# **OpenStack Course Summary**

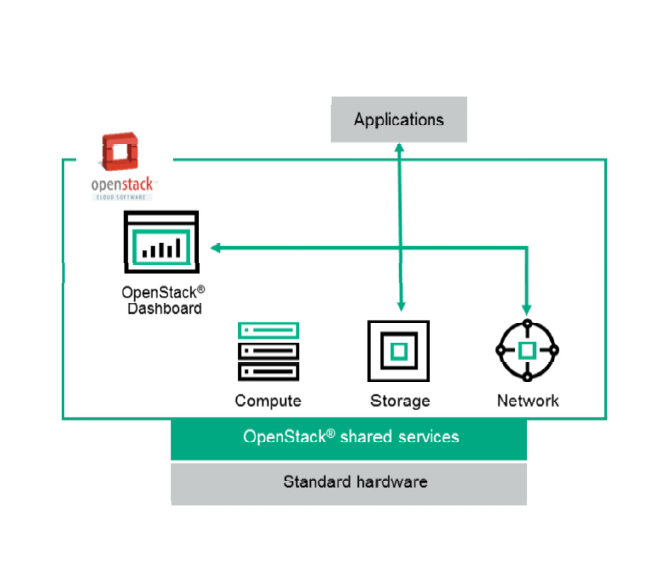
## **Documentation**

* <http://docs.openstack.org>
* <https://docs.openstack.org/ocata/>
* <https://docs.openstack.org/ocata/admin/index.html>
* <https://developer.openstack.org/api-guide/quick-start/>
* <https://www.openstack.org/software/project-navigator/>

OpenStack is a cloud operating system. OS provides pool of compute, networking, and storage resources. OS is an abstraction of the “cloud operating system” that

* Is capable of scaling to thousands of computers.
* Provides a secure remote access to all resources through API.
* Offers a web and CLI UI.

The end user server instances running in the cloud are available over private or public network.

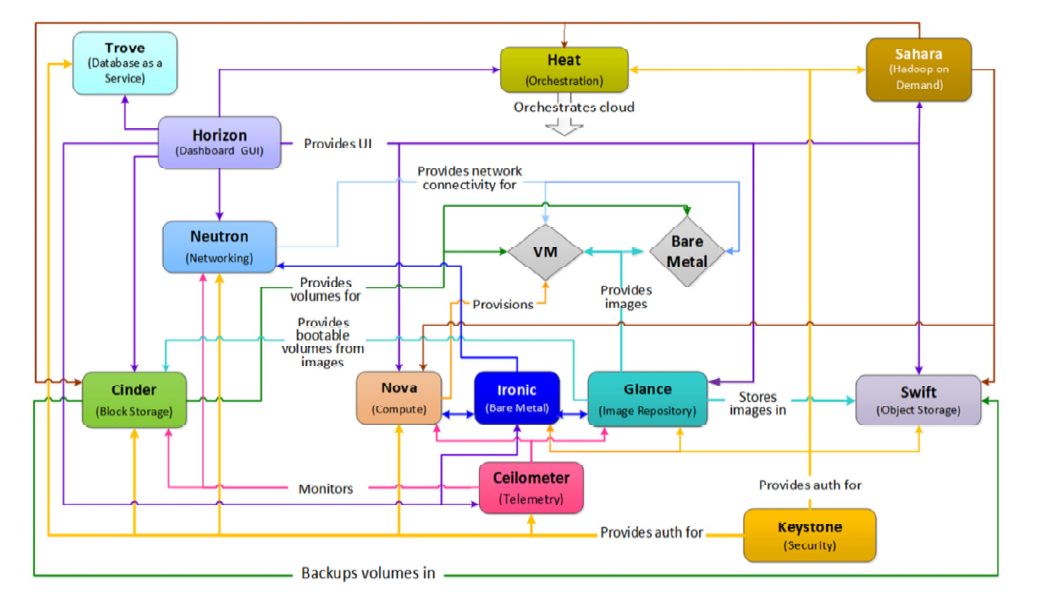


Administrators often deploy OpenStack® Compute using one of multiple supported hypervisors in a virtualized environment.

OpenStack® has support for both Object Storage and Block Storage. Object Storage is ideal for cost effective, scale-out storage. It provides a fully distributed, API-accessible storage platform that can be integrated directly into applications or used for backup, archiving, and data retention. Block Storage allows block devices to be exposed and connected to compute instances for expanded storage, better performance, and integration with enterprise storage platforms.

OpenStack® Networking is a pluggable, scalable, and API-driven system for managing networks and IP addresses. OpenStack® Networking ensures the network will not be the bottleneck or a limiting factor in a cloud deployment, and gives you real self-service, even over your network configurations.

## **OpenStack Architecture**



**Core Services:**

**Swift** provides object storage. It allows you to store or retrieve files (but not mount directories like a fileserver). Several companies provide commercial storage services based on Swift. These include KT, Rackspace (from which Swift originated) and Internap. Swift is also used internally at many large companies to store their data.

**Glance** provides a catalog and repository for virtual disk images. These disk images are most commonly used in OpenStack® Compute. While this service is technically optional, any large cloud requires it.

**Nova** provides virtual servers upon demand. Rackspace and HPE provide commercial compute services built on Nova. Nova is used internally at companies like Mercado Libre and NASA (where it originated).

**Horizon** provides a modular web-based user interface for performing most cloud operations, such as launching an instance, assigning IP addresses, and setting access controls.

**Keystone** provides authentication and authorization for all OpenStack® services. Keystone also provides a service catalog of services within a particular OpenStack® cloud.

**Neutron** provides "network connectivity as a service" between interface devices managed by other OpenStack® services (most likely Nova). Neutron allows you to create your own networks and then attach interfaces to them. OpenStack® Network has a pluggable architecture to support many popular networking vendors and technologies.

**Cinder** provides persistent block storage to guest VMs.

**Ceilometer** provides metering of the various aspects of OpenStack® deployments.

**Heat** is a template-based orchestration solution for provisioning the resources required to deploy a cloud.

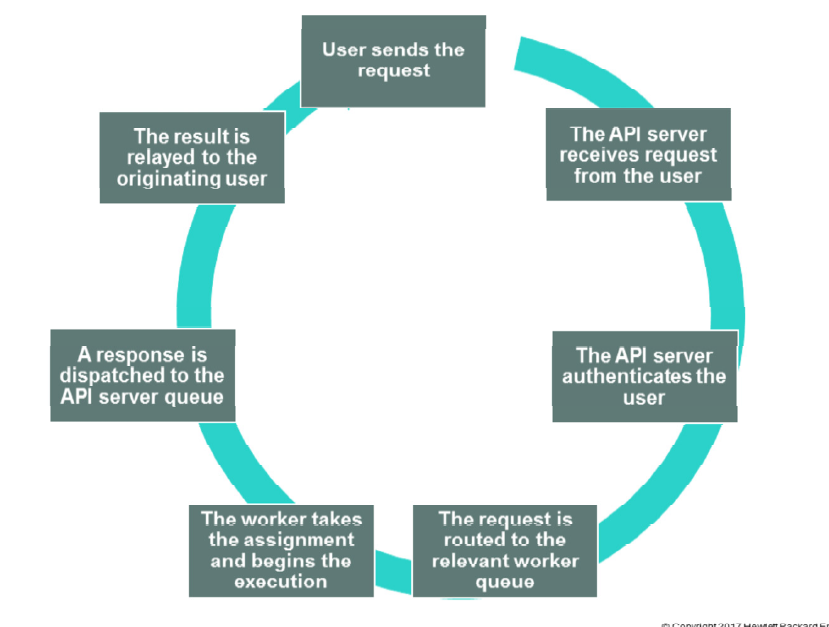
**Trove** is a database service.

End users can interact through a common web interface (Horizon) or directly to each service through their API. All services authenticate through a common source (facilitated through Keystone). Individual services interact with each other through their public APIs (except where the privileged administrator commands are necessary).

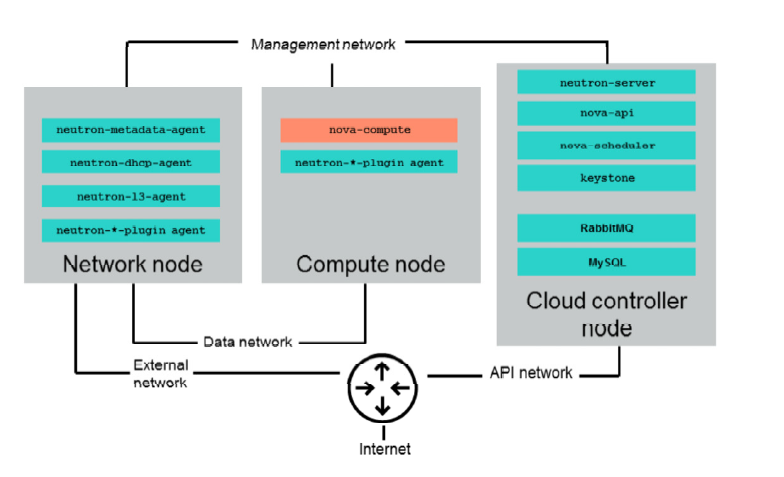
## **OpenStack Service API**

http://developer.openstack.org/api-guide/quick-start

We can communicate with a service using Web GUI (Dashboard), CLI client, REST client, cURL. Python, other bindings. Message lifecycle:



## **OpenStack Installation and Configuration**



Three primary functional components:

The **cloud controller** is a functional unit that provides the OpenStack® Nova, Keystone, Glance, Cinder, Swift, and dashboard services, as well as hosting of the databases and message queue services.

The **network node** generally provides the virtual bridging, Dynamic Host Configuration Protocol (DHCP) server, virtual routing services, and plug-in agents for the associated network hardware.

The **compute node** generally provides the hypervisor, the nova-compute to communicate with the cloud controller node, and the neutron plug-in agent to communicate with the network node.

Also, four types of functional networks are identified:

– Management network: Provides communications between and to the OpenStack® services

– API network: Supports the standard CRUD operations

– Data network: Provides the data stream between the network nodes and the compute nodes

– External network

All of these functional components can exist on a single server, such as the one used for the lab activities for this training, or they can be distributed throughout the cloud across multiple servers.

OpenStack® architecture allows for wide range of configuration options.

**General purpose:** As a baseline product, general purpose clouds do not provide optimized performance for any particular function.

**Compute-focused:** Configuration that is primarily driven by the type of instance workloads you provision in the OpenStack® cloud.

**Storage-focused:** Addresses a range of storage management-related considerations, as well as storage focused input-output performance, security, and scalability requirements.

**Network-focused:** A configuration model that takes in consideration the content delivered over the network, management functions, service offerings and connected web portals, network high availability, and so on.

**Multi-site: A** configuration model that provides the OpenStack® capable of running in a multi-region configuration.

**Hybrid:** Configuration design providing the OpenStack® that uses more than one cloud. Hybrid clouds are designs that use, for example, both an OpenStack®-based private cloud and an OpenStack®-based public cloud, or an OpenStack® cloud and a non-OpenStack® cloud.

**Massively scalable:** A massively scalable architecture is a cloud implementation that supports a very large deployment. An example is an infrastructure in which requests to service 500 or more instances at a time.

**Specialized cases:** This includes specialized networking and software-defined networking configurations,

Desktop-as-a-Service and OpenStack®-on-OpenStack® deployments, as well as the deployments that support specialized hardware.

DevStack is a script used to quickly create an OpenStack® development environment

## **Keystone Identity Service**

Keystone has 2 primary responsibilities, identity management and service catalog management.

Identity management:

– Tracks users and their permissions

– Provides authentication and authorization for all of the OpenStack® services

– Supports the creation and maintenance of OpenStack® users, projects, and roles

Keystone:

– Can serve as the common authentication system across the cloud

– Can be integrated with existing backend directory services like LDAP

– Supports multiple forms of authentication

– Supports standard username and password credentials and token-based authentication

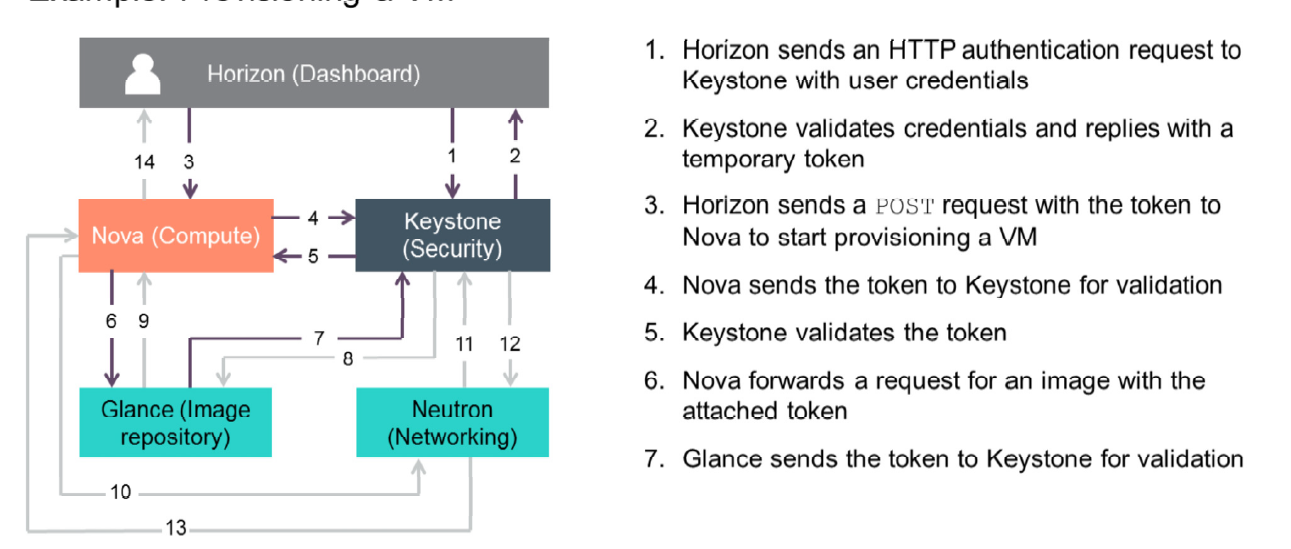
In service catalog management, Keystone:

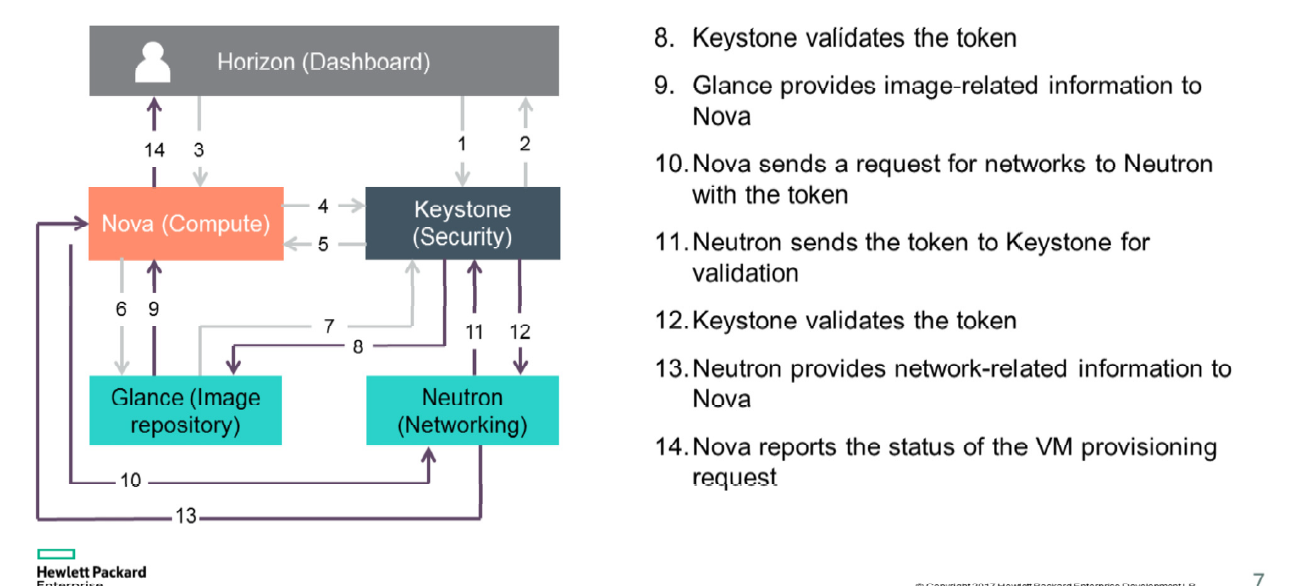
– Provides catalog of available services with the API endpoints

– Maintains a user that corresponds to each service

– Supports the use of a special service project called “service”

**Keystone Identity Authentication/Authorization**





Keystone identity-related objects:

A user is someone or something that can gain access through Keystone.

A group is a collection of users. Roles assigned to the group are automatically associated with all users of that group.

A project represents what is called the tenant in Nova originally, meaning something that aggregates the number of resources in each service.

A role represents the number of privileges or rights a user has or actions they are allowed to perform.

A domain defines administrative boundaries for the management of Keystone entities.

A region is a discrete OpenStack® environment with dedicated API endpoints that typically shares only Identity (Keystone) with other regions.

Role-based Access Control (RBAC) provides a predefined list of actions that the user can perform, such as start or stop VMs, reset passwords, and so on. RBAC is supported in both Identity and Compute and can be configured using the dashboard.

A token is an alpha-numeric string of text used to access the OpenStack® APIs and resources. Tokens can be scoped in order to describe the resources that are accessible. They are valid for limited duration, and can be revoked.

An endpoint is a network-accessible address, described by the URL, where a service might be accessed. Service catalog is the endpoint registry used for service endpoint discovery.

policy.json file is a configuration file located in the /etc/keystone/ folder that is used to set role-based policies for using the service-specific functions.

### **Common Keystone Management Tasks: CLIs**

**Managing projects:**

openstack project list/create/delete/set show

Managing users:

openstack user create/delete/list/role list/set/show

**Managing roles:**

openstack role add/create/delete/list/remove/show

**Creating a role, project, user, and adding project user to role:**

openstack role create testrole

openstack user create testuser => user\_id

openstack project create testproject =>project\_id

openstack role add –project project\_id –user user\_id

**Managing the service catalog:**

openstack service create/delete/list/show/create/delete/list/show

**To access a service, you have to know its endpoint:**

– publicurl is intended to be accessible from the global world

– internalurl can be used to access a service from a local network

– adminurl is used in case admin access to a service is separated from the common access

openstack endpoint list

The /etc/[SERVICE\_CODENAME]/policy.json file controls what users are allowed to do for a given service.

**Keystone config files:**

/etc/keystone/keystone.conf: general and driver specific config parameters

/etc/keystone/keystone-paste.conf: configuration for python WSGI-based applications

## **Glance Image Service**

Glance is an OpenStack® project with the primary function of providing discovery registration and delivery services for disk and server images. Glance enables you to choose available images or create your own from existing servers. Also, you can store images that can be used as templates to get new instances up and running quickly and consistently. Glance supports the storage and cataloging of snapshots, enabling virtual machines (VMs) to be backed up quickly. Glance is NOT used to store images, but is used to stream images between OpenStack and systems outside of OpenStack and is used to cache images.

Communication is enabled through its restful supported API.

### **Common Glance Management Tasks**

**Glance config files:**

* glance-api.conf
* glance-api-paste.ini
* glance-cache.conf
* glance-registry.conf
* glance-registry-paste.conf
* policy.json

**Common CLI commands:**

openstack image add project\_id

openstack image create/delete

openstack image list/save/show/update

openstack image remove project\_id

**Create image:**

openstack image create –name myFirstImage –is-public true –container-format bare-disk-format qcow2 < cirros-0.3.1-x86\_64-disk.img

**Glance log files:**

/var/log/glance/api.log

/var/log/glance/registry.log

# **Neutron Networking Service**

OpenStack® uses virtual networking to connect various components (for example, compute and storage node). Virtual networking works similarly to standard networking, except that the networking functions are performed by software on compute systems instead of dedicated network devices. OpenStack® does not reinvent networking; it simply provides the ability to use the existing open and vendor-specific virtual networking technology.

Another key component of virtualized networking is a virtual Layer 2 (L2) switch, which is configured from the VM host device’s OS.

**Virtual L2 switch:**

– Operates much like its physical counterpart and learns which VM MAC addresses are connected to each of its ports, which enables it to forward packets received on one of its ports to the port associated with the destination MAC.

– Is used to interconnect all VMs on the same subnet, whether the VMs existing on the same VM host device, other VM host device, or through a physical switch to baremetal machines, as shown with compute nodes A and B in the screenshot.

– Can be used to modify packets received from or going to an attached node in support of VLANs or tunnel transport.

**Virtual Local Area Networks:**

Virtual Local Area Networks (VLANs) are used to define a Layer 2 domain where only packets belonging to a specific VLAN can communicate with other networked devices belonging to that same VLAN ID. A VLAN field can be inserted into aמ Ethernet packet to identify it as belonging to a specific VLAN as shown in the diagram. Up to 4096 VLANs are supported on a network. VLANS are very useful in isolating Client A data packets from Client B’s data.

In the diagram on the above slide:

– All traffic in VLAN ID 10 is isolated from traffic in VLAN ID 20, so VM3 in hosting device A can only communicate with the bare-metal machine, VM2 in hosting device B.

– A customer assigned to VLAN 10 would only have access to networked devices that belong to VLAN 10.

– VMs on the same VM hosting device can belong to different VLANs, so VMs on the same hosting device can be isolated from one another using VLANs.

**Virtual overlay networks:**

Virtual overlay networks are another way to isolate data traffic between specific network nodes. Virtual overlay networks:

– Are a type of network virtualization that uses tunneling protocols to extend isolated network segments between servers for multi-tenant data center networks.

– Encapsulate a data packet inside of another packet which is then forwarded to an endpoint where it is de-encapsulated.

Virtual overlay networks supported by OpenStack® Neutron are:

– GRE (Generic Routing Encapsulation)

– VxLAN (Virtual Extensible LAN)

The benefits of virtual overlay networks include:

– Traffic isolation for multi-tenancy: Cloud service providers can use traffic isolation features to securely offer services to multiple customers; large enterprises can isolate business unit traffic or specific traffic types (for example, R&D network) from production traffic.

– Ease of VM provisioning: IT managers have the freedom to migrate VMs to new locations without worrying about attributes (or limitations) of the physical network. All the higher level elements of the network, including policy, security and VLANs, migrate with the VMs.

– Scalability: IT managers can scale their data center networks beyond the 4096 VLAN limit.

– Physical network independence: Overlay networks enable you to migrate to low-cost data center switches.

– Simplified connection to OpenStack®

**Floating IP address:**

Most virtual networks support two types of IP addresses to be assigned: private and public. Private IP addresses are assigned automatically by an internal DHCP service on instance boot. Instances in the same broadcast domain communicate using the private IP addresses by using the L2 networking service. A private IP address is used for communication with an external network, but it cannot be used for inbound connections. This setup is similar to the setup of a computer behind NAT or firewall in the physical world; the computer has the private IP and can communicate with external networks, but outside clients cannot initiate a connection with the computer behind the firewall.

To make an instance visible from the data center network, the administrator has to assign a Floating IP address from the predefined pool of Floating IPs. When assigned, the public IP address remains connected to the instance for as long as the instance exists. When you terminate the instance, the Floating IP address is released and returned to the pool of available IP addresses.

The delivery of packets to the interface with an assigned floating address is the responsibility of the virtual router.

**Open Virtual Switch (OVS) components—Compute host**

Open vSwitch consists of bridges and ports. Ports represent connections to other things, such as physical interfaces and patch cables. Packets from any given port on a bridge are shared with all other ports on that bridge. Bridges can be connected through Open vSwitch virtual patch cables or through Linux virtual Ethernet cables (veth). Additionally, bridges look like network interfaces to Linux, so they can be assigned IP addresses.

Open vSwitch uses several main bridges. The integration bridge, called “br-int,” connects directly to the VMs and associated services. The external bridge, called “br-ex,” connects to the external network. The VLAN configuration of the Open vSwitch plug-in uses bridges associated with each physical network.

In addition to defining bridges, Open vSwitch uses OpenFlow, which enables you to define networking flow rules. These rules are used in certain configurations to transfer packets between VLANs. The most common security group driver used with Open vSwitch is the Hybrid iptables/Open vSwitch plug-in. It uses a combination for iptables and OpenFlow rules. iptables is a tool used for creating firewalls and setting up NATs on Linux. It uses a complex rule system and “chains” of rules to allow for the complex rules required by the Neutron security groups.

In the example, four types of virtual networking devices are shown: test access point (TAP) devices, veth pairs, Linux bridges, and Open vSwitch bridges. For an Ethernet frame to travel from eth0 of virtual machine vm01, to the physical network, it must pass through nine devices in the host: TAP vnet0 → Linux bridge qbrXXX → veth pair (qvbXXX, qvoXXX) → Open vSwitch bridge br-int → veth pair (int-br-eth1, phy-br-eth1) → physical network interface card eth1.

Hypervisors such as KVM and Xen implement a virtual network interface card (typically called a VIF or vNIC) through a TAP device such as vnet0. An Ethernet frame sent to a TAP device is received by the guest operating system. A veth pair is a pair of directly connected virtual network interfaces. An Ethernet frame sent to one end of a veth pair is received by the other end of a veth pair. OpenStack® networking uses veth pairs as virtual patch cables to make connections between virtual bridges.

A Linux bridge behaves like a hub where you can connect multiple (physical or virtual) network interface devices to a Linux bridge. Any Ethernet frame that comes in from one interface attached to the bridge is transmitted to all other devices. An Open vSwitch bridge behaves like a virtual switch. The network interface devices connect to ports on the Open vSwitch bridge, and these ports can be configured similarly to the ports on a physical switch, including VLAN configurations.

The br-int Open vSwitch bridge is the integration bridge. All guests running on the compute host connect to this bridge. OpenStack® Networking implements isolation across these guests by configuring the br-int ports. The br-eth1 bridge provides connectivity to the physical NIC, eth1. It connects to the integration bridge through a veth pair: int-br-eth1 and phy-br-eth1. In the VLAN translation example, net01 and net02 have VLAN IDs of 1 and 2, respectively. However, the physical network in the example only supports VLAN IDs in the range of 101 through 110. The Open vSwitch agent configures flow rules on br-int and br-eth1 to enable the VLAN translation. When br-eth1 receives a frame marked with VLAN ID 1 on the port associated with phy-br-eth1, it modifies the VLAN ID in the frame to 101. Similarly, when br-int receives a frame marked with VLAN ID 101 on the port associated with int-br-eth1, it modifies the VLAN ID in the frame to 1.

**Security groups: iptables and Linux bridges**

Ideally, the TAP device vnet0 would be connected directly to the integration bridge—br-int. Unfortunately, this is not possible because of how OpenStack security groups are currently implemented. OpenStack uses iptables rules on the TAP devices, such as vnet0 to implement security groups, and Open vSwitch is not compatible with the iptables rules that are applied directly on TAP devices that are connected to an Open vSwitch port.

The OpenStack Networking service uses an extra Linux bridge and a veth pair as a workaround for this issue. Instead of connecting vnet0 to an Open vSwitch bridge, it is connected to a Linux bridge, qbrXXX. This bridge is connected to the integration bridge, br-int, by means of the veth pair qvbXXX and qvoXXX.

**Open Virtual Switch (OVS) components—Network host configuration**

As on the compute host, there is an Open vSwitch integration bridge (br-int) and an Open vSwitch bridge connected to the data network (br-eth1), and the two are connected by a veth pair. The neutron-openvswitch-plugin-agent configures the ports on both switches to do VLAN translation.

An additional Open vSwitch bridge, br-ex, connects to the physical interface that is connected to the external network. In this example, that physical interface is eth0. Although the integration bridge and the external bridge are connected by a veth pair (int-br-ex, phy-br-ex), this example uses Layer 3 connectivity to route packets from the internal networks to the public network; no packets traverse that veth pair in this example.

**Open vSwitch internal ports**

The network host uses Open vSwitch internal ports. Internal ports enable you to assign one or more IP addresses to an Open vSwitch bridge. In the example on the picture above, the br-int bridge had four internal ports: tapXXX, qr-YYY, qr-ZZZ, and tapWWW. Each internal port had a separate IP address associated with it. An internal port, qg-VVV, was on the br-ex bridge.

**DHCP agent**

By default, the OpenStack® Networking DHCP agent uses a program called dnsmasq to provide DHCP services to guests. The OpenStack® Networking service must create an internal port for each network that requires DHCP services and attach a dnsmasq process to that port. In the previous example, the interface tapXXX was on subnet net01\_subnet01, and the interface tapWWW was on net02\_subnet01.

**L3 agent (routing)**

The OpenStack® Networking L3 agent implements routing through the use of Open vSwitch internal ports and relies on the network host to route the packets across the interfaces. In this example:

– Interface qr-YYY, which is on subnet net01\_subnet01, has an IP address of 192.168.101.1/24.

– Interface qr-ZZZ, which is on subnet net02\_subnet01, has an IP address of 192.168.102.1/24.

– Interface qg-VVV, which has an IP address of 10.64.201.254/24.

Because all of these interfaces are visible to the network host operating system, they will route the packets appropriately across the interfaces, as long as an administrator has enabled IP forwarding.

The L3 agent uses iptables to implement Floating IP addresses to do the network address translation.

**Neutron functionality**

Neutron is the OpenStack® service that is used to interconnect the OpenStack® components within an Infrastructure as a Service (IaaS) environment, as well as the IaaS environment to the public network.

Neutron:

– Provides an API to define networks, subnets, and network connectivity used by other OpenStack services, in particular Nova compute instances

– Uses SDN (Software Defined Networking), a fundamental shift in how data center networks are defined, provisioned, and consumed

– Abstracts the physical network implementations through the use of industry standard and vendor-specific network resource plug-ins

– Enables projects to have multiple private networks, and allows them to choose their own IP addressing scheme and routing topology

– Supports virtual network services, including services such as firewalls, load balancers, and VPNs

**Neutron networking terminology**

Neutron uses the following terminology for networking components:

– Compute node—An OpenStack® server that runs user workloads in VMs, bare metal machine, or containers

– Layer2 interconnect—Bridge/Layer 2 switch used to provide layer 1 networking connections between end points, possibly filtering at layer2

– Network node—An OpenStack® infrastructure server that connects project networks to infrastructure networks

– OSI Layer 2—The layer just above physical connections (Layer 1) that manages traffic between servers and provides a logical separation of traffic

– Project (formerly tenant)—The owner of compute instances and their networks providing logical isolation (many project models require network traffic from one project to be invisible to others)

– Provider network—A term used to refer to the cloud-provided external network that tenants use to reach the outside

– Project network—A term used to refer to networks created and internally used by a project

– VLAN (Virtual Local Area Network)—Switch-enforced isolation zones created by adding 1 of 4096 tags in the network traffic (also known as tagged traffic)

– VXLAN (Virtual Extensible LAN)—A network virtualization technology that uses VLAN-like encapsulation of layer 2 Ethernet frames within layer 4 UDP packets (UDP port 4789 IANA-assigned destination number)

**Neutron components—Neutron plug-ins**

Neutron lets you use a set of different backends called "plug-ins" that work with a growing variety of networking technologies. Some OpenStack® Networking plugins can use basic Linux VLANs and iptables, while others can use more advanced technologies, such as L2-in-L3 tunneling or OpenFlow, to provide similar benefits.

These plugins may be distributed as part of the main Neutron release, or separately. Plug-ins can have different properties for hardware requirements, features, performance, scale, or operator tools.

Because Networking supports a large number of plug-ins, the cloud administrator can weigh options to decide on the right networking technology for the deployment.

**OpenStack® node networks**

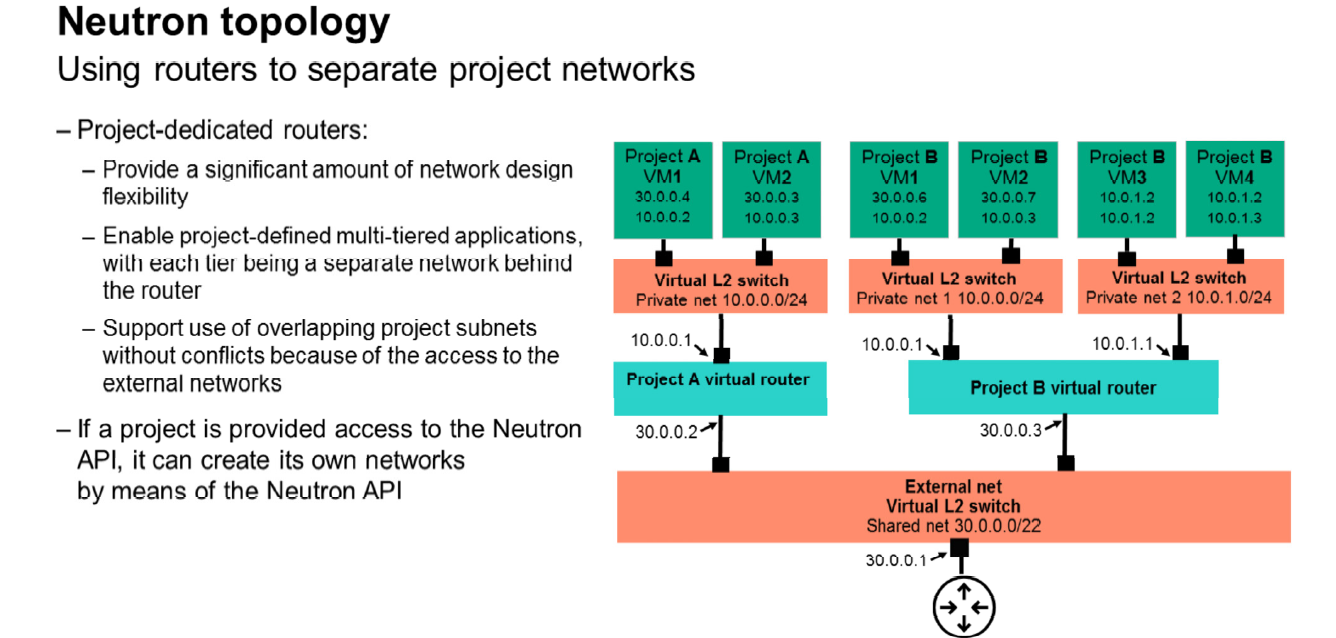
A standard OpenStack® Networking setup has at least four distinct physical data center networks (there can be others).

**Cloud management network** is used for internal communication between OpenStack® components. The IP addresses on this network should be reachable only within the data center.

**Cloud data network** is used for VM data communication within the cloud deployment. The IP addressing requirements of this network depend on the OpenStack® Networking plug-in in use.

**External network** provides VMs with Internet access in some deployment scenarios. The IP addresses on this network should be reachable by anyone on the Internet.

**API access network** exposes all OpenStack® APIs (including the OpenStack® Networking API) to tenants. The IP addresses on this network should be reachable by anyone on the Internet.



**Common CLI management tasks**

Use the following commands to perform common management tasks:

– To create a network, enter $ neutron net-create net1

– To create a subnet, enter $ neutron subnet-create net1 192.168.2.0/24 --name subnet1

– To create a port and let Neutron select the IP address, enter: $ neutron port-create net1

– To create a port and specify an IP address, enter: $ neutron port-create net1 --fixed-ip ip\_address=192.168.2.40

**Neutron Log Files**

Neutron: /var/log/neutron

Messaging: /var/log/rabbitmq

Plugin log file: /var/log/openswitch

# **Nova Compute Service**

The Nova service is the heart of OpenStack®. It controls the Infrastructure as a Service (IaaS) cloud computing platform by:

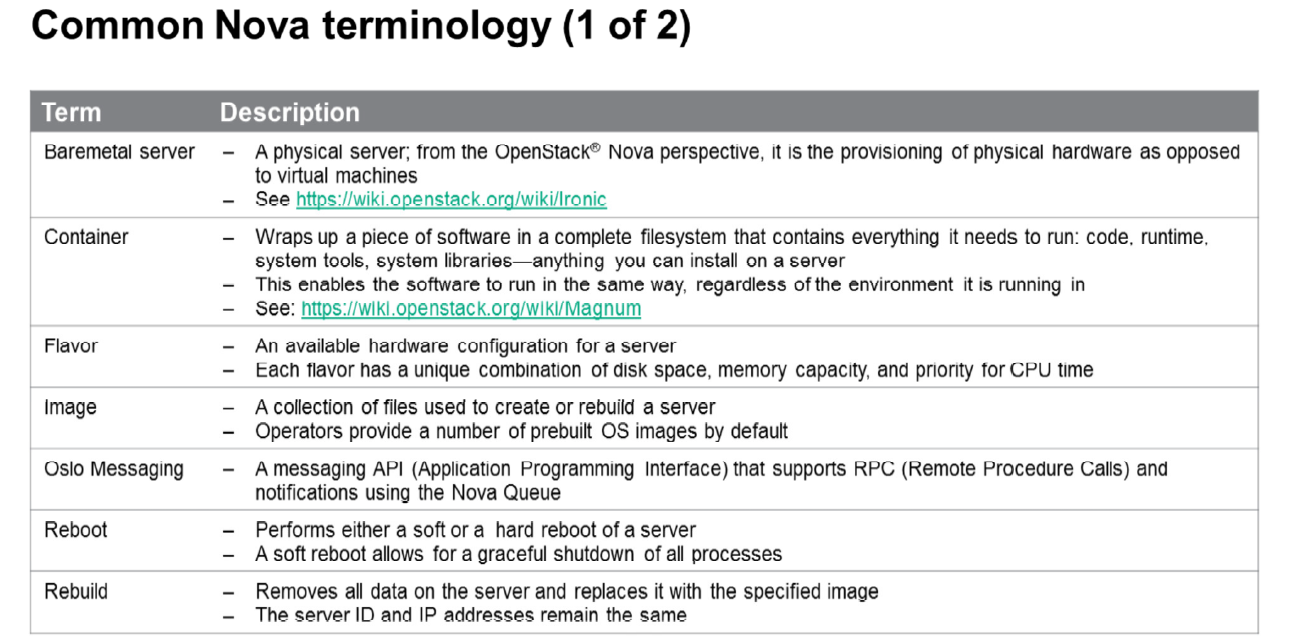
– Providing a standard Representational State Transfer (REST) interface from which VMs can be managed

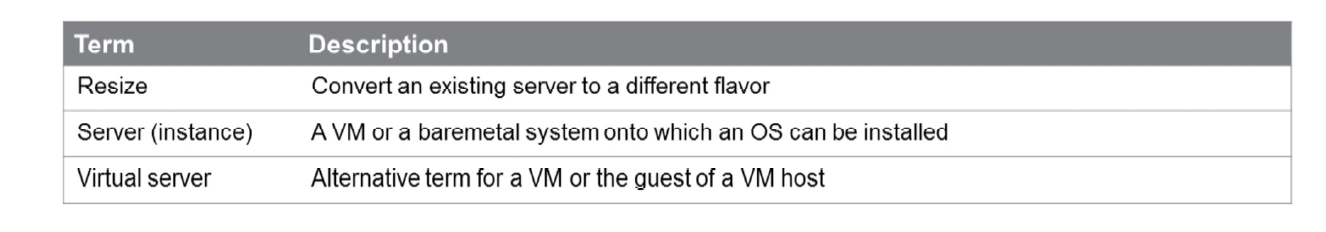
– Supporting multiple types of hypervisors

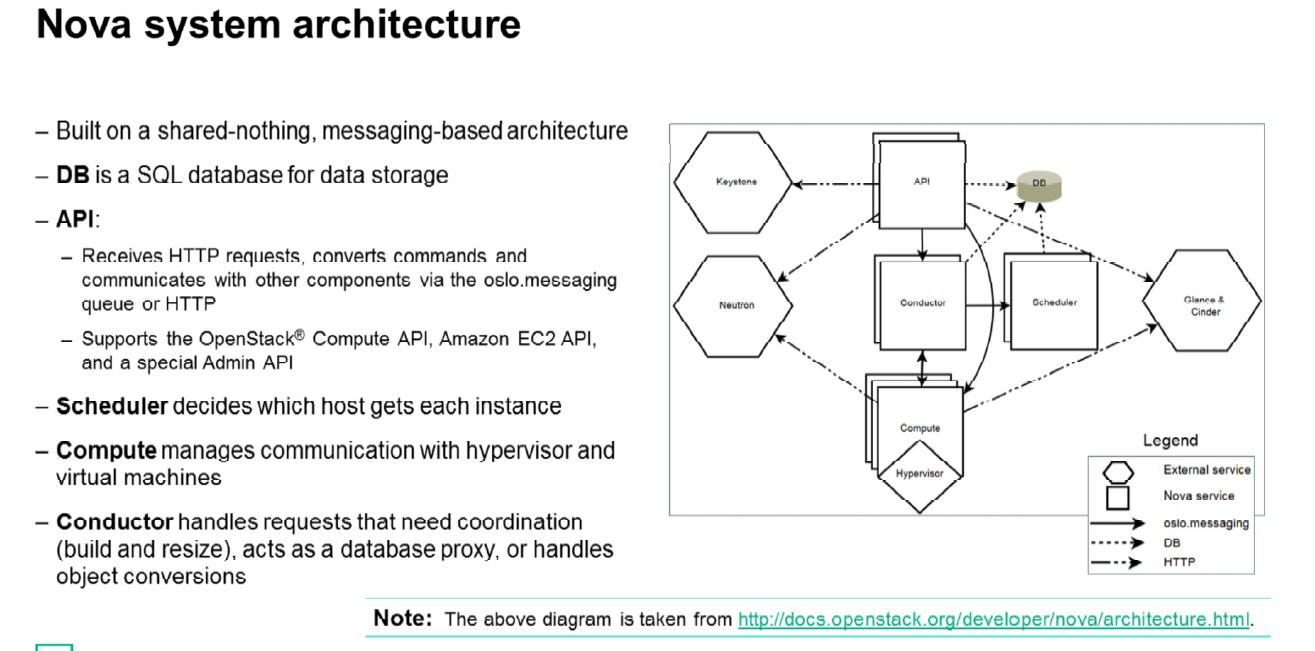
– Supporting bare-metal instances

– Starting and managing instances

– Supporting single and multi-tenancy



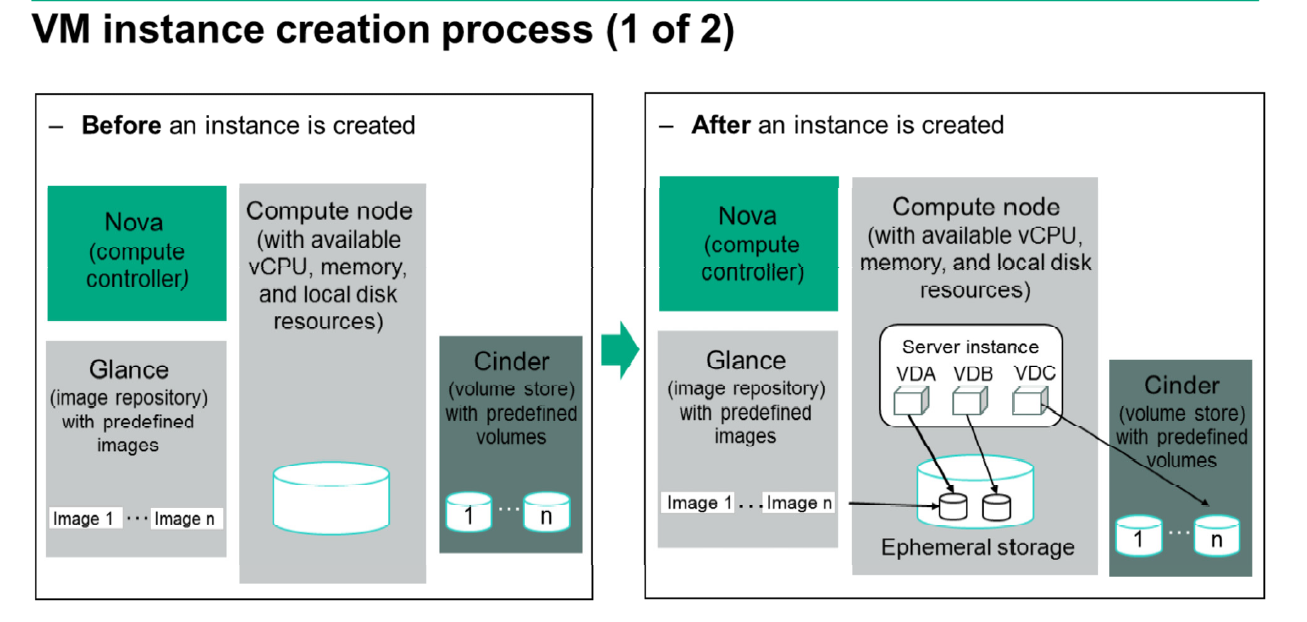


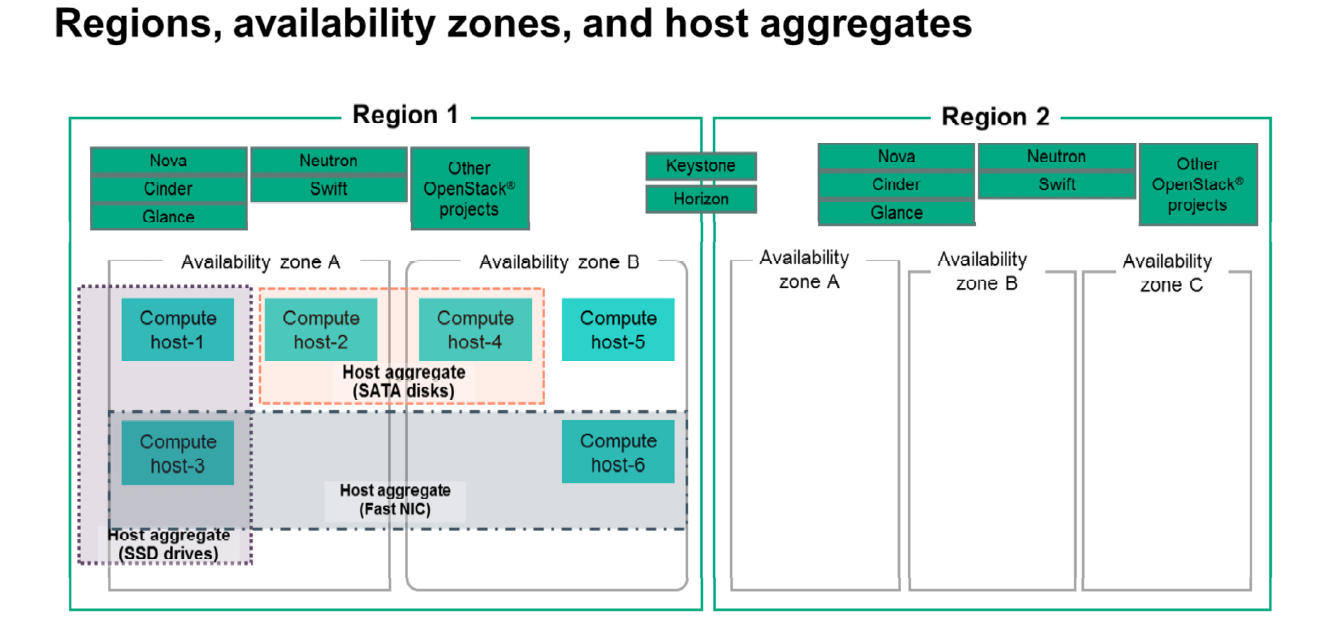


Nova comprises multiple server processes, each performing different functions. The user-facing interface is a REST API, while internally, Nova components communicate via a Remote Procedure Call (RPC) message passing mechanism.

The API servers process the REST requests, which typically involve database reads and writes, optionally sending RPC messages to other Nova services and generating responses to the REST calls. RPC messaging is done via the oslo.messaging library, an abstraction on top of message queues. Most of major Nova components can be run on multiple servers and have a manager that is listening for RPC messages. The one major exception is nova-compute, where a single process runs on the hypervisor it is managing (except when using the VMware or Ironic drivers). The manager also has periodic tasks.

Nova also uses a central database that is (logically) shared among all components. However, to aid upgrade, the database is accessed through an object layer that ensures that an upgraded control plane can still communicate with a nova-compute running the previous release. To make this possible, nova-compute proxies database requests over RPC to a central manager called “nova-conductor.”





**Cells—API cell, child cells, and grandchild cells**

The cell functionality enables you to scale an OpenStack® Compute cloud in a more distributed fashion without having to use complicated technologies like database and message queue clustering. It supports very large deployments.

OpenStack® cells essentially provide the means to create logical fences around OpenStack® resources (compute and storage) in the same manner Swift and Keystone do via regions. Cells are an imaginary line that collects resources together. When coupled with policies that are specific to each region, cells create a cloud environment that can treat some resources differently than others.

Cells are configured as a tree. The top-level cell should have a host that runs a nova-api service, but no nova-compute services. Each child cell should run all of the typical nova-\* services in a regular Compute cloud, except for nova-api. You can think of cells as a normal Compute deployment in that each cell has its own database server and message queue broker.

The nova-cells service handles communication between cells and selects cells for new instances. This service is required for every cell. Communication between cells is pluggable, and currently the only option is communication through RPC.

Cells scheduling is separate from host scheduling. Nova-cells first selects a cell. Once a cell is selected and the new build request reaches its nova-cells service, it is sent over to the host scheduler in that cell and the build proceeds, as it would have without cells.

**Nova configuration files**

The main Nova configuration file is located at /etc/nova/nova.conf. The nova.conf configuration file is an INI file format, and it provides settings for: Drivers, Networking, Service endpoints, Message Q, VNC, Notification, Instance usage auditing, Logging, Volume management, Scheduler.

You can use a particular configuration option file by using the option (nova.conf) parameter when you run one of the nova-\* services. This parameter inserts configuration option definitions from the specified configuration file name, which might be useful for debugging or performance tuning.

**Create an Instance**

Openstack server create –flavor [flavorId] –key-name [key] –image [imageId] –nic net-id=[subnet-id][instance name]

openstack server list

openstack server show [server id]

openstack keypair list

ssh-keygen

nova keypair-add --pub\_key id\_rsa.pub testkey

List all of the instances on the affected node: # nova list --host <compute\_host> --all-tenants

If you are using shared storage, migrate all of the instances to the other compute node: # nova live-migration <vm\_uuid> <backup\_compute\_host>

If you are not using shared storage, enter the following syntax to migrate the instances: # nova liva-migration --block-migrate <vm\_uuid> <backup\_compute\_host>

Stop or start a compute instance: stop nova-compute or # start nova-compute

Check if Nova is connected to the AMPQ service after a reboot: grep AMPQ /var/log/nova/nova-compute

Reboot instances that are missing the libvirt XML file (/etc/libvirt/qemu/instance-xxxxxxxx.xml): nova reboot --hard <vm\_uuid>

Reboot instances: nova reboot <vm\_uuid>

**Troubleshooting Nova—Common issues**

Some common issues and resolutions thereof include:

– Service endpoints and associated users were not created properly—The Keystone integration is incorrect or the Glance endpoint is not set up properly.

– The hypervisor is not set up correctly—Use virsh tools to ensure that the hypervisor is set up correctly using bridge networking.

– MySQL database parameters are incorrect.

– IP addresses that are provided by the network admin (public and private) are incorrect.

– DNSmasq is not running.

– The Nova CLI is not responding—The API is probably down.

– An instance gets stuck in a scheduling state—The compute is probably down.

– An instance gets stuck in a networking state—Check if the network is down.

– An instance has an error in the spawning state—Check if libvirt is down.

– You cannot use SSH with a public IP or private IP:

– Ensure that the addresses are legitimate and have routes.

– Enter iptables –list.

– The console log of instances provides details with regard to networking.

– Ensure that the stats for queues are legitimate: rabbitmqctl list\_queues ; rabbitmqctl

list\_consumers

– nova-manage service list shows ☺ for all processes.

– ps –ef | grep keystone

– ps –ef | grep glance

– openstack servers how --diagnostics <instance ID>

– An instance gets stuck on a scheduled status:

– The scheduler is probably down.

– Run the nova list command to see if it works.

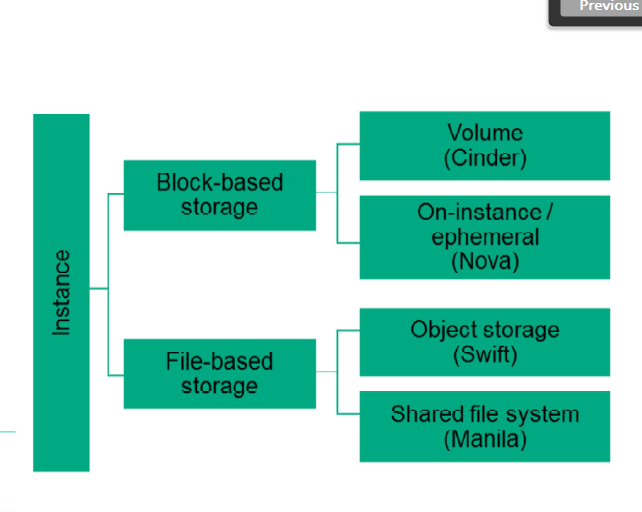
– An image is incorrect—Check the original image configuration.

In case of any inconsistency, check the information in the following Nova log files:

– Nova is located at /var/log/nova (multiple log files).

– Instance logs are located at /var/lib/nova/instances.

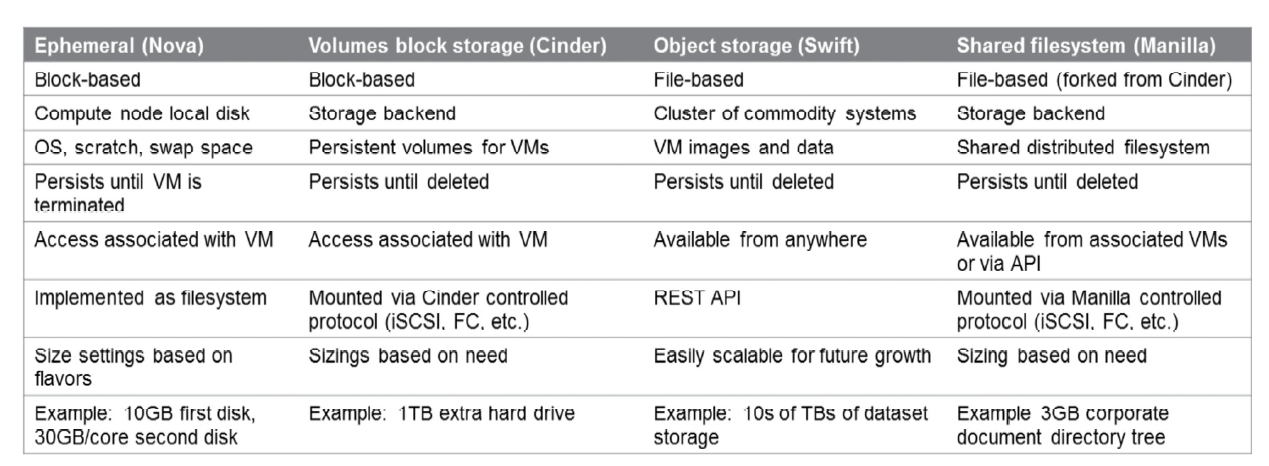
# **Cinder Block Storage**

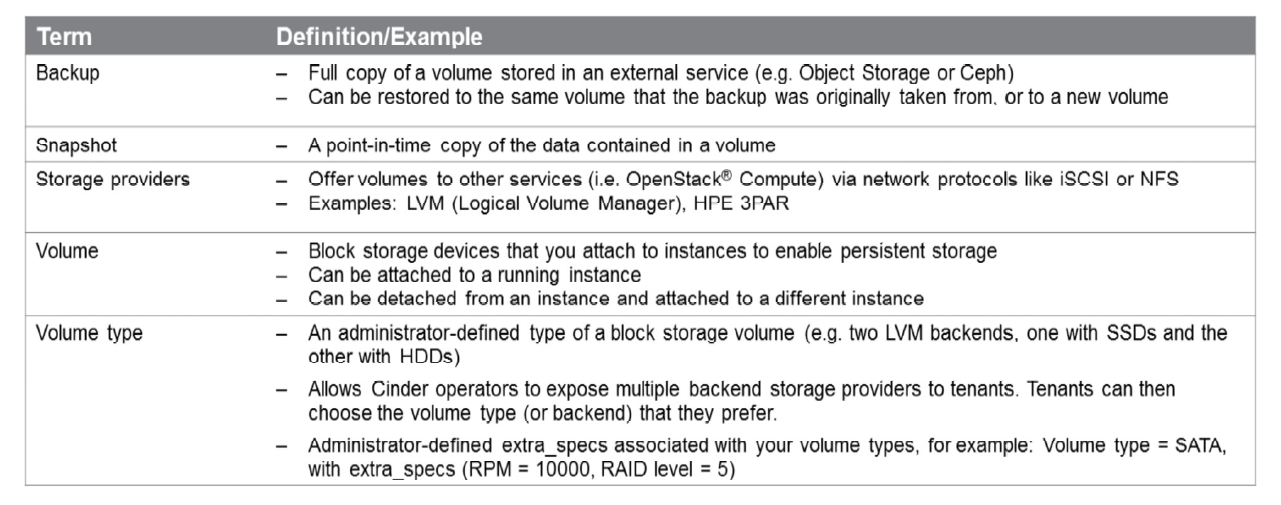


OpenStack® supports two types of persistent storage: object storage and block storage.

– Block storage (sometimes referred to as volume storage) provides access to block-storage devices. You can interact with block storage by attaching volumes to your running VM instances. These volumes are persistent; they can be detached from one instance and re-attached to another, and data remains intact. Block storage is implemented in OpenStack® by the OpenStack® Block Storage project (Cinder).

– Object storage allows you to store binary objects in an object store and access them by using a REST API. This capability is provided by the OpenStack® Swift project. Manila is an OpenStack® service used to present the management of file shares (for example, NFS and C IFS) as a core service to OpenStack®.





## Cinder Common Management Tasks

List volumes: cinder service-list (=openstack volume service list)

Create a volume: openstack volume create <name> --size <size in GB>

Create a boot volume from an image: openstack volume create <name> --size <size in GB> --description <decription> --image <image id> <name of vol>

Create an instance using a boot volume: openstack server create –volume <volume id> --flavor 1 myserv2

Attach a volume to an instance: openstack add volume <server id> <volume id>

## Troubleshooting Cinder

The cinder log files are located at /var/log/cinder:

– cinder-api.log—Keeps the API access log

– cinder-scheduler.log—Keeps information on how the scheduler allocated volumes by host

– cinder-volume.log—Keeps information on how Cinder manages the attaching and detaching of volumes

Potential issues to be checked:

– The service endpoints and associated users were not created properly

– The Keystone integration is incorrect

– iSCSI is not created and attached properly

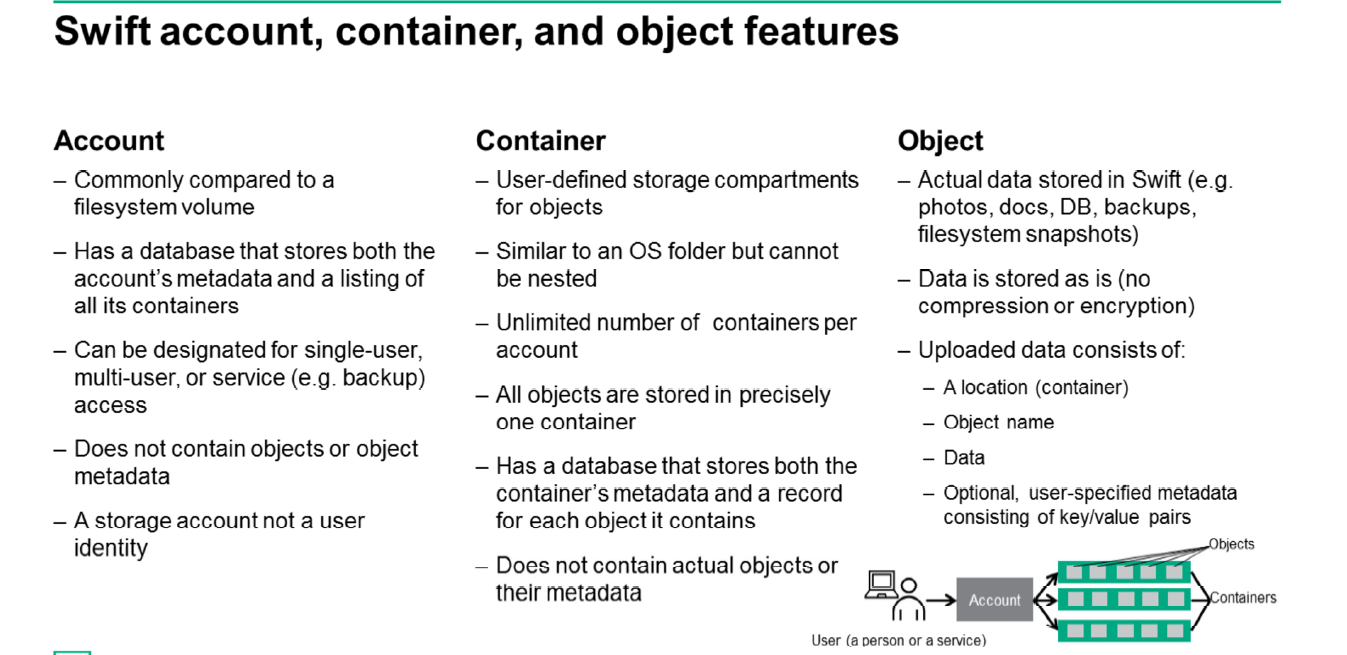
If you try to create a volume and the volume immediately goes into an error state, the best way to troubleshoot the problem is to grep the Cinder log files for the Universal Unique Identifier (UUID) of the volume. First check the log files on the cloud controller, and then check the storage node where you attempted to create the volume: grep 903b85d0-bacc-4855-a261-10843fc2d65b /var/log/cinder/\*.log

# **Swift Object Storage**

Swift is open source software used for managing scalable data storage using clusters of commodity hardware to store large amounts of data. Swift uses a distributed architecture to provide scalability and redundancy. Internally, Swift writes objects to multiple hardware devices (called “nodes”), and Swift ensures data replication and integrity across the cluster. Since Swift manages the replication and distribution of objects across different devices, inexpensive commodity hard drives and servers can be used.

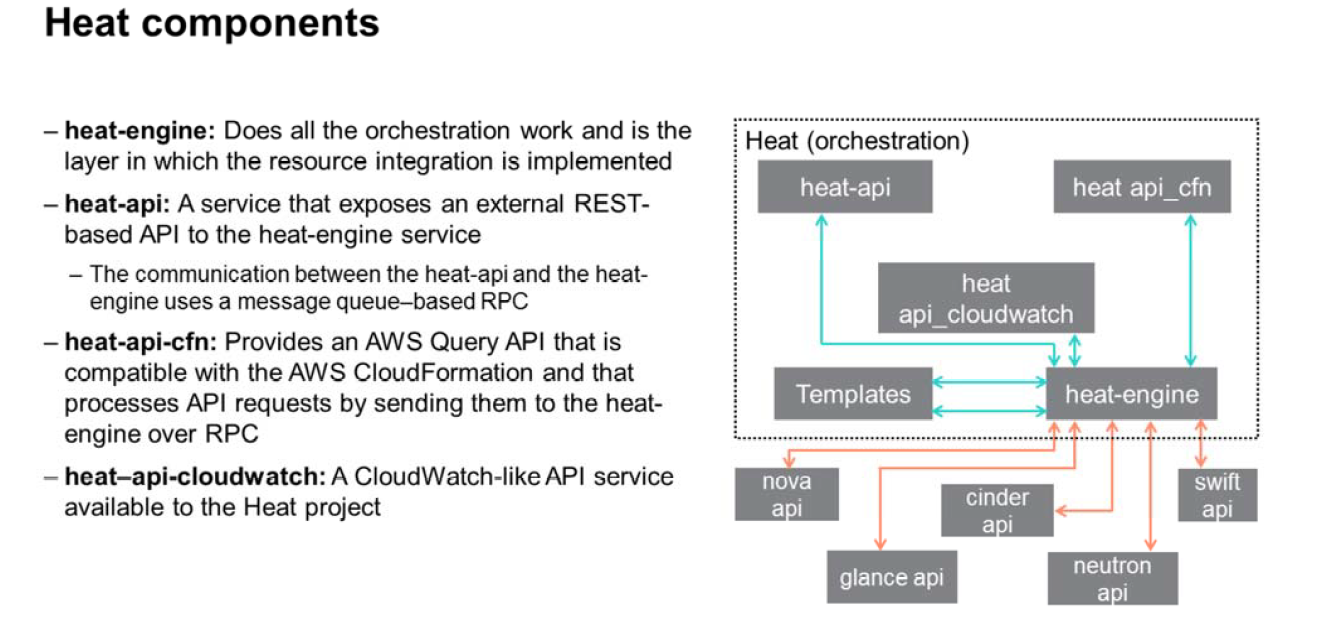
Swift achieves high scalability by relaxing constraints on consistency. Listings and aggregate metadata (like usage information) might not be immediately accurate. Similarly, reading an object that has been overwritten with new data may return an older version of the object data. Swift provides the ability for the client to request the most up-to-date version at the cost of request latency.

Upgrades and cluster resizes can be easily performed on a production cluster with zero end-user downtime. Swift provides both forward and backwards compatibility of its API, so a Swift cluster can be running multiple versions of the Swift software at the same time, as is common while the software is being upgraded. During the process of resizing, the incongruent data about where data lives is simply seen as a failure and the replication process ensures that the data will be moved to its correct location.



# **Heat Orchestration Service**

Heat is the main project in the OpenStack® Orchestration program. It implements an orchestration engine to launch multiple composite cloud applications based on templates that are in the form of text files that can be treated like code. A native Heat template format is evolving, but Heat is also compatible with the Amazon Web Services (AWS) CloudFormation template format, so many existing CloudFormation templates can be launched on OpenStack®. Heat provides both an OpenStack®-native REST API and a CloudFormation compatible Query API.



# **Ceilometer Telemetry Service**

Because OpenStack® provides Infrastructure as a Service (IaaS), it is necessary to be able to meter its performance and utilization for billing, benchmarking, scalability, and statistics purposes. The Ceilometer project provides a single point of contact for billing systems to acquire the measurements needed to establish your billing. This service currently works across all OpenStack® core components.

Ceilometer provides efficient collection of metering data, in terms of processor (CPU) and network costs. It enables those who deploy it to integrate it with the metering system directly or by replacing components, and to configure the type of data collected to meet their operating requirements.

Data can be collected by monitoring the notifications sent from existing services or by polling the infrastructure. The data collected by the metering system is made visible to some users through a RESTAPI. Metering messages are signed and cannot be repudiated.

Some of the common ceilometer cases are:

– Providing Cloud Providers with the measurements they need to establish customer billing

– Providing the live metrics necessary to trigger auto-scaling (used by Heat Scaling Groups)

– Publishing information for monitoring, debugging and graphing tools

– Providing the data for data mining and capacity planning