



Red Hat OpenStack Platform 16.0

IPv6 Networking for the Overcloud

Configuring an Overcloud to Use IPv6 Networking

Red Hat OpenStack Platform 16.0 IPv6 Networking for the Overcloud

Configuring an Overcloud to Use IPv6 Networking

OpenStack Team
rhos-docs@redhat.com

Legal Notice

Copyright © 2020 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, the Red Hat 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 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 achieve this.

Table of Contents

CHAPTER 1. INTRODUCTION	3
1.1. DEFINING IPV6 NETWORKING	3
1.2. USING IPV6 IN RED HAT OPENSTACK PLATFORM	4
1.3. SETTING REQUIREMENTS	6
1.4. DEFINING THE SCENARIO	6
CHAPTER 2. CONFIGURING THE OVERCLOUD BEFORE CREATION	8
2.1. INITIALIZING THE STACK USER	8
2.2. CONFIGURING AN IPV6 ADDRESS ON THE UNDERCLOUD	8
2.3. SETTING UP YOUR ENVIRONMENT	9
2.3.1. Registering Nodes	9
2.3.2. Inspecting the Hardware of Nodes	11
2.3.3. Manually Tagging the Nodes	11
2.4. CONFIGURING THE NETWORK	12
2.4.1. Configuring Composable Network Details	12
2.4.2. Network Isolation	13
2.4.3. Configuring Interfaces	14
2.4.4. Configuring the IPv6 Isolated Network	15
2.5. COMPLETING OVERCLOUD CONFIGURATION	15
CHAPTER 3. CREATING THE OVERCLOUD	16
3.1. ACCESSING THE OVERCLOUD	16
CHAPTER 4. CONFIGURING THE OVERCLOUD AFTER CREATION	17
4.1. CREATING THE OVERCLOUD TENANT NETWORK	17
4.2. CREATING THE OVERCLOUD PUBLIC NETWORK	17
CHAPTER 5. CONCLUSION	18

CHAPTER 1. INTRODUCTION

Red Hat 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

Bits	Representation
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 0db8 bit grouping as just db8 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. For more information, see ["RFC 5952: A Recommendation for IPv6 Address Text Representation"](#)

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 OPENSTACK PLATFORM

Red Hat 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 the [Director Installation and Usage](#) guide.

Red Hat 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 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	
Tenant Networks	Yes	Yes	Yes	
Tenant Network Endpoints	Yes	Yes	Yes	This refers to the IP address of the network hosting the tenant network tunnels, not the tenant networks themselves. IPv6 for network endpoints supports only VXLAN and Geneve. Generic routing encapsulation (GRE) is not yet supported.
External - Public API (and Horizon)		Yes	Yes	
External - Floating IPs	Yes	Yes	Yes	Dual stack and single stack (IPv6) only. neutron tenant networks that are assigned Global Unicast Address (GUA) prefixes and addresses do not require NAT on the external gateway port for the neutron router to access the outside world.
Provider Networks	Yes	Yes	Yes	IPv6 support is dependent on the tenant operating system.

Network Type	Dual Stack (IPv4/v6)	Single Stack (IPv6)	Single Stack (IPv4)	Notes
Provisioning (PXE/DHCP)			Yes	Interfaces on this network are IPv4 only.
IPMI or other BMC			Yes	RHOSP communicates with baseboard management controller (BMC) interfaces over the Provisioning network, which is IPv4. If BMC interfaces support dual stack IPv4 or IPv6, tools that are not part of RHOSP can use IPv6 to communicate with the BMCs.
Overcloud Provisioning network				The Provisioning network used for ironic in the overcloud.
Overcloud Cleaning network				The isolated network used to clean a machine before it is ready for reuse.

1.3. SETTING REQUIREMENTS

This guide acts as supplementary information for the [Director Installation and Usage](#) guide. This means the same requirements specified in [Director Installation and Usage](#) also apply to this guide. Implement these requirements as necessary.

This guide also requires the following:

- An Undercloud host with the Red Hat OpenStack Platform director installed. See the [Director Installation and Usage](#) guide.
- 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 [Director Installation and Usage](#) guide. The main difference is the omission of the Ceph Storage nodes.

For more information about this scenario, see the [Director Installation and Usage](#) guide.



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 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 [Configuring Basic Overcloud Requirements with the CLI Tools](#) in the *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 overcloud node import ~/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 overcloud node configure
```

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 overcloud node introspect --all-manageable
```



IMPORTANT

The nodes must be in the **manageable** state. 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-c7fbd12129a 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 automatic profile tagging to tag larger numbers of nodes based on benchmarking data.

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 Composable Network Details

1. Copy the default **network_data** file:

```
$ cp /usr/share/openstack-tripleo-heat-templates/network_data.yaml /home/stack/.
```

2. Edit the local copy of the **network_data.yaml** file and modify the parameters to suit your IPv6 networking requirements. For example, the External network contains the following default network details:

```

- name: External
  vip: true
  name_lower: external
  vlan: 10
  ipv6: true
  ipv6_subnet: '2001:db8:fd00:1000::/64'
  ipv6_allocation_pools: [{'start': '2001:db8:fd00:1000::10', 'end':
'2001:db8:fd00:1000:ffff:ffff:ffff:fffe'}]
  gateway_ipv6: '2001:db8:fd00:1000::1'

```

- **name** is the only mandatory value, however you can also use **name_lower** to normalize names for readability. For example, changing **InternalApi** to **internal_api**.
- **vip: true** creates a virtual IP address (VIP) on the new network with the remaining parameters setting the defaults for the new network.
- **ipv6** defines whether to enable IPv6.
- **ipv6_subnet** and **ipv6_allocation_pools**, and **gateway_ip6** set the default IPv6 subnet and IP range for the network.

Include the custom **network_data** file with your deployment using the **-n** option. Without the **-n** option, the deployment command uses the default network details.

2.4.2. Network Isolation

The overcloud assigns services to the provisioning network by default. However, the director can divide overcloud network traffic into isolated networks. To use isolated networks with IPv6, the overcloud contains an environment file that enables this feature. The **environments/network-isolation-v6.j2.yaml** file in the director's core Heat templates is a Jinja2 file that defines all ports and VIPs for each IPv6 network in your composable network file. When rendered, it results in a **network-isolation-v6.yaml** file in the same location with the full resource registry. For example:

```
resource_registry:
  # networks as defined in network_data.yaml
  OS::TripleO::Network::Storage: ../network/storage_v6.yaml
  OS::TripleO::Network::StorageMgmt: ../network/storage_mgmt_v6.yaml
  OS::TripleO::Network::InternalApi: ../network/internal_api_v6.yaml
  OS::TripleO::Network::Tenant: ../network/tenant.yaml
  OS::TripleO::Network::External: ../network/external_v6.yaml

  # Port assignments for the VIPs
  OS::TripleO::Network::Ports::StorageVipPort: ../network/ports/storage_v6.yaml
  OS::TripleO::Network::Ports::StorageMgmtVipPort: ../network/ports/storage_mgmt_v6.yaml
  OS::TripleO::Network::Ports::InternalApiVipPort: ../network/ports/internal_api_v6.yaml
  OS::TripleO::Network::Ports::ExternalVipPort: ../network/ports/external_v6.yaml
  OS::TripleO::Network::Ports::RedisVipPort: ../network/ports/vip_v6.yaml

  # Port assignments by role, edit role definition to assign networks to roles.
  # Port assignments for the Controller
  OS::TripleO::Controller::Ports::StoragePort: ../network/ports/storage_v6.yaml
  OS::TripleO::Controller::Ports::StorageMgmtPort: ../network/ports/storage_mgmt_v6.yaml
  OS::TripleO::Controller::Ports::InternalApiPort: ../network/ports/internal_api_v6.yaml
  OS::TripleO::Controller::Ports::TenantPort: ../network/ports/tenant.yaml
  OS::TripleO::Controller::Ports::ExternalPort: ../network/ports/external_v6.yaml

  # Port assignments for the Compute
  OS::TripleO::Compute::Ports::StoragePort: ../network/ports/storage_v6.yaml
  OS::TripleO::Compute::Ports::InternalApiPort: ../network/ports/internal_api_v6.yaml
  OS::TripleO::Compute::Ports::TenantPort: ../network/ports/tenant.yaml

  # Port assignments for the CephStorage
  OS::TripleO::CephStorage::Ports::StoragePort: ../network/ports/storage_v6.yaml
  OS::TripleO::CephStorage::Ports::StorageMgmtPort: ../network/ports/storage_mgmt_v6.yaml
```

The first section of this file has the resource registry declaration for the **OS::TripleO::Network::*** resources. By default these resources use the **OS::Heat::None** resource type, which does not create any networks. By redirecting these resources to the YAML files for each network, you enable the creation of these networks.

The following sections create the IP addresses for the nodes in each role. The controller nodes have IPs on each network. The compute and storage nodes each have IPs on a subset of the networks.

The rendered file also contains a parameters defaults section that enables IPv6 for certain services:

```
parameter_defaults:
```

```
# Enable IPv6 for Ceph.
CephIPv6: True
# Enable IPv6 for Corosync. This is required when Corosync is using an IPv6 IP in the cluster.
CorosyncIPv6: True
# Enable IPv6 for MongoDB. This is required when MongoDB is using an IPv6 IP.
MongoDbIPv6: True
# Enable various IPv6 features in Nova.
NovalIPv6: True
# Enable IPv6 environment for RabbitMQ.
RabbitIPv6: True
# Enable IPv6 environment for Memcached.
MemcachedIPv6: True
# Enable IPv6 environment for MySQL.
MySQLIPv6: True
# Enable IPv6 environment for Manila
ManilaIPv6: True
# Enable IPv6 environment for Redis.
RedisIPv6: True
```

2.4.3. Configuring Interfaces

The Overcloud requires a set of network interface templates. The director contains a set of Jinja2-based Heat templates, which render based on your **network_data** file:

NIC directory	Description	Environment file
single-nic-vlans	Single NIC (nic1) with control plane and VLANs attached to default Open vSwitch bridge.	environments/net-single-nic-with-vlans-v6.j2.yaml
single-nic-linux-bridge-vlans	Single NIC (nic1) with control plane and VLANs attached to default Linux bridge.	environments/net-single-nic-linux-bridge-with-vlans-v6.yaml
bond-with-vlans	Control plane attached to nic1 . Default Open vSwitch bridge with bonded NIC configuration (nic2 and nic3) and VLANs attached.	environments/net-bond-with-vlans-v6.yaml
multiple-nics	Control plane attached to nic1 . Assigns each sequential NIC to each network defined in the network_data file. By default, this is Storage to nic2 , Storage Management to nic3 , Internal API to nic4 , Tenant to nic5 on the br-tenant bridge, and External to nic6 on the default Open vSwitch bridge.	environments/net-multiple-nics-v6.yaml

For this example, we use the **single-nic-vlans** template collection.

2.4.4. Configuring the IPv6 Isolated Network

The default Heat template collection contains a Jinja2-based environment file for the default networking configuration. This file is **environments/network-environment.j2.yaml**. When rendered with our **network_data** file, it results in a standard YAML file called **network-environment.yaml**. Some parts of this file might require overrides, which is why you should create your own custom **network-environment.yaml** file. For this scenario, create a custom environment file (**/home/stack/network-environment.yaml**) with the following details:

```
parameter_defaults:
  DnsServers: ["8.8.8.8","8.8.4.4"]
  ControlPlaneDefaultRoute: 192.0.2.1
  ControlPlaneSubnetCidr: "24"
  EC2MetadataIp: 192.0.2.1
```

The **parameter_defaults** section contains the customization for certain services that remain on IPv4.

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 /usr/share/openstack-tripleo-heat-templates/environments/net-single-nic-with-vlans-v6.yaml \
-e /home/stack/templates/network-environment.yaml \
--ntp-server pool.ntp.org \
[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 /usr/share/openstack-tripleo-heat-templates/environments/net-single-nic-with-vlans-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/network-environment.yaml** - Adds an additional environment file to the Overcloud deployment. In this case, it includes overrides related to IPv6. Ensure that `network_data.yaml` includes the setting **ipv6: true**. Previous versions of Red Hat OpenStack director, included two routes: one for IPv6 on the External network (default) and one for IPv4 on the Control Plane. To use both default routes, ensure that the controller definition in `roles_data.yaml` contains both networks in `default_route_networks` (for example, **default_route_networks: ['External', 'ControlPlane']**).
- **--ntp-server pool.ntp.org** - Sets our NTP server.

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, the [Director Installation and Usage](#) guide.