DHCP requests to the undercloud on the provisioning network segment where the undercloud is connected.

**NOTE**

The undercloud uses two DHCP servers. One for baremetal node introspection, and another for deploying overcloud nodes. Make sure to read DHCP relay configuration to understand the requirements when configuring `dhcp-relay`.

**1.4. SPINE-LEAF LIMITATIONS**
- Some roles, such as the Controller role, use virtual IP addresses and clustering. The mechanism behind this functionality requires layer-2 network connectivity between these nodes. These nodes are all be placed within the same leaf.

- Similar restrictions apply to Networker nodes. The network service implements highly-available default paths in the network using Virtual Router Redundancy Protocol (VRRP). Since VRRP uses a virtual router IP address, you must connect master and backup nodes to the same L2 network segment.

- When using tenant or provider networks with VLAN segmentation, you must share the particular VLANs between all Networker and Compute nodes.

**NOTE**

It is possible to configure the network service with multiple sets of Networker nodes. Each set share routes for their networks, and VRRP would provide highly-available default paths within each set of Networker nodes. In such configuration all Networker nodes sharing networks must be on the same L2 network segment.
CHAPTER 2. CONFIGURING THE UNDERCLOUD

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

2.1. 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 as defined in the following procedure.

Procedure

1. Log into the undercloud as the stack user.

2. If you do not already have an undercloud.conf, copy the sample template file:

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

3. Edit your undercloud.conf.

4. 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 layer 2 segment in the routed spine and leaf:
      ```
      subnets = leaf0,leaf1,leaf2
      ```
   g. Specify the subnet associated with the physical layer 2 segment local to the undercloud using the local_subnet parameter:
h. Set the value of `undercloud_nameservers`.

```plaintext
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 per each subnet defined with the `subnets` parameter:

```plaintext
[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:

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

This creates three subnets on the provisioning network / 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. The next section provides some information on how to configure a DHCP relay.

### 2.2. CONFIGURING A DHCP RELAY

The undercloud uses two DHCP servers on the provisioning network:
• one for introspection.
• one for provisioning.

When configuring a DHCP relay make sure 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 devices 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 is available below. Please refer to manual page dhcrelay(8) for further details on how to use this implementation.

Broadcast DHCP relay

This method relays DHCP requests using UDP broadcast traffic onto the L2 network segment where the DHCP server(s) resides. 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 implementation this is typically configured by specifying either the interface or IP network address:

Interface
Specifying an interface connected to the L2 network segment where the DHCP requests are relayed.

IP network address
Specifying the network address of the IP network where the DHCP request are relayed.

Unicast DHCP relay

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

The interface used for introspection is the one defined as **inspection_interface** in **undercloud.conf**. 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 defined as **local_ip** in **undercloud.conf** is on the **br-ctlplane** interface.

The IP address allocated to the Neutron DHCP namespace is the first address available in the IP range configured for the local_subnet in undercloud.conf. The first address in the IP range is the one defined as **dhcp_start** in the configuration. For example: **192.168.10.10** would be the IP address when the following configuration is used:

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

The IP address for the DHCP namespace is automatically allocated. In most cases, it will be the first address in the IP range. Ensure sure to verify this is the case by running the following commands on the undercloud:

```bash
$ 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:

```bash
$ 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 our provisioning network.

Add the IP address of the leaf.

Forward UDP and BOOTP requests to the introspection DHCP server listening on IP address: 192.168.10.1.

Forward UDP and BOOTP requests to the provisioning DHCP server listening 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. You accomplish this with a series of configuration files.

### 2.3. CREATING FLAVORS AND TAGGING NODES FOR LEAF NETWORKS

Each role in each leaf network requires a flavor and role assignment so you can tag nodes into their respective leaf. This procedure shows how to create each flavor and assign them to a role.

**Procedure**

1. Source the `stackrc` file:
   
   ```
   $ 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 4096 --disk 40 --vcpus 1 $ROLE ; done
   $ for ROLE in $ROLES; do openstack flavor set --property "cpu_arch"="x86_64" --property "capabilities:boot_option"="local" --property "capabilities:profile"="$ROLE" --property resources:CUSTOM_BAREMETAL='1' --property resources:DISK_GB='0' --property resources:MEMORY_MB='0' --property resources:VCPU='0' $ROLE ; done
   ```

3. Tag nodes to their respective leaf networks. For example, run the following command to tag a node with UUID 58c3d07e-24f2-48a7-bbb6-6843f0e8ee13 to the compute role on Leaf2:
   
   ```
   $ openstack baremetal node set --property capabilities='profile:compute_leaf2,boot_option:local' 58c3d07e-24f2-48a7-bbb6-6843f0e8ee13
   ```

4. Create an environment file (```~/templates/node-data.yaml```) that contains the mapping of flavors to roles:
   
   ```
   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

You can also set the number of nodes to deploy in the overcloud using each respective *Count parameter.

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

To enable deployment onto a L3 routed network the bare metal ports must have its physical_network field configured. Each baremetal port is associated with a bare metal node in the OpenStack Bare Metal (ironic) service. The physical network names are the ones used in the subnets option in the undercloud configuration.

NOTE

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

Procedure

1. Source the stackrc file:

   $ source ~/stackrc

2. Check the bare metal nodes:

   $ openstack baremetal node list

3. Ensure 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 used to set the physical_network property on the baremetal port will fail. 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. For example:

   $ 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: \texttt{leaf0}, \texttt{leaf1}, and \texttt{leaf2}. The local\_subnet is \texttt{leaf0}. Since the physical network for the \texttt{local\_subnet} is always \texttt{ctlplane}, the baremetal port connected to \texttt{leaf0} uses \texttt{ctlplane}. The remaining ports use the other leaf names:

\begin{verbatim}
$ 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>
$ openstack baremetal port set --physical-network leaf2 <port-uuid>
\end{verbatim}

6. Make sure the nodes are in available state before deploying the overcloud:

\begin{verbatim}
$ openstack overcloud node provide --all-manageable
\end{verbatim}
CHAPTER 3. ALTERNATIVE PROVISIONING NETWORK METHODS

This section contains information about other methods to configure the provisioning network to accommodate routed spine-leaf with composable networks.

3.1. VLAN PROVISIONING NETWORK

In this example, the director deploys new overcloud nodes through the provisioning network and uses a VLAN tunnel across the layer 3 topology (see Figure 3.1, “VLAN provisioning network topology”). This allows the director’s DHCP servers to send DHCPOFFER broadcasts to any leaf. To establish this tunnel, trunk a VLAN between the Top-of-Rack (ToR) leaf switches. In this diagram, the StorageLeaf networks are presented to the Ceph storage and Compute nodes; the NetworkLeaf represents an example of any network you may want to compose.

Figure 3.1. VLAN provisioning network topology

3.2. VXLAN PROVISIONING NETWORK

In this example, the director deploys new overcloud nodes are deployed through the provisioning network and uses a VXLAN tunnel to span across the layer 3 topology (see Figure 3.2, “VXLAN provisioning network topology”). This allows the director’s DHCP servers to send DHCPOFFER broadcasts to any leaf. To establish this tunnel, configure VXLAN endpoints on the Top-of-Rack (ToR) leaf switches.
Figure 3.2. VXLAN provisioning network topology
CHAPTER 4. CONFIGURING THE OVERCLOUD

Now that you have configured the undercloud, you can configure the remaining overcloud leaf networks. You accomplish this with a series of configuration files. Afterwards, you deploy the overcloud and the resulting deployment has multiple sets of networks with routing available.

4.1. CREATING A NETWORK DATA FILE

To define the leaf networks, you create a network data file, which contain a YAML formatted list of each composable network and its attributes. This includes a subnets parameter to define the additional Leaf subnets with a base network.

Procedure

1. Create a new network_data_spine_leaf.yaml file in your stack user’s local directory. Use the default network_data_subnets_routed.yaml file as a basis:

   ```bash
   $ cp /usr/share/openstack-tripleo-heat-templates/network_data_subnets_routed.yaml /home/stack/network_data_spine_leaf.yaml
   ```

2. In the network_data_spine_leaf.yaml file, edit the YAML list to define each base network and respective leaf subnets as a composable network item. The following example is the syntax for defining a base leaf and two leaf subnets:

   ```yaml
   - name: <base_name>
     name_lower: <lowercase_name>
     vip: <true/false>
     vlan: '<vlan_id>'
     ip_subnet: '<network_address>/<prefix>'
     allocation_pools: [{'start': '<start_address>', 'end': '<end_address>'}]
     gateway_ip: '<router_ip_address>'
     subnets:
       <leaf_subnet_name>:
         vlan: '<vlan_id>'
         ip_subnet: '<network_address>/<prefix>'
         allocation_pools: [{'start': '<start_address>', 'end': '<end_address>'}]
         gateway_ip: '<router_ip_address>'
       <leaf_subnet_name>:
         vlan: '<vlan_id>'
         ip_subnet: '<network_address>/<prefix>'
         allocation_pools: [{'start': '<start_address>', 'end': '<end_address>'}]
         gateway_ip: '<router_ip_address>

   The following example demonstrates how to define the Internal API network and its leaf networks:

   ```yaml
   - name: InternalApi
     name_lower: internal_api
     vip: true
     vlan: 10
     ip_subnet: '172.18.0.0/24'
     allocation_pools: [{'start': '172.18.0.4', 'end': '172.18.0.250'}]
     gateway_ip: '172.18.0.1'
     subnets:
   ```
internal_api_leaf1:
  vlan: 11
  ip_subnet: '172.18.1.0/24'
  allocation_pools: [{'start': '172.18.1.4', 'end': '172.18.1.250'}]
  gateway_ip: '172.18.1.1'
internal_api_leaf2:
  vlan: 12
  ip_subnet: '172.18.2.0/24'
  allocation_pools: [{'start': '172.18.2.4', 'end': '172.18.2.250'}]
  gateway_ip: '172.18.2.1'

NOTE

You do not define the Control Plane networks in the network data file since the undercloud has already created these networks. However, you need to manually set the parameters so that the overcloud can configure its NICs accordingly.

NOTE

Define vip: true for the networks that contain the Controller-based services. In this example, InternalApiLeaf0 contains these services.

4.2. CREATING A ROLES DATA FILE

This section demonstrates how to define each composable role for each leaf and attach the composable networks to each respective role.

Procedure

1. Create a custom roles director in your stack user’s local directory:

   $ mkdir ~/roles

2. Copy the default Controller, Compute, and Ceph Storage roles from the director’s core template collection to the roles director. Rename the files for Compute and Ceph Storage to suit Leaf 0:

   $ cp /usr/share/openstack-tripleo-heat-templates/roles/Controller.yaml ~/roles/Controller.yaml
   $ cp /usr/share/openstack-tripleo-heat-templates/roles/Compute.yaml ~/roles/Compute0.yaml
   $ cp /usr/share/openstack-tripleo-heat-templates/roles/CephStorage.yaml ~/roles/CephStorage0.yaml

3. Copy the Leaf 0 Compute and Ceph Storage files as a basis for your Leaf 1 and Leaf 2 files:

   $ cp ~/roles/Compute0.yaml ~/roles/Compute1.yaml
   $ cp ~/roles/Compute0.yaml ~/roles/Compute2.yaml
   $ cp ~/roles/CephStorage0.yaml ~/roles/CephStorage1.yaml
   $ cp ~/roles/CephStorage0.yaml ~/roles/CephStorage2.yaml

4. Edit the name, HostnameFormatDefault, and deprecated_nic_config_name parameters in the Leaf 0, Leaf 1, and Leaf 2 files so that they align with the respective Leaf parameters. For example, the parameters in the Leaf 0 Compute file have the following values:
The Leaf 0 Ceph Storage parameters have the following values:

- name: CephStorageLeaf0
  HostnameFormatDefault: '%stackname%-cephstorage-leaf0-%index%'
  deprecated_nic_config_name: 'ceph-strorageleaf0.yaml'

5. Edit the `network` parameter in the Leaf 1 and Leaf 2 files so that they align with the respective Leaf network parameters. For example, the parameters in the Leaf 1 Compute file have the following values:

- name: ComputeLeaf1
  networks:
    InternalApi:
      subnet: internal_api_leaf1
    Tenant:
      subnet: tenant_leaf1
    Storage:
      subnet: storage_leaf1

The Leaf 1 Ceph Storage parameters have the following values:

- name: CephStorageLeaf1
  networks:
    Storage:
      subnet: storage_leaf1
    StorageMgmt:
      subnet: storage_mgmt_leaf1

NOTE

This applies only to Leaf 1 and Leaf 2. The `network` parameter for Leaf 0 retains the base subnet values, which are the lowercase names of each subnet combined with a `_subnet` suffix. For example, the Internal API for Leaf 0 is set to `internal_api_subnet`.

6. When your roles are ready, generate the full roles data file using the following command:

```
$ openstack overcloud roles generate --roles-path ~/roles -o roles_data_spine_leaf.yaml
```

This creates a full `roles_data_spine_leaf.yaml` file that includes all the custom roles for each respective leaf network.

Each role has its own NIC configuration. Before configuring the spine-leaf configuration, you need to create a base set of NIC templates to suit your current NIC configuration.

4.3. CREATING A CUSTOM NIC CONFIGURATION
Each role requires its own NIC configuration. This procedure shows how to create a copy of the base set of NIC templates and map them to their respective NIC configuration resources.

**Procedure**

1. Change to the core Heat template directory:

   ```bash
   $ cd /usr/share/openstack-tripleo-heat-templates
   ```

2. Render the Jinja2 templates using the `tools/process-templates.py` script, your custom `network_data` file, and custom `roles_data` file:

   ```bash
   $ tools/process-templates.py \
   -n /home/stack/network_data_spine_leaf.yaml \
   -r /home/stack/roles_data_spine_leaf.yaml \
   -o /home/stack/openstack-tripleo-heat-templates-spine-leaf
   ```

3. Change to the home directory:

   ```bash
   $ cd /home/stack
   ```

4. Copy the content from one of the default NIC templates to use as a basis for your spine-leaf templates. For example, copy the `single-nic-vlans`:

   ```bash
   ```

5. Edit each NIC configuration in `/home/stack/templates/spine-leaf-nics/` and change the location of the configuration script to an absolute location. Scroll to the network configuration section, which resembles the following snippet:

   ```json
   resources:
   OsNetConfigImpl:
     type: OS::Heat::SoftwareConfig
     properties:
       group: script
       config:
         str_replace:
           template:
             get_file: ../../scripts/run-os-net-config.sh
       params:
         $network_config:
           network_config:
   ```

   Change the location of the script to the absolute path:

   ```json
   resources:
   OsNetConfigImpl:
     type: OS::Heat::SoftwareConfig
     properties:
       group: script
       config:
         str_replace:
           template:
   ```
Make this change in each file for each Leaf and save the changes.

NOTE

For further NIC changes, see "Custom network interface templates" in the Advanced Overcloud Customization guide.

1. Create a file called `spine-leaf-nics.yaml` and edit the file.

2. Create a `resource_registry` section in the file and add a set of `*::Net::SoftwareConfig` resources that map to their respective NIC templates:

   ```yaml
   resource_registry:
   OS::TripleO::Controller::Net::SoftwareConfig: /home/stack/templates/spine-leaf-nics/controller.yaml
   OS::TripleO::ComputeLeaf0::Net::SoftwareConfig: /home/stack/templates/spine-leaf-nics/computeleaf0.yaml
   OS::TripleO::ComputeLeaf1::Net::SoftwareConfig: /home/stack/templates/spine-leaf-nics/computeleaf1.yaml
   OS::TripleO::ComputeLeaf2::Net::SoftwareConfig: /home/stack/templates/spine-leaf-nics/computeleaf2.yaml
   OS::TripleO::CephStorageLeaf0::Net::SoftwareConfig: /home/stack/templates/spine-leaf-nics/ceph-storageleaf0.yaml
   OS::TripleO::CephStorageLeaf1::Net::SoftwareConfig: /home/stack/templates/spine-leaf-nics/ceph-storageleaf1.yaml
   OS::TripleO::CephStorageLeaf2::Net::SoftwareConfig: /home/stack/templates/spine-leaf-nics/ceph-storageleaf2.yaml
   ``

   These resources mappings will override the default resource mappings during deployment.

3. Save the `spine-leaf-nics.yaml` file.

4. Remove the rendered template directory:

   ```
   $ rm -rf openstack-tripleo-heat-templates-spine-leaf
   ```

As a result of this procedure, you now have a set of NIC templates and an environment file that maps the required `*::Net::SoftwareConfig` resources to them. When you eventually run the `openstack overcloud deploy` command, ensure that you include the environment files in the following order:

1. `/usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml`, which enables network isolation. Note that the director renders this file from the `network-isolation.j2.yaml` Jinja2 template.

2. `/usr/share/openstack-tripleo-heat-templates/environments/network-environment.yaml`, which is the default network environment file, including default NIC resource mappings. Note that the director renders this file from the `network-environment.j2.yaml` Jinja2 template.
3. `/home/stack/templates/spine-leaf-nics.yaml`, which contains your custom NIC resource mappings and overrides the default NIC resource mappings.

The following command snippet demonstrates the ordering:

```bash
$ openstack overcloud deploy --templates...
  -e /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/network-environment.yaml \
  -e /home/stack/templates/spine-leaf-nics.yaml \
  ...
```

**Resources**

- See "Custom network interface templates" in the *Advanced Overcloud Customization* guide for more information on customizing your NIC templates.

The next couple of sections add details to your network environment file to configure certain aspects of the spine leaf architecture. Once complete, you include this file with your `openstack overcloud deploy` command.

### 4.4. SETTING CONTROL PLANE PARAMETERS

You usually define networking details for isolated spine-leaf networks using a `network_data` file. The exception is the control plane network, which the undercloud created. However, the overcloud requires access to the control plane for each leaf. This requires some additional parameters, which you define in your `network-environment.yaml` file.

In this instance, we need to define the IP, subnet, and default route for the respective Control Plane network on Leaf 0.

**Procedure**

1. Create a file called `spine-leaf-ctlplane.yaml` and edit the file.

2. Create a `parameter_defaults` section in the file and add the control plane subnet mapping for each spine-leaf network:

   ```yaml
   parameter_defaults:
     ...
     ControllerControlPlaneSubnet: leaf0
     Compute0ControlPlaneSubnet: leaf0
     Compute1ControlPlaneSubnet: leaf1
     Compute2ControlPlaneSubnet: leaf2
     CephStorage0ControlPlaneSubnet: leaf0
     CephStorage1ControlPlaneSubnet: leaf1
     CephStorage2ControlPlaneSubnet: leaf2
   ```


### 4.5. SETTING THE SUBNET FOR VIRTUAL IP ADDRESSES

The Controller role typically hosts virtual IP (VIP) addresses for each network. By default, the overcloud takes the VIPs from the base subnet of each network except for the control plane. The control plane
uses **ctlplane-subnet**, which is the default subnet name created during a standard undercloud installation.

In this spine leaf scenario, the default base provisioning network is **leaf0** instead of **ctlplane-subnet**. This means you must add overriding values to the **VipSubnetMap** parameter to change the subnet that the control plane VIP uses.

In addition, if the VIPs for each network are not using the base subnet of one or more networks, you must add additional overrides to the **VipSubnetMap** parameter to ensure the director creates VIPs on the subnet associated with the L2 network segment connecting the Controller nodes.

**Procedure:**

1. Create a file called **spine-leaf-vips.yaml** and edit the file.

2. Create a **parameter_defaults** section in the file and add the **VipSubnetMap** parameter based on your requirements:
   - If you use **leaf0** for the provisioning / control plane network, set the **ctlplane** VIP remapping to **leaf0**:
     ```yml
     parameter_defaults:
     VipSubnetMap:
     ctlplane: leaf0
     ```
   - If you use a different Leaf for multiple VIPs, set the VIP remapping to suit these requirements. The following example changes the **VipSubnetMap** parameter to use **leaf1** for all VIPs:
     ```yml
     parameter_defaults:
     VipSubnetMap:
     ctlplane: leaf1
     redis: internal_api_leaf1
     InternalApi: internal_api_leaf1
     Storage: storage_leaf1
     StorageMgmt: storage_mgmt_leaf1
     ```

3. Save the **spine-leaf-vips.yaml** file.

**4.6. MAPPING SEPARATE NETWORKS**

By default, OpenStack Platform uses Open Virtual Network (OVN), which requires all Controller and Compute nodes to connect to a single L2 network for external network access. This means both Controller and Compute network configurations use a **br-ex** bridge, which director maps to the **datacentre** network in the overcloud by default. This mapping is usually either for a flat network mapping or a VLAN network mapping. In a spine leaf architecture, you can change these mappings so that each Leaf routes traffic through the specific bridge or VLAN on that Leaf, which is often the case with edge computing scenarios.

**Procedure**

1. Create a file called **spine-leaf-separate.yaml** and edit the file.

2. Create a **parameter_defaults** section in the file and the external network mapping for each spine-leaf network:
For flat network mappings, list each Leaf in the `NeutronFlatNetworks` parameter and set the `NeutronBridgeMappings` parameter for each Leaf:

```
parameter_defaults:
  NeutronFlatNetworks: leaf0,leaf1,leaf2
  Controller0Parameters:
    NeutronBridgeMappings: "leaf0:br-ex"
  Compute0Parameters:
    NeutronBridgeMappings: "leaf0:br-ex"
  Compute1Parameters:
    NeutronBridgeMappings: "leaf1:br-ex"
  Compute2Parameters:
    NeutronBridgeMappings: "leaf2:br-ex"
```

For VLAN network mappings, additionally set the `NeutronNetworkVLANRanges` to map VLANs for all three Leaf networks:

```
NeutronNetworkType: 'geneve,vlan'
NeutronNetworkVLANRanges: 'leaf0:1:1000,leaf1:1:1000,leaf2:1:1000'
```

3. Save the `spine-leaf-separate.yaml` file.

### 4.7. DEPLOYING A SPINE-LEAF ENABLED OVERCLOUD

All our files are now ready for our deployment. This section provides a review of each file and the deployment command:

**Procedure**

1. Review the `/home/stack/template/network_data_spine_leaf.yaml` file and ensure it contains each network and subnet for each leaf.

   **NOTE**
   
   There is currently no validation performed for the network subnet and `allocation_pools` values. Be certain you have defined these consistently and there is no conflict with existing networks.

2. Review the `/home/stack/templates/roles_data_spine_leaf.yaml` values and ensure you have defined a role for each leaf.

3. Review the NIC templates contained in `~/templates/spine-leaf-nics/` and ensure the interfaces for each role on each leaf are correctly defined.

4. Review the custom `spine-leaf-nics.yaml` environment file and ensure it contains a `resource_registry` section that references the custom NIC templates for each role.

5. Review the `/home/stack/templates/nodes_data.yaml` file and ensure all roles have an assigned flavor and a node count. Also check that you have correctly tagged all nodes for each leaf.

6. Run the `openstack overcloud deploy` command to apply the spine leaf configuration. For example:

   ```
   $ openstack overcloud deploy --templates \\
   ```
-n /home/stack/templates/network_data_spine_leaf.yaml \
-r /home/stack/templates/roles_data_spine_leaf.yaml \
-e /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml \
-e /usr/share/openstack-tripleo-heat-templates/environments/network-environment.yaml \
-e /home/stack/templates/spine-leaf-nics.yaml \
-e /home/stack/templates/spine-leaf-ctlplane.yaml \
-e /home/stack/templates/spine-leaf-vips.yaml \
-e /home/stack/templates/spine-leaf-separate.yaml \
-e /home/stack/templates/nodes_data.yaml \
-e [OTHER ENVIRONMENT FILES]

- The **network-isolation.yaml** is the rendered name of the Jinja2 file in the same location (network-isolation.j2.yaml). Include this file to ensure the director isolates each networks to its correct leaf. This ensures the networks are created dynamically during the overcloud creation process.

- Include the **network-environment.yaml** file after the **network-isolation.yaml**. The **network-environment.yaml** file provides the default network configuration for composable network parameters.

- Include the **spine-leaf-nics.yaml** file after the **network-environment.yaml**. The **spine-leaf-nics.yaml** file overrides the default NIC template mappings from the **network-environment.yaml** file.

- If you created any other spine leaf network environment files, include these environment files after the **spine-leaf-nics.yaml** file.

- Add any additional environment files. For example, an environment file with your container image locations or Ceph cluster configuration.

7. Wait until the spine-leaf enabled overcloud deploys.