Lab 12. Building an OpenStack Cloud Platform

Chan-Hyun YOUN

Dept of Electrical Engineering, KAIST

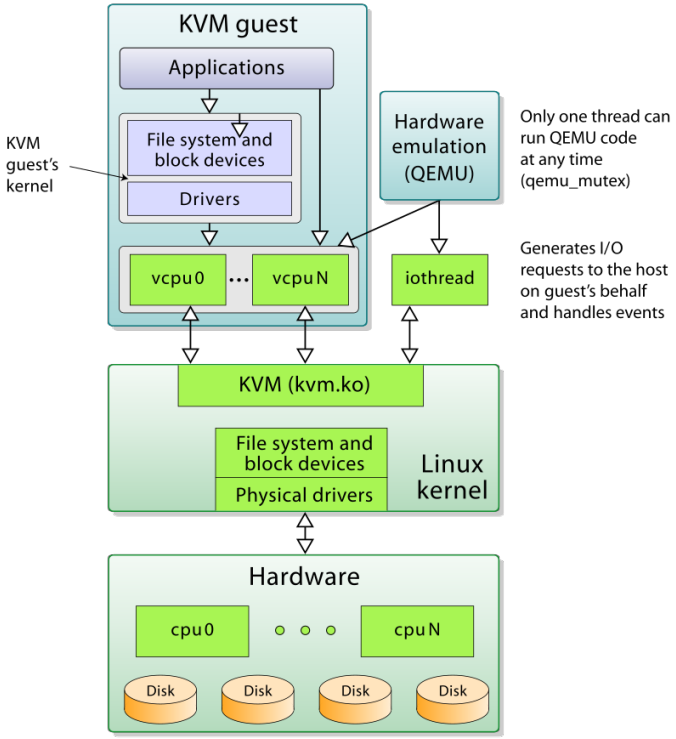
LAB 12-1)
KVM HYPERVISOR

KVM Hypervisor

- Kernel-based Virtual Machine (KVM)
  - KVM is a virtualization infrastructure for the Linux kernel that turns it into a hypervisor
  - It was merged into the Linux kernel mainline in kernel version 2.6.20
  - It requires a processor with hardware virtualization extensions

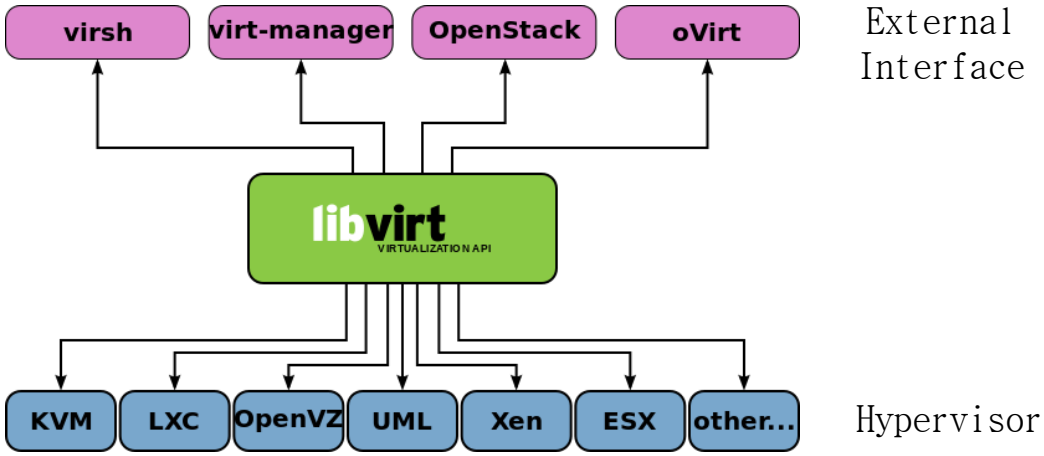
KVM Hypervisor

- Internals
  - Set up the guest VM's address space. The host must also supply a firmware image that the guest can use to bootstrap into its main OS.
  - Feed the guest simulated I/O.
  - Map the guest's video display back onto the host.
  - QEMU support host emulation



Libvirt – Hypervisor Orchestrator

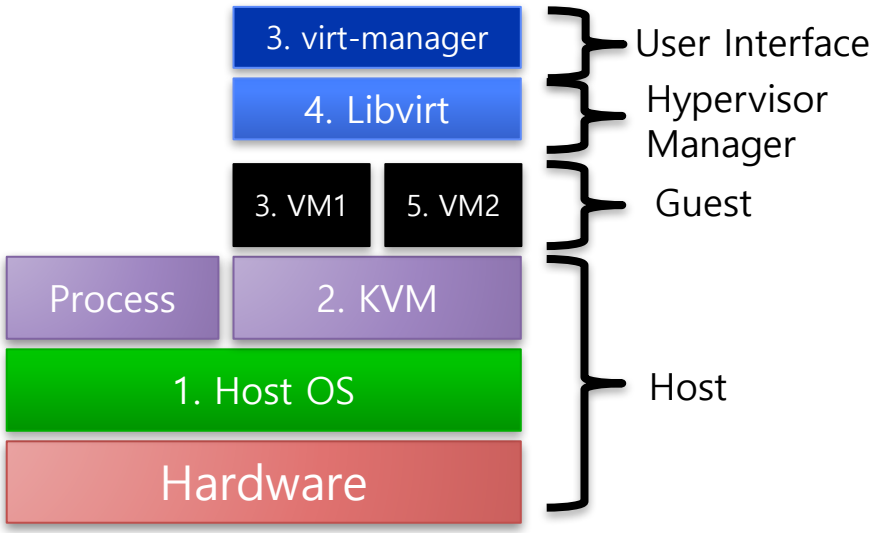
- Libvirt
  - Open-source API, daemon and management tool for managing platform virtualization
  - It can be used to manage KVM, Xen, VMware ESX, QEMU and other virtualization technologies



https://libvirt.org/

Overview of Lab 12-1

1. Prepare Host OS on your computing environment (Bare metal PC or VirtualBox)
2. Operate KVM Hypervisor to your Host OS
3. Create Virtual Machine with user interface ‘virt-manager’
4. Test the performance and manage the VM resources via virt-manager Virt-manager is operated with calling (libvirt api)



1. Prepare Host OS

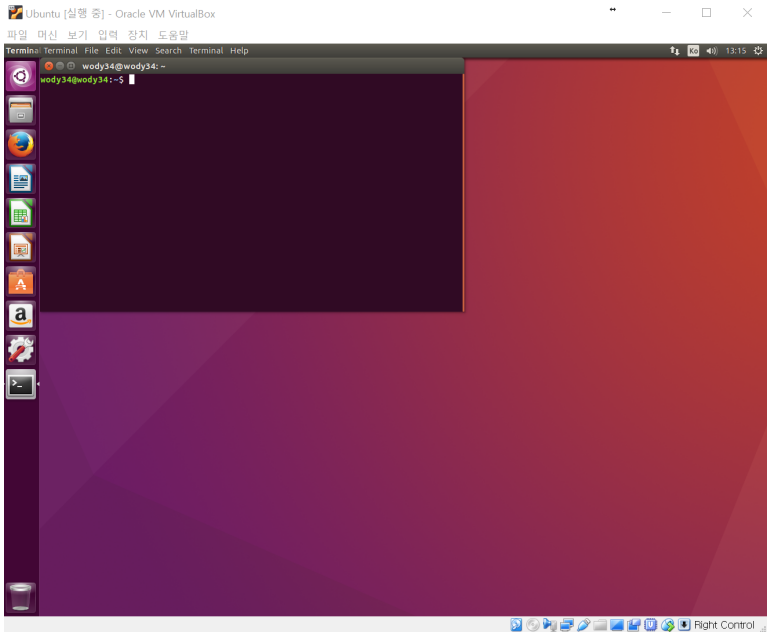
- Prepare Host OS (Linux Ubuntu 16.04.3 LTS)
  - Your CPU should support **Hardware-assisted virtualization**
    - Intel VT-x or AMD-V
    - If not, the VM is launched under emulated mode (very slow)
    - You can check your CPU information at /proc/cpuinfo

```
processor       : 0
vendor_id       : AuthenticAMD
cpu family      : 16
model           : 4
model name      : AMD Phenom(tm) II X4 925 Processor
cpu MHz         : 2731.284
cache size      : 512 KB
…
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36
clflush mmx fxsr sse sse2 ht syscall nx fxsr_opt rdtscp lm 3dnowext 3dnow rep_good nopl
extd_apicid pni lahf_lm cmp_legacy cr8_legacy svm
…
```

- Intel: vmx(Hardware virtualization)
- AMD: svm(Secure virtual machine)

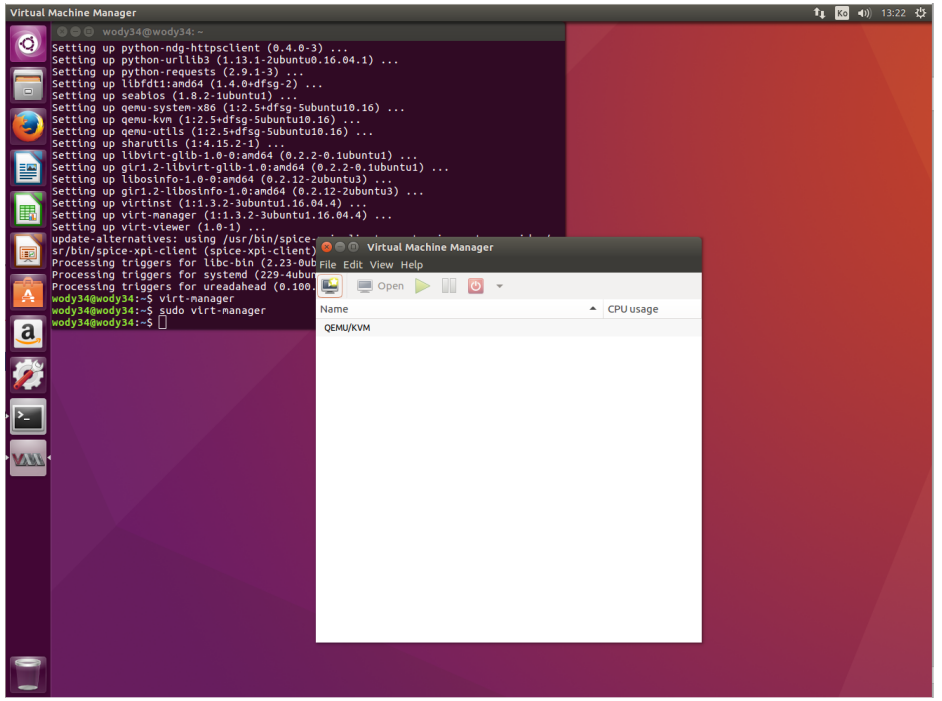
1. Prepare Host OS

- Prepare Ubuntu 16.04.3 LTS Desktop version in your computing environment
  - There is way to install Ubuntu in VirtualBox Environment
  - Refer Ubuntu Installation (Virtual Box).pdf in Lab2 supplement
- Update packages
  - # sudo apt-get update –y
  - # sudo apt-get upgrade –y
  - # sudo apt-get dist-upgrade –y
  - # sudo apt-get install openssh-server git -y



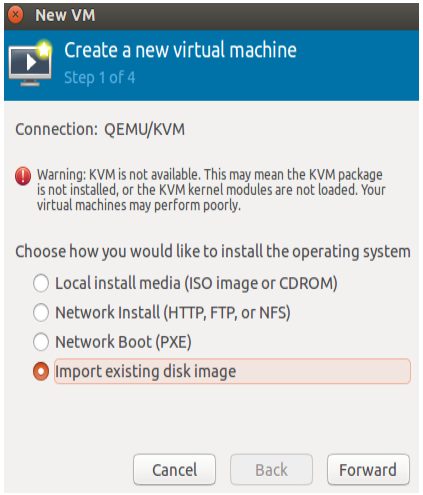
2. Install KVM and libraries

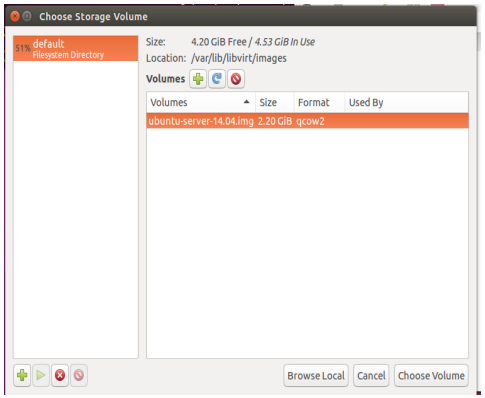
- Prepare KVM Hypervisor and virt-manager
  - # sudo apt-get install qemu-kvm libvirt-bin virt-manager
- Run virt-manager
  - # sudo virt-manager

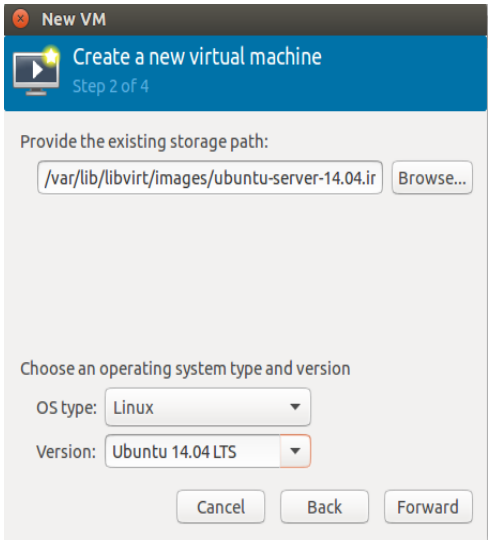


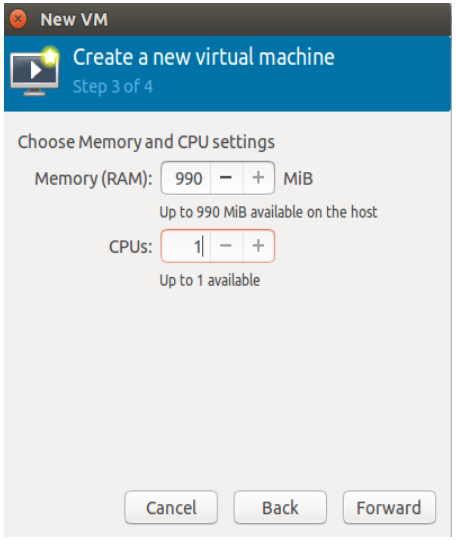
3. Create Virtual Machine

- Prepare VM image
  - Download Ubuntu image to /var/lib/libvirt/images
    $ sudo wget http://143.248.147.118/~djshin/ubuntu-server-14.04.img
  - Click button ‘Create a new virtual machine’



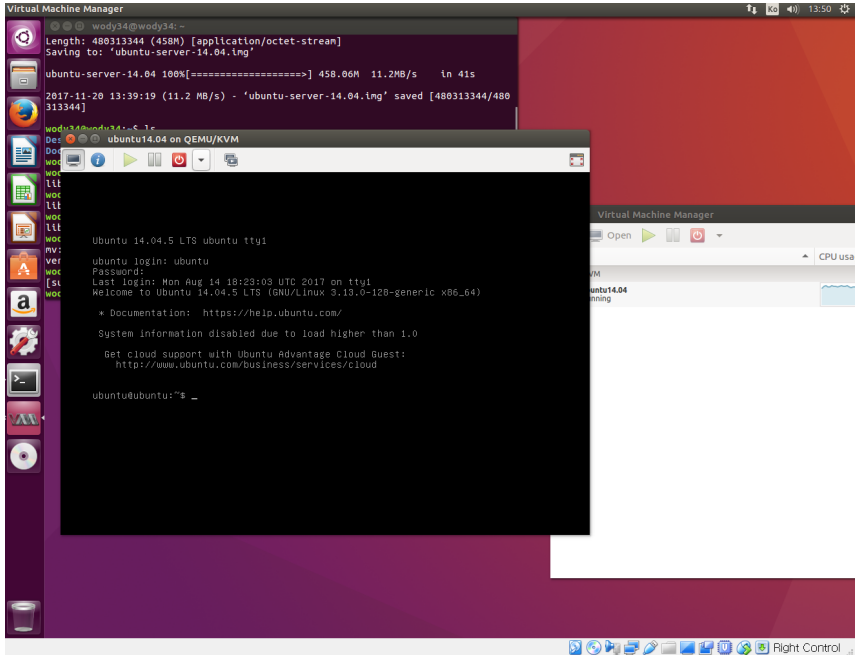






3. Create Virtual Machine

- Log-in to Guest OS
  - ubuntu / ubuntu
- Set up root password
  - # sudo passwd root
  - Type Your Password!
- Log-in with Root account
  - # su – root
  - Type Your Password!
- Check Guest's IP address
  - # ifconfig
- Power-off your guest

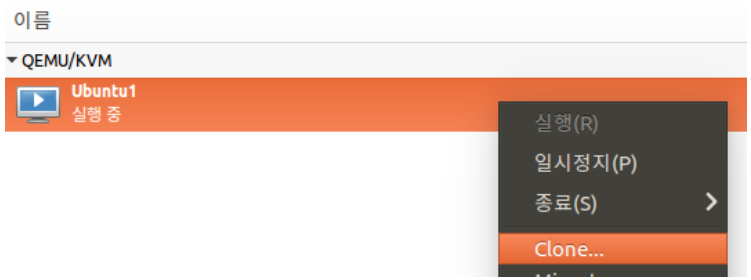



```
ubuntu@ubuntu:~$ su - root
Password:
root@ubuntu:~# _
```

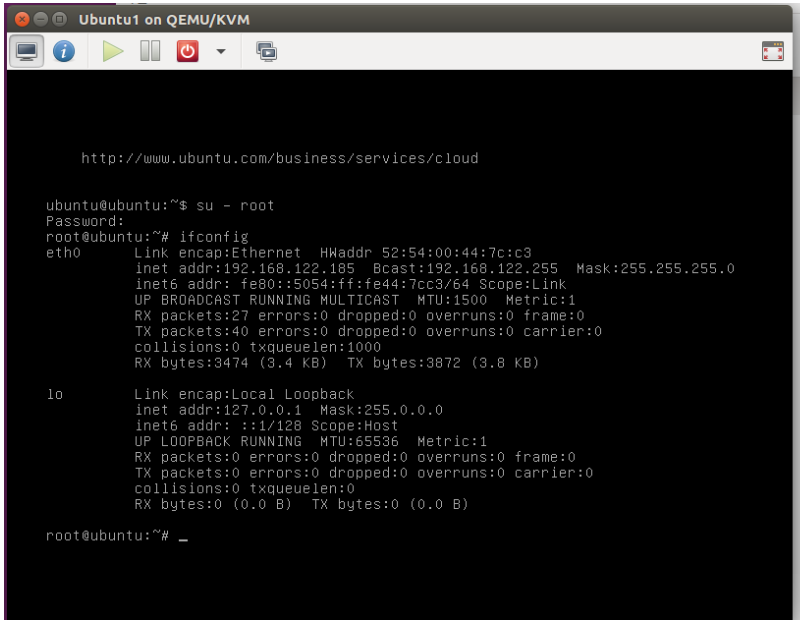
```
root@ubuntu:~# ifconfig
eth0      Link encap:Ethernet  HWaddr
          inet addr:192.168.122.185
```

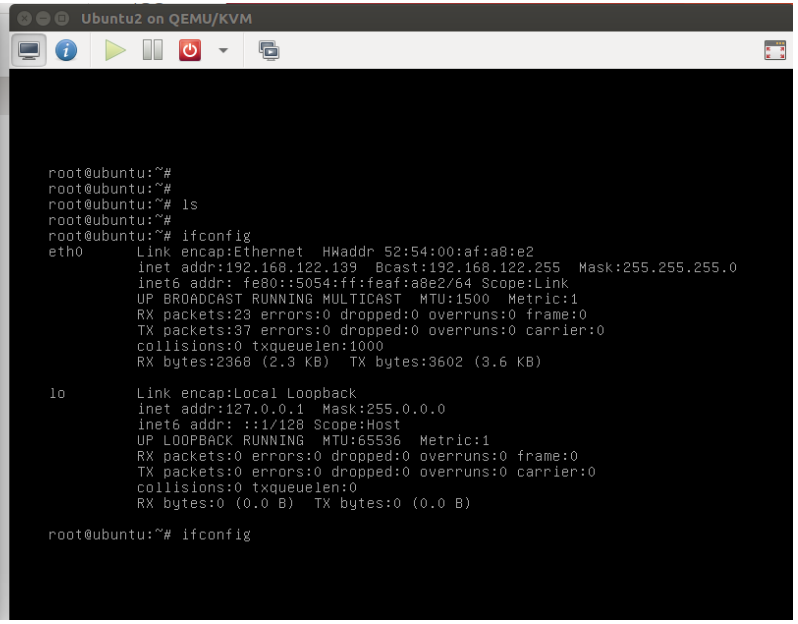
3. Create Virtual Machine

- Clone for 2nd VM
  - Clone : Make a new copied VM
  - Right click -> Clone



- Boot both of VMs
  - Check its IP address
    - # ifconfig



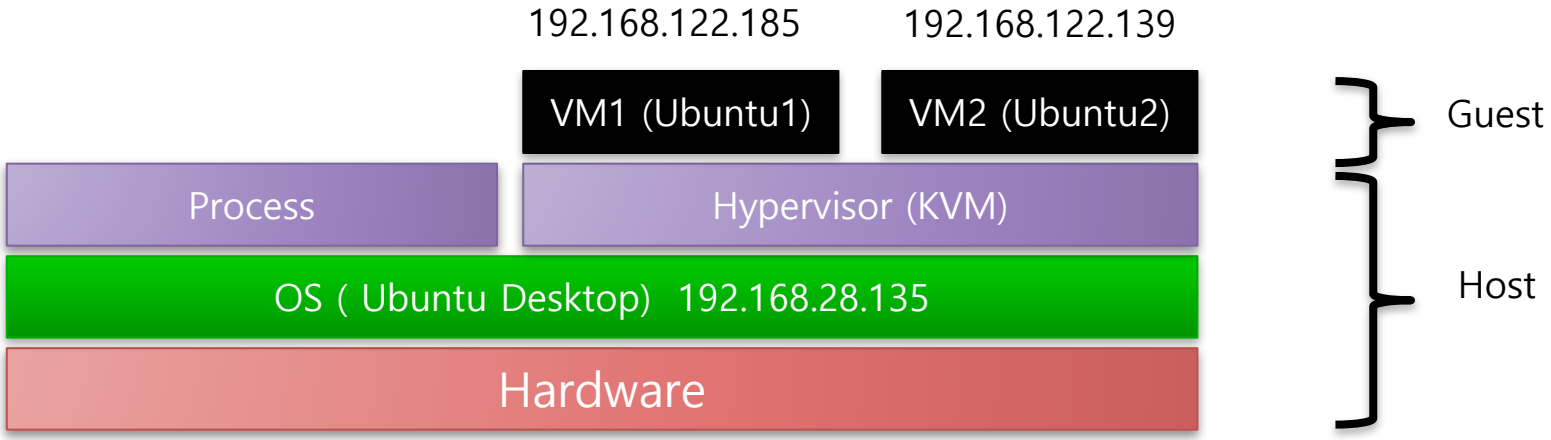


192.168.122.185

192.168.122.139

3. Create Virtual Machine

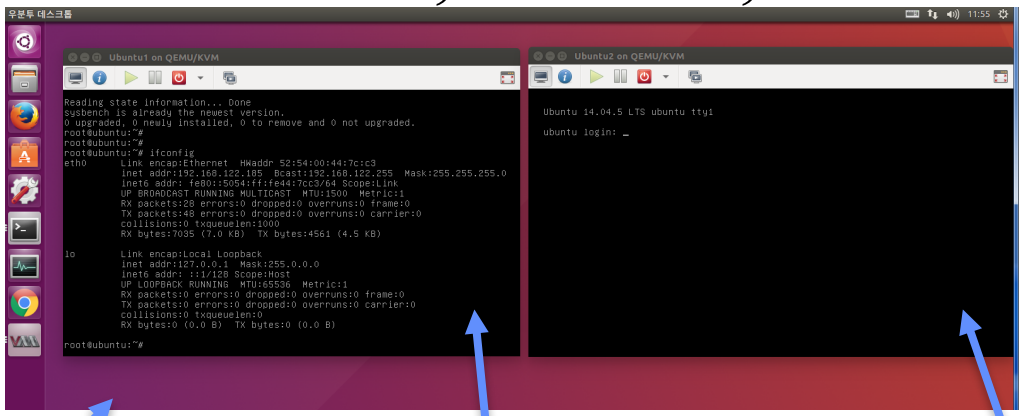
- Configured Cloud Environment

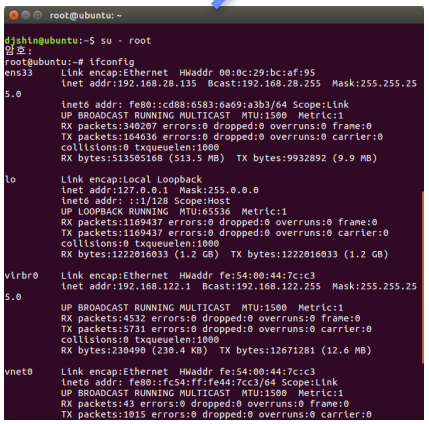


- IP address
  - # ifconfig
- CPU Info
  - # cat /proc/cpuinfo
- Process Info
  - # top
- Free memory
  - # free –m
- Disk Info
  - # df -hT

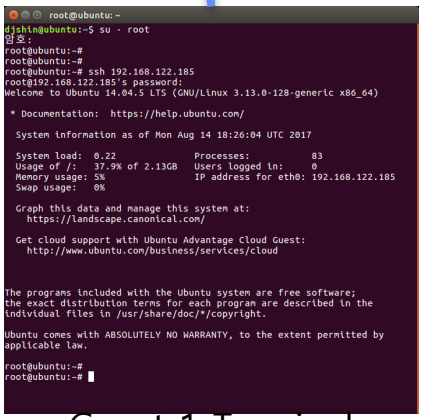
3. Create Virtual Machine

- 3 Terminals for Host, Guest1 , Guest2

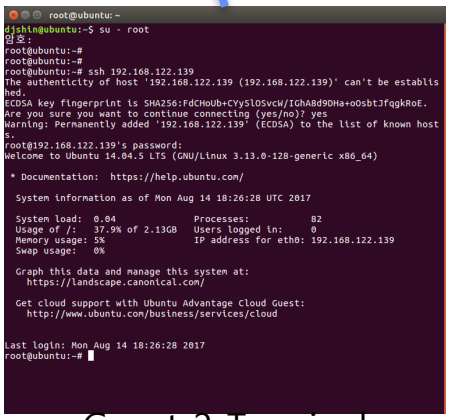




Host Terminal



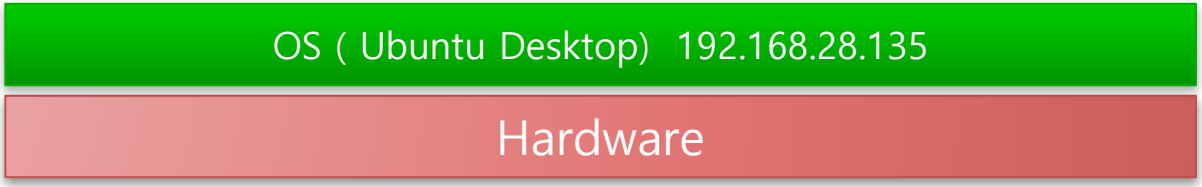
Guest 1 Terminal
(192.168.122.185)



Guest 2 Terminal
(192.168.122.139)

3. Create Virtual Machine

- Host's View



- 6 CPU Cores
- 8GB of RAM
- 52GB of Disk

- 2 VMs as a Process

```
root@ubuntu:~# cat /proc/cpuinfo | grep processor
processor       : 0
processor       : 1
processor       : 2
processor       : 3
processor       : 4
processor       : 5
```

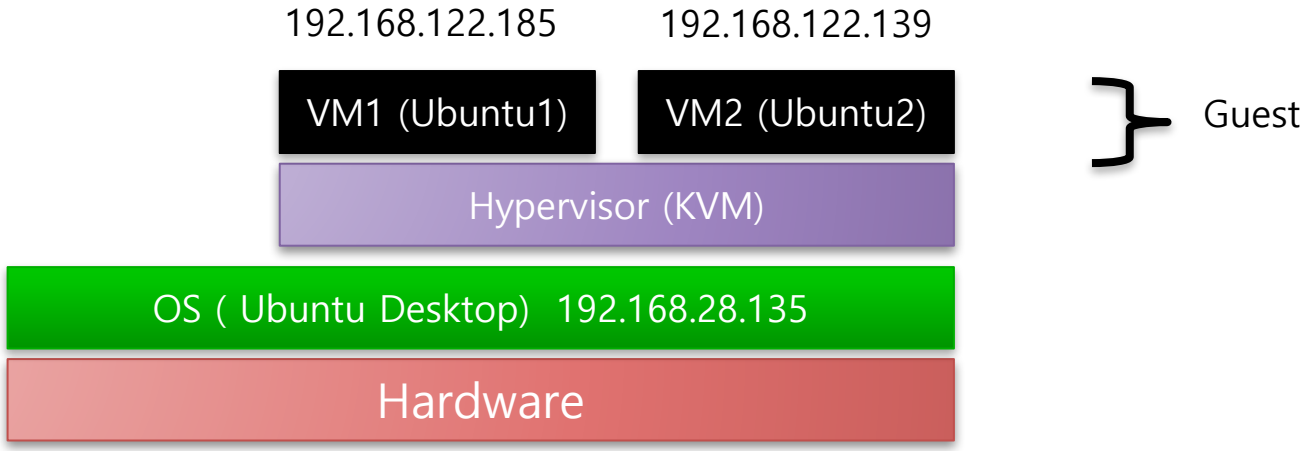
```
root@ubuntu:~# free -m
              total        used        free
Mem:           7963        2037        1328
Swap:          8189           0        8189
```

```
/dev/sda1       ext4        52G   17G   32G  35% /
```

```
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
11447 libvirt+  20   0 1870744 307608  18836 S 122.6  3.8   0:52.92 qemu-system-x86
11301 libvirt+  20   0 1965864 318832  19116 S 113.6  3.9   0:53.79 qemu-system-x86
```

3. Create Virtual Machine

- Guest's View



- 2 CPU Cores
- 1GB of RAM
- 2.2GB of Disk

```
root@ubuntu:/# cat /proc/cpuinfo |grep processor
processor       : 0
processor       : 1
```

```
root@ubuntu:/# free -m
             total       used       free
Mem:           993        160        832
-/+ buffers/cache:         65        928
Swap:            0          0          0
```

```
/dev/sda1       ext4      2.2G  842M  1.2G  41% /
```

3. Create Virtual Machine

- Benchmark
  - A number of standard test (computer program) to assess the relative performance
- We'll use 2 benchmarks
  - Sysbench, stress-ng
  - For each host and guests
  - # apt-get install stress-ng sysbench

3. Create Virtual Machine

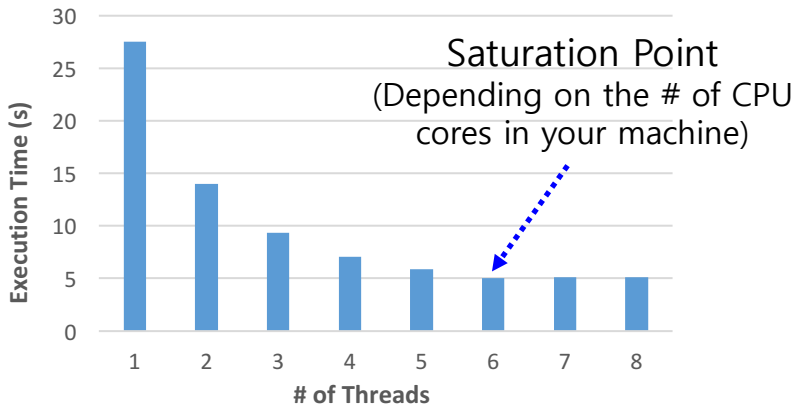
- sysbench CPU Test in Host
  - Monitor the result of the test and CPU usage with “top” command
  - # sysbench --test=cpu **--num-threads=1** --cpu-max-prime=25000 run

```
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 4934 djshin    20   0   28480   2364   2016 S 100.0  0.0   0:23.87 sysbench
```

  - increasing # of threads
  - # sysbench --test=cpu **--num-threads=2** --cpu-max-prime=25000 run
  - …

```
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 5055 djshin    20   0   28512   2384   2040 S 200.0  0.0   0:17.24 sysbench
```

  - # sysbench --test=cpu **--num-threads=8** --cpu-max-prime=25000 run



4. Virtual Machine Management

- Lifecycle Management
  - VM creation
  - VM deletion
  - VM migration
  - VM snapshot management (save & restore)
- Resource Management
  - CPU Management
    - Change number of assigned cores
  - Memory Management
    - Give or Take Memory
  - Virtual Disk Management
    - Create new disk image and mount

4. Virtual Machine Management

- In your host terminal
  - Host node info : # virsh nodeinfo
  - VM list : # virsh list
  - VM info : # virsh dominfo <VM name>

```
root@ubuntu:~# virsh nodeinfo
CPU 모델:           x86_64
CPU:                6
CPU 주파수:         3500 MHz
CPU 소켓:           2
소켓당 코어:        3
코어당 스레드: 1
NUMA cell:          1
메모리 용량:        8154208 KiB
```

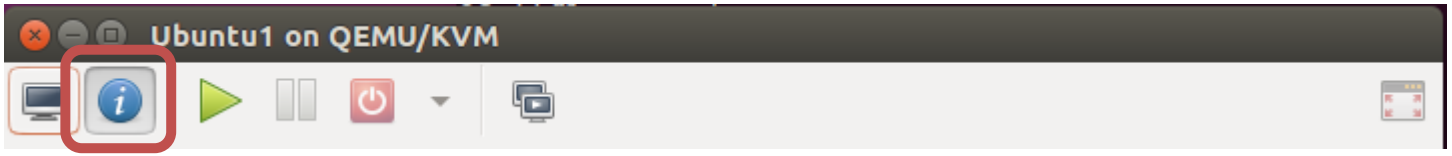
```
root@ubuntu:~# virsh list
 Id    이름                           상태
----------------------------------------------------
 1     Ubuntu1                        실행중
 2     Ubuntu2                        실행중
```

```
root@ubuntu:~# virsh dominfo Ubuntu1
Id:             1
이름:           Ubuntu1
UUID:           0906832b-e51c-4a5c-b76d-d2dbdaa8dba7
OS 유형:        hvm
상태:           실행중
CPU:            2
CPU 시간:       74.8s
최대 메모리: 1048576 KiB
사용된 메모리: 1048576 KiB
Persistent:     예
Autostart:      사용 안 함
Managed save:   아니요
모안 모델:  apparmor
보안 DOI:       0
보안 레이블: libvirt-0906832b-e51c-4a5c-b76d-d2dbdaa8dba7 (enforcing)
```

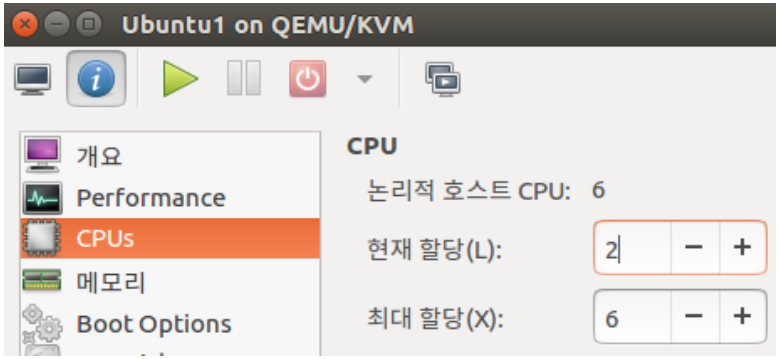
```
root@ubuntu:~# virsh dominfo Ubuntu2
Id:             2
이름:           Ubuntu2
UUID:           bf1fbdcd-57d4-42c1-a667-73c9a615d954
OS 유형:        hvm
상태:           실행중
CPU:            2
CPU 시간:       31.0s
최대 메모리: 1048576 KiB
사용된 메모리: 1048576 KiB
Persistent:     예
Autostart:      사용 안 함
Managed save:   아니요
모안 모델:  apparmor
보안 DOI:       0
보안 레이블: libvirt-bf1fbdcd-57d4-42c1-a667-73c9a615d954 (enforcing)
```

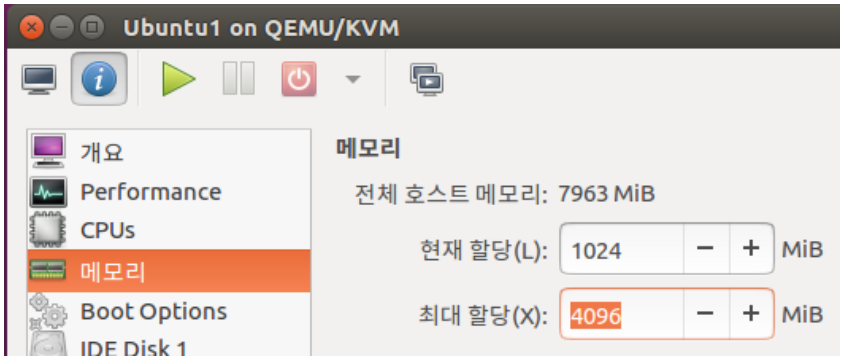
4. Virtual Machine Management

- Turn of your VMs
- virt-manager -> VM open -> VM Info



  - Set each of VMs
    - CPU 2, Memory 1G





- **Turn on all VMs**

4. Virtual Machine Management

- In your guest terminal
  - Run CPU Benchmark
    - # sysbench --test=cpu **--num-threads=1** --cpu-max-prime=25000 run
    - …
    - # sysbench --test=cpu **--num-threads=6** --cpu-max-prime=25000 run

```
Test execution summary:
    total time:                          14.1875s
```

- In your host terminal
  - Give more cores to VM1 (2 to 6)
    - # virsh setvcpus Ubuntu1 **--count 6**
- In your guest terminal
  - Turn on the cores
    - # echo 1 > /sys/devices/system/cpu/cpu2/online
    - # echo 1 > /sys/devices/system/cpu/cpu3/online
    - # echo 1 > /sys/devices/system/cpu/cpu4/online
    - # echo 1 > /sys/devices/system/cpu/cpu5/online
    - # lscpu

```
root@ubuntu:~# lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                6
On-line CPU(s) list:   0-5
```

- Run benchmark again in the guest VM
  - # sysbench --test=cpu **--num-threads=6** --cpu-max-prime=25000 run

```
Test execution summary:
    total time:                          5.2964s
```

4. Virtual Machine Management

- In your guest terminal
  - Check the size of free memory
    - # free -m
  - Run Memory Benchmark
    - # stress-ng --vm 1 --vm-bytes 1G --timeout 20s --metrics-brief
  - Cannot run the benchmark due to the out of memory .

```
root@ubuntu:~# free -m
             total       used       free     shared    buffers     cached
Mem:           880        123        757          0         11         50
-/+ buffers/cache:         61        819
Swap:            0          0          0
```

```
root@ubuntu:~# stress-ng --vm 1 --vm-bytes 1G  --timeout 20s --metrics-brief
stress-ng: info: [1237] dispatching hogs: 1 vm
stress-ng: info: [1237] successful run completed in 20.00s
stress-ng: info: [1237] stressor      bogo ops real time  usr time  sys time
ogo ops/s   bogo ops/s
stress-ng: info: [1237]                           (secs)    (secs)    (secs)
eal time) (usr+sys time)
```

- In your host terminal
  - Give more memory to VM1
    - # virsh setmem Ubuntu1 --size 4G

- In your guest terminal
  - Check the size of free memory
    - # free -m
  - Run Memory Benchmark
    - # stress-ng --vm 1 --vm-bytes 1G --timeout 20s --metrics-brief

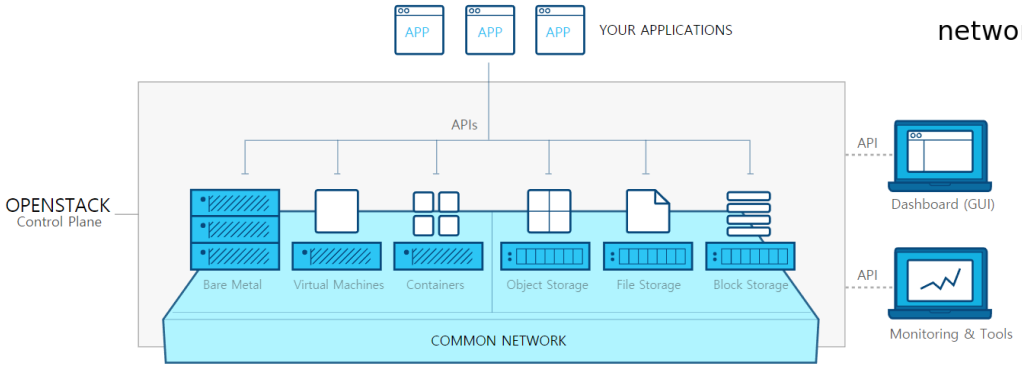
```
root@ubuntu:~# free -m
             total       used       free     shared    buffers     cached
Mem:          3952        123       3828          0         11         50
-/+ buffers/cache:         62       3890
Swap:            0          0          0
```

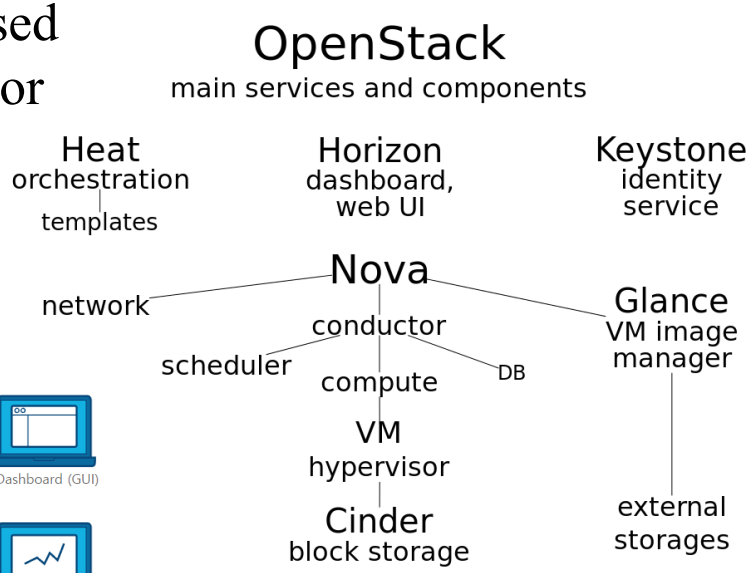
```
root@ubuntu:~# stress-ng --vm 1 --vm-bytes 1G  --timeout 20s --metrics-brief
stress-ng: info: [1242] dispatching hogs: 1 vm
stress-ng: info: [1242] successful run completed in 20.04s
stress-ng: info: [1242] stressor      bogo ops real time  usr time  sys time   bogo ops/s   bogo ops/s
stress-ng: info: [1242]                           (secs)    (secs)    (secs)   (real time) (usr+sys time)
stress-ng: info: [1242] vm            1250432     20.04     15.88      4.14     62403.02     62459.14
```

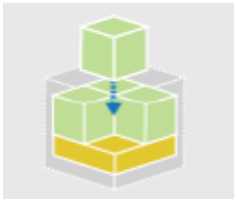
LAB 12-2) OPENSTACK CLOUD PLATFORM

OpenStack

- OpenStack is a free and open-source software platform for cloud computing, mostly deployed as infrastructure-as-a-service (IaaS), whereby virtual servers and other resources are made available to customers.
- The software platform consists of interrelated components that control diverse, multi-vendor hardware pools of processing, storage, and networking resources throughout a data center.
- Users either manage it through a web-based dashboard, through command-line tools, or through RESTful web services.
- http://www.openstack.org/



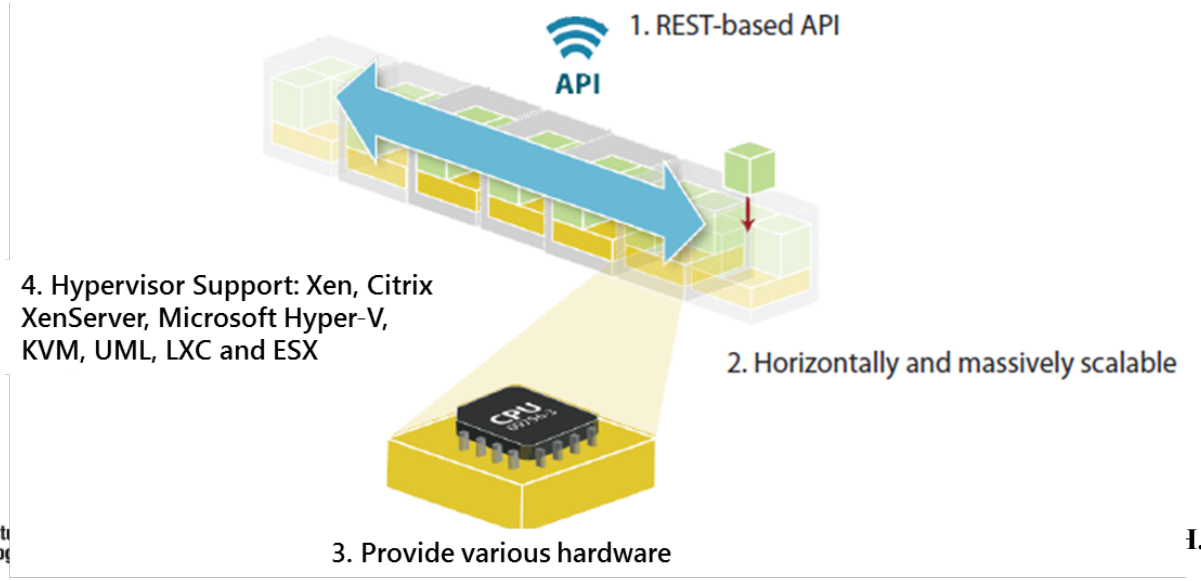




Refered from the article:
http://ken.pepple.info/openstack/2012/09/25/openstack-folsom-architecture/

OpenStack Components (Nova)

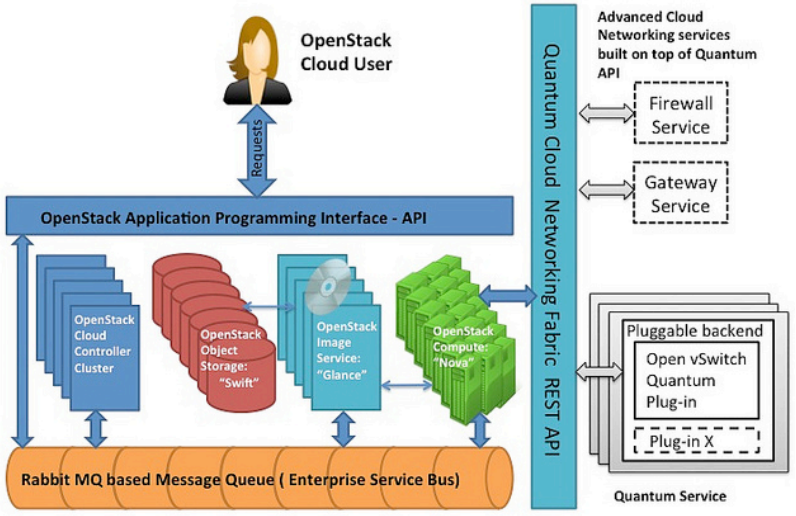
- Compute (code-named "**Nova**"):
  - Provides virtual servers upon demand
  - Manage and automate the large scale virtual compute instances pools using hypervisor
  - Work with widely available virtualization technologies, as well as bare metal and high-performance computing (HPC) configurations



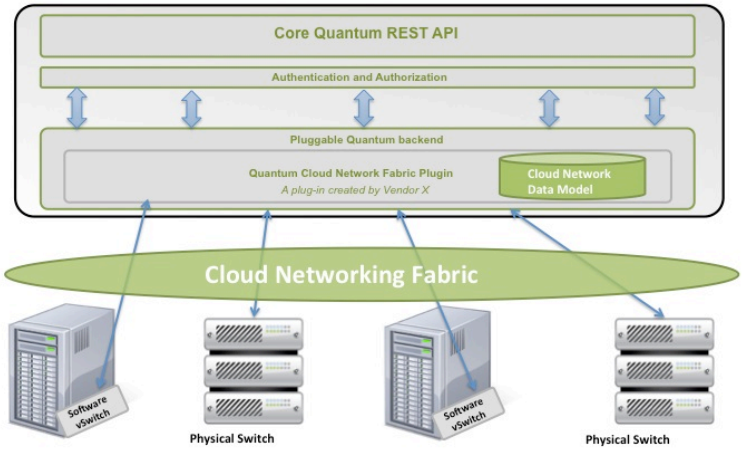
Refered from the article: http://openvswitch.org/openstack/2011/08/01/openstack-quantum-and-open-vswitch-part-ii/

OpenStack Components (Neutron)

- Network (code-named "**Neutron**"):
  - "Network connectivity as a service" between interface devices(e.g., vNICs) managed by other OpenStack services
  - Application-level abstraction of networking that relies on plug-in implementations to map the abstraction(s) to reality
  - Quantum has a pluggable, scalable architecture to support many popular networking vendors and technologies
  - **Quantum is new in the Folsom release**



OpenStack + Quantum Integration Architecture

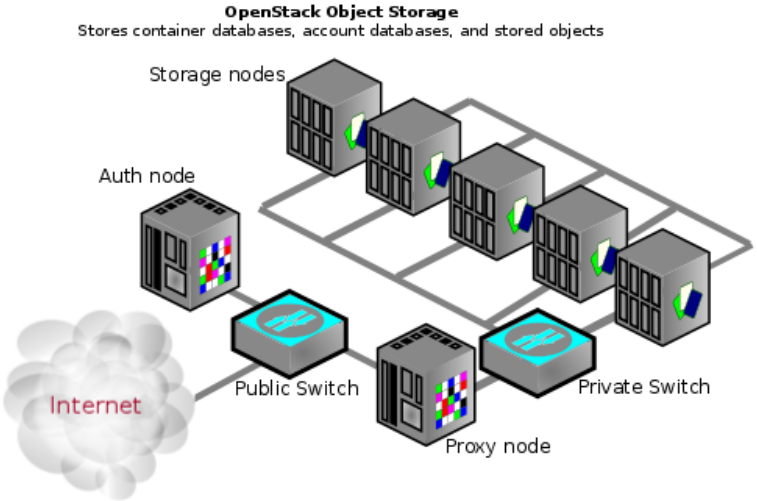


Reference Architecture – A Quantum Plugin

2017)

OpenStack Components (Swift, Cinder)

- Object Store (code-named "**Swift**"):
  - Store or retrieve files like key-value data structure (Not general file system)
  - Open source software for implementing a reliable and large-scale cloud object storage service
  - Massively scalable redundant storage system
  - Objects and files are written to multiple disk drives spread throughout servers in the data center, with the OpenStack software responsible for ensuring data replication and integrity across the cluster
- Block Storage (code-named "**Cinder**"):
  - Provides persistent block level storage devices for use with OpenStack compute instances
  - Block storage system manages the creation, attaching and detaching of the block devices to servers



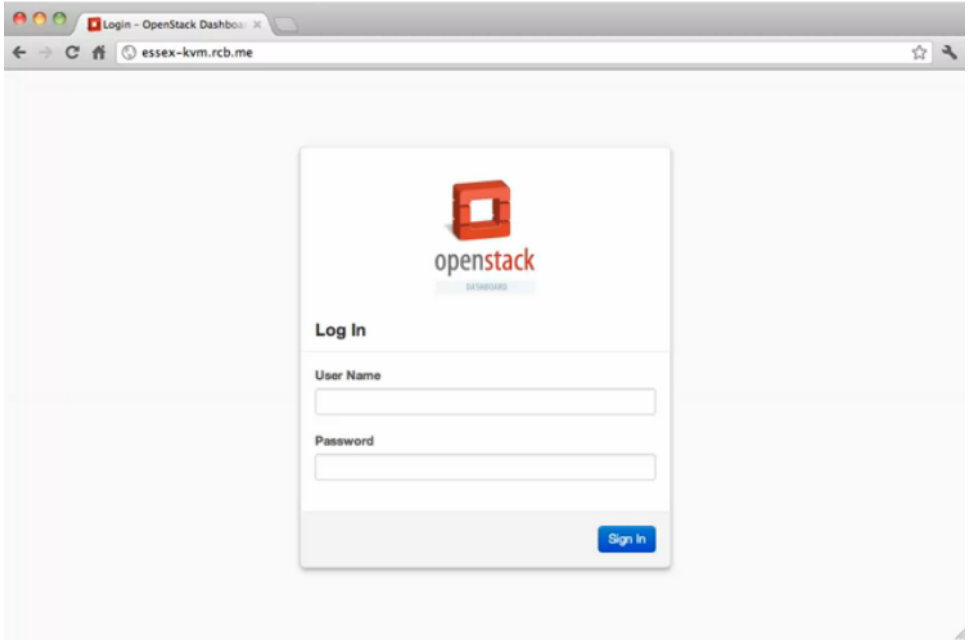
High Scalability of Swift!

OpenStack Components (Glance)

- Image (code-named "**Glance**"):
  - Provides a catalog and repository for virtual disk images
  - Discovery, registration and delivery services for disk and server images
  - Disk images are mostly commonly used for creating Virtual Machine instance in OpenStack Compute
  - Ability to copy or snapshot a server image and immediately store it away

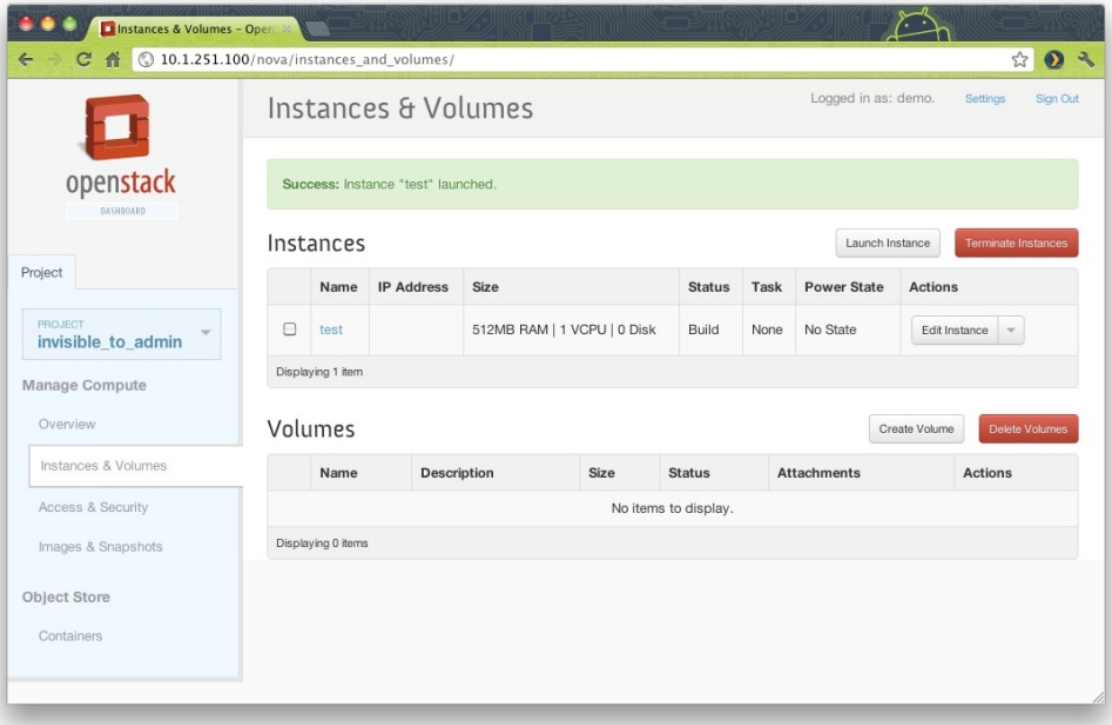
OpenStack Components (**Keystone**)

- Identity (code-named "**Keystone**"):
  - Provides authentication and authorization for all the OpenStack services
  - In API Service, keystone issues token to manage the session with clients
  - Provides a service catalog of services within a particular OpenStack Cloud

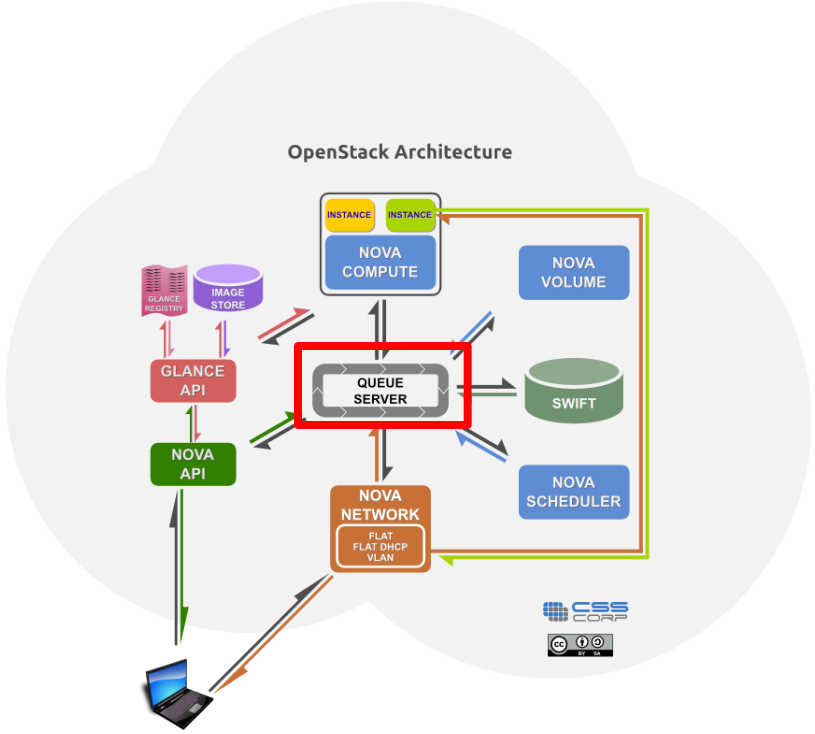


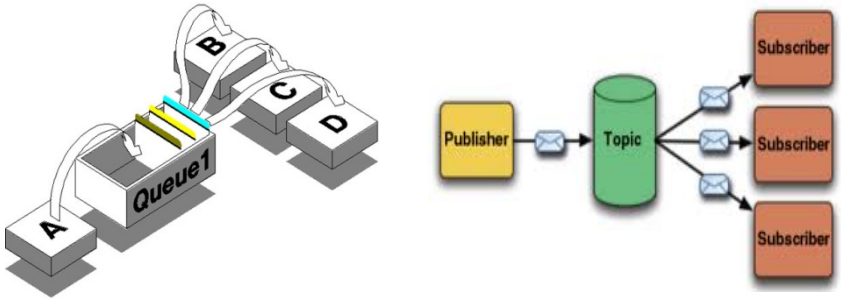
OpenStack Components (Horizon)

- Dashboard (code-named "**Horizon**"):
  - A modular web-based user interface for all the OpenStack services. With web GUI
  - Perform most operations on cloud like launching an instance, assigning IP addresses and setting access controls



RabbitMQ



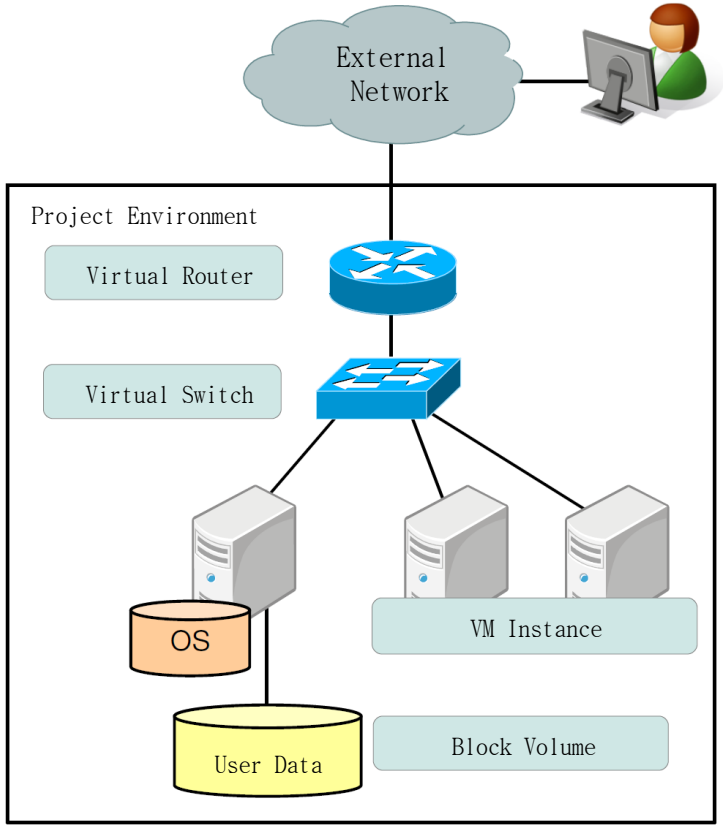


- RabbitMQ is open source message broker software (i.e., message-oriented middleware, message queue system) that implements the Advanced Message Queuing Protocol(AMQP) standard
- Software or hardware infrastructure supporting sending and receiving messages between distributed systems
- Allows application modules to be distributed over heterogeneous platforms and reduces the complexity of developing applications that span multiple operating systems and network protocols

  In OpenStack, RabbitMQ mediates various modules and forwards messages from one to others
- Can download at http://www.rabbitmq.com/

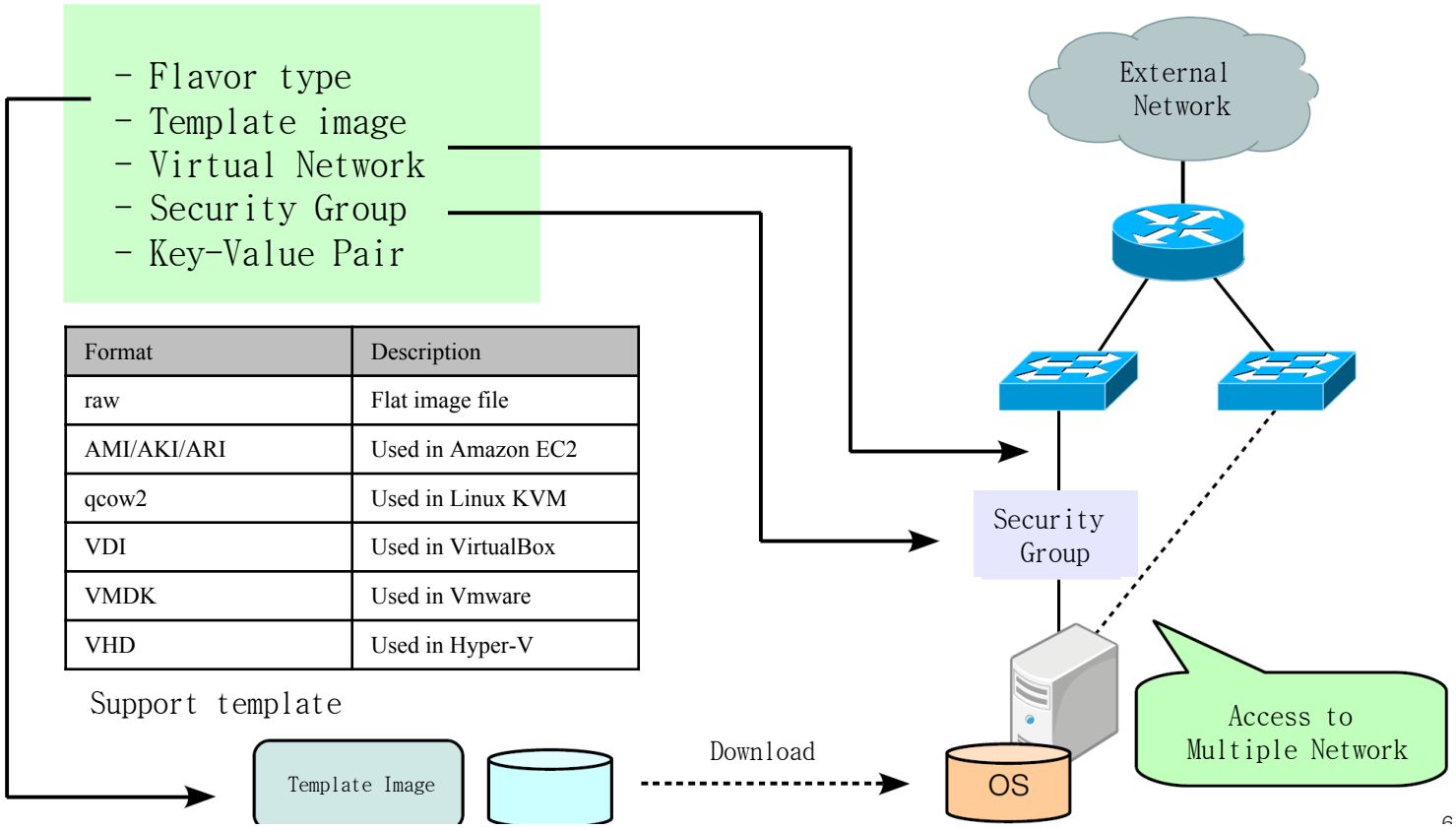
Create VM Instance

- The end-user of the open stack can create and configure the following computing resources in the user's private tenants through a web console or API.
  - Virtual network
  - VM Instant
  - Block volume
- Each user belongs to some projects.
  - Users in the same project share common computing resources in the project environment.
  - Each project owns (virtual) computing resources that are independent of other projects



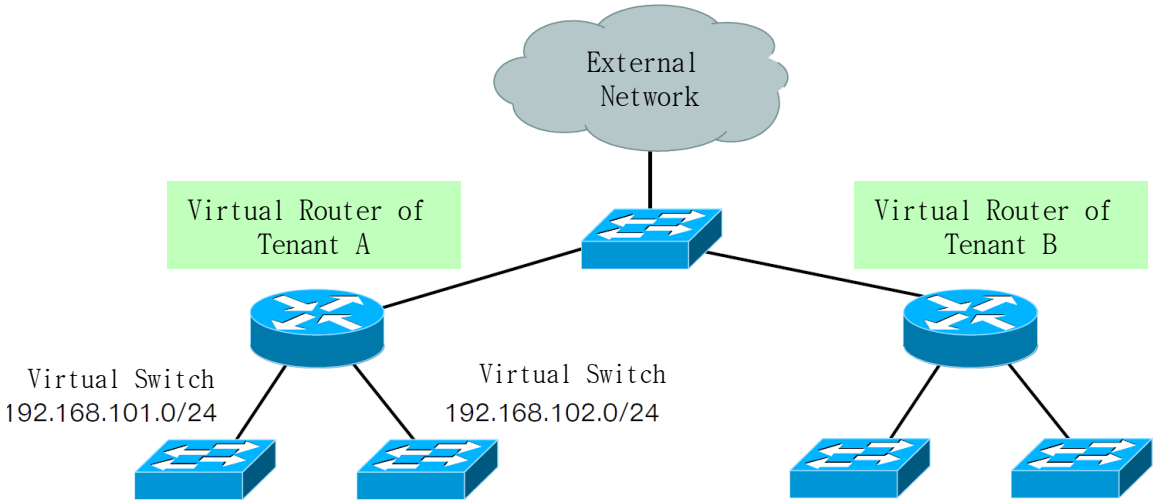
Create VM Instance

- When starting a new VM instance, you must specify the following options:



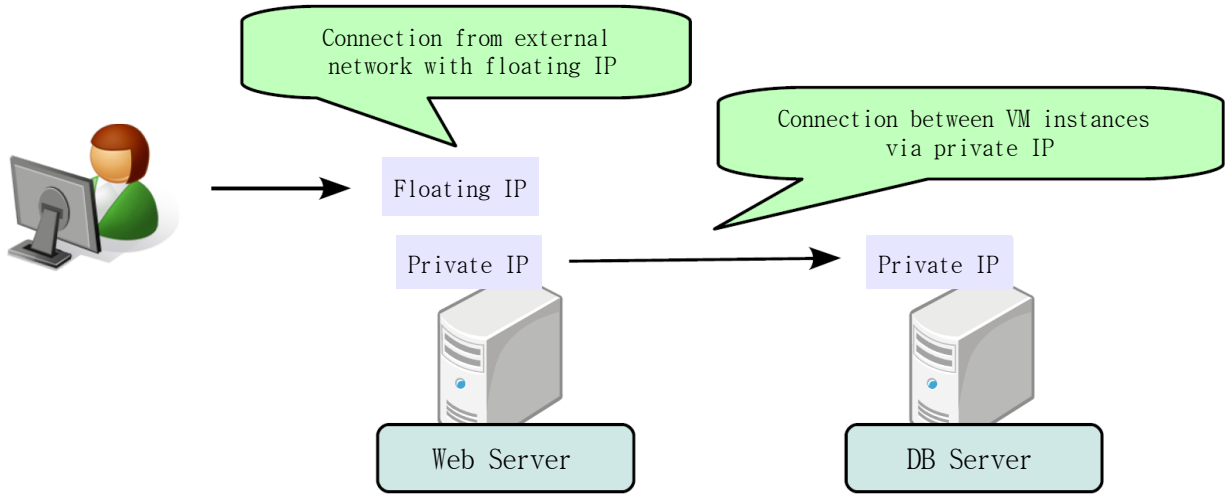
Logical View of Open Stack Virtual Networks

- Tenant users add virtual switches to the bottom of the router and assign a private subnet address. Subnets can be used in conjunction with other tenants.
- When starting the instance, the end user selects the virtual switch to connect to.
- The number of virtual NICs in the instance corresponds to the number of switches to connect. It assigns private IPs through DHCP.



Private IP and Floating IP

- When accessing from an external network, assign "Floating IP" to the VM instance.
  - IP addresses of external networks that can be used as floating IP are pooled in advance and distributed to each tenant.
  - Floating IP is NAT-converted from the virtual router to the corresponding private IP.
  - Accessing the external network from the VM instance is possible without floating IP allocation. In this case, the IP Masquerade function of the virtual router is used.



The instance type and disk space

- The following is the default instance type.
  - The root disk is copied from the template image and expanded to the specified size (except m1.tiny).

Flavor type	vCPU	Memory	Root disk	Temp disk	Swat disk
m1.tiny	1	512MB	0GB	0	0
m1.small	1	2GB	20GB	0	0
m1.medium	2	4GB	40GB	0	0
m1.large	4	4GB	80GB	0	0
m1.xlarge	8	8GB	160GB	0	0

- Administrators can define new instance types.
  - The following is an example using temp disk and swap disk.
  - Persistent data is stored elsewhere (typically block volumes) because these disks are deleted when the instance is destroyed.

```
NAME   MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
vda    252:0    0  20G  0 disk
└─vda1 252:1    0  20G  0 part /
vdb    252:16   0   5G  0 disk /mnt
vdc    252:32   0   1G  0 disk [SWAP]
```

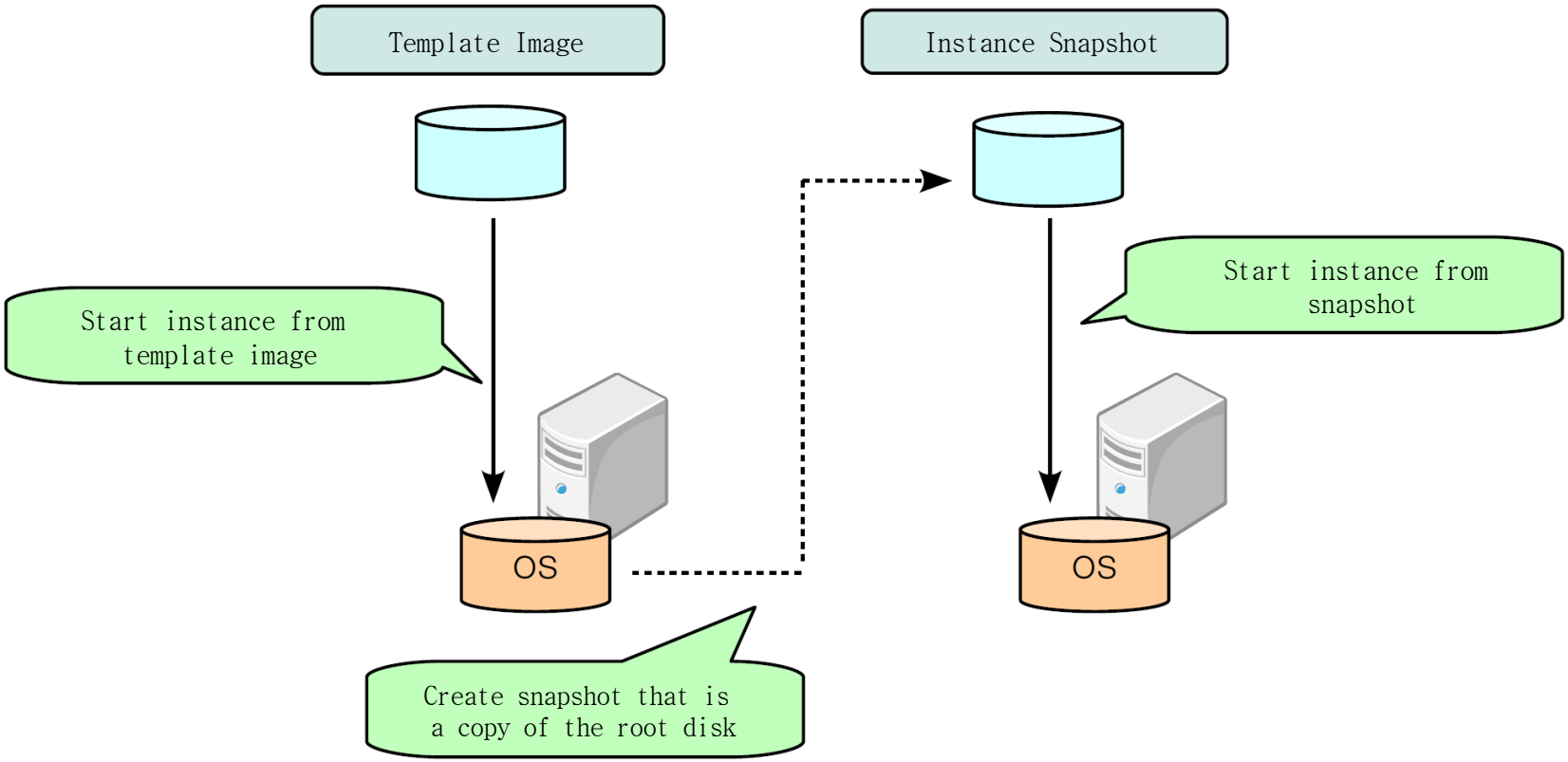
root disk

temp disk

swap disk

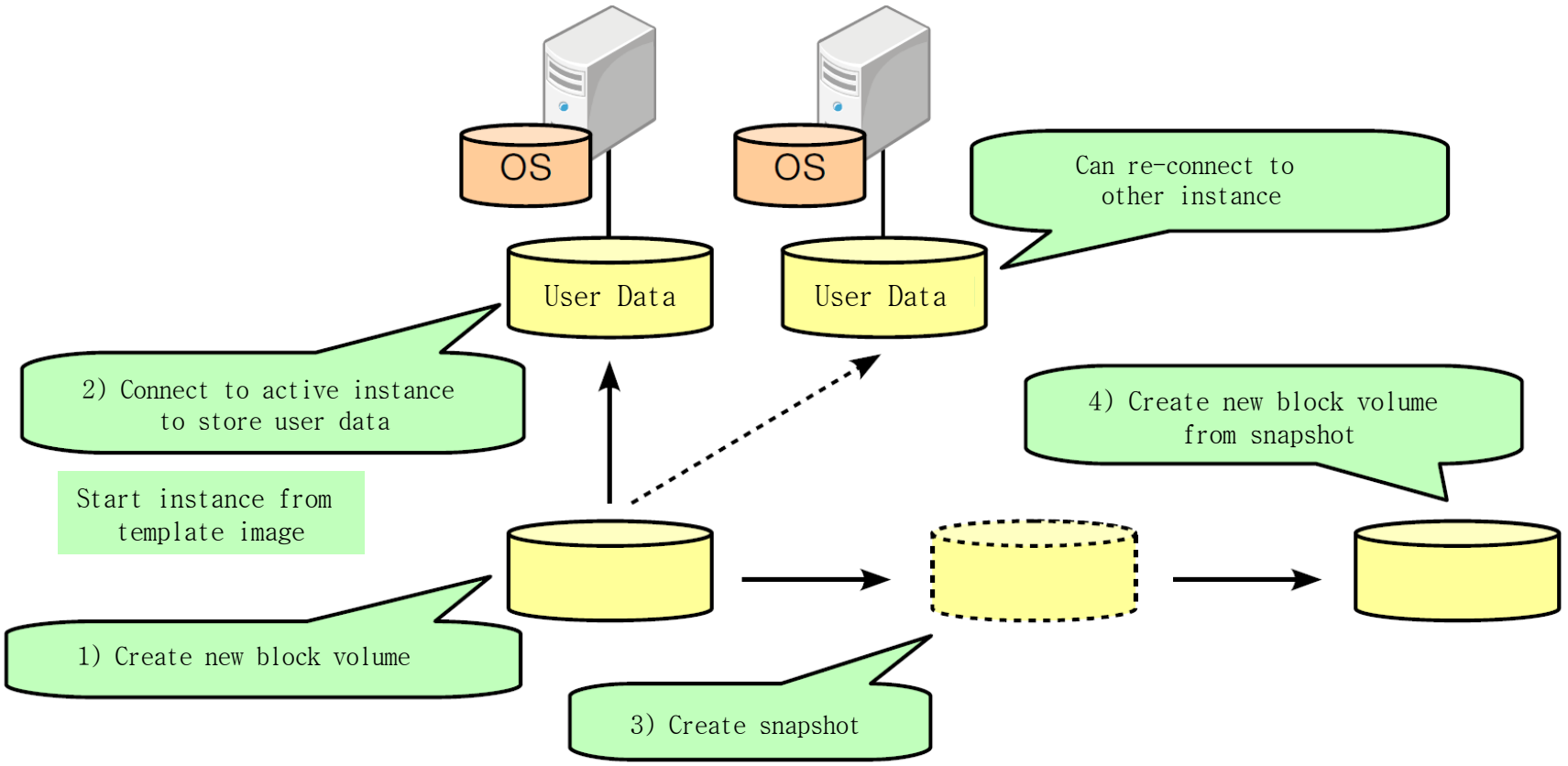
Snapshot of VM instance

- You can create a snapshot of the running instance, copy the root disk, and reuse it as a template image.



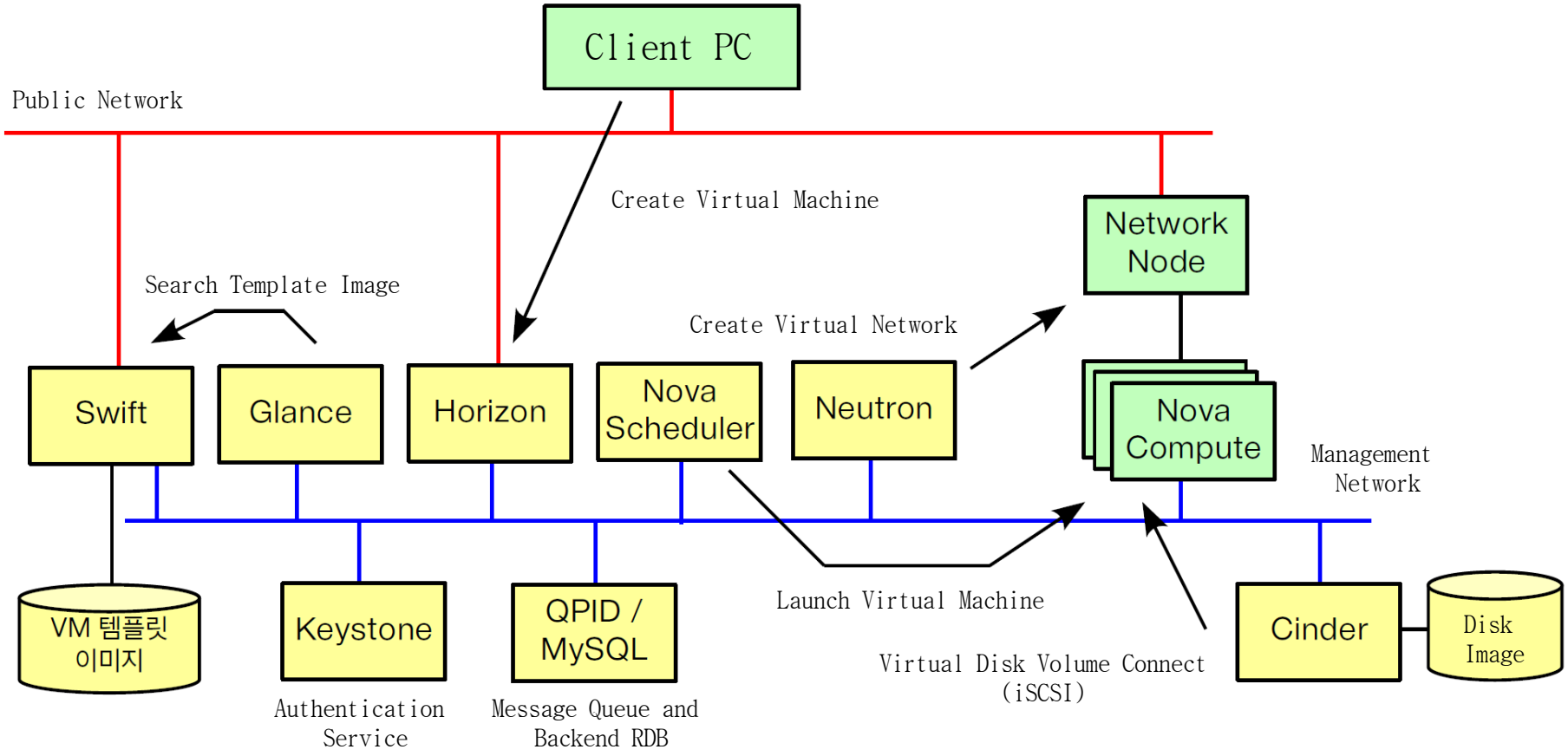
Persistent data store, block volume

- The block volume is not deleted when the VM instance is removed. It can be used as a persistent data store.



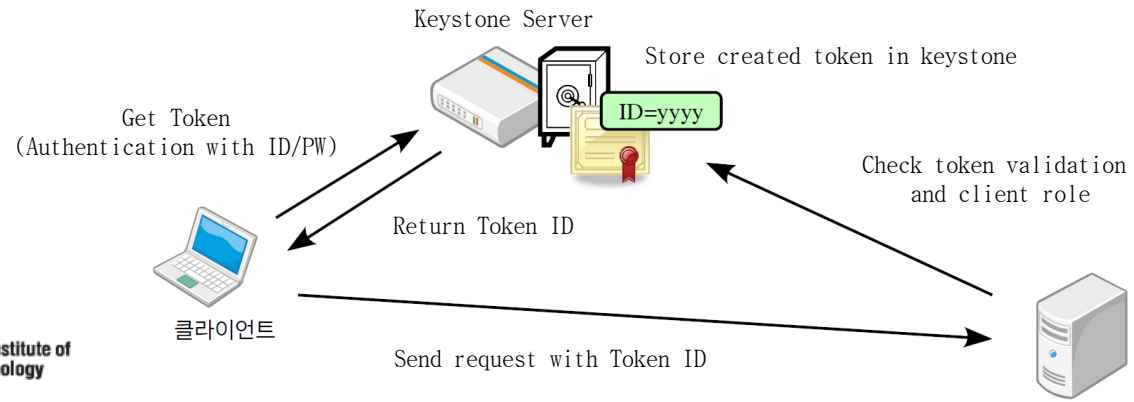
System Operation

- Modules work together through REST API calls and message queues.
  - Operation can be automated by external program through REST API.

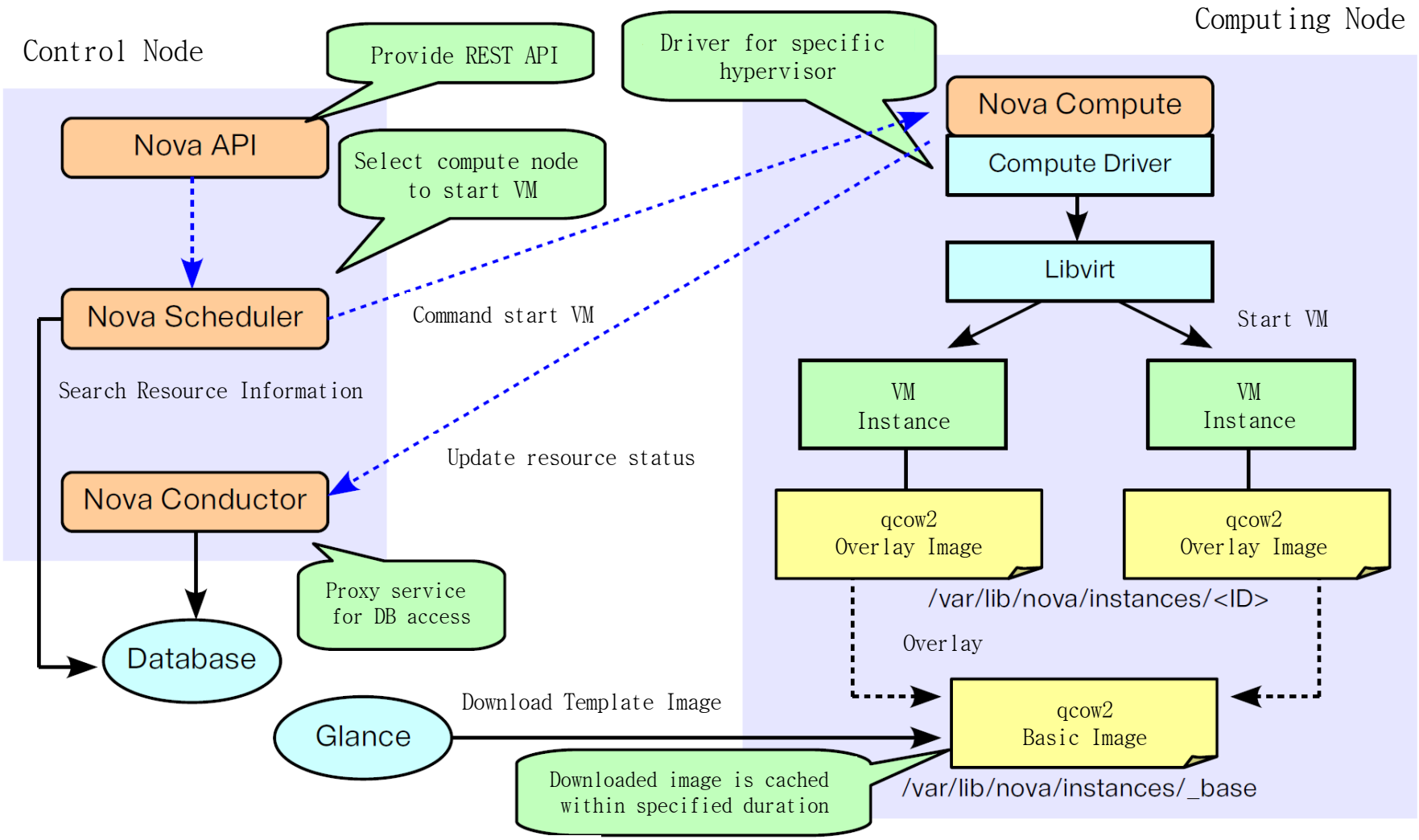


User authentication for API requests

- Authentication is required before sending a request to the API.
  - The end user / component obtains a "token" for the API operation from the keystone before sending the request to the API. (The user ID of each component is in the Keystone.)
  - Once the token is obtained, the URL for the target API is retrieved from the keystone. The end user must know only the URL to the Keystone API in advance.
- Since the open stack client makes many API calls to various components,
- Authentication using ID / password is not desirable in terms of security and performance.
- Instead, the client obtains a "token" as a "license" for the API call in advance,
- Sends the token ID to the component to be used.
  - The component receiving the request validates the token ID in the Keystone before approving the request.
  - The generated token is stored in the Keystone for a period of time (default: 24 hours). When the client expires
  - The token can be used again. The client does not need to obtain a token for each request invocation.

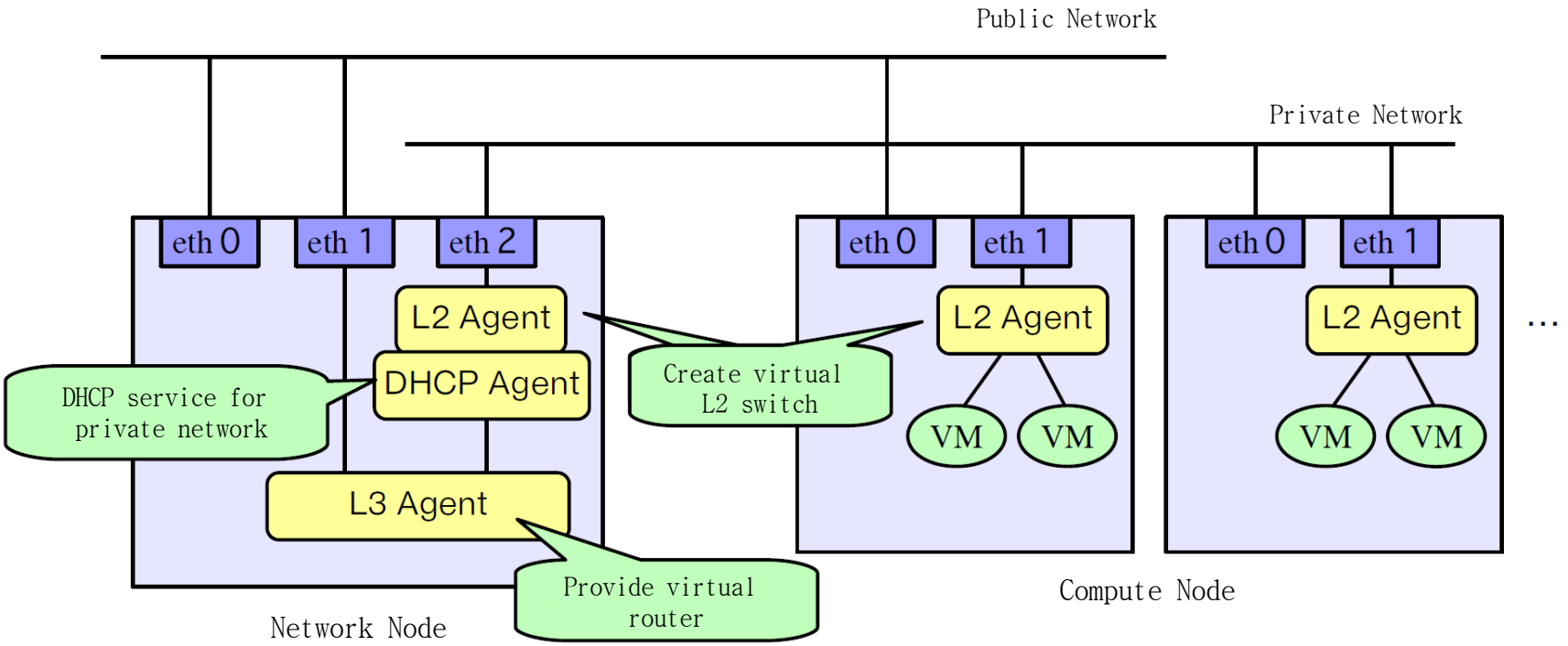


Nova Internal



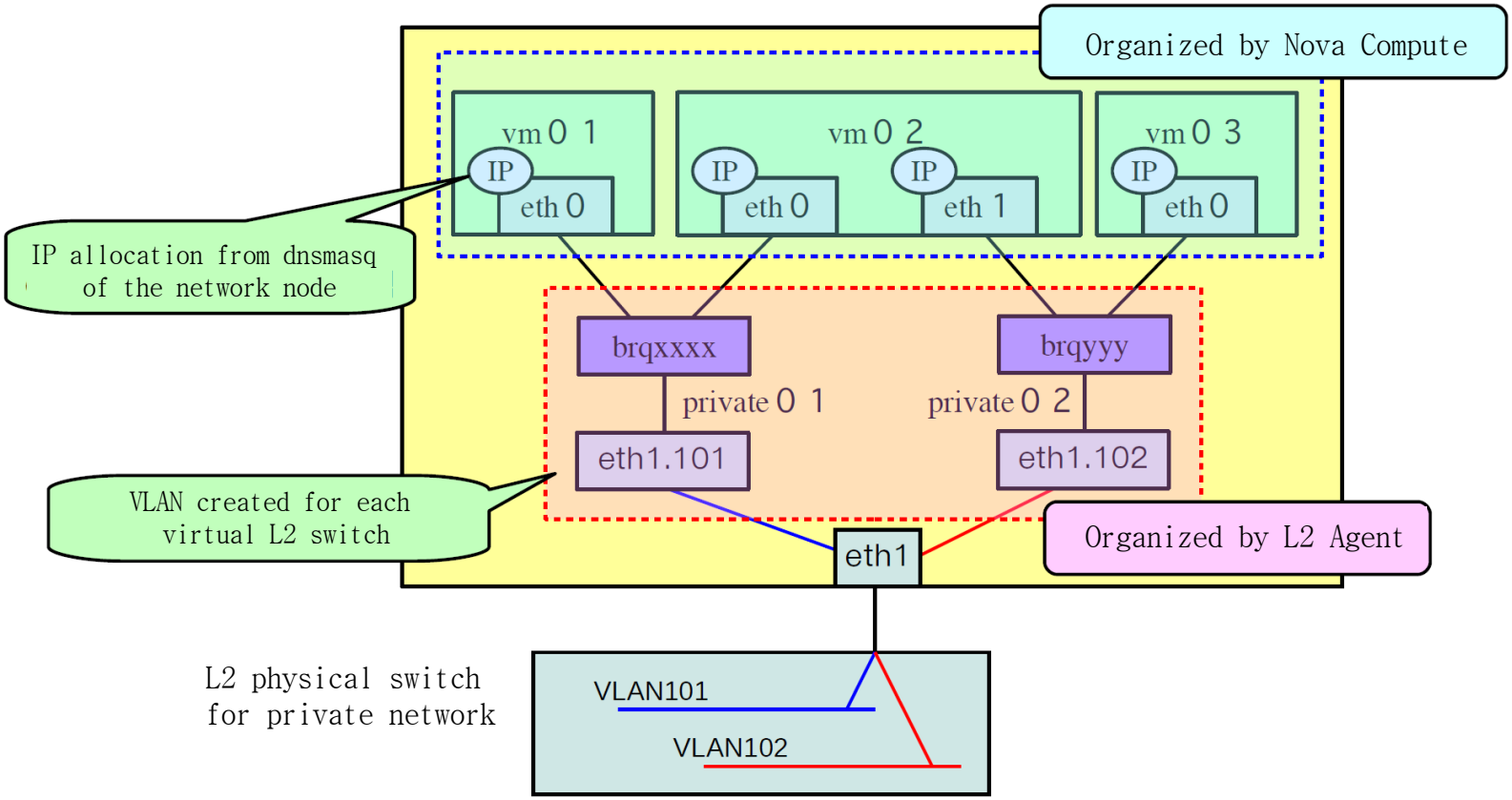
Network Composition

- Here is an example of a typical configuration using the LinuxBridge plugin or the Open vSwitch plugin:
  - The L3 Agent of a network node provides a virtual router function to connect private and public networks. (“eth0" on each node is used to access host Linux, not on VM instance communication)

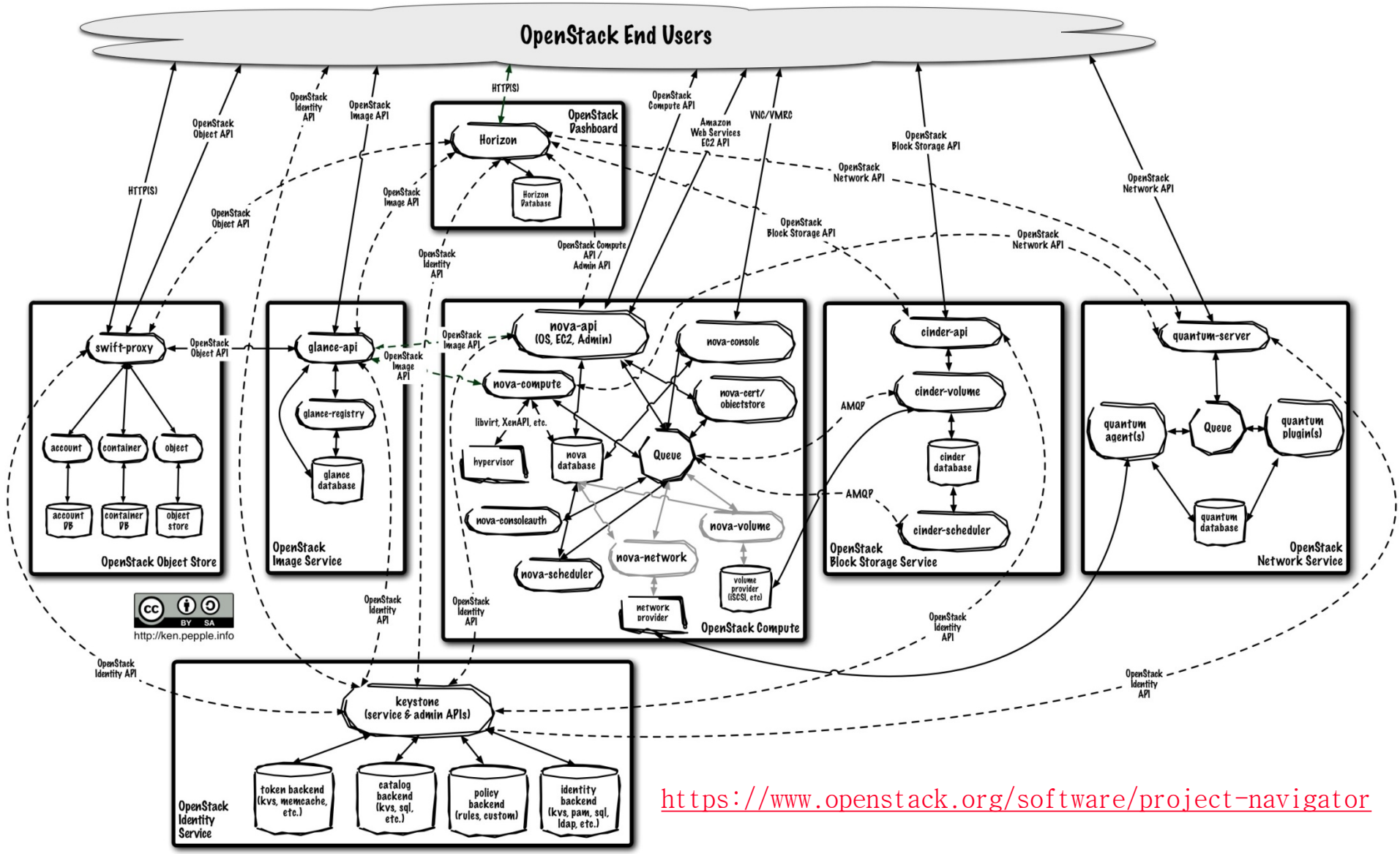


Compute Node Internal

- A Linux bridge is created for each virtual switch. The network traffic of each switch other than the Compute node is separated into VLANs.

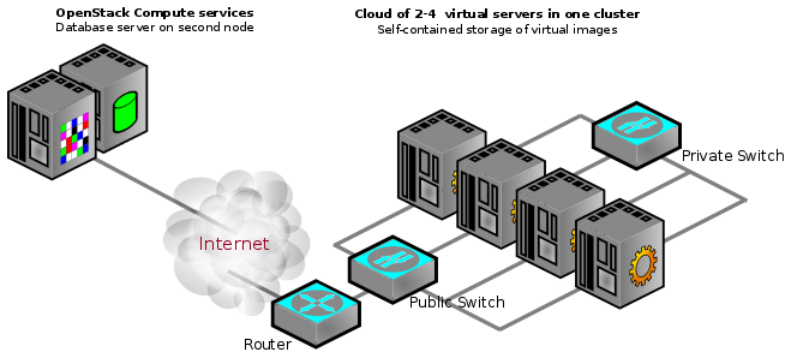


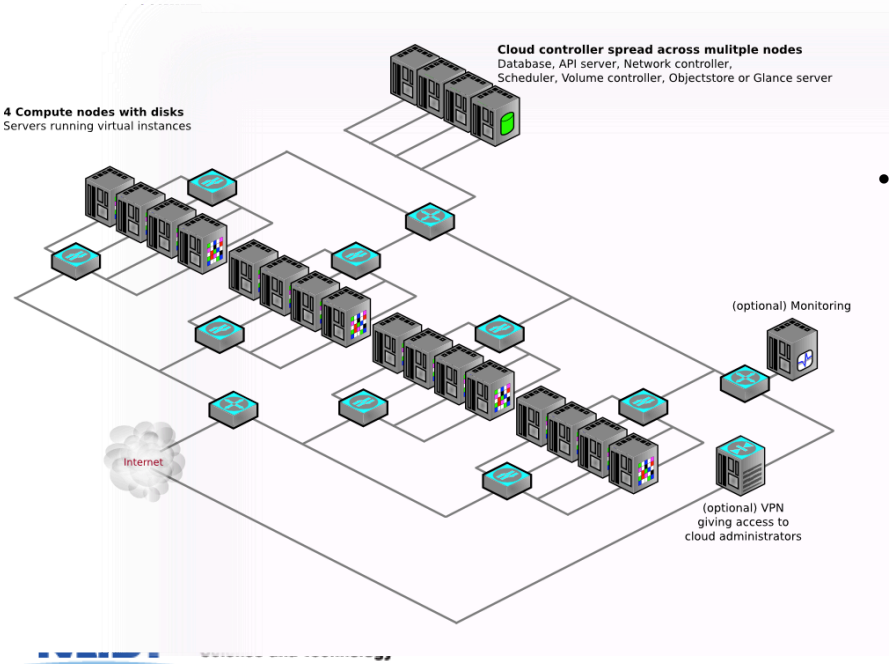
Detail OpenStack Functions



https://www.openstack.org/software/project-navigator

Examples of Installation Architectures



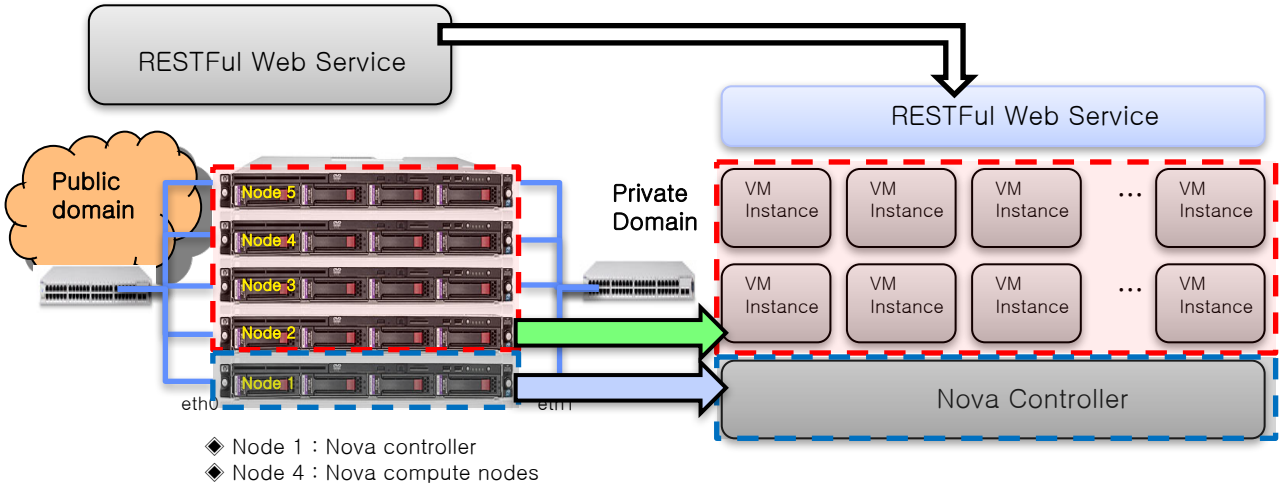


- Single node: Only one server runs all nova-services and also drives all the virtual instances. Use this configuration only for trying out OpenStack Compute, or for development purposes.
- Two nodes: A cloud controller node runs the nova- services except for **nova-compute**, and a compute node runs **nova-compute**. A client computer is likely needed to bundle images and interfacing to the servers, but a client is not required. Use this configuration for proof of concepts or development environments.
- Multiple nodes: You can add more compute nodes to the two node installation by simply installing **nova-compute** on an additional server and copying a `nova.conf` file to the added node. This would result in a multiple node installation. You can also add a volume controller and a network controller as additional nodes in a more complex multiple node installation. A minimum of 4 nodes is best for running multiple virtual instances that require a lot of processing power.

Installation Environment

- Rack Mounted Cluster Machines at E3-2 #3211

	OpenStack Platform
Hypervisor	KVM
H/W Specification	Intel Xeon E5620 2.40GHz, Core 16, MEM 16G, HDD 1T, 5 Node
S/W Specification	OS: Ubuntu 14.04
IP addresses	Eth0 (143.248.152.61~143.248.152.65)



OpenStack Installation

- To install OpenStack in basic way, we have to follow these documentations: https://docs.openstack.org/install-guide/
  - → You should follow about 200 pages instructions and it will be complex and take long-time
  - → So we will use linux shall script based installation supporter DevStack (http://devstack.org/)
- DevStack needs minimum configuration for building OpenStack platform (but this property become weakness in commercialization build)

OpenStack Installation

1. Prepare the **Ubuntu Environment in 16.04.3 LTS Desktop 64 bit Version**
   - You can use VMWare of VirtualBox but not recommended (OpenStack requires minimum 1.5GB memory)
2. Add Stack User
   - You can create a separate stack user to run DevStack with

```
$ sudo useradd -s /bin/bash -d /opt/stack -m stack
```

   - Since this user will be making many changes to your system, it should have sudo privileges:

```
$ echo "stack ALL=(ALL) NOPASSWD: ALL" | sudo tee /etc/sudoers.d/stack
$ sudo su - stack
```

3. Download DevStack

```
$ git clone https://git.openstack.org/openstack-dev/devstack
$ cd devstack
```

OpenStack Installation

4. Create a local.conf file
   - Specification for installation
   - Create local.rc file with following contents

```
[[local|localrc]]
HOST_IP=192.168.122.1
ADMIN_PASSWORD=secret
DATABASE_PASSWORD=$ADMIN_PASSWORD
RABBIT_PASSWORD=$ADMIN_PASSWORD
SERVICE_PASSWORD=$ADMIN_PASSWORD
```

5. Start the devstack installation
   - Installation takes very long time (about 40 minutes)

```
./stack.sh
```

Specific configuration file and shall script files

localrc

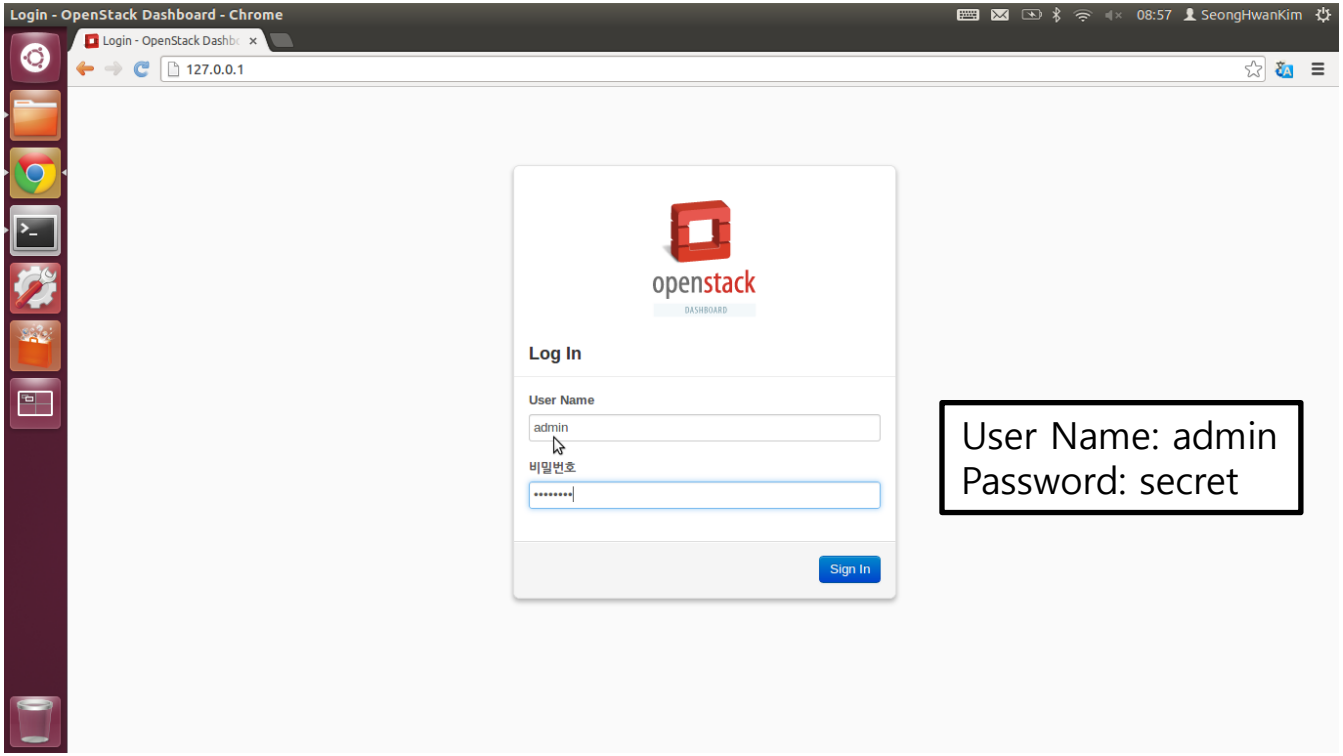
```
# network
FLAT_INTERFACE=br100
FIXED_RANGE=192.168.2.1/24
FIXED_NETWORK_SIZE=256

# vnc
VNCSERVER_LISTEN=0.0.0.0
VNCSERVER_PROXYCLIENT_ADDRESS=$HOST_IP

# logs
DEST=/opt/stack
LOGFILE=$DEST/logs/stack.sh.log
SCREEN_LOGDIR=$DEST/logs/screen

# system password
ADMIN_PASSWORD=ee614soc
MYSQL_PASSWORD=ee614soc
RABBIT_PASSWORD=ee614soc
SERVICE_PASSWORD=ee614soc
SERVICE_TOKEN=socservicetoken

# cinder
VOLUME_GROUP="cinder-volume"
VOLUME_NAME_PREFIX="volume-"
```

```
sudo iptables -A POSTROUTING -t mangle -p udp --dport
    68 -j CHECKSUM --checksum-fill
tools/upload_image.sh
    http://ancl.kaist.ac.kr/data/SOC2013/openstack/SOC.im
    g
```

* When installation meet unexpected error or exit,
  please run /home/stack/devstack/unstack.sh and clean.sh

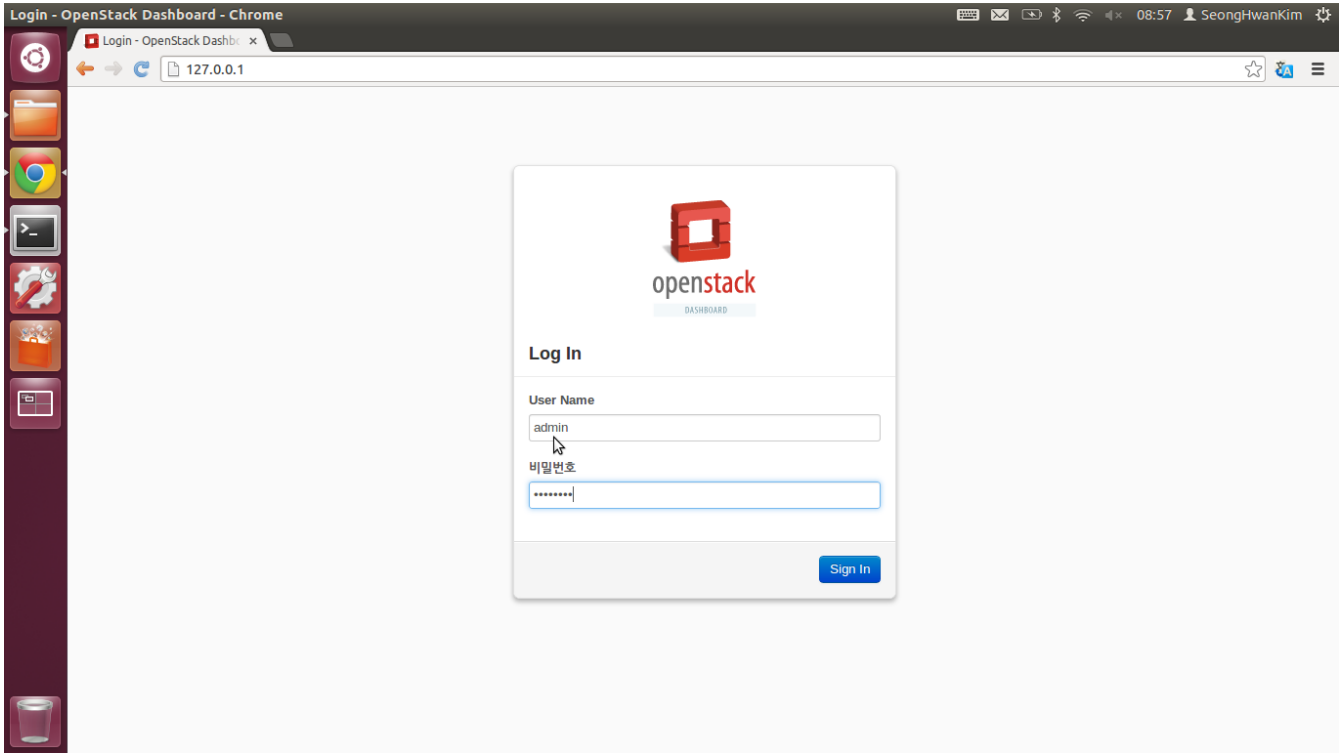
OpenStack Installation Verification

1. Open the Web Browser and access to http://127.0.0.1
   - Authenticate access with user account: 'admin', password: 'secret'



OpenStack Installation Verification

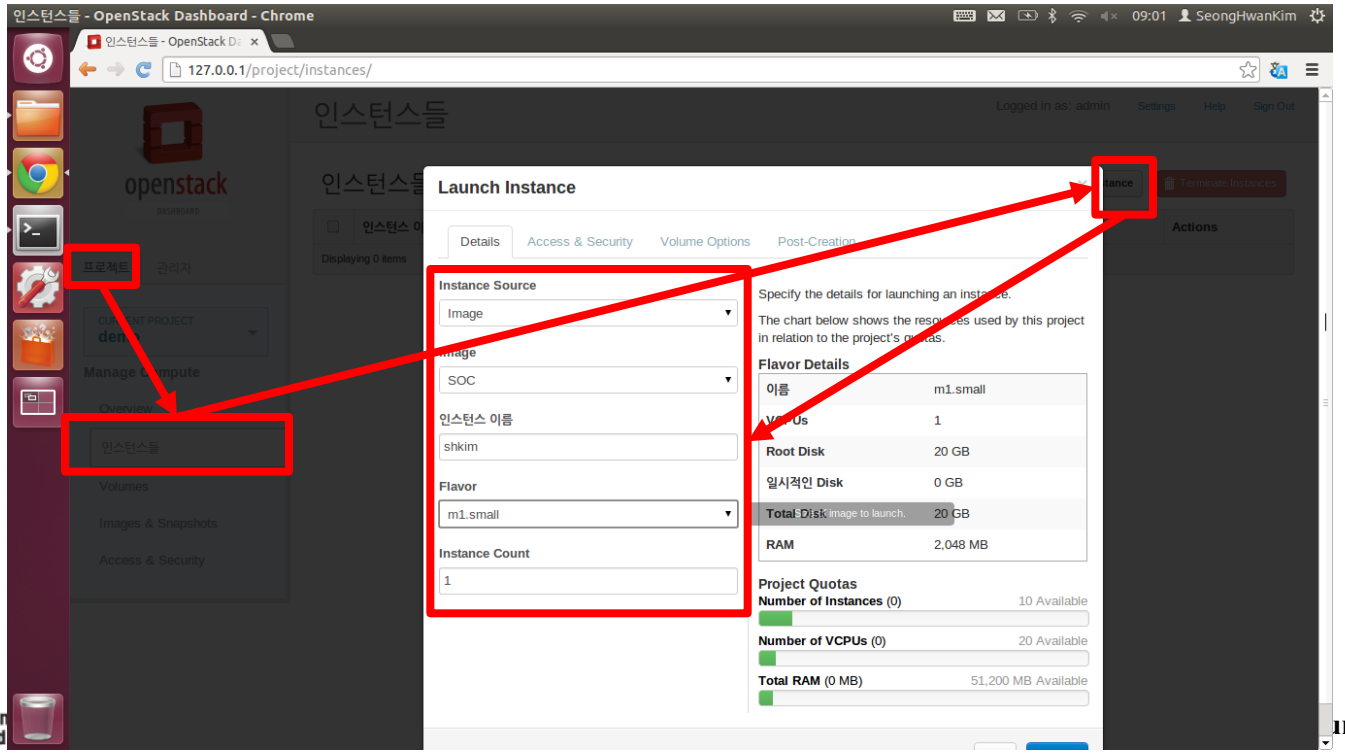
1. Open the Web Browser and access to http://143.248.146.60/horizon



OpenStack Installation Verification

2. Access ‘project→instances→Lanunch Instance’ to create instance

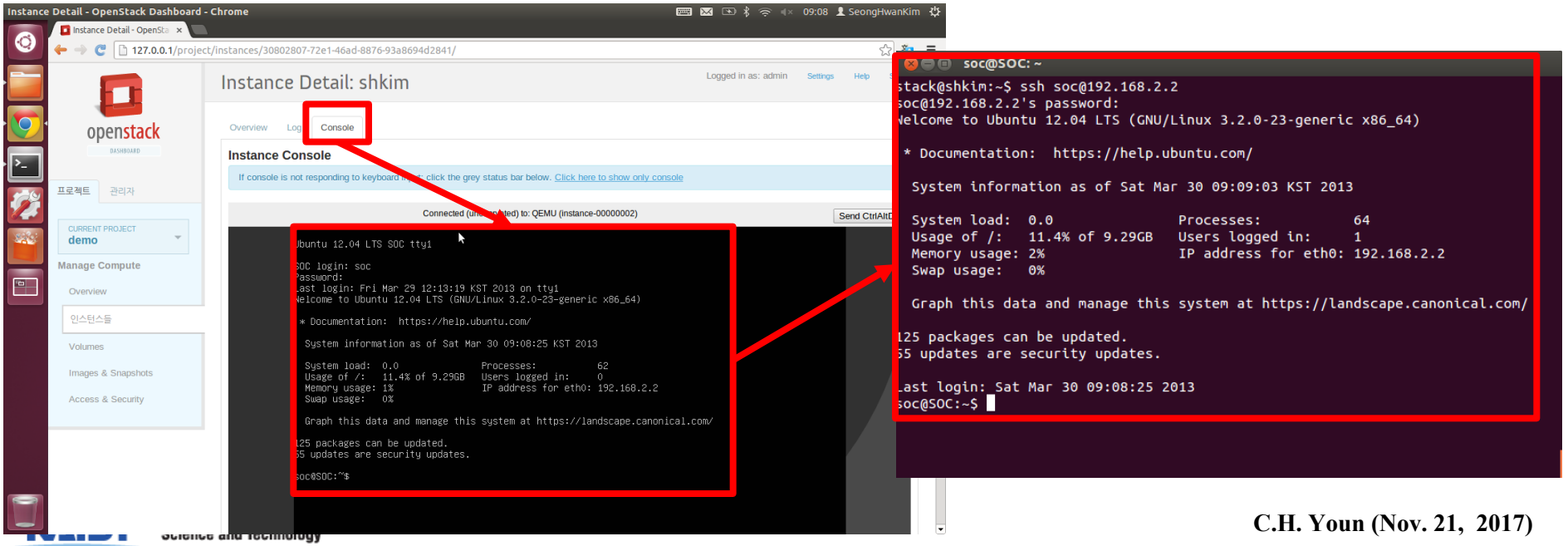
- Create instance with image=‘SOC’, instance name=‘any name’, Flavor=‘m1.small’, instance count=‘1’



OpenStack Installation Verification

3. When instance status becomes to ‘Active’, try to access VM
   - Click the instance name and access to the VM with console label (VNC Console)
   - Open the terminal and access VM with ssh protocol

     → $ ssh soc@192.168.2.2
   - VM User Account: soc, Password: ee614soc



OpenStack Control

- Please watch the movie clip in http://www.openstack.org/software/openstack-dashboard/ and be familiar with OpenStack Platform and Dashboard