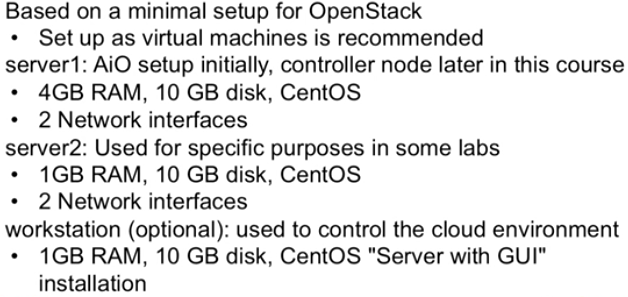
**RHCSA for Openstack - Vugt**

*Throughout this, commands complain that they are deprecated and replaced by python-openstackclient*

*Hopefully, this will not be an issue, and we are told to ignore it. Here is the link to more info on that package: https://github.com/openstack/python-openstackclient*



Installing:

- Highlight Install CentOS and hit <tab> to get cmd line option ("full configuration options")

- add "net.ifnames=0 biosdevnames=0" at the end so network hardware will show up as eth0 instead of eno353487

- output what was used to file with "grub2-mkconfig -o /boot/grub2/grub.cfg"

Nova Compute - Part of the OS talking to the hypervisor to decide where a VM will run

Swift Object Storage - physical HDs are tied to physical machines. This isn't.

Elastic alternative to local storage

Creates binary objects that are dispersed/replicated between storage nodes

Binary = easily managed, replicated= "who cares about where they exist?"

If a storage node goes down - who cares, because it is replicaed elsewhere

As long as you have a minimum of 3 physical nodes you are fine

Swift proxy is used by Swift to access filesystems. Swift itself doesnt access filesystems

Glance Images (instances) to deploy VMs from

Users don't want to install and do settings, they just want to spin something up! (think AWS)

Cirros is a 13MB image used for testing

For different Linux OS's, be sure to use ready-to-deploy cloud images when possible

You might need custom images, particularly in a corporate environment

Typically, with the image you would hava a template to specifiy hardware (flavors)

Cinder Volume You booted from an image so you need something to write on- here it is

Otherwise write is local to the machine hosting the image, and the write will be temporary

Things move around - the write cannot be guranteed, and Cinder provides **persistent storage**.

Often uses Swift object store for a backend

"No local storage! It's the last thing you want"

Neutron Networking (SDN, logical switches)

In context, SDN answers two things: tenant traffic isolation and single logical broadcast in multi-router physical infrastructure.

You don't care where VMs exist, you only care that they are on the same network

Overlaying logical network and underlaying physical network

Horizon Web user interface - dashboard, for tenants and administrators

For full control, you still need command line access

In terms of Red Hat exams, you can configure in Horizon, but you will fail since you can't determine if it works well

Keystone Identity - users and roles

Authentication tokens like kerberos

Stores in a database (MariaDB)

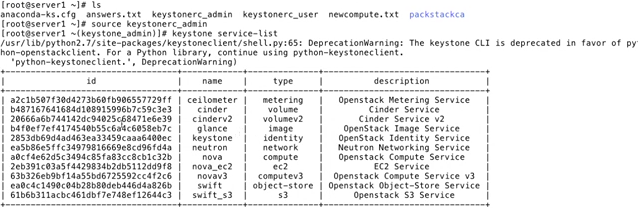
Central point to get info about cloud components:

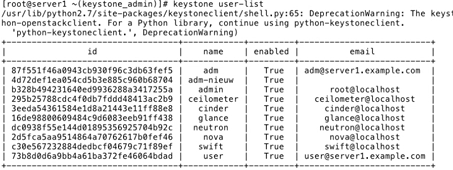
Tools: keystone service-list, keystone user-list, keystone endpoint-list

Message Broker RabbitMQ (default) or Qpid

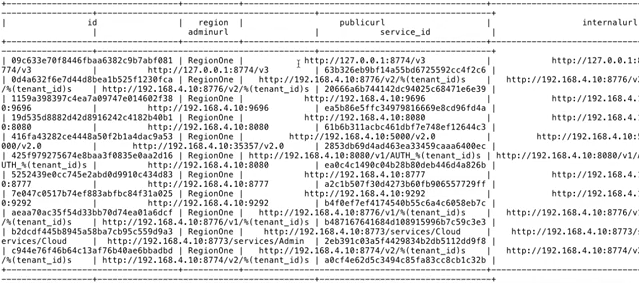
Is like an smtp server for email, ensures messages between cloud components in an orderly way

Needs to be in an HA configuration so when it goes down another can take it's place





[root@server1 ~(keystone\_admin)]# keystone endpoint-list



Overall notes on these Openstack components:

- Typically (KVM) , Glance images will be /dev/vda and Cinder volumes will be /dev/vdb, etc

- If running on a different hypervisor, this will be different (perhaps familiar like /dev/sda, sdb)

- Nova used to include networking prior to 2010

- Swift is often used as a storage backend for Glance and Cinder

- If message broker or DB stuff goes down you're screwed

**RESTful API and Python**

Standardization in components

Oslo project standardized set of Python libraries to help make things uniform.

This allows all of the Openstack commands to have the same structure

Default hypervisor is KVM. but can use Qemu, HyperV, vSphere,

Other services

Ceilometer (telemetry) - which you have to have for measurements for things like billing

Heat (orchestration) - stacks (of instances) uses templates

Ironic (bare metal deployment) - you will need this for Nova and Neutron

Trove (DB) - database as a service

Saharah (data processing) - for big data needs

Oslo (standardization)

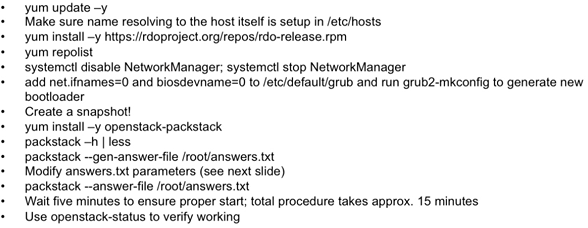
Others for DNS as a service etc...

**Deployment: Using Packstack**

- Uses answer files with Puppet to deploy a state on nodes

- Answer file generated before deployment, contains config settings

Steps:



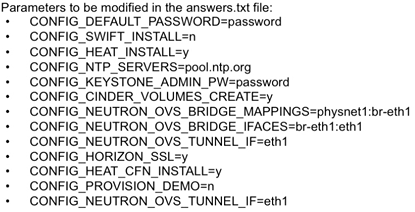
- "Network Manager is not compatible with current setup of Openstack" - doesnt explain what component. I checked and remembered teaming is not compatible with NetworkManager

- After disabling the NetworkManage you might want to check your interfaces with "ip link show"

- Check /etc/default/grub to ensure grub2-mkconfig also wrote network items here (when we ran it post-install)

- Snapshot mentioned is good if we want to roll back to pre-Openstack-installstate

These are some of the useful options to set when you deploy Openstack:



There is also a Nagios option. Not turned on for this

EXCLUDE\_SERVERS is usefull if you set up another testbed and don't want it taling to this one

When you are done, and run "packstack --answer-file /root/answers.txt" it will take about 15 or 20 minutes

In demo, said wait until after configuring ssh keys before walking off.

**Errors**

After config continued, neutron install failed- said it couldn't work with eth1. Installer quit to cmd line. Running "ip a" revealed that eth1 says it is down! Go to /etc/sysconfig/network-scripts/ then ls, then "cat ifcfg-eth1". ONBOOT=no is the problem. Set it to YES, save/exit, then run "ifup eth1". "ip a" again to check.

After fixing this, we reviewed the progress of the install, seeing no actuall configuration done yet, and merely "adding manifests for" lines (no "applying" Puppet's \*.pp files/ lines). Determining that it was safe, run "packstack --answer-file /root/answers.txt" again. If it had gone much futher, we would have needed to roll back to the previous snapshot and proceed from there.

End of install:

- A line tells us that /root/keystonerc\_admin was created - we will need that (login credentials)

- The url to Horizon dashboard

- where install logs and manifest were put

- finally, that we will need to reboot

First, run openstack-status

**Quick note on current network:**

Server 1 is the internet-connected node- the gateway for the cloud.

Internal SDN will use 172.16.0.0/24 between instances.

Server1 has Eth1 and Server2 has Eth0, but we won't be looking at those

For convenience, this allows the physical machines to have direct access to the internet for their repository stuff, but should be seen as unusual in the cloud scenario.

The OVS and BR connections are shown in "ip a" but are not configured yet. Eth0 is associated with OVS with no IP4, a EUI64 IPv6, and is considered not connected externally.

**Adding a Packstack Compute node**

Before beginning added CONFIG\_COMPUTE\_HOSTS 192.168.4.11 (dot10 was there, now there are two)

Note that this says "servers on which to install the Compute service"

You need to add dot10 to EXCLUDE\_SERVERS so it doesn't try to reinstall

**Creating br-ex, the external bridge**

/etc/sysconfig/network-scripts/

cp ifcfg-eth0 ifcfg-br-ex (duplicate a copy for br-ex)

Open **ifcfg-eth0**

Zap most of this, EXCEPT:

BOOTPROTO="none", ONBOOT="yes"

Add DEVICE=eth0, add TYPE="OVSPort", OVS\_BRIDGE=br-ex, DEVICETYPE=ovs

You should only have these 6 lines. Save.

For **ifcfg-br-ex:**

TYPE="OVSBridge", DEVICE=br-ex, BOOTPROTO="static", DEVICETYPE=ovs

(notes that there is no difference between BOOTPROTO static and none, but static is more descriptive)

Delete DEFROUTE and IPV4/6 stuff, NAME, UUID, HWADDR

Keep IPADDR0,PREFIX0, DNS1 as-is (same as eth0 was using)

GATEWAY must be called GATEWAY in this version of Openstack (using GATEWAY0 triggers a bug)

add PEERDNS=yes and USRCTL=yes

Save and reboot

Do "ip a" and "ip route" to make sure it got all the stuff it needs, plus check /etc/resolve.conf

**Machines in RHCE exam are often set up with DHCP, server will loose it's hostname and some functionality, so you may need to fix it after reboot with hostnamectl set-hostname server1.example.com**

**Deploying Openstack Behind a Proxy**

Make /etc/environment

http\_proxy=http://user@someproxy.com:80

https\_proxy=https://user@someproxy.com:443

no\_proxy= localhost, 127.0..0.1,youdomain.com, your.ip.add.ress

To ensure that your nodes aren't trying to talk to eachother over the proxy there is the no\_proxy directive.

Optionally, also include these in the /etc/yum.conf. Yum doesn't use the generic variables just defined.

proxy=

proxy\_username=

proxy\_password=

Glance Image + Flavor (4GB RAM 4 CPU 4GB storage) + Volume = instance

instance => virtual router => internal network

=> Floating IP => external network

floating IP + security group + ssh key

So if you are working with a internet browser and try to open a session to the same server such as https://server1/dashboard, you may likely get a message that server certificate failed - go to browser settings and clear certificates and servers marked "Openstack". Quit the browser and open to try again.

Forgot user/pass? Go to ~/keystonerc\_admin

**Creating Projects and Users in Horizon**

Opening and logging into Horizon as an admin, we are greeted with the Overview page with a usage summary table with date range, and list of projects including their: VCPUs, disk, RAM, VCPU hours, Disk GB hours, and Memory MB hours.

A sidebar provides us with:

Project tab>Network, Orch, and Compute sections

Admin tab>

Identity tab>Projects and Users sections

A non-admin user still has the sidebar but has a simpler summary view (showing just the main project) and limited options. The overview is under the Computer header in the sidebar where it sits next to instances, volumes, images, and "access & security".

Default Projects (tenants) are "admin" and "services"

The main listing shows name, description, projectID, enabled (or not) and an Actions column providing a "manage members" pull-down menu. Checkboxes on the side let you select multiple projects and create and delete buttons are on the top. Predictably, you add a name and description; users tab allows user add/remove. A quotas tab has many options to set project confines:

| **Quota Name (default value)** | **Defines the number of** | **Service** |
| --- | --- | --- |
| Metadata Items (128) | Metadata items allowed for each instance. | Compute |
| VCPUs (20) | Instance cores allowed for each project. | Compute |
| Instances (10) | Instances allowed for each project. | Compute |
| Injected Files (5) | Injected files allowed for each project. | Compute |
| Injected File Content Bytes (10240) | Content bytes allowed for each injected file. | Compute |
| Volumes (10) | Volumes allowed for each project. | Block Storage |
| Volume Snapshots (10) | Volume snapshots allowed for each project. | Block Storage |
| Total Size of Vols (GB) (1000) | Volume gigabytes allowed for each project. | Block Storage |
| RAM (MB) (51200) | RAM megabytes allowed for each instance. | Compute |
| Security Groups (10) | Security groups allowed for each project. | Compute |
| Security Group Rules (100) | Rules allowed for each security group. | Compute |
| Floating IPs (50) |  |  |
| Networks (10) |  |  |
| Ports (50) |  |  |
| Routers (10) |  |  |
| Subnets (10) |  |  |

Start with Identity: before we can do anything we need to create users.

Service users exist for Glance, Neutron, etc., plus Admin

New users have username, password, email (optional), primary project and role

It makes sense in larger projects to give every tenant it's own administrator, and Red Hat wants that.

Roles (shown) are admin, \_member\_, heat\_stack\_owner, heat\_stack\_user

**Images**

For this, downloaded Cirros. Google for cirros image download brings us right to Openstack's Get Image page.

Don't download the image- just copy the link

Example logs in as a regular user

The images window lists image name, type, status, public, protected, format, size, and "actions"

There are choices to view by project, shared, or public, and a "create" button.

The create window has few options: name, description, image source (file/location), location, format, architecture.

(we paste the url here)

Default image format is QCOW2 for QEMU. There is also Amazon Kernel Image (AKI), Amazon Machine Image (AMI), Amazon Ramdisk Image (ARI), Open Virtual Appliance (OVA), Docker, VDI, VHD, VMDI, ISO, Raw

There are options for Minimum disk (GB), Maximum disk (MB), and checkboxes for public and protected. None of these or architecture is required to specify.

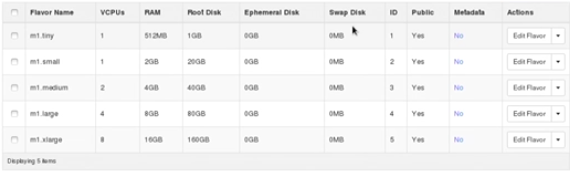
Copy data (to image service) should be checked or else it can end up downloading each use.

These options are the same for admin and regular cloud users, but cloud users can't make HW profiles (flavors) so we have to log out and log back in as admin.

Consider the "cloud user" account for the "tenant administrator" and admin account for the "cloud administrator" with wider authorization and access.

**Flavors**

Here, we get default options listed below, columns for the listing: name, VCPUs, RAM, Root Disk, Ephemeral disk, Swap disk, ID, Public (yes/no), metadata (yes/no), and edit. Here are those and the default flavor offerings:



These are pretty big. As an example we made one called "little" that is much like "tiny" (above)

**Network**

We need to define the network before spinning up instances. This can be done as a cloud user.

Create network as an internal network: give a general name and note "admin state" of up (ignore it). For subnet, we are asked to name the subnet, give IPv4 or v6, network address, gateway. You are just informing of subnets here, so they don't need to relate to virtual network card addresses. Default for GW will be first available IP in subnet, but there can be issues if you leave this blank (this example puts in 10.0.0.254 - also see "router" below). The next subnet details tab asks for dhcp pools (or disable dhcp- pools only need "to address, from address"), name servers (needed with or without DHCP on), host routes (none required)

Also create a network for an external network. This is set for a real interface, GW for internet

Disable DHCP (checkbox) but **add pool anyway for floating IP addresses** to add to nodes later on.

When we finish, we go to network topology in the sidebar (cloud user gets networks, routers, and topology links).

Since the external network is configured to directly-connected physical cards, we have to log out and log in as cloud administrator to access the rest.

When you go to "edit network" in this mode, it allows checking a (previously absent) box that says "external network". All that was needed was checking that box. No need for messing with "Ports" section or anything, and we can re-login as regular cloud admin, and click on "Routers" to add one. We can now give one an external network since we made it available. When we look at the settings (after adding) we can see SNAT applied for us. An interfaces tab lets us add them to either the internal or external network.

Important: the example set the GW for the network to 10.0.0.254 - this should be the IP address for the external network interface!

The network topology should show a router between the 10.0.0.0/24 internal and 192.168.4/24 external network.

**Instances**

The instance list shows name, image, ip addy, size, key pair, status, zone, task, power state, when created and "actions". We click "New".

Choices: Availability Zone (default is nova), flavor, instance count (1), boot source (image), image name (select)

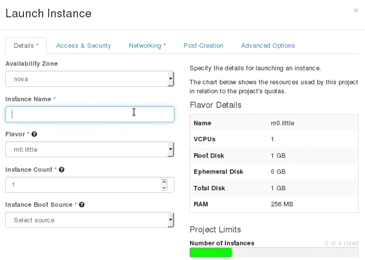
For key pair, you will need to run ssh commands into terminal (it gives you instructions)

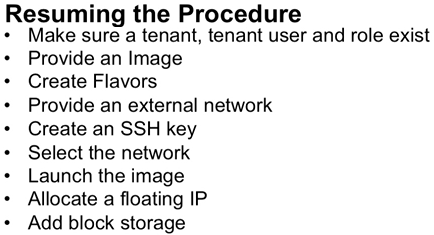
ssh-keygen -t rsa -f cloud.key <----will spit out cloud.key and cloud.key.pub for you. cat and copy contents of cloud.key to past in Horizon. You probably want to clean up by putting keys in ~/.ssh directory: "cp cloud.\* .ssh

Set security group to default (checkbox) for now. Under networking, select the internal network we created.

There are also "post-creation" and "advanced" options, we skipped for now. Hit "Launch"

We get error (to troubleshoot later) "no valid host was found... not enough hosts available"





**Troubleshooting**

openstack-status is the way to authoritatively determine the availability of services



systemctl status httpd -l (l for long format)

/var/log/horizon

Also /var/log directories for cinder, keystone, glance, heat, mariadb, neutron, nova, openvswitch, rabbitmq

If you are checing things right after restarting, give everything a minute or two on slower machines to initialize

Verifying things, you might also systemctl status rabbitmq-server, mariadb-server, and keystone

Quick notes:

- if RabbitMQ fails management will be impossible but instances won't fail. In service config, contact info must be included. Can be TLS-secured, but if manual install can be skipped. Config is in nova.conf - rabbit\_\* attributes. RabbitMQ here is fairly simple but you could do a 2-day course on it to cover everything it can do.

- Keystone relies on DB, generally SQL-compatible, and default is MaraDB. Information is supposedly synchronized with the DB, but sometimes it isn't the case, particularly with Neutron stuff (inconsistencies with neutron lists). You may have to dig into it for troubleshooting occasionally.

Storage in Openstack is ephemeral (nothing stored in a permanent way), USUALLY.

- However- while deploying instances, using the settings in a flavor, disk files are written to /var/lib/nova/instances on the Compute node. Once you move around this wont move around with the instance, stays on the same compute node. VM XML file can give details- this is persistent storage, so it may be good to make it available on an NFS (or other shared storage) as a backend. Since generally storage is ephemeral, your should use cinder volumes for truly persistent cloud storage

/var/lib/nova/ contains /buckets, /instances, /keys, /networks, /tmp Inside /instances are directories named for UUIDs

These contain console.log, disk, disk.info disk.local disk.swap libvirt.xml - disk being a disk file. all items belong to users and groups of qemu and nova. The xml file is full of the VMs info in KVM libvirt format.

In this way, storage is not necessarily always ephemeral.

**Keystone**

The backend DB everything is talking to (for authentication and authorization)

**These keystone CLI commands are replaced with python-openstackclient.**

keystone service-list -services available to users

keystone role-list -roles available to users

keystone user-list

keystone endpoint-list shows the authentication urls to use when digging deeper into Openstack. We need this when working on configuration files manually later on. Whenever configuring services endpoints must be created in Keystone. The endpoint is the url a service uses when connecting to another service

Started on these, but got message "Expecting an auth URL via either --os-auth-url or env[OS\_AUTH\_URL]

We have to fix this:

~/keystonerc\_admin <---Holds the env[ ] variable mentioned with username and pass we set up

**Run this file like a script and get a prompt!**

source keystonerc\_admin

> keystone service-list and it works.

Page two in the beginning of these notes shows the output of most of these commands

**keystone --help gives tons of info**



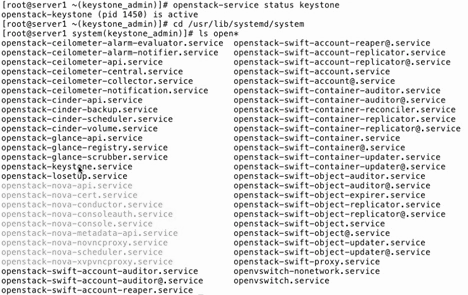
To simplify remembering:

service-, role-, user-, tenant-, and ec2-credentials-, and endpoint- all prefix create, delete, get, list commands

For user- commands, there is also user-password-update, user-update, and user-role has -add -list -remove variants

openstack-service stautus keystone

cd /usr/lib/systemd/system/; ls open\*



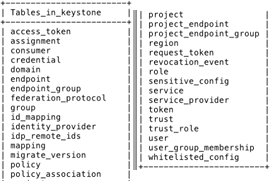
systemctl status openstack-keystone (of course) gives a lot more than the previous non-systemd command

less /etc/keystone/keystone.conf

You can also do grep -v '^#' /etc/keystone/keystone.conf

- to see all of the lines that aren't commented out

**mysql -e 'use keystone; show tables;'**

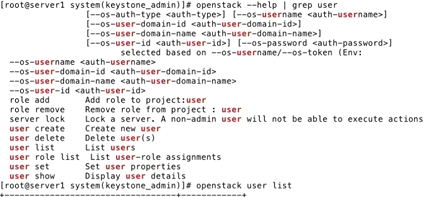


**CLI User Administration**

The openstack CLI command uses many of the similar subcommands as the keystone command, without the "-". It is a newer command so you won't find it in older versions of Openstack.

One other difference is it uses "project" instead of "tenant"

Here are it's user subcommands:

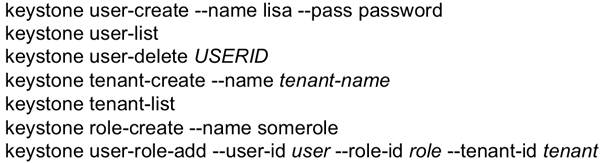


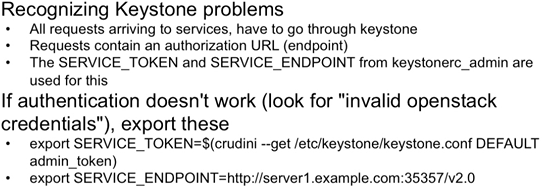
That having been said, back to the keystone commands:

keystone user-create --name list --password password

- creates the user, spits out theusername and id number

keystone user-role-add --user-id lisa role-id admin --tenant-id project





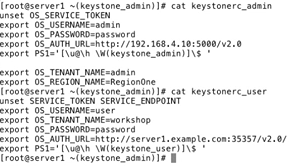
When you are looking at the logs in /var/log/keystone, you might want to run a grep exclude on INFO-level messages (there are a lot) by using "grep -v INFO /var/log/keystone/keystone.log"

Make use of keystone endpoint-list

Check keystonerc\_admin or other relevant keystonerc file (keystonerc\_user

Did you source it the way you should have for credentialing in?

Differences between user and admin versions of keystonerc files



**Swift Object storage:**

We want to work with objects- not files.

Disk-based is not elastic, and a DFS isn't elastic enough.

Swift takes care of replication, availability in a model where data is written to multiple binary objects, dispersed in a replicated way over many nodes. Metadata is similarly dispersed and replicated.

The application is required to make (RESTful) API calls to read and write, through a Swift proxy.

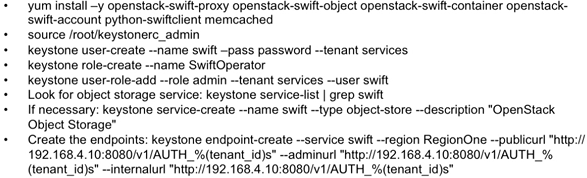
The entire structure is made of many nodes, many binary objects.

[Security note to self is that Swift data must be deabstracted through a Swift proxy of some sort to be exploited or broken into. Ideally, the existing framework would be used to accomplish this rather than obtaining files and reconstructing binaries, metadata from the entire mess of a recon grab.]

So Swift object storage also provides a flexible backend framework to support cinder volumes and glance images.

**Manually Installing Swift - Knowledge for Troubleshooting not in RHCE**

Here is the procedure (outline):

****

The install: (we omitted this from the Packstack batch install)

openstack-swift -proxy, -object, -container, -account, THEN python-swiftclient and memcached

Creating the user, example really does use name swift, and tenant services- these aren't there for special reasons

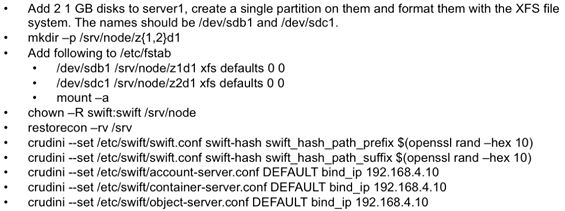
Operator role allows assigning operator permissions to storage administrators in swift

Region- just like in AWS. Can make things region specific later.

Endpoints: notice the 3 urls are the same- public, admin, or tenant will populate the "tenantID appropriately when actually used

**Deploying Swift Storage**

Since this is just a test environment to see the procedure, we are only going to work with a single node here



crudini is utility for modifying config files (is scriptable

Account, container, and server lines are for those "rings"

cat /proc/partitions

fdisk /dev/sdb and sdc - they are already both 1GB each, so no partitioning needed.

mkfs.xfs them both as well

mkdir -p /srv/node/z{1,2}d1

ls reveals z1d1, z2d1 - z for zone

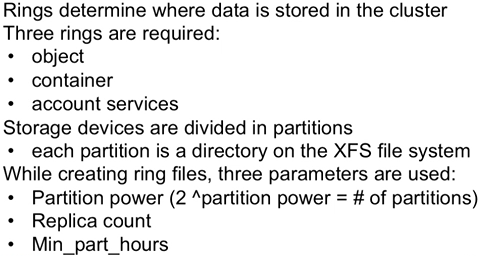
vim /etc/fstab

/dev/sdb1 /srv/node/z1d1 xfs defaults 0 0

/dev/sdc1 /srv/node/z2d1 xfs defaults 0 0

mount -a

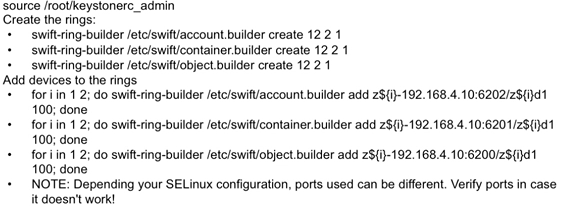
**Configuring Swift Rings**



Object ring are the binary objects themselves; container ring is about how these are stored, accounts ring is auth and access

Min\_part\_hours is the minimum time before partition synch happens- indirectly ensures that only one partition's replica is moved at a time. Moving multiple copies at the same time risks making data inaccessible. 24 is a recommended default, in a production environment. Checking logs can reveal time needed (to optimize setting).

Partition power, 4096 in this case (2^12). This is the amount of partitions needed to provision for all of Swift's operations: "maximum number of drives you expect your cluster to have at it's largest size. In general, it is considered good to have about 100 partitions per drive in large clusters." is how OReilly's Openstack Swift: Administering..." puts it, and the best definition I found. Calculate with [log2 (100 x max disks)] - e.g. is large deployment of 1800 drives that will grow to a total of 18,000 drives - part power = [log2 (100 x 18000)] = 20.77 = 21 allows for 100 partitions per disk.



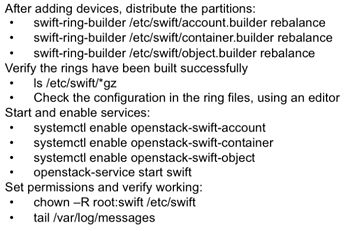
When running for "i in..." you may get a message saying region not specified, using region 1. This is normal.

In the for statement, "i" refers to our partition names we used (1 and 2).

The meaningful output for each of these is similar to this:

Device d0r1z1-192.168.4.10:6202R192.168.4.10:6202/z1d1\_"" with 100.0 weight got id0

Device d1r1z2-192.168.4.10:6202R192.168.4.10:6202/z2d1\_"" with 100.0 weight got id1



Rebalance distributes the binary objects over the different partitions that are available in the configuration

The output of the rebalance commands looks like this:

Reassigned 4096 (100.00%) partitions. Balance is now 0.00. Dispersion is now 0.00.

As noted, the configuration files in /etc/swift are indeed gz'd



Later, when verifying, you need to use netstat -ntulpe

**Deploying a Swift Proxy**

Proxy determines to which nodes gets and puts are directed (remember JSON/RESTful?)

To avoid SPOF (single point of failure) use multiple in production env

Employ a load balancer of round-robin DNS to distribute data among proxies

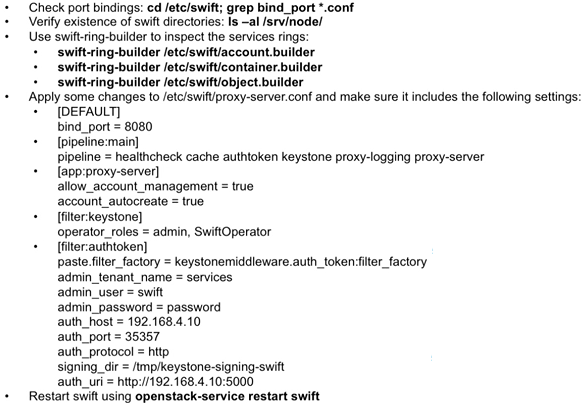


Auth host is where keystone lives.

**Managing Objects in the Swift Obj Store**

Files in the store are binary objects- not accessible as files. The Swift API and swift command allow access, and the CLI command is much like an FTP command

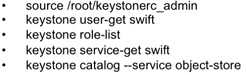
First we have to make these checks, that everything is in order.



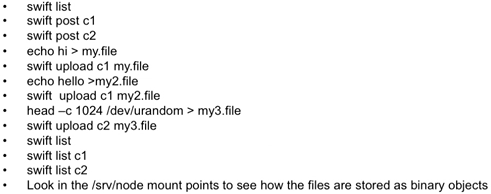
ls -al /srv/node/ shows z1d1 and z2d1, which normally would be on different servers



*Verifying correct operation of Swift:*



*Verifying through use example:*

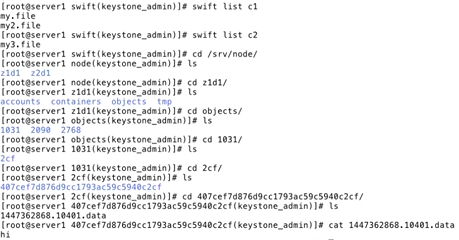


swift list should just come back like it did nothing. That's because there is nothing to display yet

swift post c1 and c2 are creating two directories (2 partitions) in the Swift object store

Here is a demo of the output when traversing directories

It shows how the binary data is distributed around

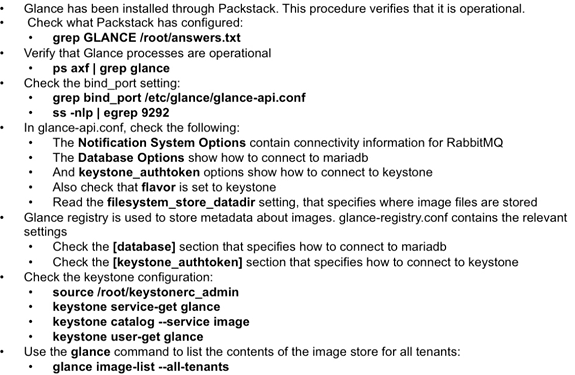


**Beyond Swift: Ceph**

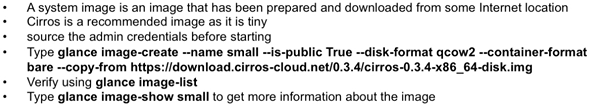
Improves on Swift - more scalable (more nodes too) and more than just API access

Ceph file system/block device lets you work with like a physical HD (same addressing), with a logical object store backend

**Verifying Glance Image Services**



**Uploading System Images**



glance-image --help

**Juno or Kilo this works, Liberty it stops working. We need Openstack command**

**It wont tell you about deprecation! Just says "glance: error: unrecognized arguments:"**

**It appears the openstack command, as said in the manpage, is to "provide a common CLI to Openstack APIs... generally equiv to CLIs provided by Openstack project libraries"**

man openstack - look for "create new image"

openstack image create --disk format=qcow2 --container-format=bare --public --copy-from https..... small

- it complains that copy-from (which was obtained from the example it gave us!) is "an Image v1 option no longer supported in Image v2"

Go to Openstack website for Liberty version- they tell us to download the image and then load- USING THE OLD COMMAND THEY TOLD US NOT TO USE!

wget http.....image.img --no-check-certificate

glance image-create --file cirros-0.3.4....disk.img

FINALLY FULL CIRCLE BACK TO GLANCE IMAGE-CREATE, EDITED:

**glance image-create --name small --visibility public --disk-format qcow2 --container-format bare --file cirros-0.3....img**

And it works.

**Building Custom Images for Glance (not in RH exam)**

virt-sysprep to remove persistent mac addresses user accounts and ssh keys from image

qemu-img info /imagefile ----if you need more info on the image type

glance image-create --file

--location to copy from remote --copy-from is an alternative

--is-public

NOTE look at last example for what WORKED - these Glance image-create options have problems we encountered, but you use the same line

glance image-list (you might also try with option --all-tenants

**Swift as a Glance backend - not needed on exam**

Local file location doesn't scaale

swift\_store\_create\_contaner\_on\_put True --- may need to be manually inserted into config





**Troubleshooting Glance**

Ensure glance-api and glance-registry processes are running

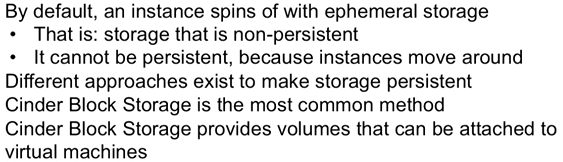
Check disk space, auth credentials, that correct ports are available

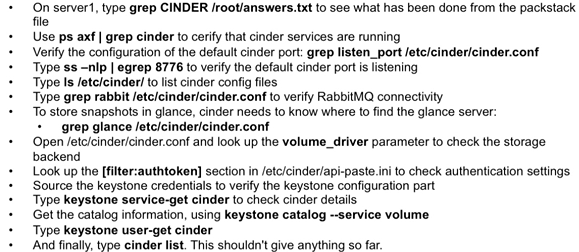
/var/log/glance/ for registry and api ..

You will probably have to use grep verboten on INFO and/or to make useful

Openstack Storage Needs

Cinder review:







Most of these have to do with being able to integrate with Netapp SAN, and the important ones for us generally are at the top. LVM=y means you should see volumes with vgs (that it is operational). This jumps to volume\_driver line.

in our list (it's LVM)

**Managing Cinder Volumes**

Creation of volumes is typically a cloud user task and not a cloud administrator task so you can log in as such for this

cinder create --display-name my-volume1

cinder list

vgs to also view stuff

lvs to see created Cinder volume

grep backup\_driver /etc/cinder/cinder.conf

systemctl status openstack-cinder-backup (start and enable)

- To create backups, you have to add the user as SwiftOperator, which has to be done by admin credentials

source/root/keystonerc\_admin

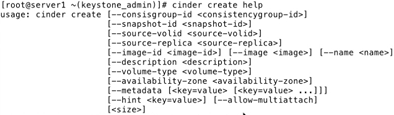
keystone user-role --role SwiftOperator --user myuser --tenant myopenstacktenant

-- log back in as user:

source /root/keystonrc\_myuser

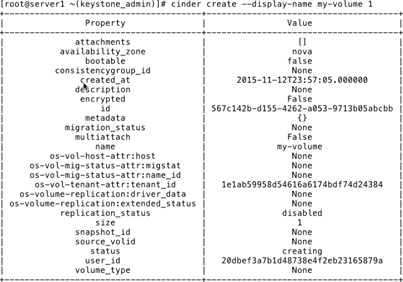
cinder backup-create vol1 --display-name vol1-backup

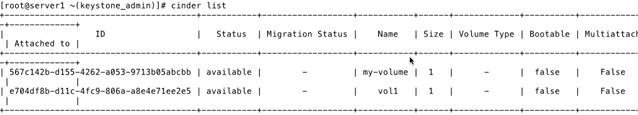
cinder backup-list



Note the the help test is incomplete, and doesn't list --display-name

On the exam you only need to know how to do this in Horizon



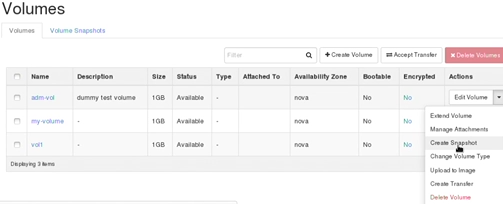




**Managing Block Storage from Horizon**

In Horizon, in "Volumes" we can see the volumes just created and say in "cinder list"

Hit "create"; call it adm-vol; under Volume Source, we go with default "No source, empty volume". We could attach it to an LVM here. Size and Zone there also. Afterward, here are "Edit Volume" options:



- Asssigning a vol to an instance, use "manage attachments"

- In troubleshooting, tail /var/log/cinder/volume.log and api.log, vgs, and cinder list commands

- Packstack provisions VGs for you but you might have to do vgcreate, etc to do it manually

**SDN and Neutron**

In the definition of SDN is something that is easily forgotten - that the data plane and HW is a separate entity:

It says that the control plane resides in the software, (true we knew that) but that the data plane is "implemented in commodity network equipment". That last part is arguable since traffic channels are often virtualized

- Think of tenants. Network traffic needs to stay separated.

- BC domains defined in SW, need to be maintained across routers (VLANs assumed)

- For some reason, it is stated that traffic load and patterns is unpredictable.

- Emphasis is placed on need to be scalable and accommodate increased BW needs on-demand.

(scale out easily)

Illustrated (as if a great mystery solved by SDN) are 3 compute nodes separated by routers, and that each of their individuated VMs are in the same BC domain. "Network Decoupling" term used for SDN doing this.

- Control plane defined as a network service, Typically in Neutron configured in ml2\_conf.ini

- ml2 is a pluggable layer, allows plugins to define the overlay networking

- The differentiation is made (finally)

VLAN is defined here as a "legacy method tough to configure because of physical dependencies"

(My critique is it isn't tough to insert 802.1q fields in frames and you don't need HW!)

VXLAN based on IP multicast to enable auto-discovery of the overlay MAC addresses

GRE P2P or multipoint handled from a central router to connect endpoints. (remember no encryption)

VXLAN has become standard, default.

VLAN is considered antiquated and confined to the stupid definition presented above

Cloud admin provides routers, subnets, external networks

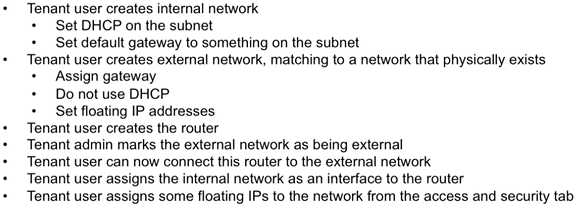
Tenant admin can have multiple private networks to plug into those through Openstack services

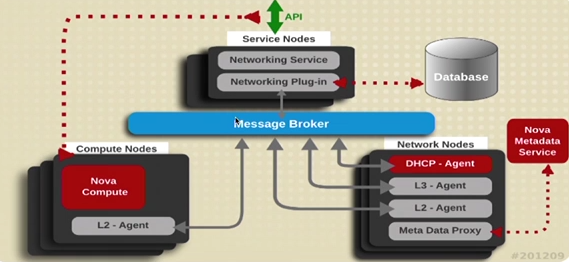
- decides IPs to assign networks, create and connect to services

Public networks on other side of router for external connectivity include floating IPs

Plugins not only define the overlay, but are provided to connect to specific physical networks (underlay)

Compares floating IPs to source-NAT - expose an instance with a publically available to the internet on inside of the private network





Compute nodes, running the instances,

- L2 OVS-agent / server component on node defines the broadcast domain in the overlay network (and the way networking is going to work in the environment)

- L2 OVS-agent on compute node talks to L2 OVS-agent on Network node

- L2 OVS-agent on the network node terminates SDN BC domain

Network node has everything it needs to define the control plane: DHCP agent, L3 stuff (routing), etc.

Often service and network node will be running on same machine

Service node:

Networking plugin defines interactions with PHYS network, uses DB to keep it straight

The networking service talks to the API, message broker glues all these different nodes together

The compute node runs neutron-server offering what the service node has (service and plugins) through it's OVS agent- the server component

**Analyzing Packet Flow on a Neutron Network**

- a VM boots, queries the neutron server, which is the software control plane

- the control plane uses OpenFlow to give info required to connect to the physical switch

- It is OpenFlow that shapes the SDN framework so the VM will know paths to the switch

- says "the switch itself is reduced to a dumb device" because the OpenFlow intelligence is instead in SW

**InterVM Comms**

VMs of the same tenant group might be on different hosts but need to be in the same subnet.

Routing all traffic through the network node to do this sucks and creates a SPOF

Instead, OpenFlow allows the VMs to talk to eachother directly

So OpenFlow generates a switching structure of a virtualized network

- Is not managed directly, but by Open vSwitch or other agents

- Replaces the control plane on switches, and works with managed and unmanaged switches

- Defines how forwarding planes should work and how they should be controlled from within the cloud

- Layer 2 based virtual interfaces for virtual machines,

- Bridges sit between the virtual and physical interfaces

- Open vSwitch in the BG calls OpenFlow to see how packets are to be dealt with

- OpenFlow rules govern forwarding between virtual machines, even on different physical nodes

**Separating traffic between tenants**

Different VMs need a different approach

VMs may belong to different customers. separation is a big deal

It should be separated enough that customers can use the same IP addresses without conflicting- which they can!

Linux Network Namespaces:

- are a container-based virtualization technology present in the Linux kernel stack (like chroot for networks)

- define separate areas and allow to distinguish the traffic between them.

- have their own network devices (i.e. virtual interfaces that are bridged into the physical device on the host) and separate IP addresses

Ultimately,

- Things in namespace A need routing to communicate with stuff in namespace B - even on the same host!

- Traffic generated in a namespace is visible only in that namespace

Command Line Tools

ovsdb-server - maintains the internal configuration database for Open vSwitch

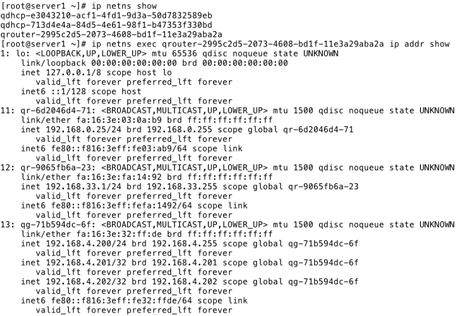
ovs-vswitchd - switching daemon doing the network configuration

ovs-vsctl - manage Open vSwitch

ovs-ofctl - manage and monitor OpenFlow - to do deep inspection

ip netns show - show namespaces available

ip netns exec <namespace-identifier> COMMAND --- runs that command inside that namespace



Notice the multiple addresses in 13- those are floating IPs.

Interfaces

br-int (integration bridge) to integrate everything

br-tun (tunnel interface) define the network between the nodes

br-ex 0 external bridge - for external devices

eth0/1/2 your physical interfaces - on the networking node you need one for the external network and one for the different physical machines in the cloud environment (I am just reporting what it said here- this doesn't sound quite right, but see below).

Typical network configuration:

Hypervisor node:

br-int - what the VMs are connected to

br-tun - used for tunneling

eth1 - internal physical interface used by br-tun

GRE tunnel - present on all hypervisor and network nodes for tunneling

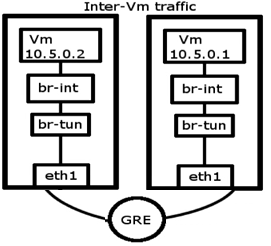
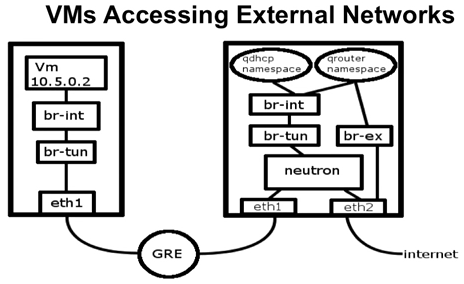
Network node:

br-int / br-tun - similar to the hypervisor

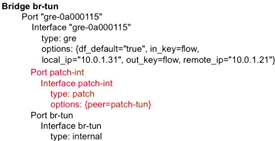
br -ex - connected to eth0 for external comms

qrouter - one for every external network for cconnections to outside

qdhcp - one for every tenant, exists within GRE tunnel





**On br-tun:**

Patch-int port shows the connection to patch-tun (which connects br-int to br-tun)

The gre interface shows which host the tunnel is connected to on the opposite end - the network node.

**On br-int:**

"tag: 1" Is the OpenFlow ID assigned to packet entering from qvo port

The patch-tun port shows it's connected to peer port patch-int (on br-tun)

The "ovs-ctl show" output can get really long/ messy on the neutron node. This is obviously split for easy viewing.

**Understanding Backends**

OpenStack + Open vSwitch the usual solution

Other backends might be provided by HW vendors or other open source solutions

Neutron just gives an API, so you can slap another onto it

Common Plugins:

Cisco UCS/ Nexus

Linux Bridge (outdated)

VMware NSX vCloud example



NSX offers:

VXLAN (Virtual eXtensible LAN)

STT- Stateless Transport Tunneling

NSX gateway for comms with non-NSX networks

NSX and RHEL sample layout, where th NSX plugin stands in for OVS plugin:



https://wiki.openstack.org/wiki/Cisco-neutron

https://wiki.openstack.org/wiki/Neutron\_Plugins\_and\_Drivers

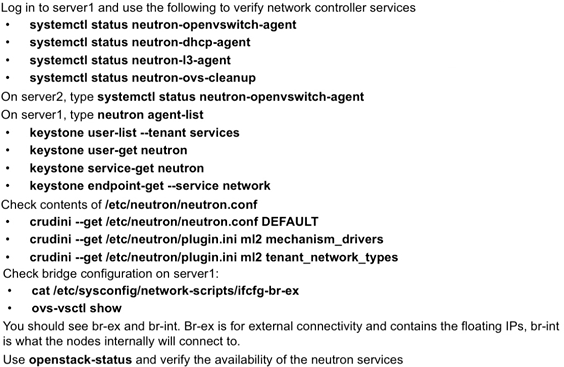
The second page says it is deprecated and does not refer to a new one; searching wiki also is "go fish"

**Verifying Neutron Networking Services**

Below, consider server1 commands for a node providing network services (Neutron node).

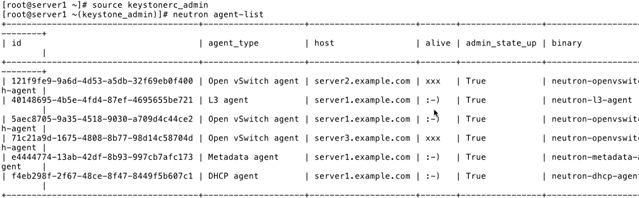
Server2 is just a compute node using network services (compute node)

On Server1's /etc/neutron/neutron.conf the ml2 entries defines which plugin to use



The "neutron" service command has many options. Before you use it, you need to "source keystonerc\_admin" in.

Below we see that Server 2 and 3 are showing down (they aren't on)



We already know what the keystone commands will show us.

One interesting item is "keystone endpoint-get --service network". Using regular "keystone endpoint-list" we don't have this service specifically identified (network.publicURL on :9696) service-get wasn't demoed

Note the use of crudini to get rather than set values. Unfortunately, a key-value pair is not returned. The query for DEFAULT just spit out an unlabeled list of values. Specific queries such as "ml2 mechanism drivers" simply dumps out the specified value.

**Configuring Tenant Networking in Horizon**

This lesson was after the command line lesson in the following notes.

We are going to do this as a tenant administrator

Had to log in as cloud admin first, go to Identity>Projects>Edit to change the max network and floating IP quotas

Logged in as tenant admin again, go to Network>Networks>Create Network

Name arbitrary, network also arbitrary unused 192.168.200.0/24, put in a gateway IP you know\*\*\*

In subnet details, allocation pools for the floating IPs range\_start,range\_end

Set a name server 8.8.8.8

For an external network, do the same an arbitrary unused IP block, make sure the gateway is correct for the external routing. Turn off DHCP on external.

Log in as cloud administrator to mark this as an external network (!) Tenant users must ask the cloud admin!

Click network, edit network, click external, save!

Log in as tenant admin again, click on routers, new, choose a external network we just approved.

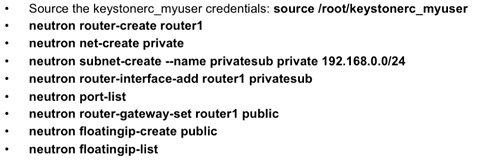
Assign interfaces with interfaces tab, add your internal network, and you are done

Network topology should show all is good.

**Configuring Tenant Networking at the command line**

A public network has already been provisioned by the cloud administrator- configured on server1, the neutron node. The tenant administrator/ cloud user sets up the rest.

**The commands:**



**The procedure, in brief:**

1. Create virtual router

2. Create private network

3. Create a subnet in that private network

4. Attach the subnet to the router

5. List the port configuration and get the ID of the public network ("public")

6. Establish a gateway to that public network

7. Make your floating IPs you'll need later and list them

**Notes on the following demo:**

- In the step after listing ports, it looks like we don't have a public network to add floating IPs to

- We need to log in as admin to do this:

neutron net-create public --tenant-id services --router:external

- It also needs a subnet so:

neutron subnet-create --tenant-id services --allocation-pool start=192.168.104.25,end=192.168.104.99 --192.168.104.2 --disable-dhcp --name subdonotuse public 192.168.104..0/24

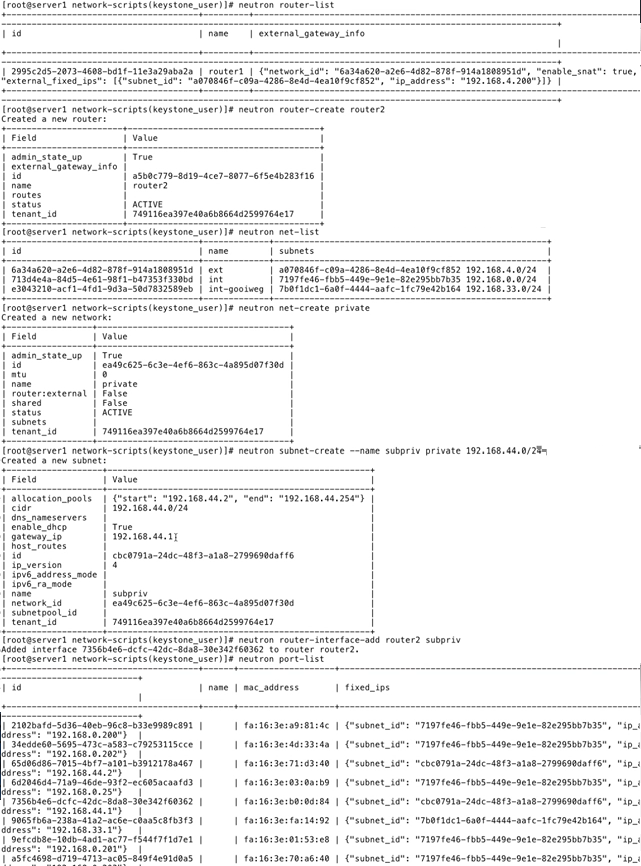
- Set the gateway

neutron router-gateway-set router2 public

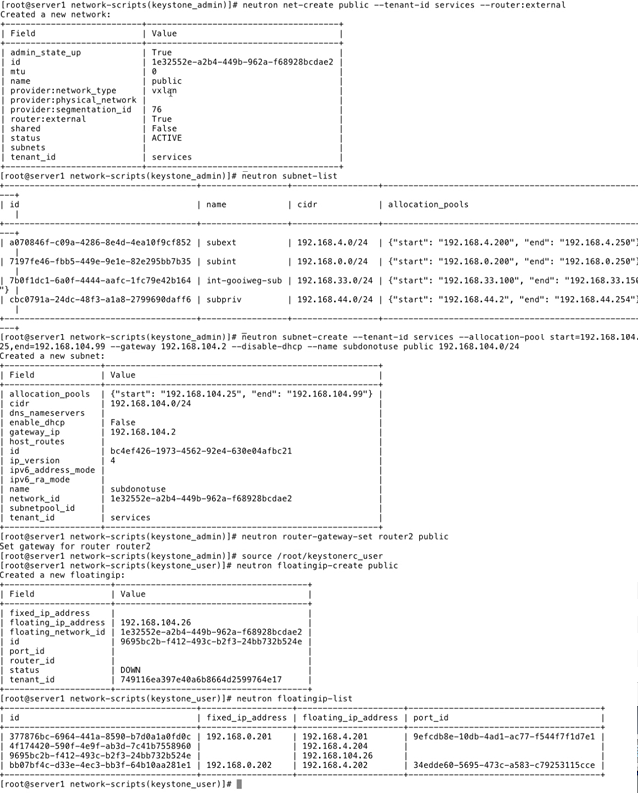
- Switch to reg user and continue:

neutron floatingip-create public

neutron floatingip-list.







**Troubleshooting Neutron Networking**

- Pinging the instance

- openstack-service status neutron

- tail /var/log/neutron/\*.log

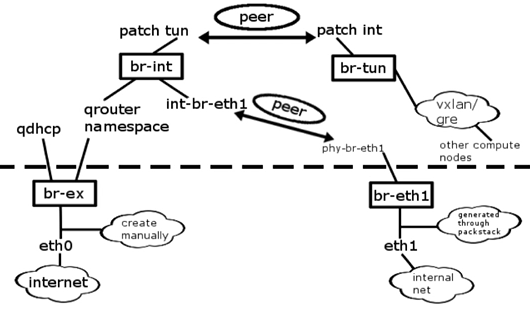
- As a tenant admin, neutron net-list, subnet-list, and router-list

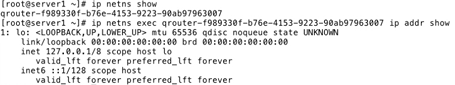
- Make sure the router is connected and has a working external gateway

Above the line is the overlay virtual network

Below is the physical network

Remember- the br-int is the integration bridge- integrating a connection into the namespace.





ip netns show "show me the namespaces!"

ipnetns exec qrouter -.... ip addr show "whats up?"

this dispaly is showing nothing is up. Router has an internal address but nothing is going on.

11.2 config tenant networking, next 2 also long - networking

The last 10 get really long esp configuring packstack... these are part of lesson14- an exam lab, and reviews just about everything configuration-wise

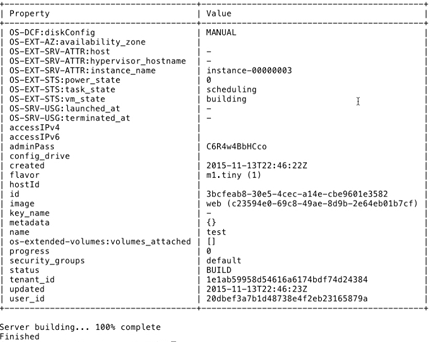
**Deploying an instance from the command line**

source keystonerc\_admin

nova list (it's blank)

nova list --all-tenants --shows instances on all tentants (not jus the admin)

nova boot --flavor m1.tiny --image web --poll test



And that's it! Off it goes!



**Troubleshooting Nova and Instances**

http://virtual2privatecloud.com/troubleshooting-openstack-nova/ |---site down- saved WBM copy

/var/log/nova/nova-compute.log

/var/log/nova/instances

/var/log/libvirt/

Check networking

mysql -u root -p

use nova;

show tables;

show columns from instances;

select \* from instances;

grep MARIA /root/answers.txt

- login info for DB incl passwd

Node comes up but can't reach it?

Might have a floating IP or routing issue.

ip netns list

ip netns exec <id> ip a

ip netns exec <id> ip ping <address> (can you ping it from inside the namespace?

**Adding and Removing Compute Nodes**

nova list --all-tenants --- check to see if it's running

nova delete <vm-id> -- to stop it

nova service-disable --reason "needstogo" server2.example.com nova-compute

- disable the compute service on Server2

nova service-list

-- verify + get the ID of the Server2 instance

nova service-delete 6; nova service-list - still shows but should be disabled

- Delete the server2 compute service ID from the list using the ID you got

- It will come back on reboot, although it is said nova service-disable

This part didn't really make things clear - like "nova delete <vm-id>" vs "nova service-delete"

**THIS SHOULD BE REVISITED AND EDITED FOR ACCURACY**

**Adding a Compute Node**

Check the instructions at the beginning of the packet, and adding the Openstack repo

Install openstack-nova-compute package

Configure /etc/nova/nova.conf to have it listen on 172.24.101.0/24

my\_ip=172.24.101.20

vncserver\_proxyclient\_address=$my\_ip

vncserver\_listen= 0.0.0.0

glance\_host=172.24.101.10

**Managing Instances**

As tenant user - source keystonerc\_user

nova list; nova show small; nova console-log small

nova hypervisor-list; source keystonerc\_admin and retry if needed

nova migrate web or live-migrate -- moves instance "web" to another location

/etc/nova/policy.json --says has more ineresting stuff

look for admin\_actions:migrate

change rule:admin\_api to rule:admin\_or\_owner

nova migrate (try again)

nova --help TONS - ignore network comands and use Neutron for those

Overview review

Essentials: compute (hypervisor), controller, object storage, networking

Controller essentials - MQ, DB, Keystone.

Obj storage - with Swift at least 3 diff servers

A typical cloud at least 10 different hosts

Bigger it gets more specific the hosts can be, more machines

HA - each host redundant - especially essentials

Security

- pub accessible and administrative parts clearly distinguished

- Sure VMs need to be accessible, but it doesn't mean every hypervisor node needs to be accessible

- Neutron using specific network nodes for external connectivity

- No public access for backend services

- API endpoints never need to be in public space - they need to be kept in a demarc'd private area

- Instances DO usually need to be accessible to be functional

- Think about cloud purpose.

Compute focus? Service orientation like AWS?

Storage focus with lots of nodes, particularly dozens of Ceph nodes?

Network focused?

How big a deal is multisite?

Is it for redundancy, regional access, both?

Deployment / Automation

- Puppet, Chef

- Triple-O (Openstack on Openstack) a common method

Nova, neutron, heat to automate management

Ironic for deployment, using PXE and IPMI; pluggable drivers, Ironic-specific glance images

- Red Hat Foreman, SUSE Crowbar

Requirements:

- Physical installation of nodes - PXE (boot) and TFTP (image delivery + other repo)

- drive setup, net configs, PXE working

IPMI (usually?) lives on a separate board with miniOS to do it's thing

**FINAL Part is comprised of the last videos, 92-99.mp4**

