Red Hat Enterprise Linux OpenStack Platform 7
IPv6 Networking for the Overcloud

Configuring an Overcloud to Use IPv6 Networking

OpenStack Team

Red Hat Enterprise Linux OpenStack Platform 7 IPv6 Networking for the Overcloud

Configuring an Overcloud to Use IPv6 Networking

OpenStack Team
rhos-docs@redhat.com

Legal Notice

Copyright © 2017 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at
http://creativecommons.org/licenses/by-sa/3.0/
. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat Software Collections is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Abstract

This guide provides information on using the Red Hat OpenStack Platform director create an Overcloud that uses IPv6 for endpoints. This includes information on how the director deploys an IPv6-based Overcloud and the configuration options to acheive this.

Table of Contents

CHAPTER 1. INTRODUCTION **3**
1.1. DEFINING IPV6 NETWORKING 3
1.2. USING IPV6 IN RED HAT ENTERPRISE LINUX OPENSTACK PLATFORM 5
1.3. SETTING REQUIREMENTS 7
1.4. DEFINING THE SCENARIO 7

CHAPTER 2. CONFIGURING THE OVERCLOUD BEFORE CREATION **9**
2.1. INITIALIZING THE STACK USER 9
2.2. CONFIGURING AN IPV6 ADDRESS ON THE UNDERCLOUD 9
2.3. SETTING UP YOUR ENVIRONMENT 10
2.4. CONFIGURING THE NETWORK 14
2.5. COMPLETING OVERCLOUD CONFIGURATION 18

CHAPTER 3. CREATING THE OVERCLOUD **19**
3.1. ACCESSING THE OVERCLOUD 20

CHAPTER 4. CONFIGURING THE OVERCLOUD AFTER CREATION **21**
4.1. CREATING THE OVERCLOUD TENANT NETWORK 21
4.2. CREATING THE OVERCLOUD PUBLIC NETWORK 21

CHAPTER 5. CONCLUSION **22**

CHAPTER 1. INTRODUCTION

Red Hat Enterprise Linux OpenStack Platform director creates a cloud environment called the **Overcloud**. As a default, the Overcloud uses Internet Protocol version 4 (IPv4) to configure the service endpoints. However, the Overcloud also supports Internet Protocol version 6 (IPv6) endpoints, which is useful for organizations that support IPv6 infrastructure. This guide provides information and a configuration example for using IPv6 in your Overcloud.

1.1. DEFINING IPV6 NETWORKING

IPv6 is the latest version of the Internet Protocol standard. Internet Engineering Task Force (IETF) developed IPv6 as a means to combat the exhaustion of IP address from the current common IPv4 standard. IPv6 has various differences from IPv4 including:

Large IP Address Range

The IPv6 range is much larger than the IPv4 range.

Better End-to-End Connectivity

The larger IP range provides better end-to-end connectivity due to less reliance on network address translation.

No Broadcasting

IPv6 does not support traditional IP broadcasting. Instead, IPv6 uses multicasting to send packets to applicable hosts in a hierarchical manner.

Stateless Address Autoconfiguration (SLAAC)

IPv6 provides features for automatically configuring IP addresses and detecting duplicate addresses on a network. This reduces the reliance on a DHCP server to assign addresses.

IPv6 uses 128 bits (represented with 4 hexadecimals using groups of 16 bits) to define addresses while IPv4 only uses only 32 bits (represented with decimal digits using groups of 8 bits). For example, a representation of an IPv4 address (192.168.0.1) looks like this:

Bits	Representation
11000000	192
10101000	168
00000000	0
00000001	1

For an IPv6 address (2001:db8:88ec:9fb3::1), the representation looks like this:

Bits	Representation
0010 0000 0000 0001	2001
0000 1101 1011 1000	0db8
1000 1000 1110 1100	88ec
1001 1111 1011 0011	9fb3
0000 0000 0000 0000	0000
0000 0000 0000 0000	0000
0000 0000 0000 0000	0000
0000 0000 0000 0001	0001

Notice you can also represent IPv6 addresses without leading zeros in each bit group and omit a set of zero bit groups once per IP address. In our example, you can represent the 0683 bit grouping as just 683 and omit the three sets of 0000 bit groups, which shortens the representation from 2001:0db8:88ec:9fb3:0000:0000:0000:0001 to 2001:db8:88ec:9fb3::1.

Subnetting in IPv6

Similar to IPv4, an IPv6 address uses a bit mask to define the address prefix as its network. For example, if you include a /64 bit mask to our sample IP address (e.g. 2001:db8:88ec:9fb3::1/64) the bit mask acts as a prefix that defines the first 64 bits (2001:db8:88ec:9fb3) as the network. The remaining bits (0000:0000:0000:0001) define the host.

IPv6 also uses some special address types, including:

Loopback

The loopback device uses an IPv6 for the internal communication within the host. This device is always ::1/128.

Link Local

A link local address is an IP address valid within a particular network segment. IPv6 requires each network device to have a link local address and use the prefix fe80::/10. However, most of the time, these addresses are prefixed with fe80::/64.

Unique local

A unique local address is intended for local communication. These addresses use a fc00::/7 prefix.

Multicast

Hosts use multicast addresses to join multicast groups. These addresses use a ff00::/8 prefix. For example, FF02::1 is a multicast group for all nodes on the network and FF02::2 is a multicast group for all routers.

Global Unicast

These addresses are usually reserved for public IP address. These addresses use a 2000::/3 prefix.

1.2. USING IPV6 IN RED HAT ENTERPRISE LINUX OPENSTACK PLATFORM

Red Hat Enterprise Linux OpenStack Platform director provides a method for mapping OpenStack services to isolated networks. These networks include:

- Internal API
- Storage
- Storage Management
- Tenant Networks (Neutron VLAN mode)
- External

For more information about these network traffic types, see Section 4.2. Planning Networks in the *Red Hat Enterprise Linux OpenStack Platform 7 Director Installation and Usage* guide.

Red Hat Enterprise Linux OpenStack Platform director also provides methods to use IPv6 communication for these networks. This means the required OpenStack services, databases, and other related services use IPv6 addresses to communicate. This also applies to environments using a high availability solution involving multiple Controller nodes. This helps organizations integrate Red Hat OpenStack Platform with their IPv6 infrastructure.

Use the following table as a guide for what networks support IPv6 in Red Hat Enterprise Linux OpenStack Platform:

Network Type	Dual Stack (IPv4/v6)	Single Stack (IPv6)	Single Stack (IPv4)	Notes
Internal API		Yes	Yes	
Storage		Yes	Yes	
Storage Management		Yes	Yes	

Network Type	Dual Stack (IPv4/v6)	Single Stack (IPv6)	Single Stack (IPv4)	Notes
Tenant Networks	Yes	Yes	Yes	
Tenant Network Endpoints			Yes	This refers to the IP address of the network hosting the tenant network tunnels, not the tenant networks themselves
External - Public API (and Horizon)		Yes	Yes	
External - Floating IPs	Yes [1]	Yes [1]	Yes	
Provider Networks	Yes	Yes	Yes	IPv6 support is dependent on tenant OS
Provisioning (PXE/DHCP)			Yes	Interfaces on this network are IPv4 only.
IPMI or other BMC			Yes	OpenStack Platform communicates with BMC interfaces over the Provisioning network, which is IPv4. However, if BMC interfaces support dual-stack IPv4/IPv6, non-OpenStack Platform tools can use IPv6 to communicate with the BMCs.

Network Type	Dual Stack (IPv4/v6)	Single Stack (IPv6)	Single Stack (IPv4)	Notes
Overcloud Provisioning network				Provisioning network for Ironic in the overcloud
Overcloud Cleaning network				Isolated network to clean a machine before it's ready for reuse

[1] Neutron tenant networks that are assigned Global Unicast Address (GUA) prefixes and addresses don't require NAT on the neutron router external gateway port to access the outside world.

1.3. SETTING REQUIREMENTS

This guide acts as supplementary information for the Red Hat Enterprise Linux OpenStack Platform 7 Director Installation and Usage guide. This means the same requirements specified in Chapter 2. Requirements also apply to this guide. Implement these requirements as necessary.

This guide also requires the following:

- An Undercloud host with the Red Hat Enterprise Linux OpenStack Platform director installed. See Chapter 3. Installing the Undercloud.
- Your network supports IPv6-native VLANs as well as IPv4-native VLANs. Both will be used in the deployment.

1.4. DEFINING THE SCENARIO

The scenario for this guide is to create an Overcloud with an isolated network that uses IPv6. The guide aims to achieve this objective through network isolation configured using Heat templates and environment files. This scenario also provides certain variants to these Heat templates and environment files to demonstrate specific differences in configuration.

Note

In this scenario, the Undercloud still uses IPv4 connectivity for PXE boot, introspection, deployment, and other services.

This guide uses a scenario similar to the Advanced Overcloud scenario in the Red Hat Enterprise Linux OpenStack Platform 7 Director Installation and Usage guide. The main difference is the omission of the Ceph Storage nodes.

For more information about this scenario, see Section 6.3. Scenario 3: Using the CLI to Create an Advanced Overcloud with Ceph Nodes.

Important

This guide uses the 2001:DB8::/32 IPv6 prefix for documentation purposes as defined in RFC 3849. Make sure to substitute these example addresses for IPv6 addresses from your own network.

CHAPTER 2. CONFIGURING THE OVERCLOUD BEFORE CREATION

The following chapter provides the configuration required before running the **`openstack overcloud deploy`** command. This includes preparing nodes for provisioning, configuring an IPv6 address on the Undercloud, and creating a network environment file that defines the IPv6 parameters for the Overcloud.

2.1. INITIALIZING THE STACK USER

Log into the director host as the **`stack`** user and run the following command to initialize your director configuration:

```
$ source ~/stackrc
```

This sets up environment variables containing authentication details to access the director's CLI tools.

2.2. CONFIGURING AN IPV6 ADDRESS ON THE UNDERCLOUD

The Undercloud requires access to the Overcloud's Public API, which is on the External network. To accomplish this, the Undercloud host requires an IPv6 address on the interface accessing the External network.

Note

The Provisioning network still requires IPv4 connectivity for every node. The Undercloud and the Overcloud nodes use this network for PXE boot, introspection, and deployment. In addition, the nodes use this network to access DNS and NTP services over IPv4.

Native VLAN or Dedicated Interface

If the Undercloud uses a native VLAN or a dedicated interface attached to the External network, use the **`ip`** command to add an IPv6 address to the interface. In this example, the dedicated interface is **`eth0`**:

```
$ sudo ip link set dev eth0 up; sudo ip addr add 2001:db8::1/64 dev dev
eth0
```

Trunked VLAN Interface

If the Undercloud uses a trunked VLAN on the same interface as the control plane bridge (**`br-ctlplane`**) to access the External network, create a new VLAN interface, attach it to the control plane, and add an IPv6 address to the VLAN. For example, our scenario uses 100 for the External network's VLAN ID:

```
$ sudo ovs-vsctl add-port br-ctlplane vlan100 tag=100 -- set interface
vlan100 type=internal
$ sudo ip l set dev vlan100 up; sudo ip addr add 2001:db8::1/64 dev
vlan100
```

Confirming the IPv6 Address

Confirm the addition of the IPv6 address with the **ip** command:

```
$ ip addr
```

The IPv6 address appears on the chosen interface.

Setting a Persistent IPv6 Address

In addition to the above, you might want to make the IPv6 address permanent. In this case, modify or create the appropriate interface file in **/etc/sysconfig/network-scripts/** (In our example, either **ifcfg-eth0** or **ifcfg-vlan100**). Include the following lines:

```
IPV6INIT=yes
IPV6ADDR=2001:db8::1/64
```

For more information, see How do I configure a network interface for IPv6? on the Red Hat Customer Portal.

2.3. SETTING UP YOUR ENVIRONMENT

This section uses a cutdown version of the process from 6.2. Scenario 2: Using the CLI to Create a Basic Overcloud in the *Red Hat Enterprise Linux OpenStack Platform 7 Director Installation and Usage*.

Use the following workflow to setup your environment:

- Create a node definition template and register blank nodes in the director.
- Inspect hardware of all nodes.
- Manually tag nodes into roles.
- Create flavors and tag them into roles.

2.3.1. Registering Nodes

A node definition template (**instackenv.json**) is a JSON format file and contains the hardware and power management details for registering nodes. For example:

```
{
    "nodes":[
        {
            "mac":[
                "bb:bb:bb:bb:bb:bb"
            ],
            "cpu":"4",
            "memory":"6144",
            "disk":"40",
            "arch":"x86_64",
            "pm_type":"pxe_ipmitool",
            "pm_user":"admin",
            "pm_password":"p@55w0rd!",
            "pm_addr":"192.0.2.205"
```

```
        },
        {
            "mac":[
                "cc:cc:cc:cc:cc:cc"
            ],
            "cpu":"4",
            "memory":"6144",
            "disk":"40",
            "arch":"x86_64",
            "pm_type":"pxe_ipmitool",
            "pm_user":"admin",
            "pm_password":"p@55w0rd!",
            "pm_addr":"192.0.2.206"
        },
        {
            "mac":[
                "dd:dd:dd:dd:dd:dd"
            ],
            "cpu":"4",
            "memory":"6144",
            "disk":"40",
            "arch":"x86_64",
            "pm_type":"pxe_ipmitool",
            "pm_user":"admin",
            "pm_password":"p@55w0rd!",
            "pm_addr":"192.0.2.207"
        },
        {
            "mac":[
                "ee:ee:ee:ee:ee:ee"
            ],
            "cpu":"4",
            "memory":"6144",
            "disk":"40",
            "arch":"x86_64",
            "pm_type":"pxe_ipmitool",
            "pm_user":"admin",
            "pm_password":"p@55w0rd!",
            "pm_addr":"192.0.2.208"
        }
        {
            "mac":[
                "ff:ff:ff:ff:ff:ff"
            ],
            "cpu":"4",
            "memory":"6144",
            "disk":"40",
            "arch":"x86_64",
            "pm_type":"pxe_ipmitool",
            "pm_user":"admin",
            "pm_password":"p@55w0rd!",
            "pm_addr":"192.0.2.209"
        }
        {
            "mac":[
                "gg:gg:gg:gg:gg:gg"
```

```
            ],
            "cpu":"4",
            "memory":"6144",
            "disk":"40",
            "arch":"x86_64",
            "pm_type":"pxe_ipmitool",
            "pm_user":"admin",
            "pm_password":"p@55w0rd!",
            "pm_addr":"192.0.2.210"
        }
    ]
}
```

Note

The Provisioning network uses IPv4 addresses. The IPMI addresses must also be IPv4 addresses, and they must either be directly attached or reachable through routing over the Provisioning network.

After creating the template, save the file to the stack user's home directory (**`/home/stack/instackenv.json`**), then import it into the director. Use the following command to accomplish this:

```
$ openstack baremetal import --json ~/instackenv.json
```

This imports the template and registers each node from the template into the director.

Assign the kernel and ramdisk images to all nodes:

```
$ openstack baremetal configure boot
```

The nodes are now registered and configured in the director.

2.3.2. Inspecting the Hardware of Nodes

After registering the nodes, inspect the hardware attribute of each node. Run the following command to inspect the hardware attributes of each node:

```
$ openstack baremetal introspection bulk start
```

Important

Make sure this process runs to completion. This process usually takes 15 minutes for bare metal nodes.

2.3.3. Manually Tagging the Nodes

After registering and inspecting the hardware of each node, tag them into specific profiles. These profile tags match your nodes to flavors, and in turn the flavors are assigned to a deployment role.

Retrieve a list of your nodes to identify their UUIDs:

```
$ ironic node-list
```

To manually tag a node to a specific profile, add a profile option to the **properties/capabilities** parameter for each node. For example, to tag three nodes to use a controller profile and one node to use a compute profile, use the following commands:

```
$ ironic node-update 1a4e30da-b6dc-499d-ba87-0bd8a3819bc0 add
properties/capabilities='profile:control,boot_option:local'
$ ironic node-update 6faba1a9-e2d8-4b7c-95a2-c7fbdc12129a add
properties/capabilities='profile:control,boot_option:local'
$ ironic node-update 5e3b2f50-fcd9-4404-b0a2-59d79924b38e add
properties/capabilities='profile:control,boot_option:local'
$ ironic node-update 484587b2-b3b3-40d5-925b-a26a2fa3036f add
properties/capabilities='profile:compute,boot_option:local'
$ ironic node-update d010460b-38f2-4800-9cc4-d69f0d067efe add
properties/capabilities='profile:compute,boot_option:local'
$ ironic node-update d930e613-3e14-44b9-8240-4f3559801ea6 add
properties/capabilities='profile:compute,boot_option:local'
```

The addition of the **profile:compute** and **profile:control** options tag the nodes into each respective profiles.

Note

As an alternative to manual tagging, use the Automated Health Check (AHC) Tools to automatically tag larger numbers of nodes based on benchmarking data,

2.3.4. Creating Flavors for Nodes

The director also needs a set of hardware profiles, or flavors, for the registered nodes. In this scenario, we'll create a profile each for the Compute and Controller nodes.

```
$ openstack flavor create --id auto --ram 6144 --disk 40 --vcpus 4
control
$ openstack flavor create --id auto --ram 6144 --disk 40 --vcpus 4
compute
```

This creates two flavors for your nodes: **control** and **compute**. Set the additional properties for each flavor.

```
$ openstack flavor set --property "cpu_arch"="x86_64" --property
"capabilities:boot_option"="local" --property
"capabilities:profile"="compute" compute
$ openstack flavor set --property "cpu_arch"="x86_64" --property
"capabilities:boot_option"="local" --property
"capabilities:profile"="control" control
```

The **capabilities:profile** defines the profile to use. This links to the same tag on each respective node tagged in the previous section.

Important

Unused roles also require a default flavor named baremetal. Create this flavor if it does not exist:

```
$ openstack flavor create --id auto --ram 4096 --disk 40 --
vcpus 1 baremetal
```

2.4. CONFIGURING THE NETWORK

This section examines the network configuration for the Overcloud. This includes isolating our services to use specific network traffic and configuring the Overcloud with our IPv6 options.

2.4.1. Configuring Interfaces

The Overcloud requires a set of network interface templates. These templates are standard Heat templates in YAML format. The director contains a set of example templates to get you started:

- **`/usr/share/openstack-tripleo-heat-templates/network/config/single-nic-vlans`** - Directory containing templates for single NIC with VLANs configuration on a per role basis.
- **`/usr/share/openstack-tripleo-heat-templates/network/config/bond-with-vlans`** - Directory containing templates for bonded NIC configuration on a per role basis.

Copy one of these template collections to the **`stack`** user's **`templates`** directory. For example:

```
$ sudo cp -r /usr/share/openstack-tripleo-heat-
templates/network/config/single-nic-vlans ~/templates/nic-configs
```

For more information on network interface configuration, see 6.3.7.1. Creating Custom Interface Templates in the *Red Hat Enterprise Linux OpenStack Platform 7 Director Installation and Usage*

Note

Both interface template collections contain two files for configuring the controller nodes: **`controller.yaml`** and **`controller-v6.yaml`**. Use the **`controller-v6.yaml`** for configuring IPv6.

2.4.2. Configuring the IPv6 Isolated Network

The Overcloud requires a network environment file to configure IPv6. This file is a Heat environment file that describes the Overcloud's network environment and points to the network interface configuration templates. For this scenario, create an environment file (**`/home/stack/network-environment.yaml`**) and include the following sections.

```
resource_registry:
  OS::TripleO::BlockStorage::Net::SoftwareConfig:
/home/stack/templates/nic-configs/cinder-storage.yaml
  OS::TripleO::Compute::Net::SoftwareConfig: /home/stack/templates/nic-
```

```
configs/compute.yaml
  OS::TripleO::Controller::Net::SoftwareConfig:
/home/stack/templates/nic-configs/controller-v6.yaml
  OS::TripleO::ObjectStorage::Net::SoftwareConfig:
/home/stack/templates/nic-configs/swift-storage.yaml
  OS::TripleO::CephStorage::Net::SoftwareConfig:
/home/stack/templates/nic-configs/ceph-storage.yaml
```

This section registers each of our interface templates as a resource. When creating the Overcloud, the main Heat template collection calls the appropriate **`OS::TripleO::*::Net::SoftwareConfig`** resource when configuring the network for each node type. Without these resources, the main Heat template collection uses a default set of network configurations from the main resource registry.

Note

Both interface template collections contain two files for configuring the controller nodes: **`controller.yaml`** and **`controller-v6.yaml`**. Use the **`controller-v6.yaml`** for configuring IPv6.

```
parameter_defaults:
  DnsServers: ["8.8.8.8","8.8.4.4"]
  ControlPlaneSubnetCidr: "24"
  EC2MetadataIp: 192.0.2.1
  ControlPlaneDefaultRoute: 192.0.2.1
  ExternalInterfaceDefaultRoute: 2001:db8::1

  ExternalNetworkVlanID: 100
  ExternalNetCidr: '2001:db8::/64'
  ExternalAllocationPools: [{'start': '2001:db8:0:1::10', 'end':
'2001:db8:0:1:ffff:ffff:ffff:fffe'}]

  InternalApiNetworkVlanID: 201
  InternalApiNetCidr: '2001:db8:fd00:2000::/64'
  InternalApiAllocationPools: [{'start': '2001:db8:fd00:2000::10',
'end': '2001:db8:fd00:2000:ffff:ffff:ffff:fffe'}]

  TenantNetworkVlanID: 202
  TenantNetCidr: 'fd00:fd00:fd00:3000::/64'
  TenantAllocationPools: [{'start': '2001:db8:fd00:3000::10', 'end':
'2001:db8:fd00:3000:ffff:ffff:ffff:fffe'}]

  StorageNetworkVlanID: 203
  StorageNetCidr: 'fd00:fd00:fd00:4000::/64'
  StorageAllocationPools: [{'start': '2001:db8:fd00:4000::10', 'end':
'2001:db8:fd00:4000:ffff:ffff:ffff:fffe'}]

  StorageMgmtNetworkVlanID: 204
  StorageMgmtNetCidr: 'fd00:fd00:fd00:5000::/64'
  StorageMgmtAllocationPools: [{'start': '2001:db8:fd00:5000::10',
'end': '2001:db8:fd00:5000:ffff:ffff:ffff:fffe'}]
```

The **`parameter_defaults`** section contains the customization for the environment. Note the following parameters for our IPv6 setup:

ExternalInterfaceDefaultRoute

The IP address for the External interface default route. In this scenario, you use the Undercloud as a default route and specify the IP address created in Section 2.2, “Configuring an IPv6 Address on the Undercloud”.

ExternalNetworkVlanID

The VLAN ID of the External network. If using VLAN for accessing the External network on the Undercloud (See Section 2.2, “Configuring an IPv6 Address on the Undercloud”), make sure to map the same VLAN for this value.

ExternalNetCidr, InternalApiNetCidr, TenantNetCidr, StorageNetCidr, StorageMgmtNetCidr

The IPv6 CIDR prefix for each respective network.

ExternalAllocationPools, InternalApiAllocationPools, TenantAllocationPools, StorageAllocationPools, StorageMgmtAllocationPools

The range of IPv6 addresses to allocate to nodes. This is useful for ensuring no IPv6 conflicts occur.

2.4.3. Using a Hybrid IPv6/IPv4 Configuration

It is possible to configure the Overcloud to use a combination of IPv4 and IPv6 networking for various services. This requires modification of the networks and ports in the Overcloud. For example, we might aim to deploy the Storage and Storage Management networks on IPv4 while the other networks use IPv6.

Copy the network isolation initialization file (**`network-isolation-v6.yaml`**):

```
$ sudo cp /usr/share/openstack-tripleo-heat-
templates/environments/network-isolation-v6.yaml ~/templates/nic-
configs/network-isolation-v6.yaml
```

Edit the file to map the Storage and Storage Management resources to use the IPv4 versions of templates. For example:

```
resource_registry:
  OS::TripleO::Network::External: ../network/external_v6.yaml
  OS::TripleO::Network::InternalApi: ../network/internal_api_v6.yaml
  OS::TripleO::Network::StorageMgmt: ../network/storage_mgmt.yaml   #
Changed for IPv4
  OS::TripleO::Network::Storage: ../network/storage.yaml            #
Changed for IPv4
  OS::TripleO::Network::Tenant: ../network/tenant_v6.yaml

# Port assignments for the VIPs
  OS::TripleO::Network::Ports::ExternalVipPort:
../network/ports/external_v6.yaml
  OS::TripleO::Network::Ports::InternalApiVipPort:
../network/ports/internal_api_v6.yaml
  OS::TripleO::Network::Ports::StorageVipPort:
../network/ports/storage.yaml            # Changed for IPv4
  OS::TripleO::Network::Ports::StorageMgmtVipPort:
../network/ports/storage_mgmt.yaml   # Changed for IPv4
  OS::TripleO::Network::Ports::RedisVipPort:
../network/ports/vip_v6.yaml
```

```
# Port assignments for the controller role
  OS::TripleO::Controller::Ports::ExternalPort:
../network/ports/external_v6.yaml
  OS::TripleO::Controller::Ports::InternalApiPort:
../network/ports/internal_api_v6.yaml
  OS::TripleO::Controller::Ports::StoragePort:
../network/ports/storage.yaml            # Changed for IPv4
  OS::TripleO::Controller::Ports::StorageMgmtPort:
../network/ports/storage_mgmt.yaml   # Changed for IPv4
  OS::TripleO::Controller::Ports::TenantPort:
../network/ports/tenant_v6.yaml

...
```

Include the custom **network-isolation-v6.yaml** instead of the original when running **openstack overcloud deploy**.

In addition, modify the **parameter_defaults** section in **/home/stack/network-environment.yaml** to define a mix of IPv6 and IPv4 allocations for their respective networks. For example:

```
parameter_defaults:
  DnsServers: ["8.8.8.8","8.8.4.4"]
  ControlPlaneSubnetCidr: "24"
  EC2MetadataIp: 192.0.2.1
  ControlPlaneDefaultRoute: 192.0.2.1
  ExternalInterfaceDefaultRoute: 2001:db8::1

  ExternalNetworkVlanID: 100
  ExternalNetCidr: '2001:db8:0:1::/64'
  ExternalAllocationPools: [{'start': '2001:db8:0:1::10', 'end':
'2001:db8:0:1:ffff:ffff:ffff:fffe'}]

  InternalApiNetworkVlanID: 201
  InternalApiNetCidr: '2001:db8:fd00:2000::/64'
  InternalApiAllocationPools: [{'start': '2001:db8:fd00:2000::10',
'end': '2001:db8:fd00:2000:ffff:ffff:ffff:fffe'}]

  TenantNetworkVlanID: 202
  TenantNetCidr: 'fd00:fd00:fd00:3000::/64'
  TenantAllocationPools: [{'start': '2001:db8:fd00:3000::10', 'end':
'2001:db8:fd00:3000:ffff:ffff:ffff:fffe'}]

  StorageNetworkVlanID: 203
  StorageNetCidr: 172.18.0.0/24
  StorageAllocationPools: [{'start': '172.18.0.10', 'end':
'172.18.0.200'}]

  StorageMgmtNetworkVlanID: 204
  StorageMgmtNetCidr: 172.19.0.0/24
  StorageMgmtAllocationPools: [{'start': '172.19.0.10', 'end':
'172.19.0.200'}]
```

Important

If using VXLAN networking in your Overcloud, set the Tenant network to use IPv4. IPv6 VXLAN is not supported for Tenant networks.

2.5. COMPLETING OVERCLOUD CONFIGURATION

This completes the necessary steps to configure an IPv6-based Overcloud. The next chapter uses the **`openstack overcloud deploy`** command to create the Overcloud using the configuration from this chapter.

CHAPTER 3. CREATING THE OVERCLOUD

The creation of an Overcloud that uses IPv6 networking requires additional arguments for the **`openstack overcloud deploy`** command. For example:

```
$ openstack overcloud deploy --templates -e /usr/share/openstack-
tripleo-heat-templates/environments/network-isolation-v6.yaml -e
/home/stack/templates/network-environment.yaml -e --control-scale 3 --
compute-scale 3 --control-flavor control --compute-flavor compute --
neutron-disable-tunneling --neutron-network-type vlan --neutron-tunnel-
types vlan --neutron-network-vlan-ranges datacenter:1:1000 --ntp-server
clock.redhat.com [ADDITIONAL OPTIONS]
```

The above command uses the following options:

- **`--templates`** - Creates the Overcloud from the default Heat template collection.
- **`-e /usr/share/openstack-tripleo-heat-templates/environments/network-isolation-v6.yaml`** - Adds an additional environment file to the Overcloud deployment. In this case, it is an environment file that initializes network isolation configuration for IPv6.
- **`-e /home/stack/templates/network-environment.yaml`** - Adds an additional environment file to the Overcloud deployment. In this case, it is the network environment file created previously.
- **`--control-scale 3`** - Scale the Controller nodes to three.
- **`--compute-scale 3`** - Scale the Compute nodes to three.
- **`--control-flavor control`** - Use a specific flavor for the Controller nodes.
- **`--compute-flavor compute`** - Use a specific flavor for the Compute nodes.
- **`--neutron-disable-tunneling`** - Disables tunneling in the Overcloud. Tunneling is not supporting with IPv6. If using IPv4 for the Tenant network configuration and you aim to enable tunneling, do not include this option.
- **`--neutron-network-type vlan`** - Sets the **`neutron`** networking type. Use VLAN mode if using IPv6 for Tenant networks. If using VXLAN, change the Tenant network to use IPv4. For more information, see Section 2.4.3, “Using a Hybrid IPv6/IPv4 Configuration”.
- **`--neutron-network-vlan-ranges datacenter:1:1000`** - Sets the mapping range for **`neutron`** to support.
- **`--ntp-server clock.redhat.com`** - Sets our NTP server.

Note

For a full list of options, run:

```
$ openstack help overcloud deploy
```

See also Appendix H. Deployment Parameters in the *Red Hat Enterprise Linux OpenStack Platform 7 Director Installation and Usage* guide for parameter examples.

The Overcloud creation process begins and the director provisions your nodes. This process takes some time to complete. To view the status of the Overcloud creation, open a separate terminal as the **stack** user and run:

```
$ source ~/stackrc
$ heat stack-list --show-nested
```

3.1. ACCESSING THE OVERCLOUD

The director generates a script to configure and help authenticate interactions with your Overcloud from the director host. The director saves this file (**overcloudrc**) in your **stack** user's home directory. Run the following command to use this file:

```
$ source ~/overcloudrc
```

This loads the necessary environment variables to interact with your Overcloud from the director host's CLI. To return to interacting with the director's host, run the following command:

```
$ source ~/stackrc
```

CHAPTER 4. CONFIGURING THE OVERCLOUD AFTER CREATION

The creation process results in a fully operational Overcloud with IPv6 network. However, the Overcloud requires some post-creation configuration.

4.1. CREATING THE OVERCLOUD TENANT NETWORK

The Overcloud requires a IPv6-based Tenant network for instances. Source the **overcloudrc** file and create an initial Tenant network in **neutron**. For example:

```
$ source ~/overcloudrc
$ neutron net-create default --provider:physical_network datacentre --
provider:network_type vlan --provider:segmentation_id 101
$ neutron subnet-create default 2001:db8:fd00:6000::/64 --ipv6-ra-mode
slaac --ipv6-address-mode slaac --ip-version 6 --name default
```

This creates a basic **neutron** network called **default**. Confirm the created network with neutron net-list:

```
$ neutron net-list
```

4.2. CREATING THE OVERCLOUD PUBLIC NETWORK

This scenario configured the node interfaces to use the External network. However, you still need to create this network on the Overcloud so that we can provide network access.

```
$ neutron net-create public --router:external --
provider:physical_network datacentre --provider:network_type vlan --
provider:segmentation_id 100
$ neutron subnet-create public 2001:db8:0:2::/64 --ip-version 6 --
gateway 2001:db8::1 --allocation-pool
start=2001:db8:0:2::2,end=2001:db8:0:2::ffff --ip-version 6 --
ipv6_address_mode=slaac --ipv6_ra_mode=slaac
```

This creates a network called **public** provides an allocation pool of over 65000 IPv6 addresses for our instances.

Create a router to route instance traffic to the External network.

```
neutron router-create public-router
neutron router-gateway-set public-router public
```

CHAPTER 5. CONCLUSION

This concludes the creation and configuration of an IPv6-based Overcloud. For general Overcloud post-creation functions, see Chapter 7. Performing Tasks after Overcloud Creation in the *Red Hat Enterprise Linux OpenStack Platform 7 Director Installation and Usage* guide.