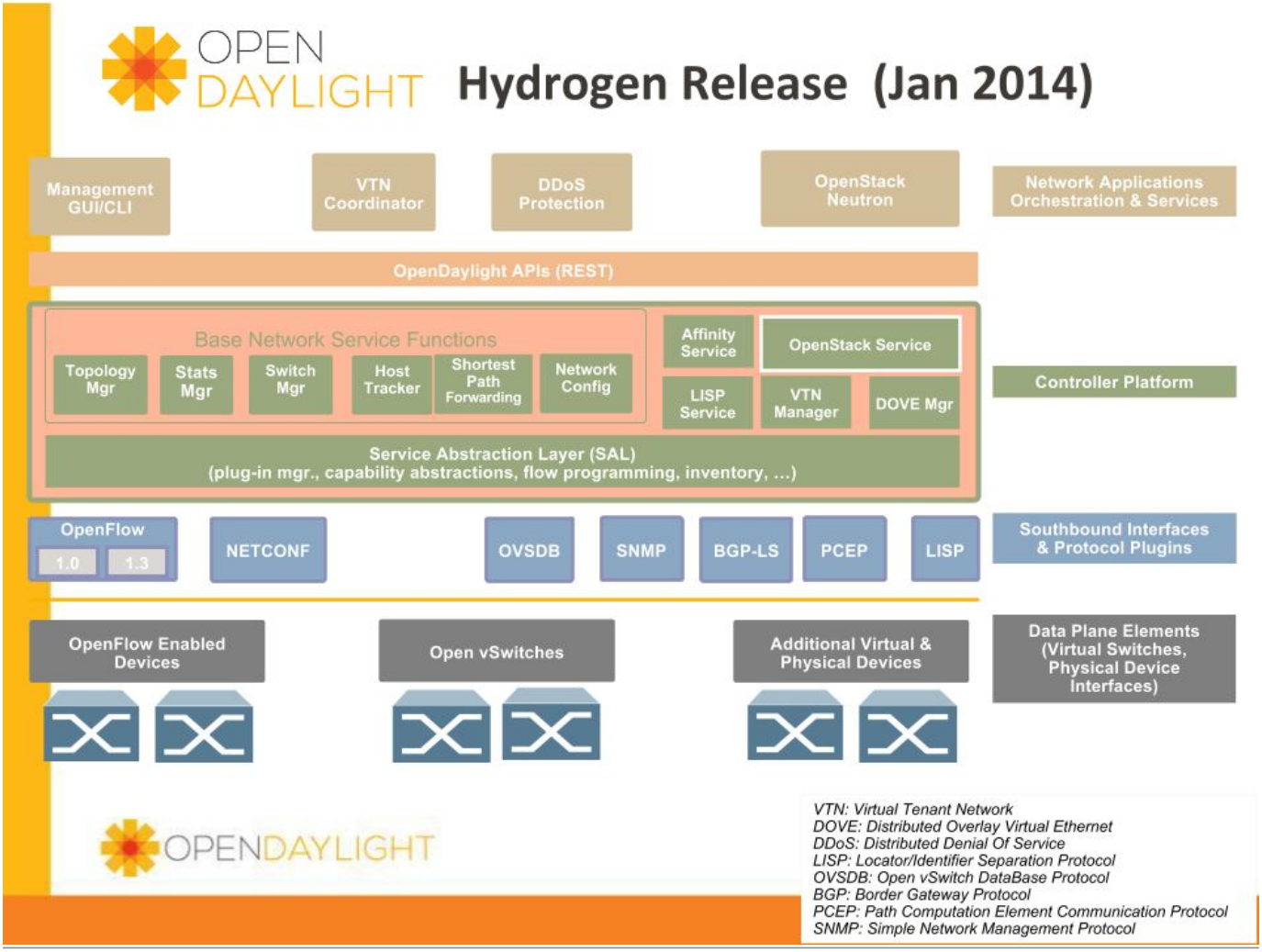
Getting Started With OpenDaylight and OpenStack

Posted by mestery on December 16, 2013

If you're a fan of networking, you are no doubt very excited by all of the recent excitement in the industry as of late. And there is no larger area of innovation in networking at the moment than Open Source networking. Two of the projects at the forefront of Open Source networking innovation are OpenStack Neutron and OpenDaylight. OpenStack Neutron is driving an API around networking for Infrastructure as a Service Clouds, and has been very successful at driving mindshare in this area. There are a large number of Plugins and ML2 MechanismDrivers for Neutron in existence already. However, so far, there is no OpenDaylight integration with OpenStack, at least upstream. I am pleased to announce that a team of us are working on making this happen, however. We have a blueprint filed and we are actively working towards the support in OpenDaylight required to support the Neutron APIs. What I'm going to show you in this blog post is how to take what we currently have for a test run and try it out yourself.

OpenDaylight Integration with OpenStack: The Details

OpenDaylight is a highly scalable controller written in Java. It is designed from the start to be modular. Perhaps the best way to understand the Modular nature of OpenDaylight is to look at an architecture diagram of it:



OpenDaylight Hydrogen Release Architecture Diagram

You can see all the pieces of OpenDaylight, and there are quite a few. Because of the modular nature of OpenDaylight it makes heavy use of the OSGI framework. I'm not going to go into extreme details of how this works, but suffice to say it allows for anyone to write a bundle which can run and interact with other bundles in OpenDaylight.

As part of this, there exists a few bundles which are relevant to the OpenStack integration efforts:

- NeutronAPIService
- OVSDB
- OpenFlow

Each of those bundles provides a necessary component in the OpenStack integration. The NeutronAPIService provides an abstraction of the Neutron APIs into OpenDaylight. It caches all of the Neutron objects inside of OpenDaylight providing access to this information to anything in OpenDaylight which requires it. The OVSDB and OpenFlow OSGI bundles in OpenDaylight provide the code which

actually programs things on each compute host. They allow for the creation and deletion of tunnel ports, flow programming for ports as they come and go, and bridge creation and deletion on the host.

The main benefit of the above is that each compute host no longer needs an Open vSwitch Agent running on each host. The combination of OpenFlow and OVSDB provide the equivalent functionality as the Open vSwitch Agent.

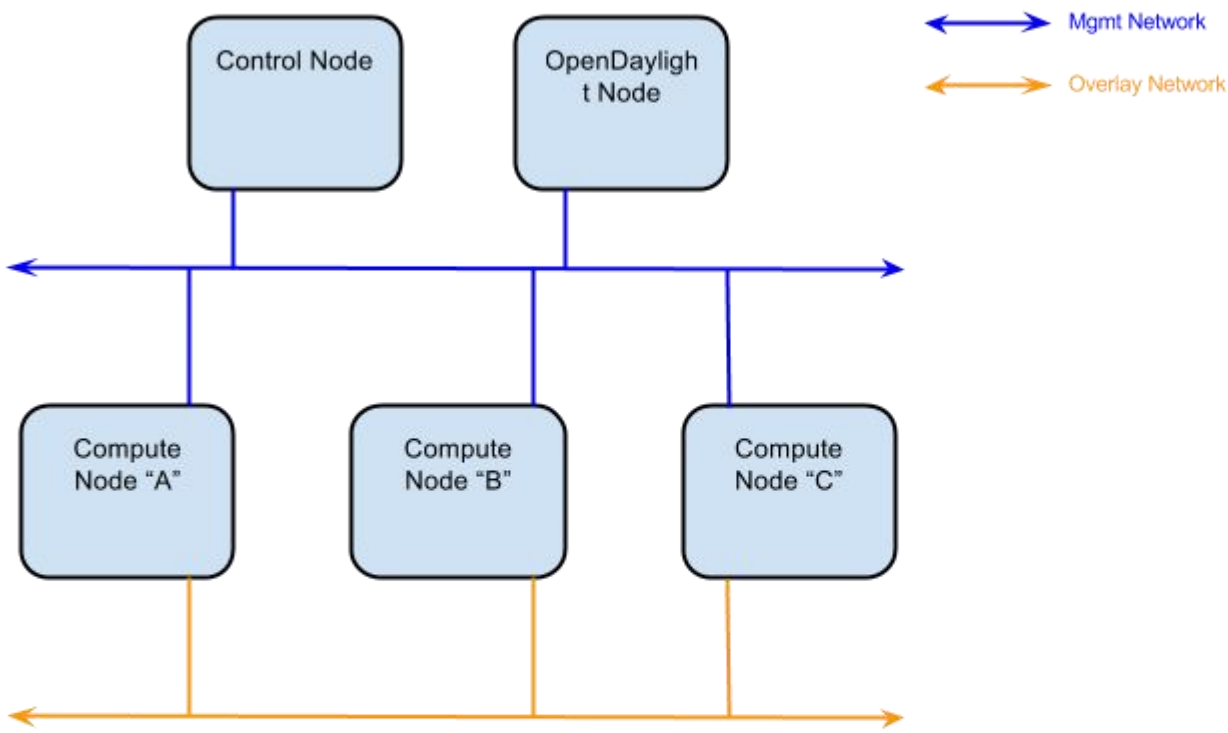
OpenDaylight and OpenStack: Getting Started

To test out the latest OpenDaylight Modular Layer2 MechanismDriver, you will need the following:

- A machine to run the OpenDaylight Controller
- A machine to run the OpenStack control software
- At least one machine to run the OpenStack Compute service to run virtual machines

Now, you can combine some of the things above, and you should most certainly run all of the above as virtual machines. I personally run all of the above as virtual machines on VMware Fusion and have one VM in which I run OpenDaylight, one VM in which I run the OpenStack control software, and 3 other VMs in which I run OpenStack compute services. A fairly minimum setup would be 3 VMs, however: One to run the OpenDaylight controller, one to run OpenStack control and compute services, and another one to run only OpenStack compute services.

In either case, your topology will look very similar to the following diagram:



OpenStack and OpenDaylight Integration

OpenDaylight and OpenStack: Building and Installing OpenDaylight

Lets get started with the actual configuration of the system now. The first piece is your OpenDaylight VM. To build and install this, follow the steps below. I should note a much larger view of building the controller is on the wiki page here, the instructions below are mostly meant to get you going very fast without having to read that wiki page in detail.

```
mkdir ~/odl
cd odl
git clone https://git.opendaylight.org/gerrit/p/controller.git
cd opendaylight/distribution/opendaylight/
mvn clean install
cd ~/odl
git clone https://git.opendaylight.org/gerrit/p/ovsdb.git
cd ovsdb
```

At this point, you can cut and paste the script below as “build_ovsdb.sh” and use that to build OVSDB and copy the bundles over to the controller:

```
#!/bin/sh
git pull
cd neutron
echo "Refreshing ovsdb/neutron.."
pwd
mvn clean install
cd ../northbound/ovsdb/
echo "Refreshing northbound/ovsdb.. "
pwd
mvn clean install
cd ../../ovsdb
echo "Refreshing ovsdb/ovsdb.."
pwd
mvn clean install
cd ..
cp ~/odl/ovsdb/neutron/target/ovsdb.neutron-0.5.0-SNAPSHOT.jar
~/odl/controller/opendaylight/distribution/opendaylight/target/distribution.opendayligh
t-osgipackage/opendaylight/plugins/
cp ~/odl/ovsdb/northbound/ovsdb/target/ovsdb.northbound-0.5.0-SNAPSHOT.jar
```

```
~/odl/controller/opendaylight/distribution/opendaylight/target/distribution.opendaylight-osgipackage/opendaylight/plugins/
cp ~/odl/ovsdb/ovsdb/target/ovsdb-0.5.0-SNAPSHOT.jar ~/odl/controller/opendaylight/distribution/opendaylight/target/distribution.opendaylight-osgipackage/opendaylight/plugins/
echo "done!"
```

Once you've created the script, simply make sure it has execute permissions (chmod +x build_ovsdb.sh) and run it and you will have the OVSDB bundles created and installed into the plugins directory. To verify they are there, look in the following location:

- odl/controller/opendaylight/distribution/opendaylight/target/distribution.opendaylight-osgipackage/opendaylight/plugins

The next step is to modify the "of.address" variable in the "configuration/config.ini" file. This file is relative to the odl/controller/opendaylight/distribution/opendaylight/target/distribution.opendaylight-osgipackage directiory. Fire up vi and add the management IP address for your ODL instance as the value for of.address.

Now it's time to fire up your controller! To do that, execute the following:

```
cd ~/odl/controller/opendaylight/distribution/opendaylight/target/distribution.opendaylight-osgipackage/opendaylight
./run.sh
```

Once the controller is running, you will want to disable the SimpleForwarding Application, so do the following:

- In the OSGI console, run "lb | grep simple" to find the bundle ID of the simpleforwarding application.
- Run "stop <bundle ID>" to disable simpleforwarding.
- Run "lb | grep simple" to verify it is in the "Resolved" state.

The entire thing looks like below:

```
osgi> lb | grep simple
 132|Active | 4|samples.simpleforwarding (0.4.1.SNAPSHOT)
 true
 osgi> stop 132
 osgi> lb | grep simple
 132|Resolved | 4|samples.simpleforwarding (0.4.1.SNAPSHOT)
 true
 osgi>
```

OpenStack and OpenDaylight: Readying the devstack nodes

At this point, you have an OpenDaylight controller running. Now it's time to fire up your devstack nodes. You will need at least two virtual machines ready for this. They can run anything which devstack supports. I am ardent user of Fedora Linux, so that's what I use, but Ubuntu works fine as well. Note if you're using Ubuntu 12.04 LTS, that particular variant of Ubuntu is using OVS 1.4, which is quite a bit old. Fedora 19 uses a much newer version of OVS.

One thing to note is that you should make sure you have passwordless "sudo" access setup for the account you're running devstack as.

So, the next thing to do on each node is to checkout devstack:

```
cd ~/
git clone  git://github.com/openstack-dev/devstack.git
cd devstack
git remote add opendaylight https://github.com/CiscoSystems/devstack.git
git fetch opendaylight
git checkout opendaylight
```

Run the above on each devstack node. It will checkout the customer OpenDaylight devstack branch. Now to configure your local.conf files.

On the control node, your local.conf will look like the below:

```
[[local|localrc]]
LOGFILE=stack.sh.log
#SCREEN_LOGDIR=/opt/stack/data/log
#LOG_COLOR=False
#OFFLINE=True
RECLONE=yes

# Only uncomment the below two lines if you are running on Fedora
disable_service rabbit
enable_service qpid
disable_service n-cpu
enable_service n-cond
disable_service n-net
enable_service q-svc
enable_service q-dhcp
enable_service q-l3
enable_service q-meta
enable_service quantum
enable_service tempest

Q_HOST=$SERVICE_HOST
HOST_IP=192.168.64.193

Q_PLUGIN=ml2
Q_ML2_PLUGIN_MECHANISM_DRIVERS=opendaylight,logger
ENABLE_TENANT_TUNNELS=True
NEUTRON_REPO=https://github.com/CiscoSystems/neutron.git
NEUTRON_BRANCH=odl_ml2

VNCSERVER_PROXYCLIENT_ADDRESS=192.168.64.193
```

```
VNCSERVER_LISTEN=0.0.0.0

HOST_NAME=km-dhcp-64-193.kmestery.cisco.com
SERVICE_HOST_NAME=${HOST_NAME}
SERVICE_HOST=192.168.64.193

FLOATING_RANGE=192.168.210.0/24
PUBLIC_NETWORK_GATEWAY=192.168.75.254
MYSQL_HOST=$SERVICE_HOST
RABBIT_HOST=$SERVICE_HOST
GLANCE_HOSTPORT=$SERVICE_HOST:9292
KEYSTONE_AUTH_HOST=$SERVICE_HOST
KEYSTONE_SERVICE_HOST=$SERVICE_HOST

MYSQL_PASSWORD=mysql
RABBIT_PASSWORD=rabbit
QPID_PASSWORD=rabbit
SERVICE_TOKEN=service
SERVICE_PASSWORD=admin
ADMIN_PASSWORD=admin

[[post-config|/etc/neutron/plugins/ml2/ml2_conf.ini]]
[agent]
minimize_polling=True

[ml2_odl]
url=http://192.168.64.131:8080/controller/nb/v2/neutron
username=admin
password=admin
```

You should note in the above you will want to change the following:

- HOST_IP: This is the management IP of the control host itself.
- VNCSERVER_PROXYCLIENT_ADDRESS: The management IP address of the control node itself.
- HOST_NAME: The host name of the control node.
- SERVICE_HOST: The management IP of the control node.
- The “url” parameter in the ml2_odl section near the bottom: Make sure the url and credentials match your OpenDaylight configuration. If you didn’t change the default username password for ODL, you can leave those bits alone.

Once you have that done, the next step is to setup your local.conf for the compute nodes:

```
[[local|localrc]]
LOGFILE=stack.sh.log
#LOG_COLOR=False
#SCREEN_LOGDIR=/opt/stack/data/log
#OFFLINE=true
RECLONE=yes

disable_all_services
enable_service neutron nova n-cpu quantum n-novnc qpid

HOST_NAME=km-dhcp-64-197.kmestery.cisco.com
HOST_IP=192.168.64.197
SERVICE_HOST_NAME=km-dhcp-64-193.kmestery.cisco.com
SERVICE_HOST=192.168.64.193
VNCSERVER_PROXYCLIENT_ADDRESS=192.168.64.197
VNCSERVER_LISTEN=0.0.0.0

FLOATING_RANGE=192.168.210.0/24

NEUTRON_REPO=https://github.com/CiscoSystems/neutron.git
NEUTRON_BRANCH=odl_ml2
```

```
Q_PLUGIN=ml2
Q_ML2_PLUGIN_MECHANISM_DRIVERS=opendaylight,linuxbridge
ENABLE_TENANT_TUNNELS=True
Q_HOST=$SERVICE_HOST

MYSQL_HOST=$SERVICE_HOST
RABBIT_HOST=$SERVICE_HOST
GLANCE_HOSTPORT=$SERVICE_HOST:9292
KEYSTONE_AUTH_HOST=$SERVICE_HOST
KEYSTONE_SERVICE_HOST=$SERVICE_HOST

MYSQL_PASSWORD=mysql
RABBIT_PASSWORD=rabbit
QPID_PASSWORD=rabbit
SERVICE_TOKEN=service
SERVICE_PASSWORD=admin
ADMIN_PASSWORD=admin

[[post-config|/etc/neutron/plugins/ml2/ml2_conf.ini]]
[agent]
minimize_polling=True

[ml2_odl]
url=http://192.168.64.131:8080/controller/nb/v2/neutron
username=admin
password=admin
```

Again, the parts to edit above on the compute nodes are:

- HOST_NAME: The host name of each compute node.
- HOST_IP: The management IP address of each host.

- SERVICE_HOST_NAME: The hostname of the control node.
- SERVICE_HOST: The management IP of the control node.
- ml2_odl: Modify the IP address there for the ODL controller.

Each local.conf file should be saved in the ~/devstack directory on each control and/or compute host.

Now you should be able to run “stack.sh” on all of the nodes (control and each compute) by doing this:

- cd ~/devstack
- ./stack.sh

Once that completes, you should have a functioning OpenStack setup with OpenDaylight.

Possible Issues With devstack on Fedora

One possible issue you may hit if you’re using a fresh VM on Fedora is mysql errors. You will see keystone errors and mysql access errors in the stack.sh run. To get around this, follow the workaround listed in this post here. It’s worked for me every time I hit this error running devstack on Fedora. One other issue with Fedora is that the latest devstack fails to kill all the nova processes when you run “unstack.sh.” To workaround this, simply run the following after “unstack.sh”:

- killall nova-api nova-cert nova-scheduler nova-consoleauth nova-api nova-conductor

OpenStack and OpenDaylight: Verifying The Install

At this point, you should have the entire system up and running. To verify this, you can do the following:

- Point your web browser at the OpenStack Horizon GUI:
    - http://<control node IP>/auth/login/
    - Login using “admin/admin” and you can see you OpenStack install.
- Point your web browser at the OpenDaylight GUI:
    - http://<odl IP>:8080/
    - Login using “admin/admin”

You can play around in the GUIs, launch VMs, etc. As you launch VMs, you will see ODL create tunnel ports and links between compute hosts, which will become visible with a refresh in the OpenDaylight GUI.

OpenStack and OpenDaylight: Getting Help

The most appropriate place to get help at this early stage is on #opendaylight-ovsdb on Freenode. A long list of OpenStack Neutron and OpenDaylight developers hang out there and can provide help. Besides myself (IRC nick “mestery”), you can also expect to find the following people online:

- Madhu Venugopal (IRC Nick “Madhu”)
- Brent Salisbury (IRC Nick “networkstatic”)
- Keith Burns (IRC Nick “alagalah”)
- Florian Otel (IRC Nick “FlorianOtel”)

You can ping any of us and we should be able to help you debug any issues. Florian in particular has some VM images which may expedite the process above for folks trying this out for the first time. The instructions above were meant to walk through all of the steps necessary to get this up and running from scratch.

OpenStack and OpenDaylight: What’s Next

In a future post, I will walk through debugging this setup and using it see flows and how different pieces interact. In particular, I’ll walk through debugging this system so you understand exactly how things are done when networks, subnets, ports, routers and other OpenStack Neutron API objects are created and how OpenDaylight handles programming them onto each host.