Red Hat OpenStack Platform 16.0

Spine Leaf Networking

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

Last Updated: 2020-03-26

Red Hat OpenStack Platform 16.0 Spine Leaf Networking

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

OpenStack Team
rhos-docs@redhat.com

Abstract

This guide provides a basic scenario about how to configure a routed spine-leaf network on the overcloud. This includes configuring the undercloud, writing the main configuration files, and creating roles for your nodes.

Table of Contents

CHAPTER 1. INTRODUCTION

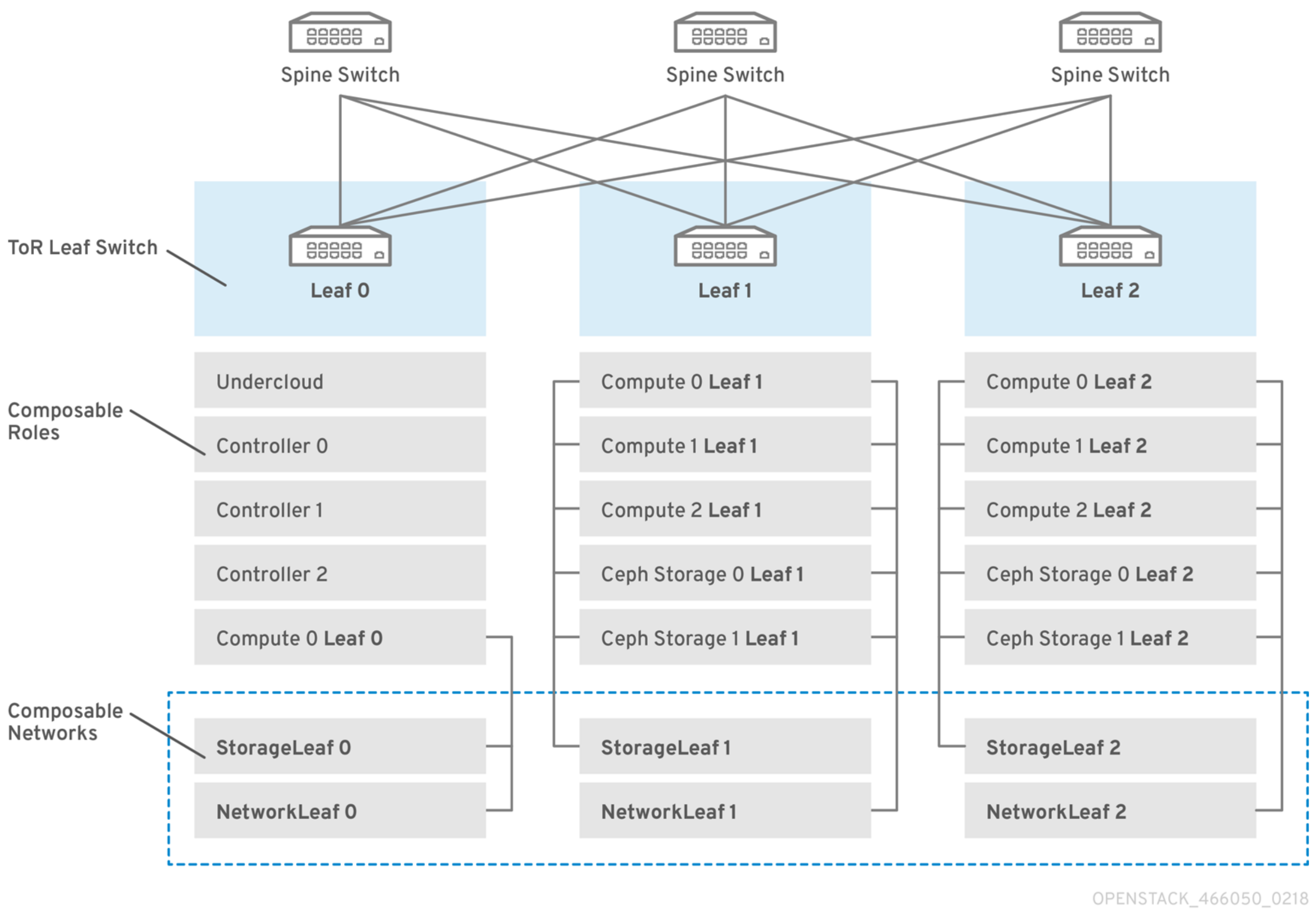
This guide provides information about constructing a spine-leaf network topology for your Red Hat OpenStack Platform environment. This includes a full end-to-end scenario and example files to help replicate a more extensive network topology within your own environment.

1.1. SPINE-LEAF NETWORKING

Red Hat OpenStack Platform has a composable network architecture that you can use to adapt your networking to the 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, Controller nodes, and Compute nodes. The composable networks are presented to the nodes, which have been assigned to composable roles. The following diagram contains the following configuration:

- 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



1.2. SPINE-LEAF NETWORK TOPOLOGY

The spine-leaf 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 of the main network.

This scenario uses the following networks:

Table 1.1. Leaf 0 Networks (base networks)

Network	Roles attached	Subnet
Provisioning / Ctlplane / Leaf0	Controller, ComputeLeaf0, CephStorageLeaf0	192.168.10.0/24
Storage	Controller, ComputeLeaf0, CephStorageLeaf0	172.16.0.0/24
StorageMgmt	Controller, CephStorageLeaf0	172.17.0.0/24
InternalApi	Controller, ComputeLeaf0	172.18.0.0/24
Tenant	Controller, ComputeLeaf0	172.19.0.0/24
External	Controller, ComputeLeaf0	10.1.1.0/24

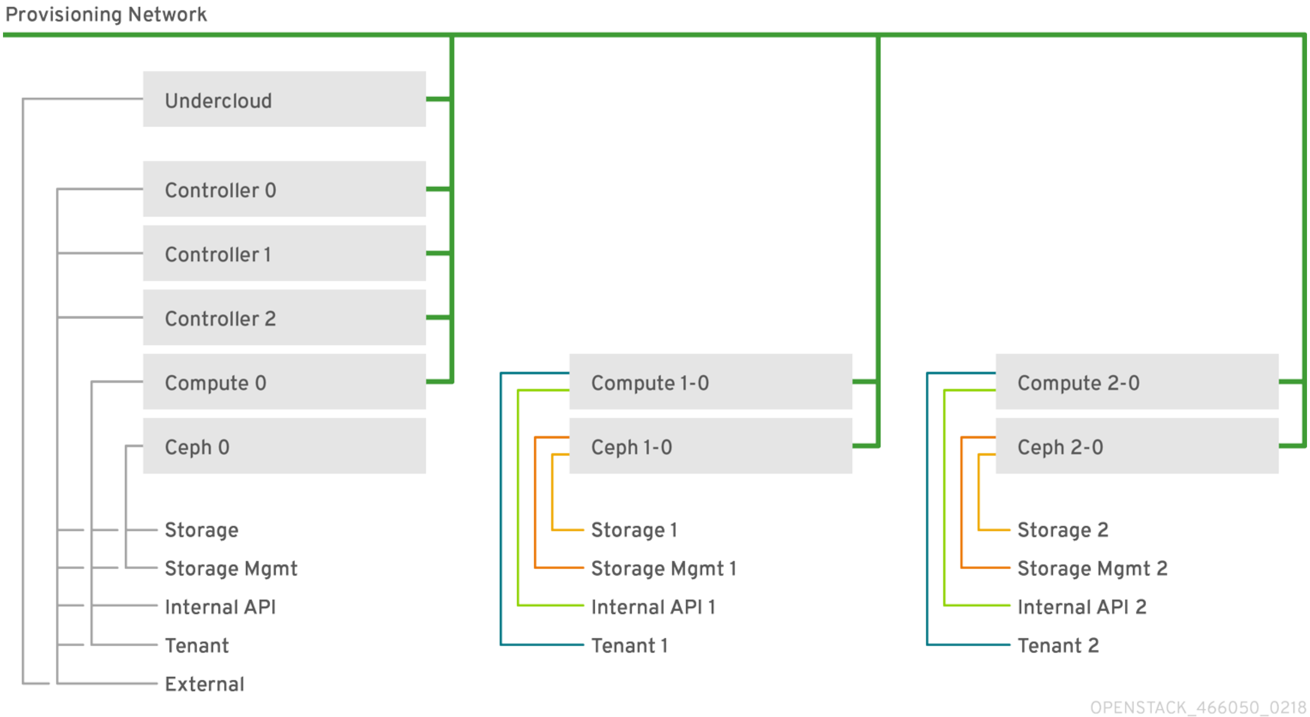
Table 1.2. Leaf 1 Networks

Network	Roles attached	Subnet
Provisioning / Ctlplane / Leaf1	ComputeLeaf1, CephStorageLeaf1	192.168.11.0/24
StorageLeaf1	ComputeLeaf1, CephStorageLeaf1	172.16.1.0/24
StorageMgmtLeaf1	CephStorageLeaf1	172.17.1.0/24
InternalApiLeaf1	ComputeLeaf1	172.18.1.0/24
TenantLeaf1	ComputeLeaf1	172.19.1.0/24

Table 1.3. Leaf 2 Networks

Network	Roles attached	Subnet
Provisioning / Ctlplane / Leaf2	ComputeLeaf2, CephStorageLeaf2	192.168.12.0/24
StorageLeaf2	ComputeLeaf2, CephStorageLeaf2	172.16.2.0/24
StorageMgmtLeaf2	CephStorageLeaf2	172.17.2.0/24

Network	Roles attached	Subnet
InternalApiLeaf2	ComputeLeaf2	172.18.2.0/24
TenantLeaf2	ComputeLeaf2	172.19.2.0/24



1.3. SPINE-LEAF REQUIREMENTS

To deploy the overcloud on a network with a L3 routed architecture, complete the following prerequisite steps:

Layer-3 routing

Configure the routing of the network infrastructure to enable traffic between the different L2 segments. You can configure this routing statically or dynamically.

DHCP-Relay

Each L2 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. Ensure that you read DHCP relay configuration to understand the requirements when you configure **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 L2 network connectivity between these nodes. You must place these nodes within the same leaf.
- Similar restrictions apply to Networker nodes. The network service implements highly-available default paths in the network with Virtual Router Redundancy Protocol (VRRP). Because VRRP uses a virtual router IP address, you must connect master and backup nodes to the same L2 network segment.
- When you use 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 of Networker nodes share routes for their networks, and VRRP provides highly-available default paths within each set of Networker nodes. In this type of configuration, all Networker nodes that share networks must be on the same L2 network segment.

CHAPTER 2. CONFIGURING ROUTED SPINE-LEAF IN THE UNDERCLOUD

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

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 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.

2.2. 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
!
```

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

2.3. CREATING FLAVORS AND TAGGING NODES FOR LEAF NETWORKS

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

Procedure

1. Source the **stackrc** file:

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

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

Procedure

1. Source the **stackrc** file:

```
$ source ~/stackrc
```

2. Check the bare metal nodes:

```
$ openstack baremetal node list
```

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

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

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

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

5. Set the **physical-network** parameter for the ports. In the example below, three subnets are

defined in the configuration: **leaf0**, **leaf1**, and **leaf2**. The local_subnet is **leaf0**. Because the physical network for the **local_subnet** is always **ctlplane**, the baremetal port connected to **leaf0** uses ctlplane. The remaining ports use the other leaf names:

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

6. Ensure that the nodes are in available state before deploying the overcloud:

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

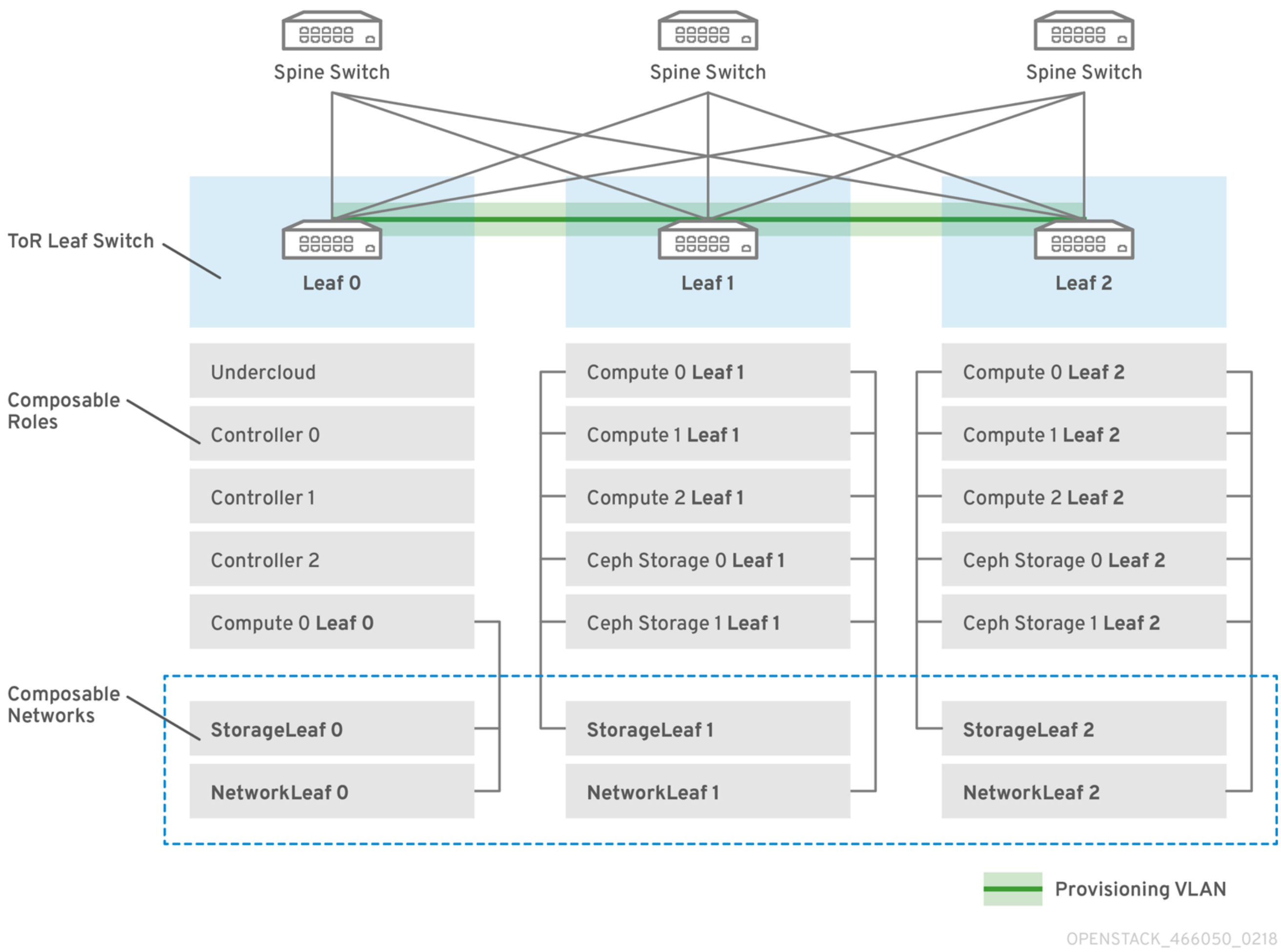
CHAPTER 3. ALTERNATIVE PROVISIONING NETWORK METHODS

This section contains information about other methods that you can use 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 L3 topology. For more information, see Figure 3.1, "VLAN provisioning network topology". If you use a VLAN provisioning network, the director DHCP servers can send **DHCPOFFER** broadcasts to any leaf. To establish this tunnel, trunk a VLAN between the Top-of-Rack (ToR) leaf switches. In the following diagram, the **StorageLeaf** networks are presented to the Ceph storage and Compute nodes; the **NetworkLeaf** represents an example of any network that you want to compose.

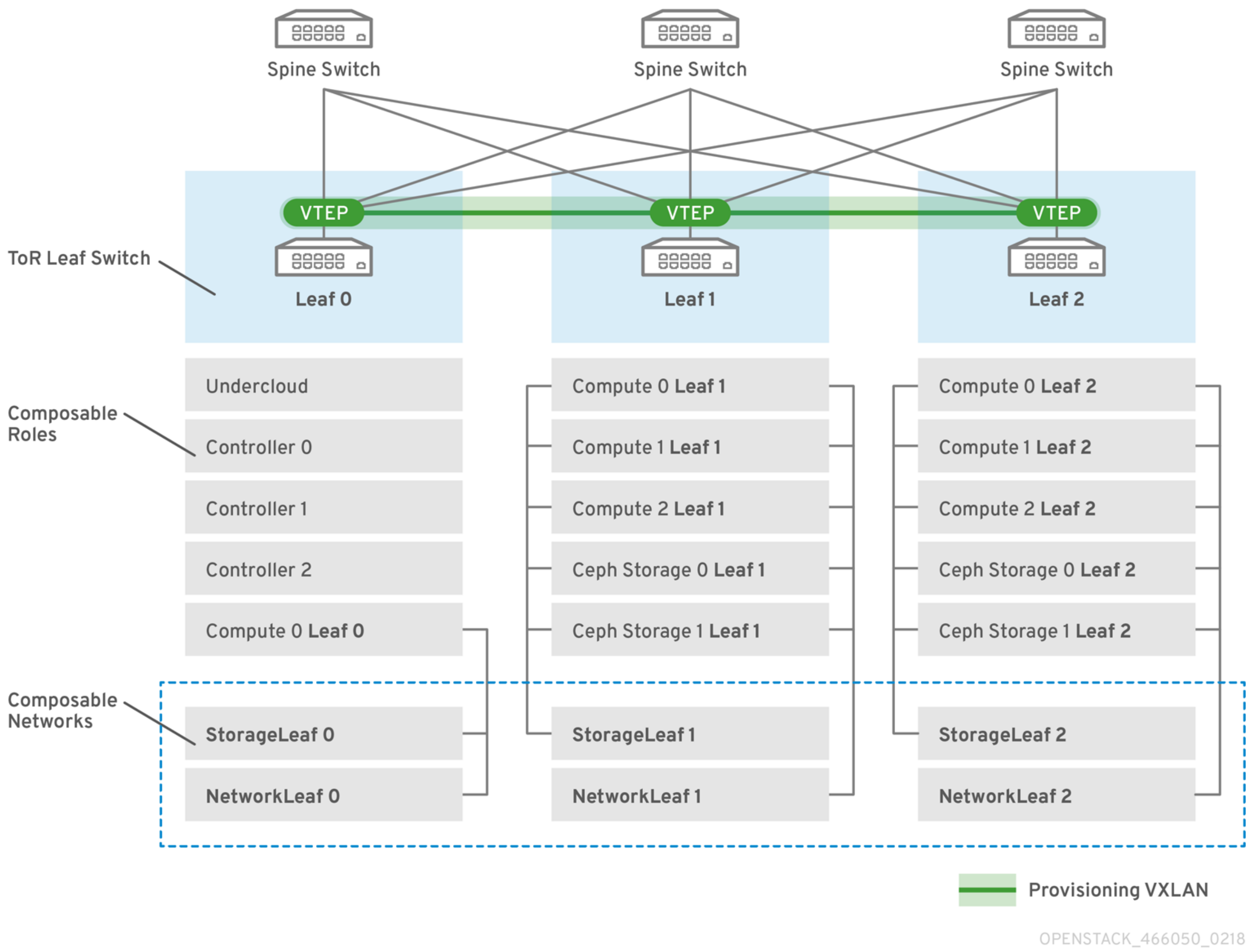
Figure 3.1. VLAN provisioning network topology



3.2. VXLAN PROVISIONING NETWORK

In this example, the director deploys new overcloud nodes through the provisioning network and uses a VXLAN tunnel to span across the layer 3 topology. For more information, see Figure 3.2, "VXLAN provisioning network topology". If you use a VXLAN provisioning network, the director DHCP servers can 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

After you configure the undercloud, you can configure the remaining overcloud leaf networks with a series of configuration files. After you configure the remaining overcloud leaf networks and deploy the overcloud, the resulting deployment has multiple sets of networks with routing available.

4.1. CREATING A NETWORK DATA FILE

To define the leaf networks, create a network data file that contains a YAML formatted list of each composable network and its attributes. Use the **subnets** parameter to define the additional Leaf subnets with a base network.

Procedure

1. Create a new **network_data_spine_leaf.yaml** file in the home directory of the **stack** user. Use the default **network_data_subnets_routed.yaml** file as a basis:

```
$ 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. Use the following example syntax to define a base leaf and two leaf subnets:

```
- 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:

```
- 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 because the undercloud has already created these networks. However, you must set the parameters manually so that the overcloud can configure the 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

To define each composable role for each leaf and attach the composable networks to each respective role, complete the following steps.

Procedure

1. Create a custom **roles** directory in the home directory of the **stack** user:

```
$ mkdir ~/roles
```

2. Copy the default Controller, Compute, and Ceph Storage roles from the director core template collection to the roles directory. 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:

```
- name: ComputeLeaf0
  HostnameFormatDefault: '%stackname%-compute-leaf0-%index%'
  deprecated_nic_config_name: 'computeleaf0.yaml'
```

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 **internal_api_subnet**.

6. When your role configuration is complete, run the following command to generate the full roles data file:

```
$ openstack overcloud roles generate --roles-path ~/roles -o roles_data_spine_leaf.yaml
Controller Compute Compute1 Compute2 CephStorage CephStorage1 CephStorage2
```

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

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

4.3. CREATING A CUSTOM NIC CONFIGURATION

Each role requires a unique NIC configuration. Complete the following steps to create a copy of the base set of NIC templates and map the new templates to the respective NIC configuration resources.

Procedure

1. Change to the core heat template directory:

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

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

```
$ 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:

```
$ 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** NIC template:

```
$ cp -r openstack-tripleo-heat-templates-spine-leaf/network/config/single-nic-vlans/*
/home/stack/templates/spine-leaf-nics/.
```

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:

```
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:

```
resources:
  OsNetConfigImpl:
    type: OS::Heat::SoftwareConfig
    properties:
      group: script
      config:
        str_replace:
          template:
```

```
            get_file: /usr/share/openstack-tripleo-heat-templates/network/scripts/run-os-net-config.sh
          params:
            $network_config:
              network_config:
```

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.

6. Create a file called **spine-leaf-nics.yaml** and edit the file.

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

```
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 override the default resource mappings during deployment.

8. Save the **spine-leaf-nics.yaml** file.

9. 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:

```
$ 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

- For more information about customizing your NIC templates, see "Custom network interface templates" in the *Advanced Overcloud Customization* guide .

Complete the procedures in the following sections to add details to your network environment file and define certain aspects of the spine leaf architecture. After you complete this configuration, include this file in the **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 creates. However, the overcloud requires access to the control plane for each leaf. To enable this access, you must include additional parameters in your **network-environment.yaml** file.

In this example, 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:

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

3. Save the **network-environment.yaml** file.

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 that you must add overriding values to the **VipSubnetMap** parameter to change the subnet that the control plane VIP uses.

Additionally, if the VIPs for each network do not use the base subnet of one or more networks, you must add additional overrides to the **VipSubnetMap** parameter to ensure that the director creates VIPs on the subnet associated with the L2 network segment that connects 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**:

     ```
     parameter_defaults:
       VipSubnetMap:
         ctlplane: leaf0
     ```

   - If you use a different Leaf for multiple VIPs, set the VIP remapping to suit these requirements. For example, use the following snippet to configure the **VipSubnetMap** parameter to use **leaf1** for all VIPs:

     ```
     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 that all Controller and Compute nodes connect to a single L2 network for external network access. This means that 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 **spine-leaf-separate.yaml** file and include 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

When you have completed your spine-leaf overcloud configuration, complete the following steps to review each file and then run the deployment command:

Procedure

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

> **NOTE**
>
> There is currently no automatic validation for the network subnet and **allocation_pools** values. Ensure that you define these values consistently and that there is no conflict with existing networks.

2. Review the **/home/stack/templates/roles_data_spine_leaf.yaml** values and ensure that you define a role for each leaf.

3. Review the NIC templates in the **~/templates/spine-leaf-nics/** directory and ensure that you define the interfaces for each role on each leaf correctly.

4. Review the custom **spine-leaf-nics.yaml** environment file and ensure that 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 that 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 in the deployment command to ensure that the director isolates each networks to the correct leaf. This ensures that 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.