

MidoNet Quick Start Guide

for Ubuntu 14.04 / Kilo

5.0-SNAPSHOT (2016-02-09 03:11 UTC)

DRAFT



MidoNet Quick Start Guide for Ubuntu 14.04 / Kilo

5.0-SNAPSHOT (2016-02-09 03:11 UTC)

Copyright © 2016 Midokura SARL All rights reserved.

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 necessary to use MidoNet with OpenStack.



Caution

This document is a DRAFT. It may be MISSING relevant information or contain UNTESTED information. Use it at your own risk.



Note

Please consult the [MidoNet Mailing Lists or Chat](#) if you need assistance.

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Table of Contents

Preface	iv
Conventions	iv
1. Architecture	1
Hosts and Services	2
2. Basic Environment Configuration	4
Networking Configuration	4
Repository Configuration	4
3. OpenStack Installation	6
Identity Service (Keystone)	6
Compute Services (Nova)	6
Networking Services (Neutron)	7
4. MidoNet Installation	11
NSDB Nodes	11
Controller Node	14
Midolman Installation	15
MidoNet Host Registration	16
5. Initial Network Configuration	18
6. Edge Router Setup	19
7. BGP Uplink Configuration	21
8. Further Steps	23

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

Hosts and Services	2
--------------------------	---

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 **controller**, **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	4
Repository Configuration	4

Networking Configuration



Important

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

This guide assumes that you follow the instructions in [OpenStack Networking \(neutron\)](#) of the OpenStack Documentation.

Repository Configuration

Configure necessary software repositories and update installed packages.

1. Configure Ubuntu repositories

Edit the `/etc/apt/sources.list` file to contain the following:

```
# Ubuntu Main Archive
deb http://archive.ubuntu.com/ubuntu/ trusty main
deb http://security.ubuntu.com/ubuntu trusty-updates main
deb http://security.ubuntu.com/ubuntu trusty-security main

# Ubuntu Universe Archive
deb http://archive.ubuntu.com/ubuntu/ trusty universe
deb http://security.ubuntu.com/ubuntu trusty-updates universe
deb http://security.ubuntu.com/ubuntu trusty-security universe
```

2. Configure Ubuntu Cloud Archive repository

Create the `/etc/apt/sources.list.d/cloudarchive-kilo.list` file and edit it to contain the following:

```
# Ubuntu Cloud Archive
deb http://ubuntu-cloud.archive.canonical.com/ubuntu trusty-updates/kilo
main
```

Install the repository's key:

```
# apt-get update
# apt-get install ubuntu-cloud-keyring
```

3. Configure DataStax repository

Create the `/etc/apt/sources.list.d/datastax.list` file and edit it to contain the following:

```
# DataStax (Apache Cassandra)
deb http://debian.datastax.com/community 2.0 main
```


Download and install the repository's key:

```
# curl -L https://debian.datastax.com/debian/repo_key | apt-key add -
```

4. Configure Java 8 repository

Since Ubuntu 14.04 does not provide a Java 8 runtime environment in its repositories, we are going to use the [Launchpad PPA for OpenJDK](#).

Create the `/etc/apt/sources.list.d/openjdk-8.list` file and edit it to contain the following:

```
# OpenJDK 8
deb http://ppa.launchpad.net/openjdk-r/ppa/ubuntu trusty main
```

Download and install the repository's key:

```
# apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv-keys
0x86F44E2A
```

5. Configure MidoNet repositories

Create the `/etc/apt/sources.list.d/midonet.list` file and edit it to contain the following:

```
# MidoNet
deb http://builds.midonet.org/midonet-5 stable main

# MidoNet OpenStack Integration
deb http://builds.midonet.org/openstack-kilo stable main

# MidoNet 3rd Party Tools and Libraries
deb http://builds.midonet.org/misc stable main
```

Download and install the repositories' key:

```
# curl -L https://builds.midonet.org/midorepo.key | apt-key add -
```

6. Install available updates

```
# apt-get update
# apt-get dist-upgrade
```

7. If necessary, reboot the system

```
# reboot
```


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

```
# service libvirt-bin restart
```

1. Install nova-rootwrap network filters

```
# apt-get install nova-network
```

2. Restart the Compute service

```
# service nova-compute restart
```

Networking Services (Neutron)



Important

Follow the OpenStack documentation's [Chapter 6. OpenStack Networking \(neutron\)](#) 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. To configure prerequisites

Apply as is.

2. To install the Networking components

Do **not** apply.

a. Instead, install the following packages:

```
# apt-get install neutron-server python-neutronclient python-neutron-  
plugin-midonet  
# apt-get purge neutron-plugin-ml2
```

3. To configure the Networking server component

Do **not** apply step 'd. Enable the Modular Layer 2 (ML2) plug-in, router service, and overlapping IP addresses'.

a. Instead, edit the `/etc/neutron/neutron.conf` file and configure the following key in the `[DEFAULT]` section:

```
[DEFAULT]  
...  
core_plugin = midonet.neutron.plugin_v2.MidonetPluginV2  
...  
allow_overlapping_ips = True
```



Note

Make sure to not leave any space at the starting of lines in any configuration file (this applies to all configuration files).

4. To configure the Modular Layer 2 (ML2) plug-in

Do **not** apply.

Instead, perform the following steps.

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:

```
[DATABASE]  
sql_connection = mysql://neutron:NEUTRON_DBPASS@controller/neutron  
  
[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. Edit the `/etc/default/neutron-server` file to contain the following:

```
NEUTRON_PLUGIN_CONFIG="/etc/neutron/plugins/midonet/midonet.ini"
```

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 \(LBaaS\)” \[9\]](#).

7. To 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
```

```
# midonet-db-manage upgrade head
```

b. Restart the Compute service:

```
# service nova-api restart
```

c. Restart the Networking service:

```
# service neutron-server restart
```

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

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

```
# apt-get 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 `/etc/neutron/neutron.conf` file:

```
service_plugins = lbaas
```

4. Enable load balancing in the dashboard

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

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

5. 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. To configure prerequisites

Do not apply.

2. To install the Networking components

Do not apply.

3. To configure the Networking common components

Do not apply.

4. To configure the Modular Layer 2 (ML2) plug-in

Do not apply.

5. To configure the Open vSwitch (OVS) service

Do not apply.

6. To configure Compute to use Networking

Apply as is.

7. To finalize the installation

Do not apply.

a. Instead, restart the following service:

```
# service nova-compute restart
```

4. MidoNet Installation

Table of Contents

NSDB Nodes	11
Controller Node	14
Midolman Installation	15
MidoNet Host Registration	16

NSDB Nodes

ZooKeeper Installation

1. Install ZooKeeper packages

```
# apt-get install openjdk-7-jre-headless
# apt-get install zookeeper zookeeperd zkdump
```

2. Configure ZooKeeper

a. Common Configuration

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

```
server.1=nsdb1:2888:3888
server.2=nsdb2:2888:3888
server.3=nsdb3:2888:3888
```



Important

For production deployments it is recommended to configure the storage of snapshots in a different disk than the commit log. This can be set by changing the parameter `dataDir` in `zoo.cfg` to a different disk.

b. Node-specific Configuration

i. **NSDB Node 1**

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

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

ii. NSDB Node 2

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

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

iii. NSDB Node 3

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

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

3. Restart ZooKeeper

```
# service zookeeper restart
```

4. 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

```
# apt-get install openjdk-7-jre-headless
# apt-get install dsc20
```

2. Configure Cassandra

a. Common Configuration

Edit the `/etc/cassandra/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/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/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/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. Clean existing data and restart Cassandra

```
# service cassandra stop
# rm -rf /var/lib/cassandra/*
# service cassandra start
```

4. 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
```



```
EOF
```

5. Start the MidoNet Cluster

```
# service midonet-cluster start
```

MidoNet CLI Installation

1. Install MidoNet CLI package

```
# apt-get install python-midonetclient
```

2. Configure MidoNet CLI

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

```
[cli]
api_url = http://controller:8181/midonet-api
username = admin
password = ADMIN_PASS
project_id = admin
```

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

```
# apt-get install openjdk-8-jre-headless
# apt-get 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://
/nova_metadata_host:nova_metadata_port\" | 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
```

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

nova_metadata_host and **nova_metadata_port** 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

```
# service midolman start
```

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



Important

Follow the OpenStack documentation's [Create initial networks](#) instructions, but **note the following differences**.

1. To create the external network

Use the following command to create the external network:

```
$ neutron net-create ext-net --router:external
```

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 its 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 infiltrer 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” \[19\]](#) 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
address 198.51.100.2 net 198.51.100.0/30
port port1 device router0 state up mac fa:16:3e:22:22:22
address 203.0.113.2 net 203.0.113.0/30
[...]
```

3. Configure basic BGP settings

8. Further Steps

MidoNet installation and integration into OpenStack is completed.



Note

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