Setup a virtual Openstack environment

Milestones 01: Build the virtual infrastructure

1 .Vagrant 101

After installing vagrant and the resizing plugin, inside the Vagrantfile directory, run

```
vagrant up
```

This command will create the typology based on the Vagrantfile.

To check that the Virtual machines are running

```
vagrant status
```

To connect to the created VMs:

```
vagrant ssh <vm name >
```

If you like to use putty or another ssh client, you can use the public IP with :
username: vagrant
password vagrant

To destroy the created infra:

```
vagrant destroy  -f
```

2. Create bare metal nodes

Note :The next commands must be runned on the bare metal VM

Change the directory

```
# cd /home/vagrant/openstack_scripts/baremetal/
```

Create a node

```
./create_baremetal.sh <node number > <disk size > <VCPU> <RAM>
```

Example:

```
./create_baremetal.sh 01  30G 1 4096
```

3. Add nodes to VirtualBMC

Note :The next commands must be runned on the bare metal VM

Add the node to VirtualBMC

```
vbmc add --port 6230 node_01
vbmc start node_01
```

Check the status of the vm

```
ipmitool -I lanplus -U admin -P password -H 127.0.0.1 -p 6230 power status
```

Turn off the vm

```
ipmitool -I lanplus -U admin -P password -H 127.0.0.1 -p 6230 power off

virsh list --all
```

In virsh you can see the VM in a shutdown state

Turn on the vm

```
ipmitool -I lanplus -U admin -P password -H 127.0.0.1 -p 6230 power on

virsh list --all
```

Milestones 02: Setup OpenStack LAB

Now our lab is ready we'll move to the install of OpenStack on the infrastructure that we built. In each VM you will find a directory called *openstack_scripts*

```
root@controller01:/home/vagrant# ls openstack_scripts/
baremetal  compute  controller  storage
```

For each VM use the right folder. inside you will find a script each one is used to install a node, you can check the used configurations in the *configs* directory.

1. Setup controller node

openstack_scripts/controller/deploy_controller.sh is used to deploy controller components on the controller, the script is splitted into functions, each function is responsible of installing a component, the *deploy_controller* script contains the following functions:

Function	Description
init	Upgrade system, install OpenStack repository and client
mysql	Deploy and configure a MySQL server
rabbitmq	Install and Create a rabbitmq user
memcache	Install memcache server
keystone	Install keystone and configure keystone
glance	Install keystone and configure glance
placement	Install keystone and configure placement
nova	Install keystone and configure nova
cinder	Install keystone and configure cinder
network	Create baremetal and public bridge and
neutron	Install keystone and configure neutron
horizon	Install keystone and configure horizon

ironic	Install keystone and configure ironic

To start deploying the controller node :

Login as root:

```
$ sudo su
```

Change the directory to :

```
# cd /home/vagrant/openstack_scripts/controller/
```

Run the script with the needed function

```
# ./deploy_controller.sh  <function>
```

Run the functions one by one and in each step check that the service is working properly, it is prefered to use the following order :

1. init
2. mysql
3. rabbitmq
4. memcache
5. keystone
6. glance
7. placement
8. nova
9. neutron
10. network
11. cinder
12. horizon
13. ironic

Note: *Make sure to run the network function after the neutron function.*

2. Setup compute node

Same as the controller, the deploy_compute.sh is responsible of deploying a compute node, it contains 3 functions :

Function	Description
init	Upgrade system, install OpenStack repository and client
compute	Install nova-compute and hypervisor
neutron	Install OpenVswitch and OVS agent

To start deploying the compute node :

Login as root:

```
$ sudo su
```

Change the directory to :

```
# cd /home/vagrant/openstack_scripts/compute/
```

Run the script with the needed function

```
# ./deploy_compute.sh  <function>
```

3. Setup storage node

Same as the controller, the deploy_storage.sh is responsible of deploying a storage node, it contains 2 functions :

Function	Description
init	Upgrade system, install OpenStack repository and client
storage	Install and configure cinder-volume, create a PV and VG using LVM

To start deploying the storage node :

Login as root:

```
$ sudo su
```

Change the directory to :

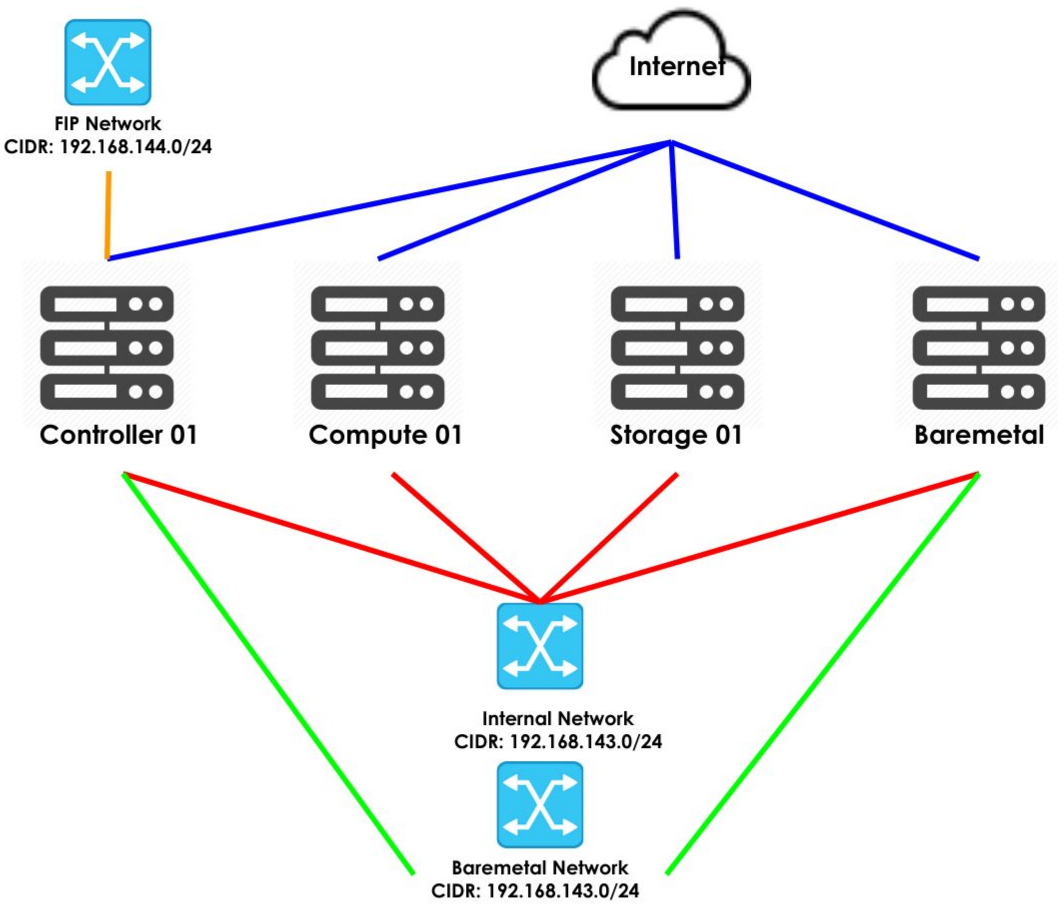
```
# cd /home/vagrant/openstack_scripts/storage/
```

Run the script with the needed function

```
# ./deploy_storage.sh  <function>
```

Milestones 03: Documentation

The Final part of the documentation will explain how to deploy and validate each openstack component, After deploying the Vagrant file, a topology as below will be created.



As shown in part two of the documentation, in each VM you can find scripts to deploy components on that particular node, please refer to the second part of the documentation to know how to use the scripts.

Note : *Before running any of the functions, please for controller, compute and storage run init function first, refer to the second part of the documentation for more information.*

1. **Requirements:**

Before running any openstack environments we need to install a set of requirements that will be used by Openstack

MySQL : Is the most important requirement, it is used by all components to persist the state of the openstack infrastructure: roles, users, VMs, ports, networks…, so make sure to run periode backups in a production environments.

```
# ./deploy_controller.sh  mysql
```

To validate it run :

```
# mysql -uroot
```

RabbitMQ : It is used for communications between components of the same project, for example between neutron server and neutron agents or between nova-api and nova-compute, if your RabbitMQ is down you can't create any resources in the infrastructure except for keystone and glance

```
# ./deploy_controller.sh  rabbitmq
```

To validate it run :

```
# rabbitmqctl status
```

Memcache: The Identity service authentication mechanism for services uses Memcached to cache tokens**,** it is not mandatory but preferred**,** in a production environments please make sure to secure it using encryption and Firewalling

```
# ./deploy_controller.sh  memcache
```

To validate it run :

```
# ss -laputen  | grep  11211
```

You can see a process in Listening mode.

2. **Identity Service (Keystone)**

Keystone is an OpenStack service that provides API client authentication, service discovery, and authorization, it is used for authenticating the clients with the OpenStack platform and authenticating each project with another project, for example if nova wants neutron to create a port for a new created VMs it sends a API call to neutron with nova credentials, so each service is authenticated to another. Keystone is used also as service discovery, so for example if nova is search for neutron API address it will search for it in keystone before sending the request to neutron.

```
# ./deploy_controller.sh  keystone
```

To validate it run :

Source login information to your bash.

```
# source ~/admin.sh
```

List created users, you will find only admin user create for now, you can try it with each install of new service.

```
# openstack user list
```

3. **Image service (glance)**

Glance image services include discovering, registering, and retrieving virtual machine (VM) images. Glance has a RESTful API that allows querying of VM image metadata as well as retrieval of the actual image.

```
# ./deploy_controller.sh  glance
```

To validate it run :

Source login information to your bash.

```
# source ~/admin.sh
```

List created images, the script download and add a cirros image to glance so you should see it if you run :

```
# openstack image list
```

4. Placement

This project used to be related to nova, but since stein release it becomes a separate project, This is a REST API stack and data model used to track resource provider inventories and usages, along with different classes of resources. In a simple word placement tracks all resources created on nodes including : RAM, CPU and storage.

```
# ./deploy_controller.sh  placement
```

5. Compute Service (Nova)

Nova is the OpenStack project that provides a way to provision compute instances (aka virtual servers). Nova supports creating virtual machines, baremetal servers (through the use of ironic). Nova runs as a set of daemons on top of existing Linux servers to provide that service; theses services are :

- **Nova API**

Accepts and responds to end user compute API calls. The service supports the OpenStack Compute API.

- **Nova Scheduler**

Takes a virtual machine instance request from the queue and determines on which compute server host it runs.

- **Nova novncproxy**

Provides a proxy for accessing running instances through a VNC connection.

- **Nova Conductor**

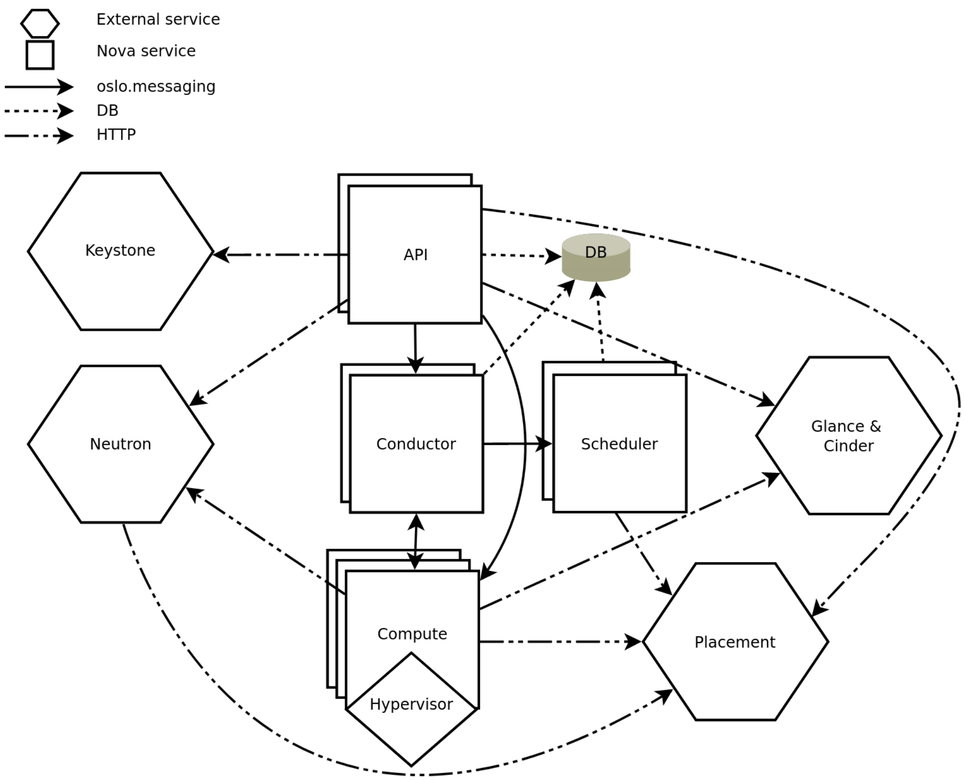
Mediates interactions between the nova-compute service and the database. It eliminates direct accesses to the cloud database made by the nova-compute service

- **Nova Compute**

A worker daemon that creates and terminates virtual machine instances through hypervisor APIs. it supports various hypervisors

- **Nova API Metadata**

The metadata service provides a way for instances to retrieve instance-specific data. Instances access the metadata service at http://169.254.169.254.



In our setup and usual setups, all components are installed on the controller with the exception of nova-compute, it must be installed on the computer node next to the hypervisor.

Note: Do not install nova-conductor on a compute node, it is not recommended

To deploy, on the controller run

```
# ./deploy_controller.sh  nova
```

On the compute node run

```
# ./deploy_compute.sh  compute
```

To validate it run :

```
#  openstack hypervisor list
```

If you don't see any output, repeat the command, there is cron that scans new nova-compute instances.

Create a Flavor, we will use it later.

```
#  openstack flavor create  --ram  256 --disk 2 --vcpus 1 m1.tiny
```

6. Networking Services (Neutron)

OpenStack Networking (neutron) allows you to create and attach interface devices managed by other OpenStack services to networks, we can call it *OpenStack SDN*. Like nova it is composed of multiples components :

- **Neutron Server :** Accepts and routes API requests to the appropriate OpenStack Networking plug-in for action.
- **Networking plug-ins and agents :** They are used to perform needed actions on the infrastructures, some of the important plugins and agents are:
  - **OpenVSwitch Agent :** Receive commands (create a port, attache a port, add flow, remove flow ...) from neutron-server and execute them as commands to OpenVSwitch.
  - **L3 agent :** Work with IPtables to apply security groups and handles also Floatings IPs NAT operations

- **DHCP agent :** Create and managed a DHCP for each network if DHCP is enabled on that network.

There are other plugins but in general cases the agent mentioned above are used.

To deploy, on the controller run

```
# ./deploy_controller.sh  neutron network
```

On the compute node run

```
# ./deploy_compute.sh  neutron
```

To validate it run, we'll create a private network, a public one and a router.

- Create a public network

```
# openstack network create  public --provider-network-type flat
--provider-physical-network physnet1 --share --external
```

- Create a public subnet

```
# openstack subnet create  --network  public  --subnet-range 192.168.144.0/24  --gateway
192.168.144.1  --allocation-pool start=192.168.144.100,end=192.168.144.200
public-subnet
```

- Create a private network

```
# openstack network create  --provider-network-type vxlan  private
```

- Create a private subnet

```
# openstack subnet create  --network  private --subnet-range  10.0.0.0/24  --gateway
10.0.0.1 private-subnet
```

- Create a router

```
# openstack router create  router1
# openstack router add subnet router1 private-subnet
# openstack router set router1 --external-gateway public
```

7. **Deploy a VM**

Create a security policy to allow SSH and PING

```
# openstack security group  create ping_ssh
# openstack security group rule create --ingress  --protocol icmp  --remote-ip 0.0.0.0/0 ping_ssh
# openstack security group rule create --ingress  --protocol tcp --dst-port 22   --remote-ip 0.0.0.0/0  ping_ssh
```

- Create a VM

```
#  openstack server create --flavor m1.tiny --network  private --image  cirros --security-group ping_ssh vm_01
```

- List VM status

```
# openstack server list
```

- Associate a Floating IP to the created VM

```
#  openstack floating ip create  public
```

Save the given IP address we'll use it later

- Associate a Floating IP to the created VM

```
#  openstack server add floating ip vm_01 <Floating IP>
```

From the local windows machine ping the Floating IP address

```
#  ping <Floating IP>
```

8. Block Storage Services (Cinder)

The OpenStack Block Storage service (Cinder) adds persistent storage to a virtual machine. Block Storage provides an infrastructure for managing volumes, and interacts with OpenStack Compute to provide volumes for instances. The service also enables management of volume snapshots, and volume types.

- **Cinder API**

Accepts API requests, and routes them to the cinder-volume for action.

- **Cinder Volume**

Interacts directly with the Block Storage service, and processes such as the cinder-scheduler. It also interacts with these processes through a message queue. The cinder-volume service responds to read and write requests sent to the Block Storage service to maintain state. It can interact with a variety of storage providers through a driver architecture.

- **Cinder Scheduler**

Selects the optimal storage provider node on which to create the volume. A similar component to the nova-scheduler.

- **Cinder Backup**

The cinder-backup service provides backing up volumes of any type to a backup storage provider. Like the cinder-volume service, it can interact with a variety of storage providers through a driver architecture.

To deploy, on the controller run

```
# ./deploy_controller.sh  cinder
```

On the storage node run

```
# ./deploy_compute.sh  storage
```

To validate it run, we'll create a volume

```
# openstack volume create --size 1 vol_1
```

- Attach the volume to the VM created earlier

```
# openstack server add volume vm_01 vol_1
```

9. Dashboard (Horizon)

To deploy, on the controller run

```
# ./deploy_controller.sh horizon
```

To validate it log into horizon :
URL : http://192.168.143.50/horizon
Username : *admin*
Password : *adminpass*

Note: *before proceeding to the install of ironic, make sure to create a Baremetal node using the documentation provided in the first milestone documentation*

10. Bare Metal service (Ironic)

The Bare Metal service, codenamed ironic, is a collection of components that provides support to manage and provision physical service. It is composed from various components :

- **Ironic API**

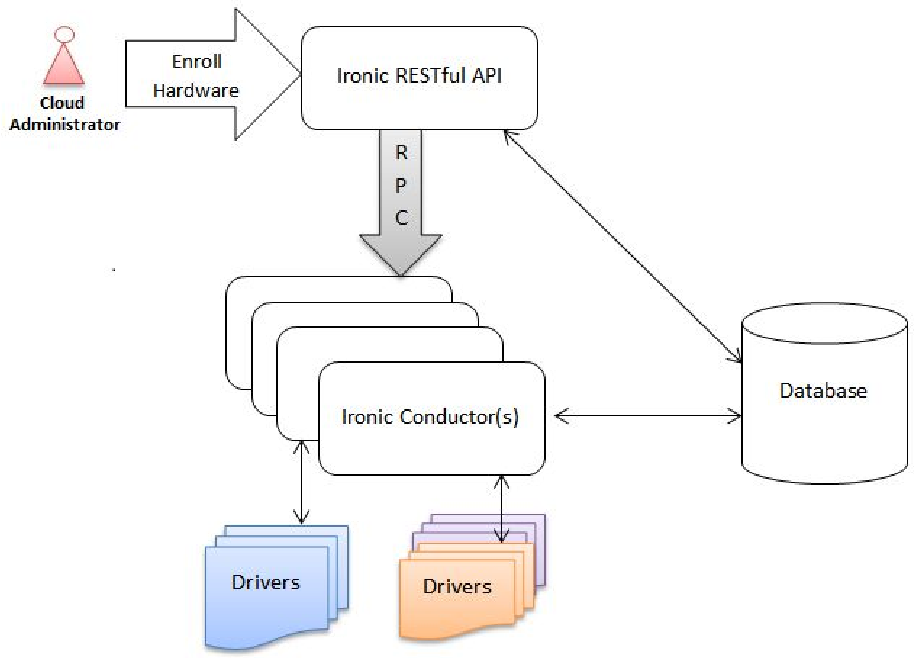
A RESTful API that processes application requests by sending them to the ironic-conductor using RabbitMQ.

- **Ironic Conductor**

Adds/edits/deletes nodes; powers on/off nodes with IPMI or other vendor-specific protocol; provisions/deploys/cleans bare metal nodes.

- **Ironic Python Agent**

A python service which is run in a temporary ramdisk to provide ironic-conductor and ironic-inspector services with remote access to gather information about the node.



To deploy, on the controller run

 # ./deploy_controller.sh ironic

- Create Bare metal network

 # openstack network create baremetal --provider-network-type flat *--provider-physical-network physnet2 --share*

- Create Bare metal subnet

 # openstack subnet create --network baremetal --subnet-range 192.168.145.0/24 *--gateway 192.168.145.1 --allocation-pool start=192.168.145.100,end=192.168.145.200* *baremetal-subnet*

- Adapt the Ironic configuration

In the file */etc/ironic/ironic.conf*, under neutron section change the value of *cleaning_network* and *provisioning_network* with the ID of the new created bare metal network and restart ironic-conductor service.

 # openstack network show bare metal *# vim /etc/ironic/ironic.conf* # systemctl restart ironic-api ironic-conductor

- Create Bare metal flavor

```
# openstack flavor create bm.small --vcpu 1--ram 2 --disk 30 \
  --property resources:CUSTOM_BM_SMALL=1 \
 --property resources:VCPU=0 \
 --property resources:MEMORY_MB=0 \
 --property resources:DISK_GB=0
```

For more information on how to create flavors visite the link

- Upload Deploy images to glance

```
# openstack image create   --container-format ari --disk-format ari --file
ironic-python-agent-fedora-train.initramfs ironic-deploy-initrd

# openstack image create   --container-format aki --disk-format aki --file
ironic-python-agent-fedora-train.kernel  ironic-deploy-kernel
```

- Get ID of both Kernel and Ram image

```
# openstack image list
```

- Register the node

```
# openstack baremetal node create --name node_01 \
 --driver ipmi \
 --driver-info ipmi_address=192.168.142.70 \
 --driver-info ipmi_port=6230 \
 --driver-info ipmi_username="admin" \
 --driver-info ipmi_password="password" \
 --driver-info deploy_kernel="<ID Image ironic-deploy-kernel>" \
 --driver-info deploy_ramdisk="<ID Image ironic-deploy-initrd>"
```

On the Bare metal node copy the mac address of the VM port

```
# virsh dumpxml  node_01 | grep  mac
```

Create a Bare metal node for node_01

```
# openstack baremetal port create <MAC ADDRESS> --node node_01
```

- Assign a Flavor to the node based on its ressources

```
# openstack baremetal node set node_01 --resource-class bm.small
```

- Put the node in manageable state

```
# openstack baremetal node manage node_01
```

To validate the state

```
# openstack baremetal node show node_01 -f value -c provision_state
```

- Make node available for deployments. This will automatically trigger clean process if you have enabled autoclean

```
# openstack baremetal node provide node_01
```

To validate the state

```
# openstack baremetal node show node_01 -f value -c provision_state
```

When then node switches to ***available*** state, create a Bare metal server

```
# openstack server create --flavor bm.small  --network baremetal  --image  cirros  node_04
```

To check the deployment state

```
# openstack server list
```

Note: The next part of the documentation is not tested and it is based on OpenStack documentation

Bonus

In Ironic example, Bare metal instances are created on the same network,to separate network flow we need to use a network of type VLAN, to separate the traffic we need to use either :

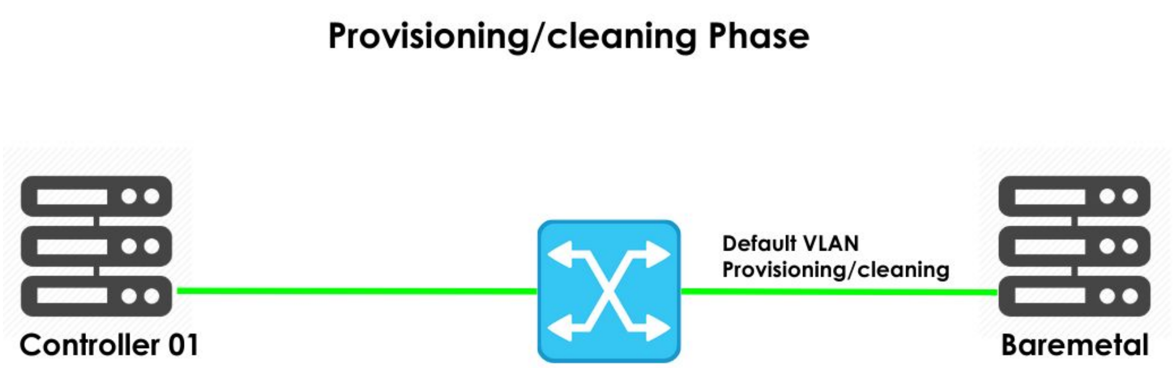
- *Networking-Ansible*
- *Networking-generic-switch ML2 plugin.*

These modules will interact with the physical switch and configure the port attached to the server with a VLAN number.

1. Flow

- **Cleaning/Provisioning**

In the cleaning and introspection phase all nodes will be on the same lan so the switch port is configured with the default VLAN number.

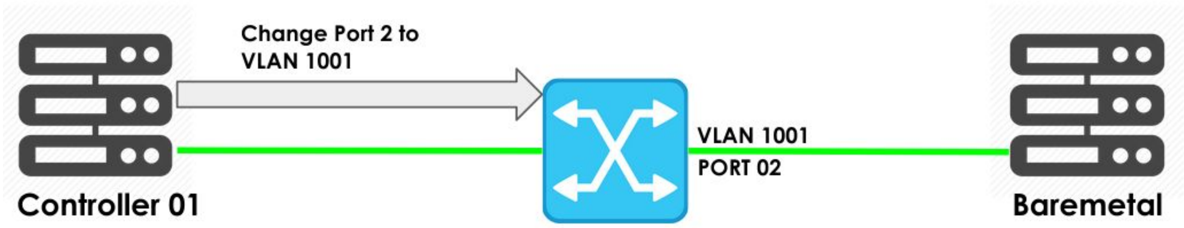


So This phase is still the same as before, we create a flat network bare metal network we provide the ID to Ironic.

- **Deployment**

When the node is being deployed and attached to network, the used network must be a network of type VLAN, this will move the node from the cleaning/provisioning network to the VLAN network by commanding the switch to change to VLAN number to the VLAN number used by the OpenStack network.

Deployment



2. **Installing and configuration**

- **Networking-Ansible**

Networking-Ansible is a python library that abstracts management and interaction with switching hardware to Ansible Networking.

First we need to install it on the controller:

```
# apt-get install python2-networking-ansible
```

In the file */etc/neutron/plugins/ml2/ml2_conf.ini,* add the following configuration

```
[ml2]
type_drivers = flat,vlan,vxlan
tenant_network_types = vxlan
mechanism_drivers = openvswitch,l2population,ansible
[ml2_type_vlan]
network_vlan_ranges = physnet2:1000:1099
```

Create a section for each host with a section name prefixed by ansible: in /etc/neutron/plugins/ml2/ml2_conf.ini.

```
[ansible:switch_1]
ansible_network_os= <switch_OS_Name>
ansible_host=<switch ip address>
ansible_user=<username>
ansible_pass=<password>
```

<switch_1> : Internal hostname used by Ironic link_local_information
<switch ip address> : Switch management IP address

<username> : Switch SSH username
<password> : Switch password

```
# systemctl restart neutron-server
```

- **Networking-generic-switch ML2 plugin**

Before trying the next tutorial make sure that the switch connected to the servers supports Neutron ML2 driver *Networking-generic-switch,* to check the supported switches please refer to the link.

First we need to install it on the controller:

```
# pip install networking-generic-switch
```

Activate the plugin in the ML2 configuration file

```
[ml2]
type_drivers = flat,vlan,vxlan
tenant_network_types = vxlan
mechanism_drivers = openvswitch,l2population,genericswitch
[ml2_type_vlan]
network_vlan_ranges = physnet2:1000:1099
```

Add Switch configuration to the file */etc/neutron/plugins/ml2/ml2_conf_genericswitch.ini*

```
[genericswitch:sw-hostname]
device_type = switch_type ex:  netmiko_cisco_ios
ngs_mac_address = <switch mac address>
username = admin
password = password
ip = <switch mgmt ip address>
```

For extra examples visit the link

Restart the switch including the new configuration file

```
neutron-server \
   --config-file /etc/neutron/neutron.conf \
```

```
    --config-file /etc/neutron/plugins/ml2/ml2_conf.ini \
    --config-file /etc/neutron/plugins/ml2/ml2_conf_genericswitch.ini
```

3. **Ironic extra configuration**

Even though the provisioning and the cleaning is still the same we need to change some ironic configuration, in */etc/ironic/ironic.conf*, change the following options

```
[DEFAULT]
default_network_interface=neutron
enabled_network_interfaces=noop,flat,neutron
```

Restart Ironic API and conductor.

```
# systemctl restart ironic-api ironic-conductor
```

4. **Node registry changes.**

A node registry node is the the same except for the node port creating, we need to add more information about the switch hostname and port

```
# openstack baremetal port create $HW_MAC_ADDRESS --node $NODE_UUID \
    --local-link-connection switch_info=$SWITCH_HOSTNAME \
    --local-link-connection port_id=$SWITCH_PORT \
    --pxe-enabled true \
    --physical-network physnet3
```

Switch hostname is the name provided in the configuration ML2 configuration file. The rest is the same as a flat network.

5. **Node deployment changes.**

In the deployment you need to create a network of type VLAN and attach the node to it.

```
# openstack network create --provider-network-type vlan --provider-physical-network
datacentre my-tenant-net
# openstack subnet create --network tenant-net --subnet-range 192.168.3.0/24
--allocation-pool start=192.168.3.10,end=192.168.3.20 tenant-subnet
```

When the network is create, deploy the node using the new created network.