Astara Documentation

Release 9.0.0.0b3.dev19

Akanda, Inc

November 30, 2016

Contents

Astara is an open source network virtualization platform built by OpenStack operators for real OpenStack clouds. Originally developed by DreamHost for their OpenStack-based public cloud, DreamCompute, Astara eliminates the need for complex SDN controllers, overlays, and multiple plugins by providing a simple integrated networking stack (routing, firewall, and load balancing via a *virtual Service VM*) for connecting and securing multi-tenant OpenStack environments.

CHAPTER 1

Narrative Documentation

1.1 What Is Astara

Astara an open source network virtualization solution built by OpenStack operators for OpenStack clouds.

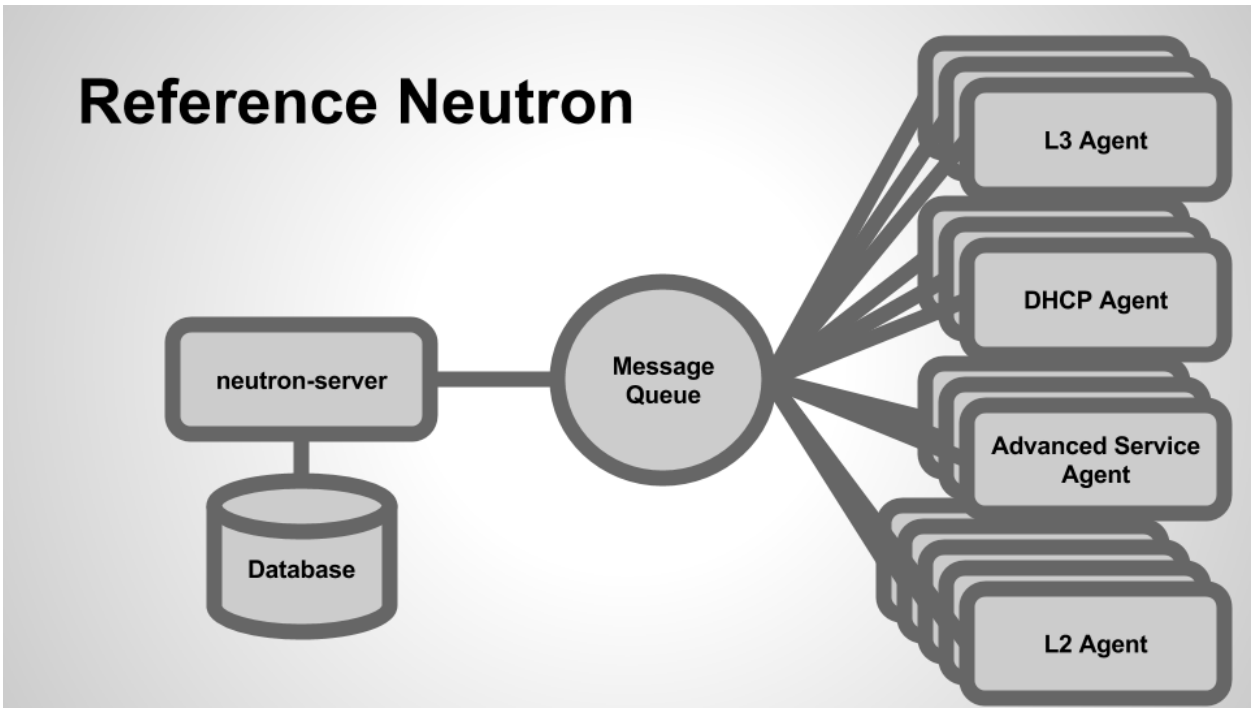
Astara follows core principles of simple, compatible, and open development.

The Astara architecture is broken down by describing the building blocks. The most important of those building blocks, the Astara Orchestrator, is a multi-process, multi-threaded Neutron Advanced Services orchestration service which manages the lifecycle of the Neutron Advanced Services. Astara currently supports a layer 3 routing and load balancing. Astara will support additional Neuton Advanced services such as VPN, and Firewalls in the open driver model.

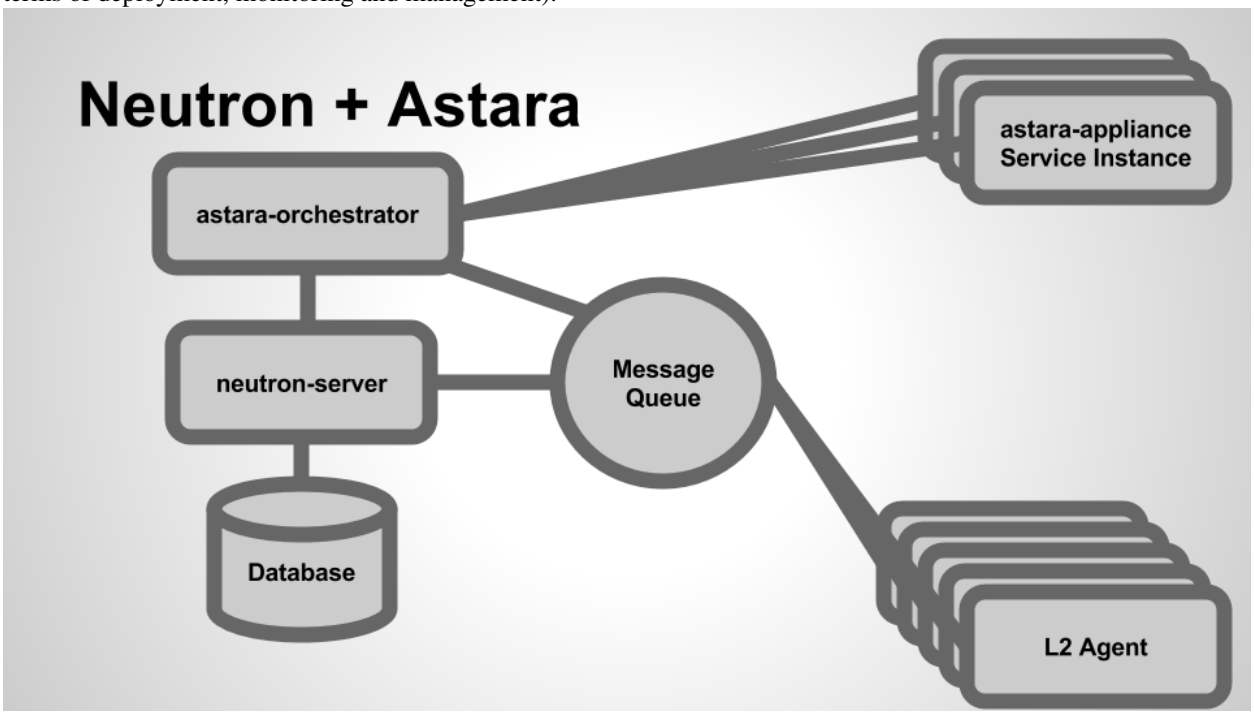
1.1.1 High-Level Architecture

Astara is a network orchestration platform that delivers network services (L3-L7) via service instances that provide routing, load balancing, and eventually more. Astara also interacts with any L2 overlay - including open source solutions based on OVS and Linux bridge (VLAN, VXLAN, GRE) and most proprietary solutions - to deliver a centralized management layer for all OpenStack networking decisions.

In a typical OpenStack deployment, Neutron server emits L3 and DHCP messages which are handled by a variety of Neutron agents (the L3 agent, DHCP agent, agents for advanced services such as load balancing, firewall, and VPN as a service):



When we add Astara into the mix, we're able to replace these agents with a virtualized Service Instance that manages layer 3 routing and other advanced networking services, significantly lowering the barrier of entry for operators (in terms of deployment, monitoring and management):



Astara takes the place of many of the agents that OpenStack Neutron communicates with (L3, DHCP, LBaaS, FWaaS) and acts as a single control point for all networking services. By removing the complexity of extra agents, Astara can centrally manage DHCP and L3, orchestrate load balancing and VPN Services, and overall reduce the number of components required to build, manage and monitor complete virtual networks within your cloud.

Astara Building Blocks

From an architectural perspective, Astara is composed of a few sub-projects:

- astara

  A service for managing the creation, configuration, and health of Astara Service Instances. The Orchestrator acts in part as a replacement for Neutron's various L3-L7 agents by listening for Neutron AMQP events and coalescing them into software appliance API calls (which configure and manage embedded services on the Service Instance). Additionally, the Orchestrator contains a health monitoring component which monitors health and guarantees uptime for existing Service Instances.

- astara-appliance

  The software and services (including tools for building custom service images themselves) that run on the virtualized Linux appliance. Includes drivers for L3-L7 services and a RESTful API that is used to orchestrate changes to appliance configuration.

- astara-neutron

  Addon API extensions and plugins for OpenStack Neutron which enable functionality and integration with the Astara project, notably Astara router appliance interaction.

- akanda-horizon

  OpenStack Horizon rug panels

Software Instance Lifecycle

As Neutron emits events in reaction to network operations (e.g., a user creates a new network/subnet, a user attaches a virtual machine to a network, a floating IP address is associated, etc...), Astara Orchestrator receives these events, parses, and dispatches them to a pool of workers which manage the lifecycle of every virtualized appliance.

This management of individual appliances is handled via a state machine per appliance; as events come in, the state machine for the appropriate instance transitions, modifying its configuration in a variety of ways, such as:

- Booting a virtual machine for the appliance via the Nova API
- Checking for aliveness of the Service Instance.
- Pushing configuration updates via the *REST API* to configure services (such as `iptables`, `dnsmasq`, `bird6`, etc...).
- Deleting instances via the Nova API (e.g., when a router or load balancer is deleted from Neutron).

1.1.2 The Service Instance (the Astara Appliance)

Astara uses Linux-based images (stored in OpenStack Glance) to provide layer 3 routing and advanced networking services. There is a stable image available by default, but it's also possible to build your own custom Service Instance image (running additional services of your own on top of the routing and other default services provided by the project).

1.2 Service VM Orchestration and Management

1.2.1 Astara Orchestrator

astara-orchestrator is a multi-processed, multithreaded Python process composed of three primary subsystems, each of which are spawned as a subprocess of the main `astara-orchestrator` process:

1.2.2 L3 and DHCP Event Consumption

`astara.notifications` uses kombu and a Python `multiprocessing.Queue` to listen for specific Neutron service events (e.g., `router.interface.create`, `subnet.create.end`, `port.create.end`, `port.delete.end`) and normalize them into one of several event types:

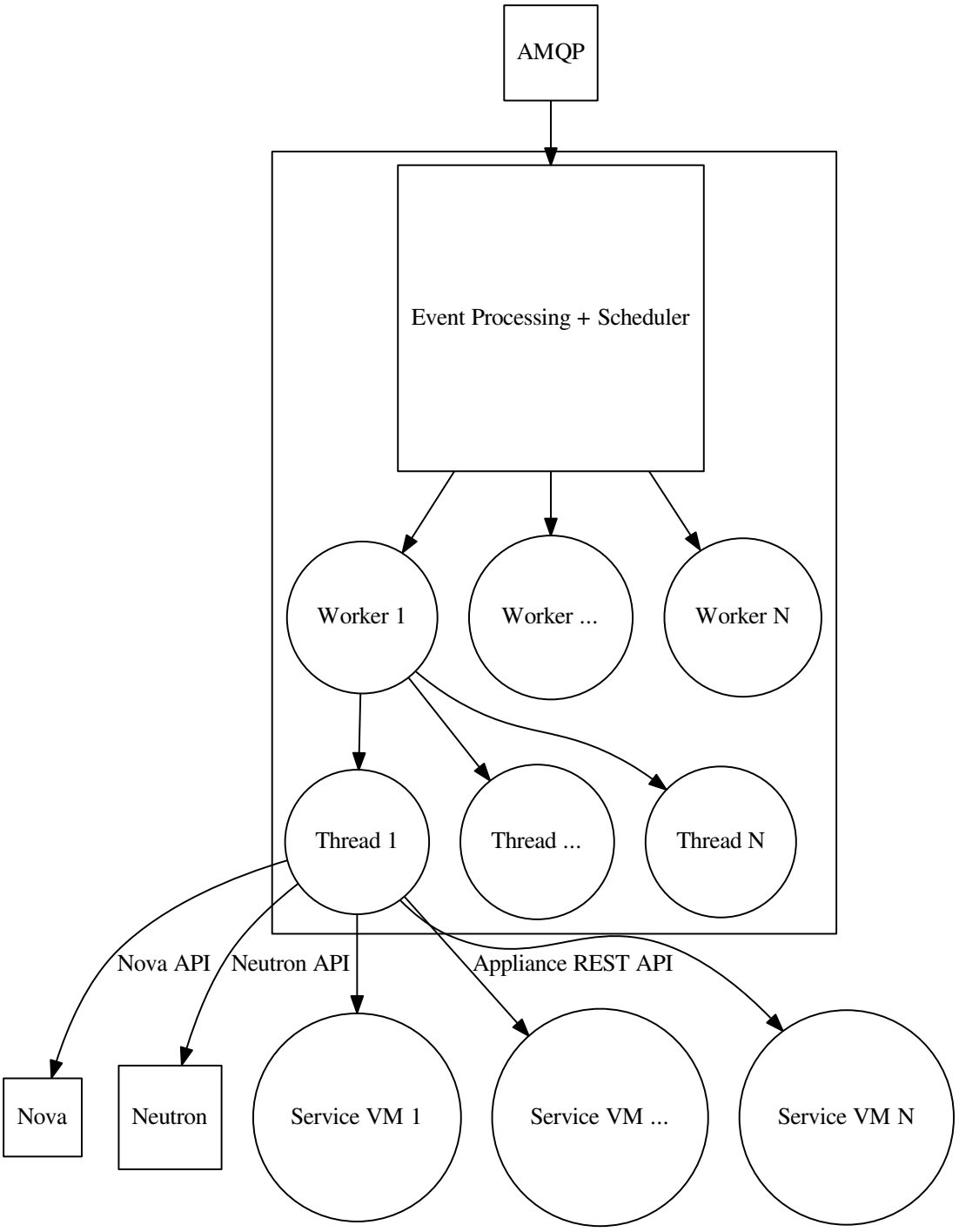
- `CREATE` - a router creation was requested
- `UPDATE` - services on a router need to be reconfigured
- `DELETE` - a router was deleted
- `POLL` - used by the *health monitor* for checking aliveness of a Service VM
- `REBUILD` - a Service VM should be destroyed and recreated

As events are normalized and shuttled onto the `multiprocessing.Queue`, `astara.scheduler` shards (by Tenant ID, by default) and distributes them amongst a pool of worker processes it manages.

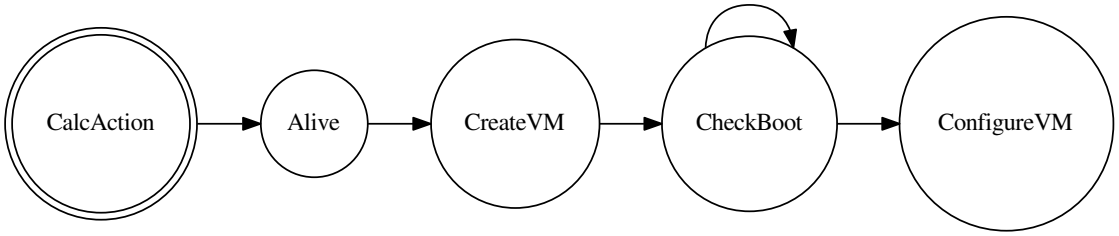
This system also consumes and distributes special `astara.command` events which are published by the **rug-ctl** *operator tools*.

1.2.3 State Machine Workers and Router Lifecycle

Each multithreaded worker process manages a pool of state machines (one per virtual router), each of which represents the lifecycle of an individual router. As the scheduler distributes events for a specific router, logic in the worker (dependent on the router's current state) determines which action to take next:



For example, let's say a user created a new Neutron network, subnet, and router. In this scenario, a `router-interface-create` event would be handled by the appropriate worker (based by tenant ID), and a transition through the state machine might look something like this:



State Machine Flow

The supported states in the state machine are:

CalcAction The entry point of the state machine. Depending on the current status of the Service VM (e.g., `ACTIVE`, `BUILD`, `SHUTDOWN`) and the current event, determine the first step in the state machine to transition to.

Alive Check aliveness of the Service VM by attempting to communicate with it via its REST HTTP API.

CreateVM Call `nova boot` to boot a new Service VM. This will attempt to boot a Service VM up to a (configurable) number of times before placing the router into `ERROR` state.

CheckBoot Check aliveness (up to a configurable number of seconds) of the router until the VM is responsive and ready for initial configuration.

ConfigureVM Configure the Service VM and its services. This is generally the final step in the process of booting and configuring a router. This step communicates with the Neutron API to generate a comprehensive network configuration for the router (which is pushed to the router via its REST API). On success, the state machine yields control back to the worker thread and that thread handles the next event in its queue (likely for a different Service VM and its state machine).

ReplugVM Attempt to hot-plug/unplug a network from the router via `nova interface-attach` or `nova-interface-detach`.

StopVM Terminate a running Service VM. This is generally performed when a Neutron router is deleted or via explicit operator tools.

ClearError After a (configurable) number of `nova boot` failures, Neutron routers are automatically transitioned into a cool down `ERROR` state (so that `astara` will not continue to boot them forever; this is to prevent further exasperation of failing hypervisors). This state transition is utilized to add routers back into management after issues are resolved and signal to `astara-orchestrator` that it should attempt to manage them again.

STATS Reads traffic data from the router.

CONFIG Configures the VM and its services.

EXIT Processing stops.

ACT(ion) Variables are:

Create Create router was requested.

Read Read router traffic stats.

Update Update router configuration.

Delete Delete router.

Poll Poll router alive status.

rEbuild Recreate a router from scratch.

VM Variables are:

Down VM is known to be down.

Booting VM is booting.

Up VM is known to be up (pingable).

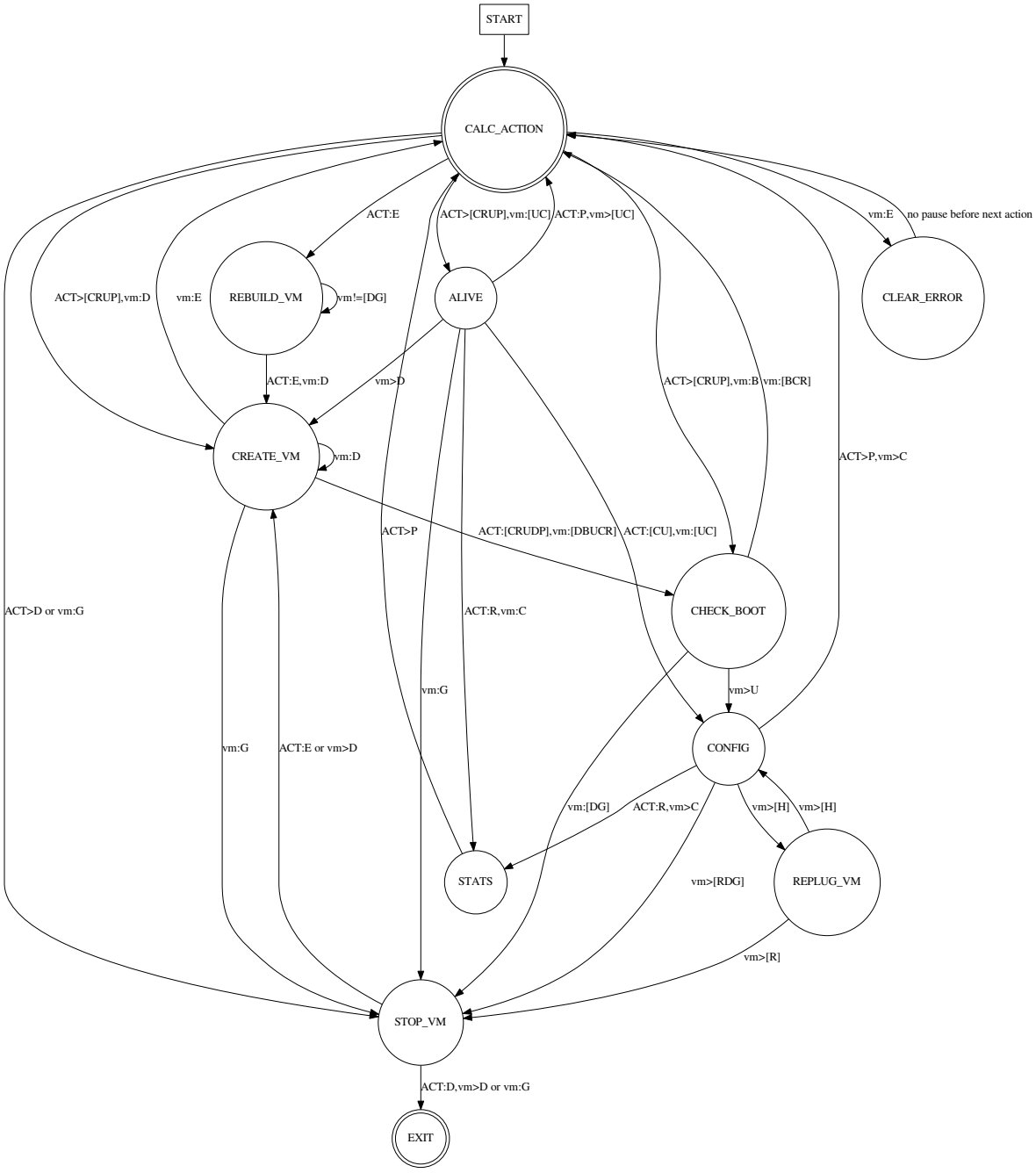
Configured VM is known to be configured.

Restart Needed VM needs to be rebooted.

Hotplug Needed VM needs to be replugged.

Gone The router definition has been removed from neutron.

Error The router has been rebooted too many times, or has had some other error.



1.2.4 Health Monitoring

`astara.health` is a subprocess which (at a configurable interval) periodically delivers `POLL` events to every known virtual router. This event transitions the state machine into the `Alive` state, which (depending on the availability of the router), may simply exit the state machine (because the router's status API replies with an `HTTP 200`) or transition to the `CreateVM` state (because the router is unresponsive and must be recreated).

1.2.5 High Availability

Astara supports high-availability (HA) on both the control plane and data plane.

The `astara-orchestrator` service may be deployed in a configuration that allows multiple service processes to span nodes to allow load-distribution and HA. For more information on clustering, see the *install docs*.

It also supports orchestrating pairs of virtual appliances to provide HA of the data path, allowing pairs of virtual routers to be clustered among themselves using VRRP and connection tracking. To enable this, simply create Neutron routers with the `ha=True` parameter or set this property on existing routers and issue a rebuild command via `astara-ctl` for that router.

1.3 The Service VM (the Astara Appliance)

Astara uses Linux-based images (stored in OpenStack Glance) to provide layer 3 routing and advanced networking services. Akanda, Inc provides stable image releases for download at akanda.io, but it's also possible to build your own custom Service VM image (running additional services of your own on top of the routing and other default services provided by Astara).

1.3.1 Building a Service VM image from source

The router code that runs within the appliance is hosted in the `astara-appliance` repository at `https://git.openstack.org/cgit/openstack/astara-appliance`. Additional tooling for actually building a VM image to run the appliance is located in that repository's `disk-image-builder` sub-directory, in the form elements to be used with `diskimage-builder`. The following instructions will walk through building the Debian-based appliance locally, publishing to Glance and configuring the RUG to use said image. These instructions are for building the image on an Ubuntu 14.04+ system.

Install Prerequisites

First, install `diskimage-builder` and required packages:

```
sudo apt-get -y install debootstrap qemu-utils
sudo pip install "diskimage-builder<0.1.43"
```

Next, clone the `astara-appliance` repository:

```
git clone https://git.openstack.org/openstack/astara-appliance
```

Build the image

Kick off an image build using diskimage-builder:

```
cd astara-appliance
ELEMENTS_PATH=diskimage-builder/elements DIB_RELEASE=jessie DIB_EXTLINUX=1 \
disk-image-create debian vm astara -o astara
```

Publish the image

The previous step should produce a qcow2 image called `astara.qcow` that can be published into Glance for use by the system:

```
# We assume you have the required OpenStack credentials set as an environment
# variables
glance image-create --name astara --disk-format qcow2 --container-format bare \
    --file astara.qcow2
+------------------+--------------------------------------+
| Property         | Value                                |
+------------------+--------------------------------------+
| checksum         | cfc24b67e262719199c2c4dfccb6c808     |
| container_format | bare                                 |
| created_at       | 2015-05-13T21:27:02.000000           |
| deleted          | False                                |
| deleted_at       | None                                 |
| disk_format      | qcow2                                |
| id               | e2caf7fa-9b51-4f42-9fb9-8cfce96aad5a |
| is_public        | False                                |
| min_disk         | 0                                    |
| min_ram          | 0                                    |
| name             | astara                               |
| owner            | df8eaa19c1d44365911902e738c2b10a     |
| protected        | False                                |
| size             | 450573824                            |
| status           | active                               |
| updated_at       | 2015-05-13T21:27:03.000000           |
| virtual_size     | None                                 |
+------------------+--------------------------------------+
```

Configure the RUG

Take the above image id and set the corresponding value in the RUG's config file, to instruct the service to use that image for software router instances it manages:

```
vi /etc/astara/orchestrator.ini
...
[router]
image_uuid=e2caf7fa-9b51-4f42-9fb9-8cfce96aad5a
```

Making local changes to the appliance service

By default, building an image in this way pulls the `astara-appliance` code directly from the upstream tip of trunk. If you'd like to make modifications to this code locally and build an image containing those changes, set DIB_REPOLOCATION_astara and DIB_REPOREF_astara in your enviornment accordingly during the image build, ie:

```
export DIB_REPOLOCATION_astara=~/src/astara-appliance  # Location of the local repository checkout
export DIB_REPOREF_astara=my-new-feature # The branch name or SHA-1 hash of the git ref to build fro
```

1.3.2 REST API

The Astara Appliance REST API is used by the orchestrator service to manage health and configuration of services on the router.

Router Health

HTTP GET /v1/status/

Used to confirm that a router is responsive and has external network connectivity.

```
Example HTTP 200 Response

Content-Type: application/json
{
    'v4': true,
    'v6': false,
}
```

Router Configuration

HTTP GET /v1/firewall/rules/

Used to retrieve an overview of configured firewall rules for the router (from `iptables -L` and `iptables6 -L`).

```
Example HTTP 200 Response

Content-Type: text/plain
Chain INPUT (policy DROP)
target     prot opt source               destination
ACCEPT     all  --  0.0.0.0/0            0.0.0.0/0
ACCEPT     icmp --  0.0.0.0/0            0.0.0.0/0           icmptype 8

...
```

HTTP GET /v1/system/interface/<ifname>/

Used to retrieve JSON data about a specific interface on the router.

```
Example HTTP 200 Response

Content-Type: application/json
{
    "interface": {
        "addresses": [
            "8.8.8.8",
            "2001:4860:4860::8888",
        ],
        "description": "",
        "groups": [],
        "ifname": "ge0",
        "lladdr": "fa:16:3f:de:21:e9",
        "media": null,
        "mtu": 1500,
        "state": "up"
    }
}
```

HTTP GET /v1/system/interfaces

Used to retrieve JSON data about a *every* interface on the router.

```
Example HTTP 200 Response

Content-Type: application/json
{
    "interfaces": [{
        "addresses": [
            "8.8.8.8",
            "2001:4860:4860::8888",
        ],
        "description": "",
        "groups": [],
        "ifname": "ge0",
        "lladdr": "fa:16:3f:de:21:e9",
        "media": null,
        "mtu": 1500,
        "state": "up"
    }, {
        ...
    }]
}
```

HTTP PUT /v1/system/config/

Used (generally, by **astara-orchestrator**) to push a new configuration to the router and restart services as necessary:

```
Example HTTP PUT Body

Content-Type: application/json
{
    "configuration": {
        "networks": [
            {
                "address_allocations": [],
                "interface": {
                    "addresses": [
                        "8.8.8.8",
                        "2001:4860:4860::8888"
                    ],
                    "description": "",
                    "groups": [],
                    "ifname": "ge1",
                    "lladdr": null,
                    "media": null,
                    "mtu": 1500,
                    "state": "up"
                },
                "name": "",
                "network_id": "f0f8c937-9fb7-4a58-b83f-57e9515e36cb",
                "network_type": "external",
                "v4_conf_service": "static",
                "v6_conf_service": "static"
            },
```

```
            {
                "address_allocations": [],
                "interface": {
                    "addresses": [
                        "..."
                    ],
                    "description": "",
                    "groups": [],
                    "ifname": "ge0",
                    "lladdr": "fa:16:f8:90:32:e3",
                    "media": null,
                    "mtu": 1500,
                    "state": "up"
                },
                "name": "",
                "network_id": "15016de1-494b-4c65-97fb-475b40acf7e1",
                "network_type": "management",
                "v4_conf_service": "static",
                "v6_conf_service": "static"
            },
            {
                "address_allocations": [
                    {
                        "device_id": "7c400585-1743-42ca-a2a3-6b30dd34f83b",
                        "hostname": "10-10-10-1.local",
                        "ip_addresses": {
                            "10.10.10.1": true,
                            "2607:f298:6050:f0ff::1": false
                        },
                        "mac_address": "fa:16:4d:c3:95:81"
                    }
                ],
                "interface": {
                    "addresses": [
                        "10.10.10.1/24",
                        "2607:f298:6050:f0ff::1/64"
                    ],
                    "description": "",
                    "groups": [],
                    "ifname": "ge2",
                    "lladdr": null,
                    "media": null,
                    "mtu": 1500,
                    "state": "up"
                },
                "name": "",
                "network_id": "31a242a0-95aa-49cd-b2db-cc00f33dfe88",
                "network_type": "internal",
                "v4_conf_service": "static",
                "v6_conf_service": "static"
            }
        ],
        "static_routes": []
    }
}
```

1.3.3 Survey of Software and Services

The Astara Appliance uses a variety of software and services to manage routing and advanced services, such as:

- `iproute2` tools (e.g., `ip neigh`, `ip addr`, `ip route`, etc...)
- `dnsmasq`
- `bird6`
- `iptables` and `iptables6`

In addition, the Astara Appliance includes two Python-based services:

- The REST API (which **`astara-orchestrator)`** communicates with to orchestrate router updates), deployed behind gunicorn.
- A Python-based metadata proxy.

1.3.4 Proxying Instance Metadata

When OpenStack VMs boot with `cloud-init`, they look for metadata on a well-known address, `169.254.169.254`. To facilitate this process, Astara sets up a special NAT rule (one for each local network):

```
-A PREROUTING -i eth2 -d 169.254.169.254 -p tcp -m tcp --dport 80 -j DNAT --to-destination 10.10.10.1
```

...and a special rule to allow metadata requests to pass across the management network (where OpenStack Nova is running, and will answer requests):

```
-A INPUT -i !eth0 -d <management-v6-address-of-router> -j DROP
```

A Python-based metadata proxy runs locally on the router (in this example, listening on `http://10.10.10.1:9602`) and proxies these metadata requests over the management network so that instances on local tenant networks will have access to server metadata.

1.4 Contributing

1.4.1 Submitting Code Upstream

All of Astara's code is 100% open-source and is hosted on git.openstack.org Patches are welcome!

1.5 Operation and Deployment

1.5.1 Installation

You can install from GitHub directly with `pip`:

```
$ pip install -e git://git.openstack.org/openstack/astara@stable/liberty#egg=astara
```

After installing `astara`, it can be invoked as:

```
$ astara-orchestrator --config-file /etc/akanda-rug/rug.ini
```

The `astara` service is intended to run on a management network (a separate network for use by your cloud operators). This segregation prevents system administration and the monitoring of system access from being disrupted by traffic generated by guests.

1.5.2 Operator Tools

rug-ctl

`astara-ctl` is a tool which can be used to send manual instructions to a running `astara-orchestrator` via AMQP:

```
$ astara-ctl browse
A curses console interface for browsing the state
of every Neutron router and issuing `rebuild` commands

$ astara-ctl poll
Sends a POLL instruction to every router to check health

$ astara-ctl router rebuild <router-id>
Sends a REBUILD instruction to a specific router

$ astara-ctl router update <router-id>
Sends an UPDATE instruction to a specific router

$ astara-ctl router debug <router-id>
Places a specific router in `debug mode`.
This causes the rug to ignore messages for the specified
router (so that, for example, operators can investigate
troublesome routers).

$ astara-ctl router manage <router-id>
Removes a specific router from `debug mode` and places
it back under astara-orchestrator management.

$ astara-ctl tenant debug <tenant-id>
Places a specific tenant in `debug mode`.
This causes the rug to ignore messages for the specified
tenant.
troublesome routers).

$ astara-ctl tenant manage <tenant-id>
Removes every router for a specific tenant from `debug mode`
and places the tenant back under astara-orchestrator management.

$ astara-ctl ssh <router-id>
Establishes an ssh connection with a specified Service VM.

$ astara-ctl workers debug
Causes the rug to print debugging diagnostics about the
current state of its worker processes and the state machines
under their management.
```

`astara-orchestrator` also exposes an RPC API on the management network, which allows non-interactive *astara-ctl* commands to be issued via HTTP, e.g.,

```
$ curl -X PUT -g6 "http://[fdca:3ba5:a17a:acda::1]:44250/poll/"
$ curl -X PUT -g6 "http://[fdca:3ba5:a17a:acda::1]:44250/workers/debug/"
```

```
$ curl -X PUT -g6 "http://[fdca:3ba5:a17a:acda::1]:44250/router/rebuild/<ID>"
```

astara-debug-router

astara-debug-router is a diagnostic tool which can be used to analyze the state machine flow of any router and step through its operation using Python's debugger. This is particularly useful for development purposes and understanding the nature of the `astara-orchestrator` state machine, but it's also useful for debugging problematic routers as an operator; a common pattern for determining why a Service VM won't boot is to place the router in *debug mode*:

```
$ astara-ctl router debug <router-id>
```

...and then step through the handling of a manual `UPDATE` event to see where it fails:

```
$ astara-debug-router --router-id <router-id>
```

1.6 Astara Installation

1.6.1 Assumptions

You have a fully operating Openstack environment with, at least: Nova, Keystone, Glance, Neutron The OpenStack environment has been tested and they VMs can be successfully created. the packages git and pip should be installed

This has been tested on Ubuntu 14.04 with OpenStack installed from source. For RHEL or CentOS path names will need to be adjusted. These instructions assume they are performed by the root user, whose home directory is /root. If another user does the installation some adjustment in the paths may be needed. This user will need sudo access and most commands will need to be prepended with sudo.

Use the neutron commands to delete all VMs, routers, networks

All neutron l3 agents should be stopped and disabled. (l3, dhcp, ..)

1.6.2 Installation

All configuration is to be performed on the controller node.

1. Set up astara user and directories:

```
mkdir -p /var/log/astara /var/lib/astara /etc/astara
useradd --home-dir "/var/lib/astara" --create-home --system --shell /bin/false astara

chown -R astara:astara /var/log/astara /var/lib/astara /etc/astara
```

   Set up log rotation:

```
cat >> /etc/logrotate.d/astara << EOF

/var/log/astara/*.log {

  daily

  missingok

  rotate 7
```

```
    compress

    notifempty

    nocreate

}

EOF
```

Give astara sudo permissions:

```
cat > '/etc/sudoers.d/astara_sudoers' << EOF
Defaults:astara !requiretty

astara ALL = (root) NOPASSWD: /usr/local/bin/astara-rootwrap  /etc/astara/rootwrap.conf *

EOF
```

2. Get the code:

```
cd ~
git clone git://git.openstack.org/openstack/astara
git clone git://git.openstack.org/openstack/astara-neutron
git clone git://git.openstack.org/openstack/astara-appliance
```

3. Install the code:

```
# If you are not building packages and just installing locally, manually install it via pip:

cd ~/astara
pip install .

cd ~/astara-neutron
pip install .
cd ~
```

4. Configure Neutron:

   Make required changes to the neutron configuration file:

   In /etc/neutron/neutron.conf, set in the [DEFAULT] section:

   To use the Astara Neutron ML2 plugin change the core_plugin and service_plugins to:

```
core_plugin = astara_neutron.plugins.ml2_neutron_plugin.Ml2Plugin
service_plugins = astara_neutron.plugins.ml2_neutron_plugin.L3RouterPlugin
```

   And also the add the API extension path (Note: append the astara path to existing list of extension paths if you have others specified):

```
api_extensions_path = /usr/local/lib/python2.7/dist-packages/astara_neutron/extensions/
```

   Note: the path shown will vary with the distribution for Ubuntu it will be /usr/lib/python2.7/dist-packages/astara_neutron/extensions/ for Red Hat installations this path will be different.

   Configure Neutron to emit event notifications:

```
notification_driver  = neutron.openstack.common.notifier.rpc_notifier
```

In /etc/neutron/plugins/ml2/ml2_conf.ini in the [ml2] section add:

```
extension_drivers = port_security
```

Ensure that l2population is enabled. On all nodes running the l2 agent, either Linuxbridge or OpenvSwitch (namely the compute nodes and nodes running the orchestrator process), in the ml2 ini file set:

Add l2population to the mechanism_drivers line

To the [agent] sections add:

```
l2_population = True
```

Depending on the layer 2 technology used in your OpenStack environment to enable layer 2 population additional parameters may need to be set. Check the OpenStack configuration guide for information about additional layer 2 setting for the layer 2 type and to tenant isolation type (VLAN, VXLAN of GRE) being used.

5. Configure Nova to use astara in the [DEFAULT] section of /etc/nova/nova.conf set:

   If using IPv6:

   ```
   use_ipv6=True
   ```

   In the [neutron] section of /etc/nova/nova.conf set:

   ```
   service_metadata_proxy = True
   ```

   In /etc/nova/policy.json, replace:

   ```
   "network:attach_external_network": "rule:admin_api"
   ```

   with:

   ```
   "network:attach_external_network": "rule:admin_api or role:service"
   ```

6. Start/restart Nova API to read the configuration changes:

   ```
   restart nova-api
   ```

   Restart the neutron services:

   ```
   restart neutron-server
   restart neutron-linuxbridge
   ```

   Stop and disable any L3 agents such as the DHCP agent, L3 agent or the metadata agent.

   Create a management network:

   ```
   neutron net-create mgt # note the ID, it is used in the orchestrator.ini config
   neutron subnet-create --name mgt-subnet mgt fdca:3ba5:a17a:acda::/64 --ip-version=6 --ipv6_addre
   ```

   Create a public network:

   ```
   neutron net-create --shared --router:external public
   neutron subnet-create --name public-subnet public 172.16.0.0/24
   ```

7. Configure Astara:

   For this configuration, we assume an IPv6 Neutron network /w prefix fdca:3ba5:a17a:acda::/64 has been created to be used as the management network:

```
mkdir /etc/astara
cp -r ~/astara/etc/* /etc/astara/
mv /etc/astara/orchestrator.ini.sample /etc/astara/orchestrator.ini
chown astara:astara /etc/astara/*.{ini,json}
```

Create a ssh keypair to enable ssh key based logins to the router:

```
ssh-keygen
```

It is best to copy the public ssh key into the astara configuration directory:

```
cp ~/.ssh/id_rsa.pub /etc/astara
chmod 600 /etc/astara
```

In the astara orchestrator configuration file (/etc/astara/orchestrator.ini) make the following changes:

In the [oslo_messaging_rabbit] section set:

```
rabbit_userid = guest
rabbit_password = guest
rabbit_hosts = 10.0.1.4
```

Set up logging:

```
log_file = /var/log/astara/orchestrator.log
```

Set the prefix of the existing Neutron network to be used used as management network used during subnet creation (above):

```
management_prefix = fdca:3ba5:a17a:acda::/64
```

The neutron subnet id of the management network and subnet:

```
management_net_id = $management_net_uuid
management_subnet_id = $management_subnet_uuid
```

The neutron network if of the external network:

```
external_network_id=$public_network_id
external_subnet_id=$public_subnet_id
```

Public SSH Key used for SSH'ing into the appliance VMs as user 'astara' (this is optional):

```
ssh_public_key = $path_to_readable_ssh_pub_key #From the above step this should be /etc/asta
```

The interface driver is used for bringing up a local port on the astara control node that plugs into the management network. This is specific to the underlying L2 implementation used, set accordingly:

```
interface_driver=astara.common.linux.interface.BridgeInterfaceDriver  #For Linuxbridge
interface_driver=astara.common.linux.interface.OVSInterfaceDriver #For OpenvSwitch
```

Correct the provider rules path:

```
provider_rules_path=/etc/astara/provider_rules.json
```

In the [keystone_authtoken] section, configure the credentials for the keystone service tenant as configured in your environment, specifically:

```
auth_uri = http://127.0.0.1:5000     # Adjust the IP for the current installation
project_name = service
```

```
password = neutron
username = neutron
auth_url = http://127.0.0.1:35357    # Adjust the IP for the current installation
auth_plugin = password
```

In the [database] section, configure URL to supported oslo.db backend, ie:

```
connection = mysql+pymysql://astara:astara@127.0.0.1/astara?charset=utf8
```

8. Create and Migrate the DB:

   Install the PyMySQL pip package:

```
pip install PyMySQL
```

   And create the database set database access permissions:

```
mysql -u root -pmysql -e 'CREATE DATABASE astara;'
mysql -u root -pmysql -e "GRANT ALL PRIVILEGES ON astara.* TO 'astara'@'localhost' IDENTIFIED BY
mysql -u root -pmysql -e "GRANT ALL PRIVILEGES ON astara.* TO 'astara'@'%' IDENTIFIED BY 'astara
astara-dbsync --config-file /etc/astara/orchestrator.ini upgrade
```

9. Create or download an Appliance Image

   If you don't plan to build your own appliance image, one can be downloaded for testing at: http://tarballs.openstack.org/akanda-appliance/images/

   If you want to build one yourself instructions are found in the appliance documentation In either case, upload the image to Glance (this command must be performed in the directory where the image was downloaded/created):

```
openstack image create astara --public --container-format=bare --disk-format=qcow2 --file astara
```

   Note the image id for the next step

   Update /etc/astara/orchestrator.ini and set this in the [router] section:

```
image_uuid=$image_uuid_in_glance
```

   You may also want to boot appliances with a specific nova flavor, this may be specified in the [router] section as: Create a new flavor:

```
nova flavor-create m1.astara 6 512 3 1 --is-public True
```

   Set the flavor in /etc/astara/orchestrator.ini:

```
instance_flavor=$nova_flavor_id
```

10. Start astara:

    astara-orchestrator –config-file /etc/astara/orchestrator.ini

    For Ubuntu or Debian systems use the following to create an upstart script to automatically start astara-orchestrator on boot:

```
cat > /etc/init/astara.conf << EOF
description "Astara Orchestrator server"

start on runlevel [2345]
stop on runlevel [!2345]

respawn
```

```
    exec start-stop-daemon --start --chuid astara --exec /usr/local/bin/astara-orchestrator -- --con

    EOF
```

Note: For RHEL or CentOS use the command:

```
    sudo -u astara  /usr/local/bin/astara-orchestrator --config-file=/etc/astara/orchestrator.ini &
```

Note: to automatically start the orchestrator process a systemd startup script will need to be created. Start the astara orchestrator process:

```
    start astara
```

1.6.3 Use Astara

If you have existing routers in your environment, astara will find them and attempt to boot appliances in Nova. If not, create a router and it should react accordingly. Otherwise use the following to create a privte network, create a router and add the network interface to the rputer:

```
neutron net-create private
neutron subnet-create --name private-subnet private 10.2.0.0/24

neutron router-create MyRouter
neutron router-interface-add MyRouter private
```

Boot a VM (replacing the <—> with the appropriate information):

```
nova boot --image <VM image name> --flavor 1 --nic net-id=<private network UUID> <name>
```

At this time sourcing the admin's credentials and using the command:

```
nova list --all-tenants
```

Output similar to:

```
+--------------------------------------+--------------------------------------------------+-----------
| ID                                   | Name                                             | Tenant ID
+--------------------------------------+--------------------------------------------------+-----------

| 1003335d-640c-4492-8054-80c4d23f9552 | Three                                            | fbf54d3e3f
| e75a0429-15cb-41a2-ae7b-890315b75922 | ak-router-6aa27c79-8ed4-4c59-ae83-4c4da725b3ec | d9aa8deb2d
+--------------------------------------+--------------------------------------------------+-----------
```

The line with the ak-router shows that astara has built the router VM. Further operation and debug information can be found in the *operator tools* section.

1.6.4 Clustering astara-orchestrator

The `astara-orchestartor` service supports clustering among multiple processes spanning multiple nodes to provide active/active clustering for purposes of load-distribution and high-availability (HA). In this setup, multiple `astara-orchestrator` processes form a distributed hash ring, in which each is responsible for orchestrating a subset of virtual appliances. When one `astara-orchestrator` falls offline, management of its resources are redistributed to remaining nodes. This feature requires the use of an external coordination service (ie, zookeeper), as provided by the tooz library. To find out more about which services `tooz` supports, see http://docs.openstack.org/developer/tooz/drivers.html.

To enable this feature, you must set the following in `orchestrator.ini`:

```
[coordination]
enabled=True  # enable the feature
url=kazoo://zookeeper.localnet:2181?timeout=5  # a URL to a tooz-supported coordination service
group_id=astara.orchestrator # optional, change this if deploying multiple clusters
heartbeat_interval=1 # optional, tune as needed
```

1.7 Install an Astara Load Balancer

1.7.1 How to configure Astara to be able to create load balancers

In this example we will create an image that can be used for both a router or a loadbalancer. Then we will configure both astara and neutron for loadbalancer support, which will use the LBAASV2 commands. We can then use the LBAASv2 API to create a loadbalancer.

1.7.2 Build loadbalancer applicance image:

Build an image to include loadbalancer support by using one of the two following commands. If you have a license for nginx plus you will be able to take advantage of some of these nginx-plus features but you must first copy over your nginx certs. Run this commad in the astara-appliance directory:

```
ELEMENTS_PATH=diskimage-builder/elements \
DIB_RELEASE=jessie DIB_EXTLINUX=1 \
DIB_ASTARA_ADVANCED_SERVICES=router,loadbalancer \
disk-image-create debian vm astara nginx -o astara-lb
```

or for nginx plus (nginx certs will need to be copied over before running this command). Run this commad in the astara-appliance directory:

```
ELEMENTS_PATH=diskimage-builder/elements \
DIB_RELEASE=jessie DIB_EXTLINUX=1 \
DIB_ASTARA_ADVANCED_SERVICES=router,loadbalancer \
disk-image-create debian vm astara nginx-plus -o astara-lb
```

1.7.3 Configure Neutron for Astara loadbalancer support

1. Ensure that neutron LBAAS packages are installed or install neutron-lbaas from source as follows:

```
    git clone https://git.openstack.org/openstack/neutron-lbaas
    cd neutron-lbaas
    pip install -U .
```

2. Make the following changes to neutron.conf in the [DEFAULT] section:

```
core_plugin = astara_neutron.plugins.ml2_neutron_plugin.Ml2Plugin
service_plugins = astara_neutron.plugins.ml2_neutron_plugin.L3RouterPlugin,astara_neutron.plugins.lba
api_extensions_path = /usr/local/lib/python2.7/dist-packages/astara_neutron/extensions:/usr/local/lib
```

in the [SERVICE_PROVIDERS] section (you may have to add this section if it doesn't exist):

```
service_provider = LOADBALANCERV2:LoggingNoop:neutron_lbaas.drivers.logging_noop.driver.LoggingNoopLo
```

3. Create the loadbalancer tables in the neutron database:

```
    neutron-db-manage --subproject neutron-lbaas upgrade head
```

1.7.4 Configure Astara for loadbalancer support

1. Make the following changes to orchestrator.conf.

in the [DEFAULT] section:

```
enabled_drivers = router,loadbalancer
```

in the [LOADBALANCER] section:

```
image_uuid = <loadbalancer image ID>
instance_flavor = 6
```

(If you are using this image for the router also, in the [ROUTER] section, set the image_uuid to this value also.)

2. Restart the neutron-server and astara services to pick up the changes:

```
    restart neutron-server
    restart astara
```

1.7.5 Create a loadbalancer

1. Build a loadbalancer (this assumes that you have two web servers at ips -WEB1_IP, WEB2_IP which will used in the following commands):

```
neutron lbaas-loadbalancer-create --name lb1 private-subnet
neutron lbaas-loadbalancer-show lb1 # Note the VIP address
neutron lbaas-listener-create --loadbalancer lb1 --protocol HTTP --protocol-port 80 --name listener1
neutron lbaas-pool-create --lb-algorithm ROUND_ROBIN --listener listener1 --protocol HTTP --name pool
neutron lbaas-member-create  --subnet private-subnet --address 10.2.0.4 --protocol-port 80 --name mem
neutron lbaas-member-create  --subnet private-subnet --address 10.2.0.5 --protocol-port 80 --name mem
neutron lbaas-healthmonitor-create --delay 3 --type HTTP  --max-retries 3 --timeout 3 --pool pool1 --
```

2. Once finished you can delete everything using the following:

```
    neutron lbaas-member-delete mem1 pool1
    neutron lbaas-member-delete mem2 pool1
    neutron lbaas-pool-delete pool1
    neutron lbaas-listener-delete listener1
    neutron lbaas-loadbalancer-delete lb1
```

1.8 Astara Developer Quickstart

This guide provides guidance for new developers looking to get up and running with an Astara development environment. The Astara components may be easily deployed alongside OpenStack using DevStack. For more information about DevStack visit `http://docs.openstack.org/developer/devstack/`.

1.8.1 Deploying Astara using DevStack

Preparation and prerequisites

Deploying DevStack on your local workstation is not recommended. Instead, developers should use a dedicated virtual machine. Currently, Ubuntu Trusty 14.04 is the tested and supported base operating system. Additionally, you'll need at least 4GB of RAM (8 is better) and to have `git` installed:

```
sudo apt-get -y install git
```

First clone the DevStack repository:

```
sudo mkdir -p /opt/stack/
sudo chown `whoami` /opt/stack
git clone https://git.openstack.org/openstack-dev/devstack /opt/stack/devstack
```

Configuring DevStack

Next, you will need to enable the Astara plugin in the DevStack configuration and enable the relevant services:

```
cat >/opt/stack/devstack/local.conf <<END
[[local|localrc]]
enable_plugin astara https://github.com/openstack/astara
enable_service q-svc q-agt astara
disable_service n-net

HOST_IP=127.0.0.1
LOGFILE=/opt/stack/logs/devstack.log
DATABASE_PASSWORD=secret
RABBIT_PASSWORD=secret
SERVICE_TOKEN=secret
SERVICE_PASSWORD=secret
ADMIN_PASSWORD=secret
END
```

You may wish to SSH into the appliance VMs for debugging purposes. The orchestrator will enable access for the 'astara' user for a specified public key. This may be specified by setting ASTARA_APPLIANCE_SSH_PUBLIC_KEY variable in your devstack config to point to an existing public key. The default is $HOME/.ssh/id_rsa.pub.

Building a Custom Service VM

By default, the Astara plugin downloads a pre-built official Astara image. To build your own from source, enable `BUILD_ASTARA_APPLIANCE_IMAGE` and specify a repository and branch to build from:

```
cat >>/opt/stack/devstack/local.conf <<END

BUILD_ASTARA_APPLIANCE_IMAGE=True
ASTARA_APPLIANCE_REPO=http://github.com/openstack/astara-appliance.git
ASTARA_APPLIANCE_BRANCH=master
END
```

To build the appliance using locally modified `astara-appliance` code, you may point devstack at the local git checkout by setting the ASTARA_APPLIANCE_DIR variable. Ensure that any changes you want included in the image build have been committed to the repository and it is checked out to the proper commit.

Deploying

Simply run DevStack and allow time for the deployment to complete:

```
cd /opt/stack/devstack
./stack.sh
```

After it has completed, you should have a `astara_orchestrator` process running alongside the other services and an Astara router appliance booted as a Nova instance.

1.9 Configuration Options

`astara-orchestrator` uses `oslo.config` for configuration, so it's configuration file format should be very familiar to OpenStack deployers

1.10 Astara Release Notes

1.10.1 Astara Mitaka Series Release Notes (UNRELEASED)

8.0.0

Astara has dropped a number of legacy convenience hooks available in earlier releases. The hooks complicated automation and created potential for mismatch of end state and the desired state.

Astara Mitaka Series Release v8.0.0.

New Features

- Blueprint astara-rootwrap - We replace shelling out directly to sudo with the oslo.rootwrap library.
- blueprint autogen-astara-conf-file - This switches astara to use oslo-config-generator, where the contents of our sample configuration file are configured using a configuration file in etc/oslo-config-generator/.
- Operators may now associate custom drivers and image IDs to tenants, via the Neutron API, to override global configuration, providing support for dynamic user-provided network functions. To enable this feature, set `enable_byonf=True` in `orchestrator.ini` and be sure the version of `astara-neutron` loaded into Neutron supports the BYONF API.
- Astara now supports orchestrating clustered pairs of appliance VMs for Neutron routers that have the been set to highly-available.
- The orchestrator now pushes local orchestrator-specific configuration into the appliance, allowing services like the metadata proxy to be configured specifically for current cluster layout.

Upgrade Notes

- Astara will no longer automatically add the external gateway to a router. Previous usage was causing issues with automation tooling.
- Astara no longer requires the external network and subnet id to be known. In production deployments this step was handled externally and the internal hooks were often disabled.

Deprecation Notes

- The `amqp_url` config option has been deprecated in favor using `oslo.messaging` backend specific configution. See example configuration file for an example. The pre-Liberty rabbit options have been removed.

Critical Issues

- The devstack plugin no longer creates the external network as before and instead follows the setup used for reference implementation.

Bug Fixes

- Bug 1539786 Varible MTU support is now supported by the orchestrator and passed to appliance. This requires Neutron with MTU extension enabled to support.
- Bug 1524979, Bug 1528338 - astara-debug-router command has been fixed
- Bug 1537500 Fixes tenant_id issue when rebuilding router from the astara-ctl browser
- Bug 1527396 Fixes issue where, after a cluster rebalance, stat machines are created across all workers and instead ensures they are only created on a single target worker.
- Bug 1524595 astara-ctl warning message for deprecated AMQP configuration
- Bug 1539345 auto added resources break interoperability
- Bug 1524068 Local management port addresses are now allocated from the management subnet rather than using a hard-coded address, fixing Neutron port address conflicts when clustering astara-orchestrators.
- Bug 152492 Fixed astara-ctl ssh command
- Bug 1535857 The additional "leadership" member reported by zookeeper is now ignored to avoid hashing resources to a non-existent node.
- Bug 1531597 - Deleted resources are properly invalidated from the local tenant resource cache

CHAPTER 2

Licensing

Astara is licensed under the Apache-2.0 license and is copyright Akanda, Inc.