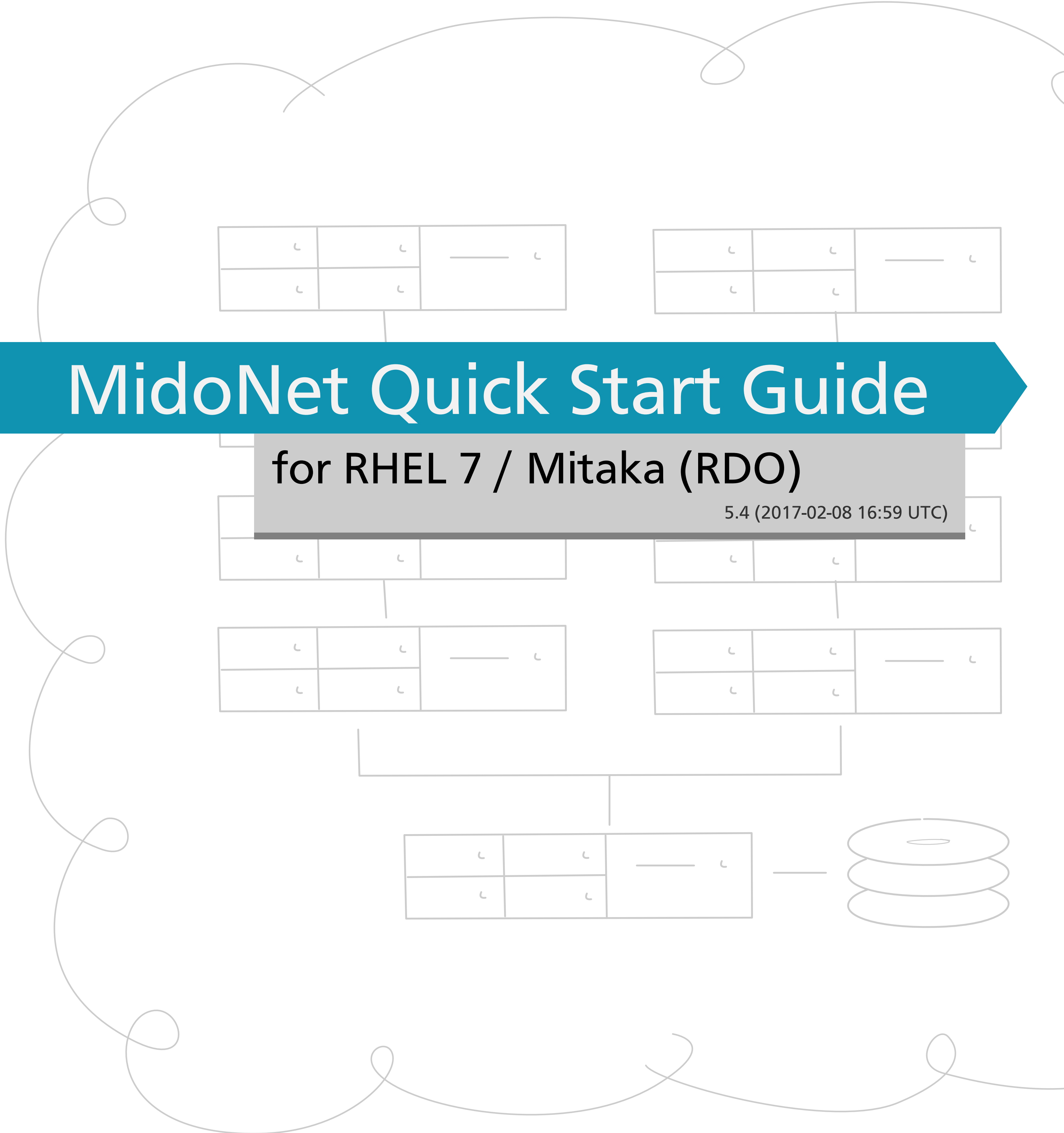
MidoNet Quick Start Guide

for RHEL 7 / Mitaka (RDO)

5.4 (2017-02-08 16:59 UTC)



docs.midonet.org

MidoNet Quick Start Guide for RHEL 7 / Mitaka (RDO)

5.4 (2017-02-08 16:59 UTC)

MidoNet is a network virtualization software for Infrastructure-as-a-Service (IaaS) clouds.

It decouples your IaaS cloud from your network hardware, creating an intelligent software abstraction layer between your end hosts and your physical network.

This guide walks through the minimum installation and configuration steps neccessary to use MidoNet with OpenStack.

Note

Please consult the MidoNet Mailing Lists or Chat if you need assistance.

Table of Contents

Preface

Conventions

The MidoNet documentation uses several typesetting conventions.

Notices

Notices take these forms:

Note

A handy tip or reminder.

Important

Something you must be aware of before proceeding.

Warning

Critical information about the risk of data loss or security issues.

Command prompts

$ prompt

Any user, including the root user, can run commands that are prefixed with the $ prompt.

 # prompt

The root user must run commands that are prefixed with the # prompt. You can also prefix these commands with the **sudo** command, if available, to run them.

1. Architecture

Table of Contents

This guide assumes the following example system architecture.

OpenStack Controller Node:

- Controller Node (**controller**)

Compute Node:

- Compute Node (**compute1**)

Since MidoNet is a distributed system, it does not have the concept of a Network Node as being used with the default OpenStack networking plugin. Instead it uses two or more Gateway Nodes that utilize Quagga to provide connectivity to external networks via the Border Gateway Protocol (BGP).

- Gateway Node 1 (**gateway1**)
- Gateway Node 2 (**gateway2**)

Three or more hosts are being used for the MidoNet Network State Database (NSDB) cluster which utilizes ZooKeeper and Cassandra to store virtual network topology and connection state information:

- NSDB Node 1 (**nsdb1**)
- NSDB Node 2 (**nsdb2**)
- NSDB Node 3 (**nsdb3**)

Important

Ideally, both the ZooKeeper transaction log and Cassandra data files need their own dedicated disks, with additional disks for other services on the host. However, for small POCs and small deployments, it is ok to share the Cassandra disk with other services and just leave the ZooKeeper transaction log on its own.

The *MidoNet Agent (Midolman)* has to be installed on all nodes where traffic enters or leaves the virtual topology. In this guide this are the **gateway1**, **gateway2** and **compute1** hosts.

The *Midonet Cluster* can be installed on a separate host, but this guide assumes it to be installed on the **controller** host.

The *Midonet Command Line Interface (CLI)* can be installed on any host that has connectivity to the MidoNet Cluster. This guide assumes it to be installed on the **controller** host.

The *Midonet Neutron Plugin* replaces the ML2 Plugin and has to be installed on the **controller**.

Hosts and Services

Controller Node (controller)

- General
  - Database (MariaDB)
  - Message Broker (RabbitMQ)
- OpenStack
  - Identity Service (Keystone)
  - Image Service (Glance)
  - Compute (Nova)
  - Networking (Neutron)
    - Neutron Server
  - Dashboard (Horizon)
- MidoNet
  - Cluster
  - CLI
  - Neutron Plugin

Compute Node (compute1)

- OpenStack
  - Compute (Nova)
  - Networking (Neutron)
- MidoNet
  - Agent (Midolman)

NSDB Nodes (nsdb1, nsdb2, nsdb3)

- Network State Database (NSDB)
  - Network Topology (ZooKeeper)
  - Network State Information (Cassandra)

Gateway Nodes (gateway1, gateway2)

- BGP Daemon (Quagga)

- MidoNet
  - Agent (Midolman)

2. Basic Environment Configuration

Table of Contents

Networking Configuration

Important

All hostnames must be resolvable, either via DNS or locally.

This guide assumes that you follow the instructions in Host networking of the OpenStack Documentation.

SELinux Configuration

Important

This guide assumes that SELinux (if installed) is either in `permissive` state or `disabled`.

To change the mode, execute the following command:

```
# setenforce Permissive
```

To permanently change the SELinux configuration, edit the `/etc/selinux/config` file accordingly:

```
SELINUX=permissive
```

Repository Configuration

Configure necessary software repositories and update installed packages.

1. **Enable Red Hat base repository**

```
# subscription-manager repos --enable=rhel-7-server-rpms
```

2. **Enable additional Red Hat repositories**

```
# subscription-manager repos --enable=rhel-7-server-extras-rpms
# subscription-manager repos --enable=rhel-7-server-optional-rpms
```

3. **Enable RDO repository**

```
# yum install https://rdoproject.org/repos/openstack-mitaka/rdo-release-mitaka.rpm
```

4. **Enable DataStax repository**

Create the `/etc/yum.repos.d/datastax.repo` file and edit it to contain the following:

```
# DataStax (Apache Cassandra)
[datastax]
name = DataStax Repo for Apache Cassandra
baseurl = http://rpm.datastax.com/community
enabled = 1
gpgcheck = 1
gpgkey = https://rpm.datastax.com/rpm/repo_key
```

1. **Enable MidoNet repositories**

   Create the `/etc/yum.repos.d/midonet.repo` file and edit it to contain the following:

```
[midonet]
name=MidoNet
baseurl=http://builds.midonet.org/midonet-5.4/stable/el7/
enabled=1
gpgcheck=1
gpgkey=https://builds.midonet.org/midorepo.key

[midonet-openstack-integration]
name=MidoNet OpenStack Integration
baseurl=http://builds.midonet.org/openstack-mitaka/stable/el7/
enabled=1
gpgcheck=1
gpgkey=https://builds.midonet.org/midorepo.key

[midonet-misc]
name=MidoNet 3rd Party Tools and Libraries
baseurl=http://builds.midonet.org/misc/stable/el7/
enabled=1
gpgcheck=1
gpgkey=https://builds.midonet.org/midorepo.key
```

2. **Install available updates**

```
# yum clean all
# yum upgrade
```

3. **If necessary, reboot the system**

```
# reboot
```

3. OpenStack Installation

Table of Contents

Important

Follow the OpenStack Mitaka Installation Guide for Red Hat Enterprise Linux 7 , but **note the following differences**.

Identity Service (Keystone)

Important

Follow the OpenStack documentation's Identity service instructions, but **note the following differences and additions**.

1. **Verify operation**

   Do **not** apply step '1. For security reasons, disable the temporary authentication token mechanism' from the 'Verify operation' section.

2. **Create MidoNet API Service**

   As Keystone admin, execute the following command:

   ```
   $ openstack service create --name midonet --description "MidoNet API
    Service" midonet
   ```

3. **Create MidoNet Administrative User**

   As Keystone admin, execute the following commands:

   ```
   $ openstack user create --domain default --password-prompt midonet
   $ openstack role add --project service --user midonet admin
   ```

Compute Services (Nova)

Important

Follow the OpenStack documentation's Compute service instructions, but **note the following differences**.

Controller Node

Note

Follow the OpenStack documentation's Install and configure controller node instructions as is.

Compute Node

Important

Follow the OpenStack documentation's Install and configure a compute node instructions, but **note the following additions**.

1. **Configure libvirt**

   Edit the `/etc/libvirt/qemu.conf` file to contain the following:

```
user = "root"
group = "root"

cgroup_device_acl = [
    "/dev/null", "/dev/full", "/dev/zero",
    "/dev/random", "/dev/urandom",
    "/dev/ptmx", "/dev/kvm", "/dev/kqemu",
    "/dev/rtc","/dev/hpet", "/dev/vfio/vfio",
    "/dev/net/tun"
]
```

2. **Restart the libvirt service**

```
# systemctl restart libvirtd.service
```

3. **Install nova-rootwrap network filters**

```
# yum install openstack-nova-network
# systemctl disable openstack-nova-network.service
```

4. **Restart the Compute service**

```
# systemctl restart openstack-nova-compute.service
```

Networking Services (Neutron)

Important

Follow the OpenStack documentation's Networking service instructions, but **note the following differences**.

Controller Node

Important

Follow the OpenStack documentation's Install and configure controller node instructions, but **note the following differences and additions**.

1. **Prerequisites**

   Apply as is.

2. **Configure networking options**

   Do **not** apply.

   a. Instead, install the following packages:

```
# yum install openstack-neutron python-networking-midonet python-
neutronclient
# yum erase openstack-neutron-ml2
```

b. Configure the server component:

Edit the `/etc/neutron/neutron.conf` file and configure the following keys:

```
[DEFAULT]
...
core_plugin = midonet.neutron.plugin_v2.MidonetPluginV2
service_plugins
 = midonet.neutron.services.l3.l3_midonet.MidonetL3ServicePlugin
...
dhcp_agent_notification = False
...
allow_overlapping_ips = True
...
rpc_backend = rabbit
...
auth_strategy = keystone
...
notify_nova_on_port_status_changes = True
notify_nova_on_port_data_changes = True
nova_url = http://controller:8774/v2.1

[database]
...
connection = mysql+pymysql://neutron:NEUTRON_DBPASS@controller/neutron

[oslo_messaging_rabbit]
...
rabbit_host = controller
rabbit_userid = openstack
rabbit_password = RABBIT_PASS

[keystone_authtoken]
...
auth_uri = http://controller:5000
auth_url = http://controller:35357
memcached_servers = controller:11211
auth_plugin = password
project_domain_id = default
user_domain_id = default
project_name = service
username = neutron
password = NEUTRON_PASS

[nova]
...
auth_url = http://controller:35357
auth_plugin = password
project_domain_id = default
user_domain_id = default
region_name = RegionOne
project_name = service
username = nova
password = NOVA_PASS

[oslo_concurrency]
...
lock_path = /var/lib/neutron/tmp
```

Note

When using multiple service plugins, separate them with commas:

```
[DEFAULT]
service_plugins = foo,bar,midonet.neutron.services.l3.
l3_midonet.MidonetL3ServicePlugin
```

3. **Configure the MidoNet plug-in**

   a. Create the directory for the MidoNet plugin:

   ```
   mkdir /etc/neutron/plugins/midonet
   ```

   b. Create the `/etc/neutron/plugins/midonet/midonet.ini` file and edit it to contain the following:

   ```
   [MIDONET]
   # MidoNet API URL
   midonet_uri = http://controller:8181/midonet-api
   # MidoNet administrative user in Keystone
   username = midonet
   password = MIDONET_PASS
   # MidoNet administrative user's tenant
   project_id = service
   ```

   c. Create a symbolic link to direct Neutron to the MidoNet configuration:

   ```
   # ln -s /etc/neutron/plugins/midonet/midonet.ini /etc/neutron/plugin.
   ini
   ```

4. **Configure the metadata agent**

   Do **not** apply.

5. **To configure Compute to use Networking**

   Apply as is.

6. **Configure Load-Balancer-as-a-Service (LBaaS)**

   Additionally to the OpenStack Installation Guide, configure Load-Balancer-as-a-Service (LBaaS) as described in the section called "Configure Load-Balancer-as-a-Service (LBaaS)" [10].

7. **Configure FireWall-as-a-Service (FWaaS)**

   Additionally to the OpenStack Installation Guide, configure FireWall-as-a-Service (FWaaS) as described in the section called "Configure FireWall-as-a-Service (FWaaS)" [10].

8. **Finalize installation**

   Do **not** apply.

   Instead, perform the following steps.

   a. Populate the database:

   ```
   # su -s /bin/sh -c "neutron-db-manage --config-file /etc/neutron/
   neutron.conf --config-file /etc/neutron/plugins/midonet/midonet.ini
    upgrade head" neutron
   ```

```
# su -s /bin/sh -c "neutron-db-manage --subproject networking-midonet
 upgrade head" neutron
```

b. Restart the Compute service:

```
# systemctl restart openstack-nova-api.service
```

c. Start the Networking service and configure it to start when the system boots:

```
# systemctl enable neutron-server.service
# systemctl start neutron-server.service
```

Configure Load-Balancer-as-a-Service (LBaaS)

1. **Install Neutron Load-Balancing-as-a-Service**

```
# yum install python-neutron-lbaas
```

2. **Enable the MidoNet driver**

   Enable the MidoNet driver by using the `service_provider` option in the `/etc/neutron/neutron.conf` file:

```
[service_providers]
service_provider = LOADBALANCER:Midonet:midonet.neutron.services.
loadbalancer.driver.MidonetLoadbalancerDriver:default
```

3. **Enable the LBaaS plug-in**

   Enable the LBaaS plug-in by using the `service_plugins` option in the `[DEFAULT]` section of the `/etc/neutron/neutron.conf` file:

```
[DEFAULT]
service_plugins = lbaas
```

Note

When using multiple service plugins, separate them with commas:

```
[DEFAULT]
service_plugins = foo,bar,lbass
```

4. **Enable load balancing in the dashboard**

   Change the `enable_lb` option to `True` in the `/etc/openstack-dashboard/local_settings` file:

```
OPENSTACK_NEUTRON_NETWORK = {
   'enable_lb': True,
   ...
}
```

5. **To finalize installation**

   Finalize the installation as described in Neutron Controller Node Installation.

Configure FireWall-as-a-Service (FWaaS)

1. **Install Neutron FireWall-as-a-Service**

```
# yum install python-neutron-fwaas
```

2. **Enable the MidoNet FWaaS plug-in**

   Enable the MidoNet FWaaS plug-in by using the `service_plugins` option in the `/etc/neutron/neutron.conf` file:

   ```
   service_plugins = midonet.neutron.services.firewall.plugin.
   MidonetFirewallPlugin
   ```

   ### Note

   When using multiple service plugins, separate them with commas:

   ```
   [DEFAULT]
   service_plugins = foo,bar,midonet.neutron.services.firewall.
   plugin.MidonetFirewallPlugin
   ```

3. **Enable firewall in the dashboard**

   Change the `enable_firewall` option to `True` in the `/etc/openstack-dashboard/local_settings` file:

   ```
   OPENSTACK_NEUTRON_NETWORK = {
      'enable_firewall': True,
      ...
   }
   ```

4. **To finalize installation**

   Finalize the installation as described in Neutron Controller Node Installation.

Compute Node

Important

Follow the OpenStack documentation's Install and configure compute node instructions, but **note the following differences**.

1. **Install the components**

   Do **not** apply.

2. **Configure the common component**

   Do **not** apply.

3. **Configure networking options**

   Do **not** apply.

4. **Configure Compute to use Networking**

   Apply as is.

5. **Finalize installation**

   Do **not** apply.

   a. Instead, restart the following service:

      ```
      # systemctl restart openstack-nova-compute.service
      ```

4. MidoNet Installation

Table of Contents

NSDB Nodes

ZooKeeper Installation

1. **Install ZooKeeper packages**

```
# yum install java-1.8.0-openjdk-headless
# yum install zookeeper zkdump nmap-ncat
```

2. **Configure ZooKeeper**

   a. **Common Configuration**

   Edit the `/etc/zookeeper/zoo.cfg` file to contain the following:

```
server.1=nsdb1:2888:3888
server.2=nsdb2:2888:3888
server.3=nsdb3:2888:3888
autopurge.snapRetainCount=10
autopurge.purgeInterval =12
```

   Create data directory:

```
# mkdir /var/lib/zookeeper/data
# chown zookeeper:zookeeper /var/lib/zookeeper/data
```

   #### Important

   For production deployments it is recommended to configure the storage of snapshots in a different disk than the commit log, this is done by setting the parameters `dataDir` and `dataLogDir` in `zoo.cfg`. In addition we advice to use an SSD drive for the commit log.

   b. **Node-specific Configuration**

   #### Important

   If using `CentOS 7.3` please use `/var/lib/zookeeper/myid` as the path for the host ID

   i. **NSDB Node 1**

   Create the `/var/lib/zookeeper/data/myid` file and edit it to contain the host's ID:

```
# echo 1 > /var/lib/zookeeper/data/myid
```

ii. **NSDB Node 2**

Create the `/var/lib/zookeeper/data/myid` file and edit it to contain the host's ID:

```
# echo 2 > /var/lib/zookeeper/data/myid
```

iii. **NSDB Node 3**

Create the `/var/lib/zookeeper/data/myid` file and edit it to contain the host's ID:

```
# echo 3 > /var/lib/zookeeper/data/myid
```

3. **Create Java Symlink**

```
# mkdir -p /usr/java/default/bin/
# ln -s /usr/lib/jvm/jre-1.8.0-openjdk/bin/java /usr/java/default/bin/
java
```

4. **Enable and start ZooKeeper**

```
# systemctl enable zookeeper.service
# systemctl start zookeeper.service
```

5. **Verify ZooKeeper Operation**

After installation of all nodes has been completed, verify that ZooKeeper is operating properly.

A basic check can be done by executing the `ruok` (Are you ok?) command on all nodes. This will reply with `imok` (I am ok.) if the server is running in a non-error state:

```
$ echo ruok | nc 127.0.0.1 2181
imok
```

More detailed information can be requested with the `stat` command, which lists statistics about performance and connected clients:

```
$ echo stat | nc 127.0.0.1 2181
Zookeeper version: 3.4.5--1, built on 06/10/2013 17:26 GMT
Clients:
 /127.0.0.1:34768[0](queued=0,recved=1,sent=0)
 /192.0.2.1:49703[1](queued=0,recved=1053,sent=1053)

Latency min/avg/max: 0/4/255
Received: 1055
Sent: 1054
Connections: 2
Outstanding: 0
Zxid: 0x260000013d
Mode: follower
Node count: 3647
```

Cassandra Installation

1. **Install Cassandra packages**

```
# yum install java-1.8.0-openjdk-headless
# yum install dsc22
```

2. **Configure Cassandra**

a. **Common Configuration**

Edit the `/etc/cassandra/conf/cassandra.yaml` file to contain the following:

```
# The name of the cluster.
cluster_name: 'midonet'

...

# Addresses of hosts that are deemed contact points.
seed_provider:
    - class_name: org.apache.cassandra.locator.SimpleSeedProvider
      parameters:
          - seeds: "nsdb1,nsdb2,nsdb3"
```

b. **Node-specific Configuration**

i. **NSDB Node 1**

Edit the `/etc/cassandra/conf/cassandra.yaml` file to contain the following:

```
# Address to bind to and tell other Cassandra nodes to connect to.
listen_address: nsdb1

...

# The address to bind the Thrift RPC service.
rpc_address: nsdb1
```

ii. **NSDB Node 2**

Edit the `/etc/cassandra/conf/cassandra.yaml` file to contain the following:

```
# Address to bind to and tell other Cassandra nodes to connect to.
listen_address: nsdb2

...

# The address to bind the Thrift RPC service.
rpc_address: nsdb2
```

iii. **NSDB Node 3**

Edit the `/etc/cassandra/conf/cassandra.yaml` file to contain the following:

```
# Address to bind to and tell other Cassandra nodes to connect to.
listen_address: nsdb3

...

# The address to bind the Thrift RPC service.
rpc_address: nsdb3
```

3. **Edit the service's init script**

On installation, the `/var/run/cassandra` directory is created, but because it is located on a temporary file system it will be lost after system reboot. As a result it is not possible to stop or restart the Cassandra service anymore.

To avoid this, edit the /etc/init.d/cassandra file to create the directory on service start:

```
[...]
case "$1" in
    start)
        # Cassandra startup
        echo -n "Starting Cassandra: "
        mkdir -p /var/run/cassandra
        chown cassandra:cassandra /var/run/cassandra
        su $CASSANDRA_OWNR -c "$CASSANDRA_PROG -p $pid_file" > $log_file
 2>&1
        retval=$?
[...]
```

4. **Enable and start Cassandra**

```
# systemctl enable cassandra.service
# systemctl start cassandra.service
```

5. **Verify Cassandra Operation**

After installation of all nodes has been completed, verify that Cassandra is operating properly.

Important

If Cassandra fails to start and prints a "buffer overflow" error message in its log file, you may try associating 127.0.0.1 with the hostname in etc/hosts (so that hostname -i will show 127.0.0.1). This may solve the Cassandra start problem.

A basic check can be done by executing the nodetool status command. This will reply with UN (Up / Normal) in the first column if the servers are running in a non-error state:

```
$ nodetool --host 127.0.0.1 status
[...]
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address     Load       Tokens  Owns    Host ID
       Rack
UN  192.0.2.1  123.45 KB   256     33.3%
 11111111-2222-3333-4444-555555555555  rack1
UN  192.0.2.2  234.56 KB   256     33.3%
 22222222-3333-4444-5555-666666666666  rack1
UN  192.0.2.3  345.67 KB   256     33.4%
 33333333-4444-5555-6666-777777777777  rack1
```

Controller Node

MidoNet Cluster Installation

1. **Install MidoNet Cluster package**

```
# yum install midonet-cluster
```

2. **Set up mn-conf**

Edit /etc/midonet/midonet.conf to point mn-conf to the ZooKeeper cluster:

```
[zookeeper]
zookeeper_hosts = nsdb1:2181,nsdb2:2181,nsdb3:2181
```

3. **Configure access to the NSDB**

   This step needs to happen only once, it will set up access to the NSDB for the MidoNet Cluster and Agent nodes.

   Run the following command to set the cloud-wide values for the ZooKeeper and Cassandra server addresses:

```
$ cat << EOF | mn-conf set -t default
zookeeper {
    zookeeper_hosts = "nsdb1:2181,nsdb2:2181,nsdb3:2181"
}

cassandra {
    servers = "nsdb1,nsdb2,nsdb3"
}
EOF
```

   Run the following command to set the Cassandra replication factor:

```
$ echo "cassandra.replication_factor : 3" | mn-conf set -t default
```

4. **Configure Keystone access**

   This step needs to happen only once, it will set up access to Keystone for the MidoNet Cluster node(s).

   Determine `domain_name` and `domain_id` to be used for Keystone authentication:

```
# openstack domain list
+---------+---------+---------
+------------------------------------------------------------------------+
| ID      | Name    | Enabled | Description
                              |
+---------+---------+---------
+------------------------------------------------------------------------+
| default | Default | True    | Owns users and tenants (i.e. projects)
 available on Identity API v2. |
+---------+---------+---------
+------------------------------------------------------------------------+
```

   For `tenant_name`, use the project/tenant that the `midonet` user belongs to, as configured during user creation.

   Configure the authentication parameters for MidoNet Cluster via `mn-conf`:

```
$ cat << EOF | mn-conf set -t default
cluster.auth {
   admin_role = "admin"
   provider_class = "org.midonet.cluster.auth.keystone.KeystoneService"
   keystone {
      admin_token = ""
      protocol = "http"
      host = "controller"
      port = 35357
      domain_name = "Default"
      domain_id = "default"
      tenant_name = "$MIDONET_TENANT"
      user_name = "midonet"
      user_password = "$MIDONET_PASS"
```

```
        version = 3
    }
}
EOF
```

5. **Start the MidoNet Cluster**

```
# systemctl enable midonet-cluster.service
# systemctl start midonet-cluster.service
```

MidoNet CLI Installation

1. **Install MidoNet CLI package**

```
# yum install python-midonetclient
```

2. **Configure MidoNet CLI**

   Create the `~/.midonetrc` file and edit it to contain the following:

```
[cli]
api_url = http://controller:8181/midonet-api
username = MIDONET_USER
password = MIDONET_PASS
project_id = service
```

Midolman Installation

The *MidoNet Agent (Midolman)* has to be installed on all nodes where traffic enters or leaves the virtual topology, in this guide this are the **gateway1**, **gateway2** and **compute1** nodes.

1. **Install Midolman package**

```
# yum install java-1.8.0-openjdk-headless
# yum install midolman
```

2. **Set up mn-conf**

   Edit `/etc/midolman/midolman.conf` to point mn-conf to the ZooKeeper cluster:

```
[zookeeper]
zookeeper_hosts = nsdb1:2181,nsdb2:2181,nsdb3:2181
```

3. **Configure resource usage**

   Run these steps on **each agent host** in order to configure resource usage.

   ### Important

   For production environments the **large** templates are strongly recommended.

   a. **Midolman resource template**

   Run the following command to configure the Midolman resource template:

```
$ mn-conf template-set -h local -t TEMPLATE_NAME
```

   Replace **TEMPLATE_NAME** with one of the following templates:

```
agent-compute-large
```

```
agent-compute-medium
agent-gateway-large
agent-gateway-medium
default
```

b. **Java Virtual Machine (JVM) resource template**

Replace the default `/etc/midolman/midolman-env.sh` file with one of the below to configure the JVM resource template:

```
/etc/midolman/midolman-env.sh.compute.large
/etc/midolman/midolman-env.sh.compute.medium
/etc/midolman/midolman-env.sh.gateway.large
/etc/midolman/midolman-env.sh.gateway.medium
```

4. **Configure MidoNet Metadata Proxy for all agents**

This step needs to happen only once, it will set up MidoNet Metadata Proxy for all MidoNet Agent nodes.

Run the following commands to set the cloud-wide values for the MidoNet Metadata Proxy:

```
$ echo "agent.openstack.metadata.nova_metadata_url : \"http:/
/controller:8775\"" | mn-conf set -t default
$ echo "agent.openstack.metadata.shared_secret : shared_secret" | mn-
conf set -t default
$ echo "agent.openstack.metadata.enabled : true" | mn-conf set -t
 default
```

controller, **8775**, and **shared_secret** should be replaced with appropriate values. They need to match with the corresponding Nova Metadata API configuration.

controller and **8775** specify the address on which Nova accepts Metadata API requests. **shared_secret** has to be the same as specified by the "metadata_proxy_shared_secret" field in the "neutron" section of nova.conf.

The Nova side of the configuration for the metadata service is same as when using Neutron Metadata Proxy. See the OpenStack documentation for details:

Cloud Administrator Guide: Configure Metadata

Important

The Metadata Proxy creates an interface on the hypervisor hosts, named "metadata".

When using `iptables` it may be necessary to add a rule to accept traffic on that interface:

```
iptables -I INPUT 1 -i metadata -j ACCEPT
```

1. **Start Midolman**

```
# systemctl enable midolman.service
# systemctl start midolman.service
```

MidoNet Host Registration

1. **Launch MidoNet CLI**

```
$ midonet-cli
midonet>
```

2. **Create tunnel zone**

   MidoNet supports the Virtual Extensible LAN (VXLAN) and Generic Routing Encapsulation (GRE) protocols to communicate to other hosts within a tunnel zone.

   To use the VXLAN protocol, create the tunnel zone with type 'vxlan':

```
midonet> tunnel-zone create name tz type vxlan
tzone0
```

   To use the GRE protocol, create the tunnel zone with type 'gre':

```
midonet> tunnel-zone create name tz type gre
tzone0
```

> **Important**
>
> Make sure to allow GRE/VXLAN traffic for all hosts that belong to the tunnel zone. For VXLAN MidoNet uses UDP port 6677 as default.

1. **Add hosts to tunnel zone**

```
midonet> list tunnel-zone
tzone tzone0 name tz type vxlan

midonet> list host
host host0 name controller alive true
host host1 name gateway1 alive true
host host2 name gateway2 alive true
host host3 name compute1 alive true

midonet> tunnel-zone tzone0 add member host host0
 address ip_address_of_host0
zone tzone0 host host0 address ip_address_of_host0

midonet> tunnel-zone tzone0 add member host host1
 address ip_address_of_host1
zone tzone0 host host1 address ip_address_of_host1

midonet> tunnel-zone tzone0 add member host host2
 address ip_address_of_host2
zone tzone0 host host2 address ip_address_of_host2

midonet> tunnel-zone tzone0 add member host host3
 address ip_address_of_host3
zone tzone0 host host3 address ip_address_of_host3
```

5. Initial Network Configuration

Table of Contents

Before launching your first instance, you must create the necessary virtual network infrastructure to which the instances connect, including the external network and tenant network.

External Network

The external network typically provides Internet access for your instances. By default, this network only allows Internet access from instances using Network Address Translation (NAT). You can enable Internet access to individual instances using a floating IP address and suitable security group rules. The `admin` tenant owns this network because it provides external network access for multiple tenants.

Note

Perform these commands on the controller node.

1. **Create an external network**

   Source the `admin` credentials to gain access to admin-only CLI commands:

   ```
   $ source admin-openrc.sh
   ```

   Create the network:

   ```
   $ neutron net-create ext-net --router:external
   Created a new network:
   +---------------------------+--------------------------------------+
   | Field                     | Value                                |
   +---------------------------+--------------------------------------+
   | admin_state_up            | True                                 |
   | id                        | 893aebb9-1c1e-48be-8908-6b947f3237b3 |
   | name                      | ext-net                              |
   | provider:network_type     | flat                                 |
   | provider:physical_network | external                             |
   | provider:segmentation_id  |                                      |
   | router:external           | True                                 |
   | shared                    | False                                |
   | status                    | ACTIVE                               |
   | subnets                   |                                      |
   | tenant_id                 | 54cd044c64d5408b83f843d63624e0d8     |
   +---------------------------+--------------------------------------+
   ```

   Like a physical network, a virtual network requires a `subnet` assigned to it. The external network shares the same subnet and `gateway` associated with the physical network connected to the external interface on the network node. You should specify an exclusive slice of this subnet for `router` and floating IP addresses to prevent interference with other devices on the external network.

2. **Create a subnet on the external network**

Create the subnet:

```
$ neutron subnet-create ext-net EXTERNAL_NETWORK_CIDR --name ext-subnet
 \
  --allocation-pool start=FLOATING_IP_START,end=FLOATING_IP_END \
  --disable-dhcp --gateway EXTERNAL_NETWORK_GATEWAY
```

Replace `FLOATING_IP_START` and `FLOATING_IP_END` with the first and last IP addresses of the range that you want to allocate for floating IP addresses. Replace `EXTERNAL_NETWORK_CIDR` with the subnet associated with the physical network. Replace `EXTERNAL_NETWORK_GATEWAY` with the gateway associated with the physical network, typically the ".1" IP address. You should disable DHCP on this subnet because instances do not connect directly to the external network and floating IP addresses require manual assignment.

For example, using `203.0.113.0/24` with floating IP address range `203.0.113.101` to `203.0.113.200`:

```
$ neutron subnet-create ext-net 203.0.113.0/24 --name ext-subnet \
  --allocation-pool start=203.0.113.101,end=203.0.113.200 \
  --disable-dhcp --gateway 203.0.113.1
Created a new subnet:
+-------------------
+---------------------------------------------------+
| Field             | Value
   |
+-------------------
+---------------------------------------------------+
| allocation_pools  | {"start": "203.0.113.101", "end": "203.0.113.200"}
   |
| cidr              | 203.0.113.0/24
   |
| dns_nameservers   |
   |
| enable_dhcp       | False
   |
| gateway_ip        | 203.0.113.1
   |
| host_routes       |
   |
| id                | 9159f0dc-2b63-41cf-bd7a-289309da1391
   |
| ip_version        | 4
   |
| ipv6_address_mode |
   |
| ipv6_ra_mode      |
   |
| name              | ext-subnet
   |
| network_id        | 893aebb9-1c1e-48be-8908-6b947f3237b3
   |
| tenant_id         | 54cd044c64d5408b83f843d63624e0d8
   |
+-------------------
+---------------------------------------------------+
```

Tenant Network

The tenant network provides internal network access for instances. The architecture isolates this type of network from other tenants. The demo tenant owns this network because it only provides network access for instances within it.

Note

Perform these commands on the controller node.

1. **Create the tenant network**

   Source the `demo` credentials to gain access to user-only CLI commands:

```
$ source demo-openrc.sh
```

   Create the network:

```
$ neutron net-create demo-net
Created a new network:
+-----------------+--------------------------------------+
| Field           | Value                                |
+-----------------+--------------------------------------+
| admin_state_up  | True                                 |
| id              | ac108952-6096-4243-adf4-bb6615b3de28 |
| name            | demo-net                             |
| router:external | False                                |
| shared          | False                                |
| status          | ACTIVE                               |
| subnets         |                                      |
| tenant_id       | cdef0071a0194d19ac6bb63802dc9bae     |
+-----------------+--------------------------------------+
```

   Like the external network, your tenant network also requires a subnet attached to it. You can specify any valid subnet because the architecture isolates tenant networks. By default, this subnet uses DHCP so your instances can obtain IP addresses.

2. **Create a subnet on the tenant network**

   Create the subnet:

```
$ neutron subnet-create demo-net TENANT_NETWORK_CIDR \
  --name demo-subnet --dns-nameserver DNS_RESOLVER \
  --gateway TENANT_NETWORK_GATEWAY
```

   Replace `TENANT_NETWORK_CIDR` with the subnet you want to associate with the tenant network, `DNS_RESOLVER` with the IP address of a DNS resolver, and `TENANT_NETWORK_GATEWAY` with the gateway you want to associate with the tenant network, typically the ".1" IP address.

   Example using `192.168.1.0/24` with DNS resolver `8.8.4.4` and gateway `192.168.1.1`:

```
$ neutron subnet-create demo-net 192.168.1.0/24 \
  --name demo-subnet --dns-nameserver 8.8.4.4 --gateway 192.168.1.1
Created a new subnet:
+-------------------
+------------------------------------------------------+
| Field               | Value
   |
```

```
+-------------------
+-----------------------------------------------------+
| allocation_pools  | {"start": "192.168.1.2", "end": "192.168.1.254"}
   |
| cidr              | 192.168.1.0/24
   |
| dns_nameservers   | 8.8.4.4
   |
| enable_dhcp       | True
   |
| gateway_ip        | 192.168.1.1
   |
| host_routes       |
   |
| id                | 69d38773-794a-4e49-b887-6de6734e792d
   |
| ip_version        | 4
   |
| ipv6_address_mode |
   |
| ipv6_ra_mode      |
   |
| name              | demo-subnet
   |
| network_id        | ac108952-6096-4243-adf4-bb6615b3de28
   |
| tenant_id         | cdef0071a0194d19ac6bb63802dc9bae
   |
+-------------------
+-----------------------------------------------------+
```

3. **Create a router on the tenant network and attach the external and tenant networks to it**

```
$ neutron router-create demo-router
Created a new router:
+-----------------------+--------------------------------------+
| Field                 | Value                                |
+-----------------------+--------------------------------------+
| admin_state_up        | True                                 |
| external_gateway_info |                                      |
| id                    | 635660ae-a254-4feb-8993-295aa9ec6418 |
| name                  | demo-router                          |
| routes                |                                      |
| status                | ACTIVE                               |
| tenant_id             | cdef0071a0194d19ac6bb63802dc9bae     |
+-----------------------+--------------------------------------+
```

Attach the router to the demo tenant subnet:

```
$ neutron router-interface-add demo-router demo-subnet
Added interface b1a894fd-aee8-475c-9262-4342afdc1b58 to router demo-
router.
```

Attach the router to the external network by setting it as the gateway:

```
$ neutron router-gateway-set demo-router ext-net
Set gateway for router demo-router
```

6. Edge Router Setup

Prior to v5.0, with Neutron, you could set up the gateway in only one way, which was to have a special singleton gateway router called the Provider Router created implicitly when an external network was created in Neutron. The provider router sits at the edge of the cloud and interfaces with the uplink router. The Provider Router is where BGP was typically configured. The biggest limitation of this approach was that it took away the scenario in which you wanted to have an L2 network at the edge instead of a router. Another limitation was that only one such router could exist for the entire cloud.

These limitations are removed in v5.0, where you could design your gateway to be either L2 network or router with as many routers as you wish, all using the Neutron API.

There are two main changes:

Edge Router

The Provider Router is no longer implicitly created upon the external network creation. Instead, the edge gateway routers, called the Edge Routers, are created explicitly using standard Neutron API. With this approach, multiple Edge Routers can be created, and they are optional.

Gateway Virtual Topology

In the previous model, the Provider Router was connected directly to the tenant routers, with the external networks hanging off of the Provider Router.

In the new model, the external networks exist between the edge and the tenant routers.

To create the gateway topology issue the following Neutron commands.

Create a standard neutron router:

```
neutron router-create <EDGE_ROUTER_NAME>
```

Attach the edge router to an external network:

```
neutron router-interface-add <EDGE_ROUTER_ID> <EXT_SUBNET_ID>
```

Create a special network called `uplink` network, representing the physical network outside of the cloud:

```
neutron net-create <UPLINK_NET_NAME> --tenant_id admin --
provider:network_type uplink
```

Create a subnet for the uplink network matching the CIDR used in the uplink network (could just be /30 if linked directly to another router):

```
neutron subnet-create --tenant_id admin --disable-dhcp --name
 <UPLINK_SUBNET_NAME> <UPLINK_NET_NAME> <CIDR>
```

Create a port on the uplink network with a specific IP that you want to use and the binding details so that this virtual port gets bound to a specific NIC on the gateway host:

```
neutron port-create <UPLINK_NET_ID> --binding:host_id <HOST_NAME> --
binding:profile type=dict interface_name=<INTERFACE_NAME> --fixed-ip
 ip_address=<IP_ADDR>
```

Attach the uplink port to the Edge Router:

```
neutron router-interface-add <EDGE_ROUTER_ID> port=<UPLINK_PORT_ID>
```

7. BGP Uplink Configuration

MidoNet utilizes the Border Gateway Protocol (BGP) for external connectivity.

For production deployments it is strongly recommended to use BGP due to it's scalability and redundancy.

For demo or POC environments, alternatively static routing can be used.

The following instructions assume below sample environment:

- One floating IP network
  - *192.0.2.0/24*
- Two MidoNet gateway nodes
  - *gateway1*, connecting to *bgp1* via *eth1*
  - *gateway2*, connecting to *bgp2* via *eth1*
- Two remote BGP peers
  - *bgp1*, *198.51.100.1*, AS *64513*
  - *bgp2*, *203.0.113.1*, AS *64514*
- Corresponding MidoNet BGP peers
  - *198.51.100.2*, AS *64512*
  - *203.0.113.2*, AS *64512*

Follow these steps to configure the BGP uplinks.

1. Launch the MidoNet CLI and find the Edge Router

   ```
   midonet-cli> router list
   router router0 name Edge Router state up
   router router1 name Tenant Router state up infilter chain0 outfilter
    chain1
   ```

   In this example the Edge Router is **router0**.

2. Create and bind virtual ports for the BGP sessions

   Refer to Chapter 6, "Edge Router Setup" [24] for instructions on how to create the necessary ports and bind them to the Gateway hosts' physical network interfaces.

   You can confirm the port configuration within MidoNet CLI by listing the Edge Router's ports:

   ```
   midonet> router router0 port list
   port port0 device router0 state up mac fa:16:3e:11:11:11
    addresses 198.51.100.2/30
   port port1 device router0 state up mac fa:16:3e:22:22:22
    addresses 203.0.113.2/30
   [...]
   ```

3. Configure basic BGP settings

```
midonet> router router0 set asn 64512

midonet> router router0 add bgp-peer asn 64513 address 198.51.100.1
router0:peer0

midonet> router router0 add bgp-peer asn 64514 address 203.0.113.1
router0:peer1

midonet> router router0 list bgp-peer
peer peer0 asn 64513 address 198.51.100.1
peer peer1 asn 64514 address 203.0.113.1
```

4. If needed, configure MD5 authentication:

```
midonet> router router0 bgp-peer peer0 set password BGP_PASSWORD
midonet> router router0 bgp-peer peer1 set password BGP_PASSWORD
```

5. If needed, configure custom timers that will take precedence over the default ones defined in the MidoNet configuration:

```
midonet> router router0 bgp-peer peer0 set connect-retry 10
midonet> router router0 bgp-peer peer0 set hold-time 5
midonet> router router0 bgp-peer peer0 set keep-alive 5
midonet> router router0 bgp-peer peer1 set connect-retry 10
midonet> router router0 bgp-peer peer1 set hold-time 5
midonet> router router0 bgp-peer peer1 set keep-alive 5
midonet> router router0 list bgp-peer
peer peer0 asn 64513 address 198.51.100.1 keep-alive 5 hold-time 5
 connect-retry 10
peer peer1 asn 64514 address 203.0.113.1 keep-alive 5 hold-time 5
 connect-retry 10
```

6. Add routes to the remote BGP peers

   In order to be able to establish connections to the remote BGP peers, corresponding routes have to be added.

```
midonet> router router0 route add src 0.0.0.0/0 dst 198.51.100.0/30
 port router0:port0 type normal
router0:route0

midonet> router router0 route add src 0.0.0.0/0 dst 203.0.113.0/30
 port router0:port1 type normal
router0:route1
```

7. Advertise BGP routes

   In order to provide external connectivity for hosted virtual machines, the floating IP network has to be advertised to the BGP peers.

```
midonet> router router0 add bgp-network net 192.0.2.0/24
router0:net0

midonet> router router0 list bgp-network
net net0 net 192.0.2.0/24
```

8. Further Steps

MidoNet installation and integration into OpenStack is completed.

Note

Consult the **Operations Guide** for further instructions on operating MidoNet.