Università degli Studi di Bologna
Scuola di Ingegneria

Corso di

Reti di Calcolatori M

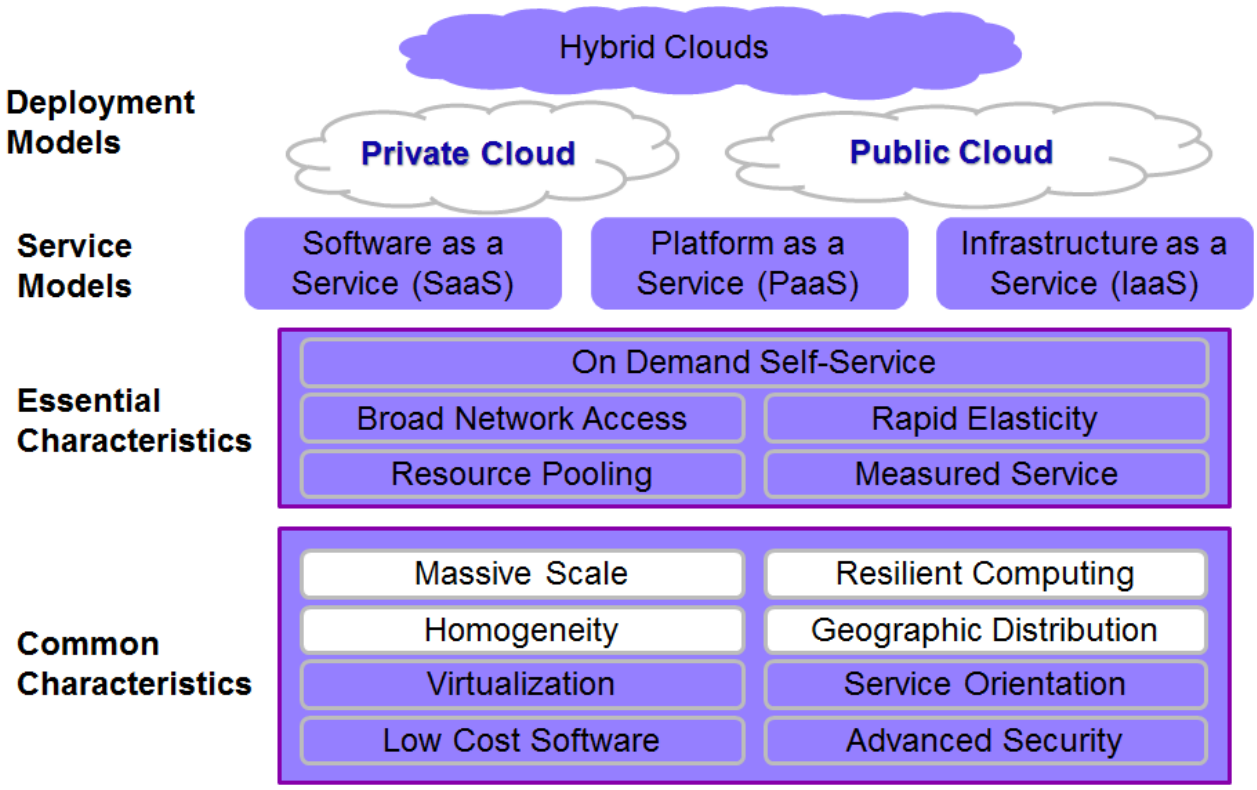
Cloud: Openstack

Antonio Corradi

Luca Foschini

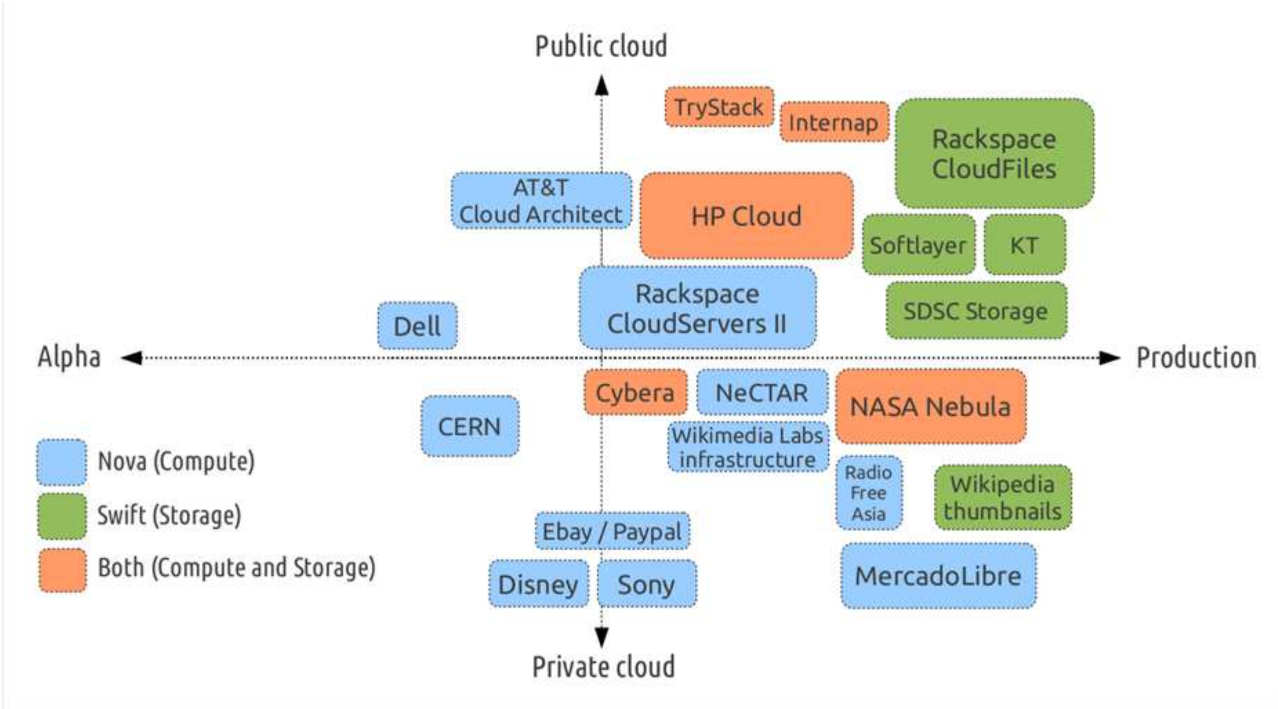
Anno accademico 2014/2015

NIST STANDARD CLOUD



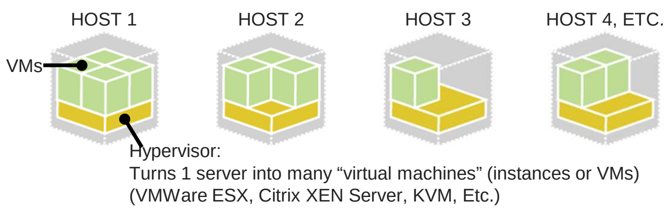
National Institute of Standards and Technology www.nist.gov/

Known Deployment Models



Cloud: resource virtualization

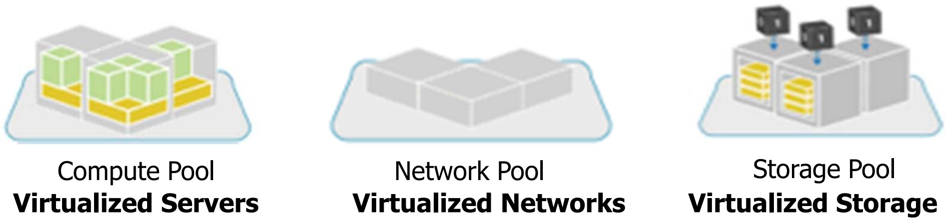
- First step: *Server virtualization*



- Hypervisors provide an abstraction layer between hardware and software
- Hardware abstraction
- Better resource utilization for every single server

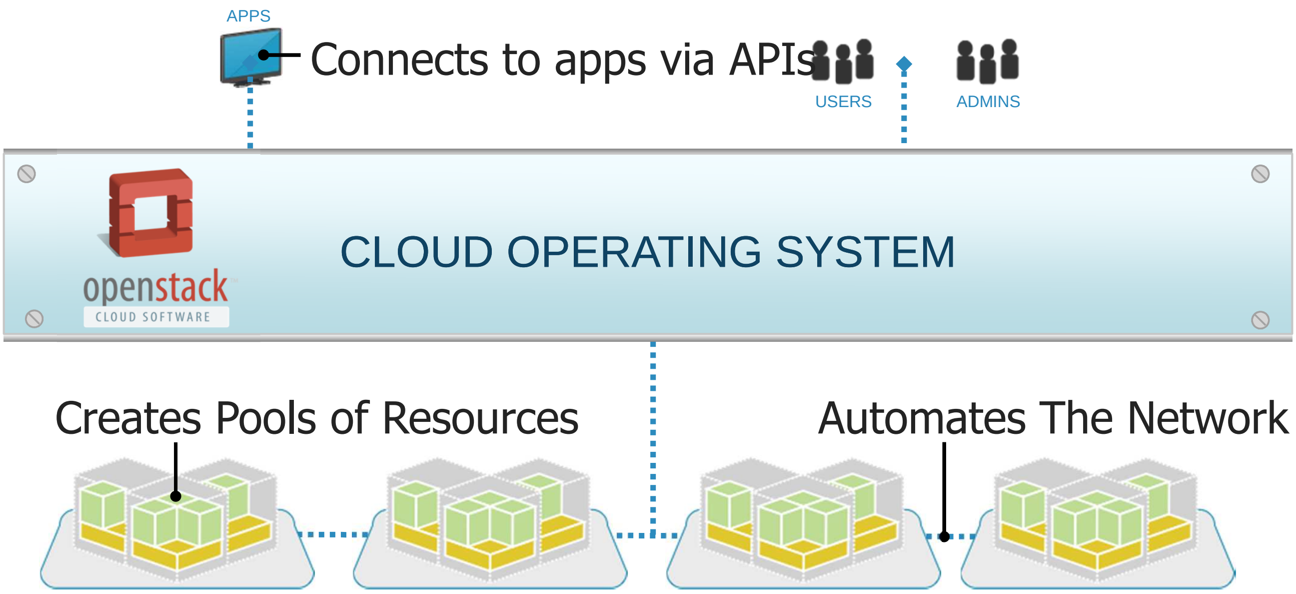
Cloud: resource virtualization

- Second step: *network and storage virtualization*



- Resource pool available for several applications
- Flexibility and efficiency

High-level Architecture of the OpenStack Cloud IaaS



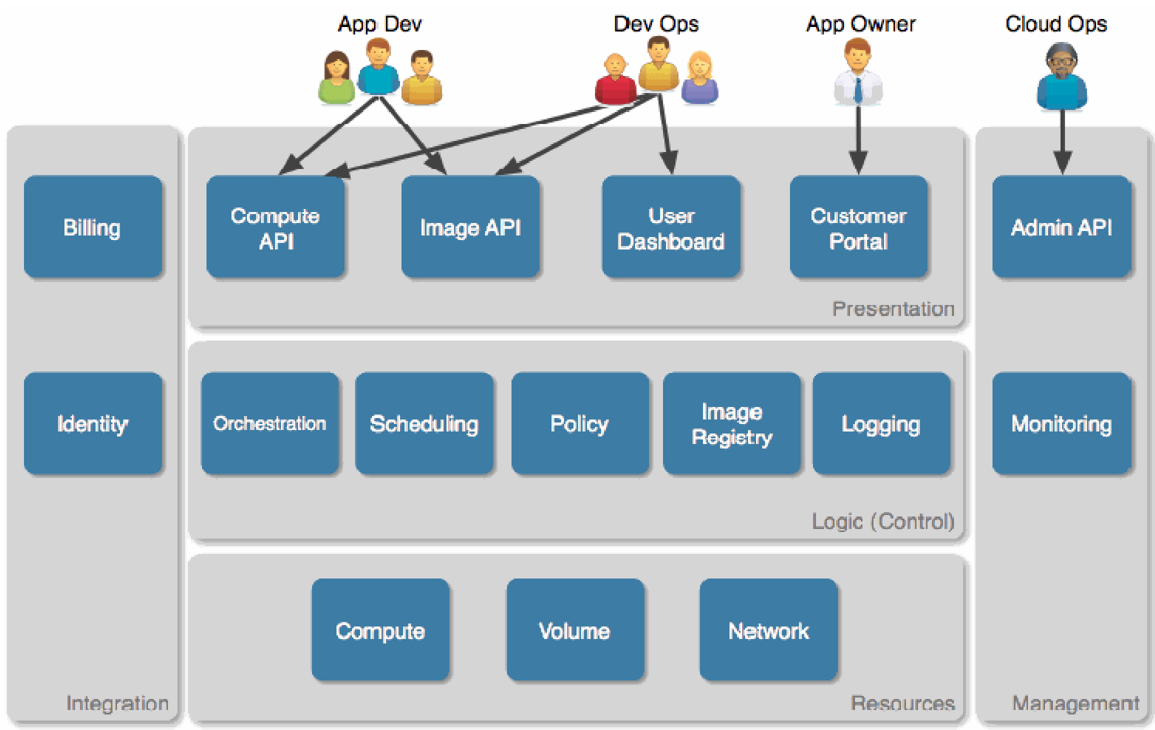
OpenStack history in a nutshell

OpenStack

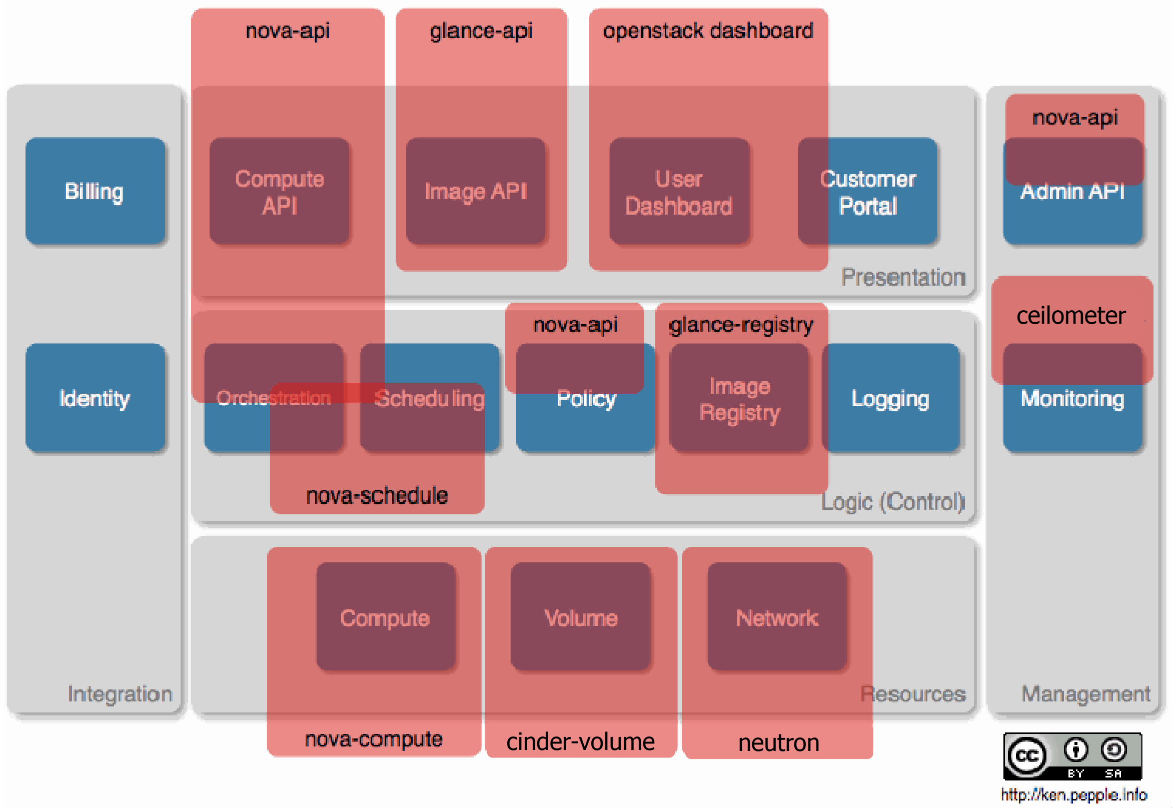
- Founded by **NASA** and **Rackspace** in 2010
- Currently supported by more than **300 companies** and **13866 people**
- Latest release: **Juno**, October 2014

- **Six-month** time-based **release cycle** (aligned with Ubuntu release cycle)
- **Open-source** vs Amazon, Microsoft, Vmware…
- **Constantly growing** project

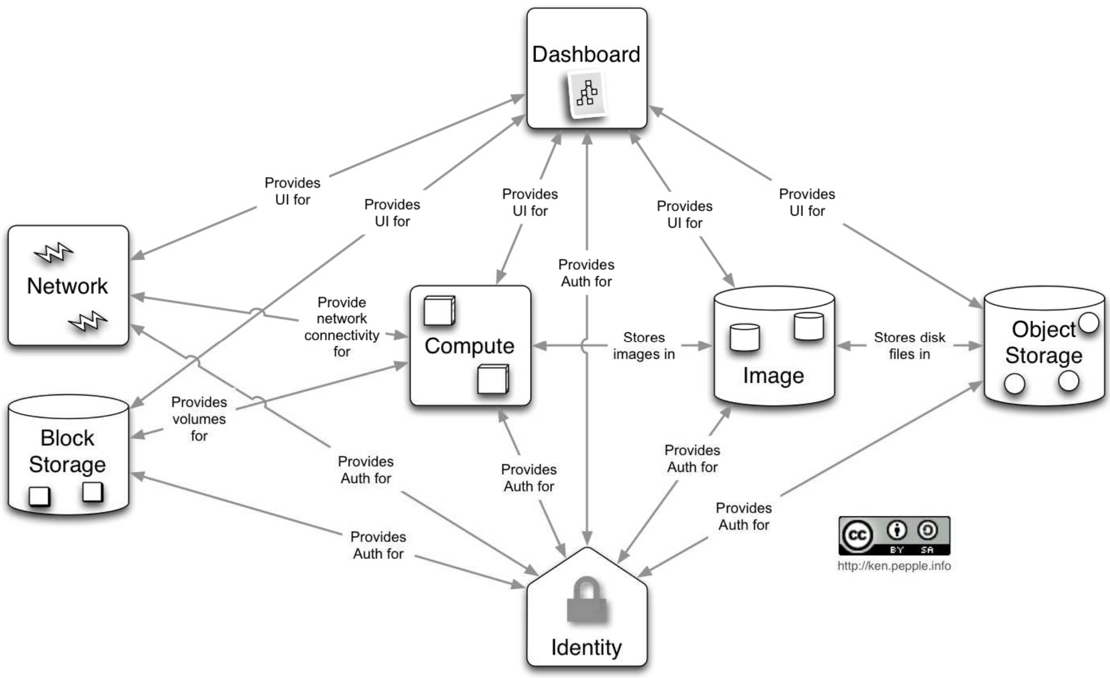
Main Function in a Cloud



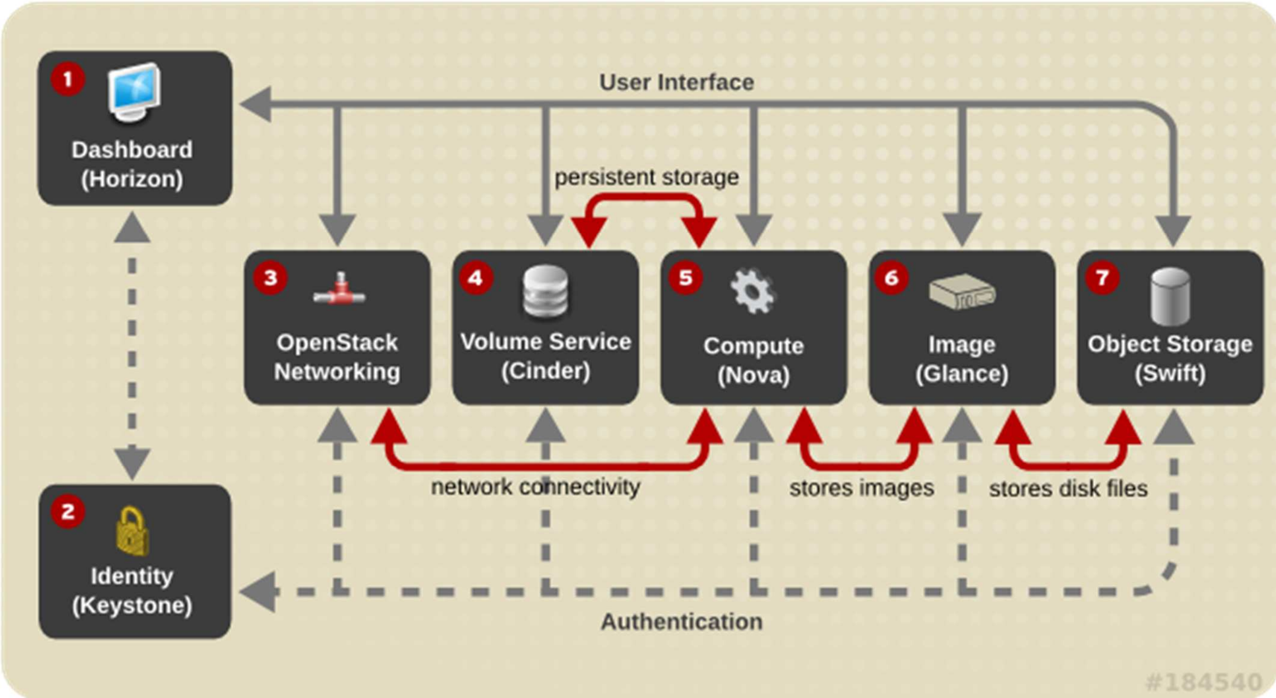
Main Function in a Cloud



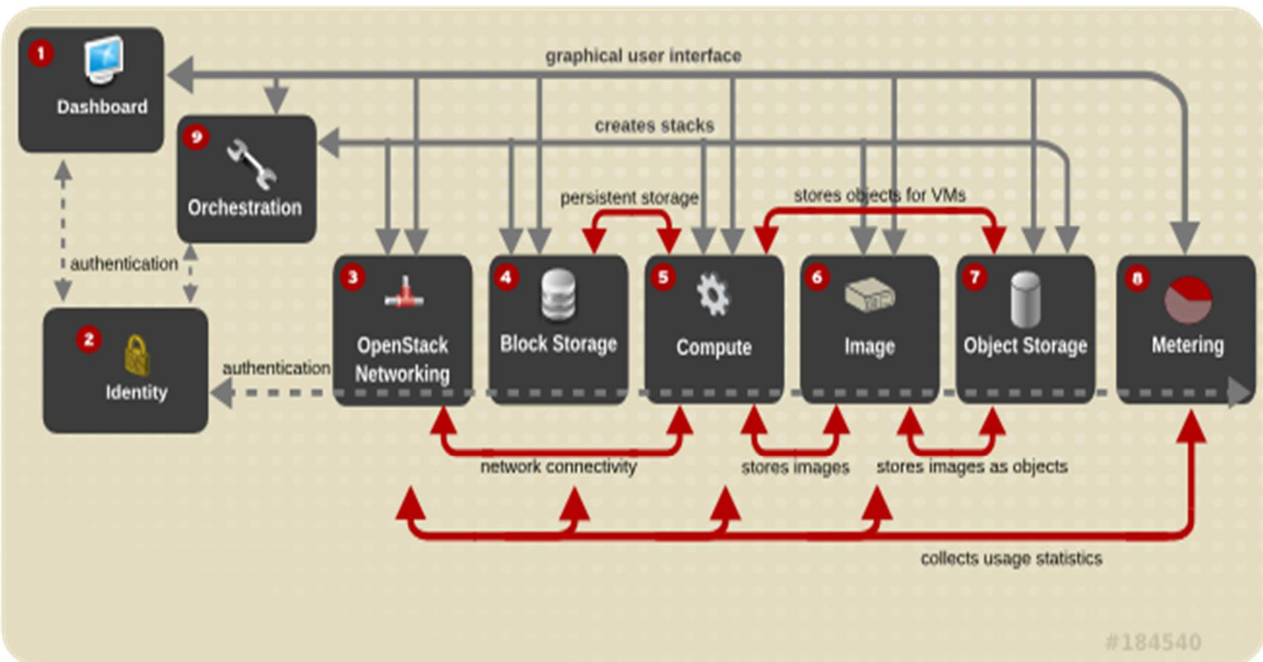
OpenStack main services



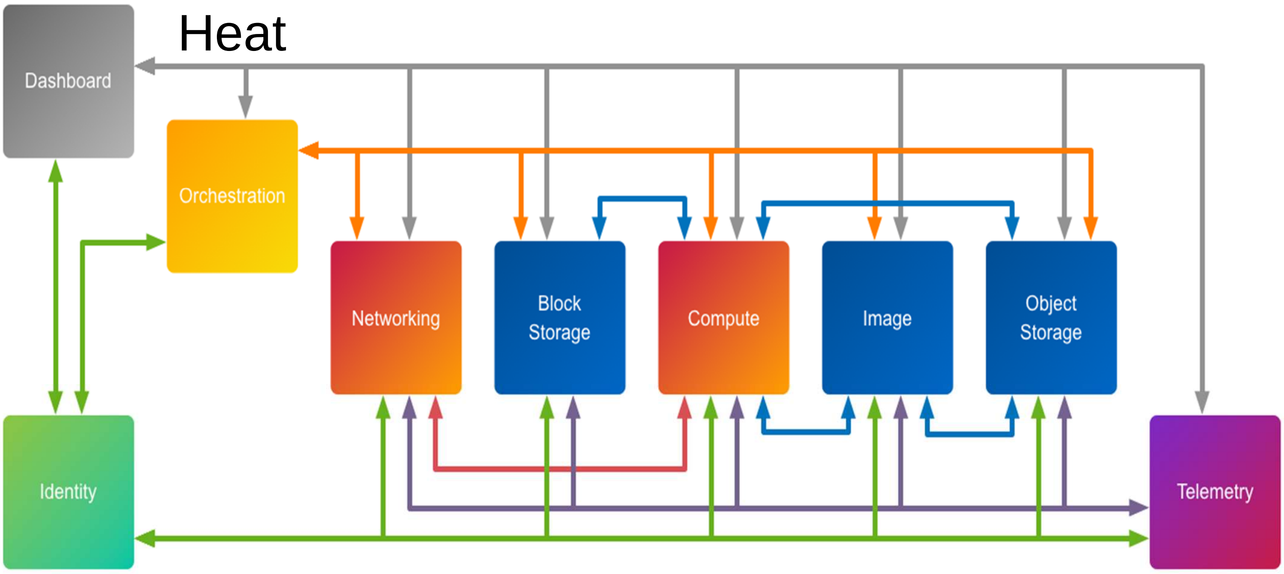
OpenStack main services



OpenStack main services



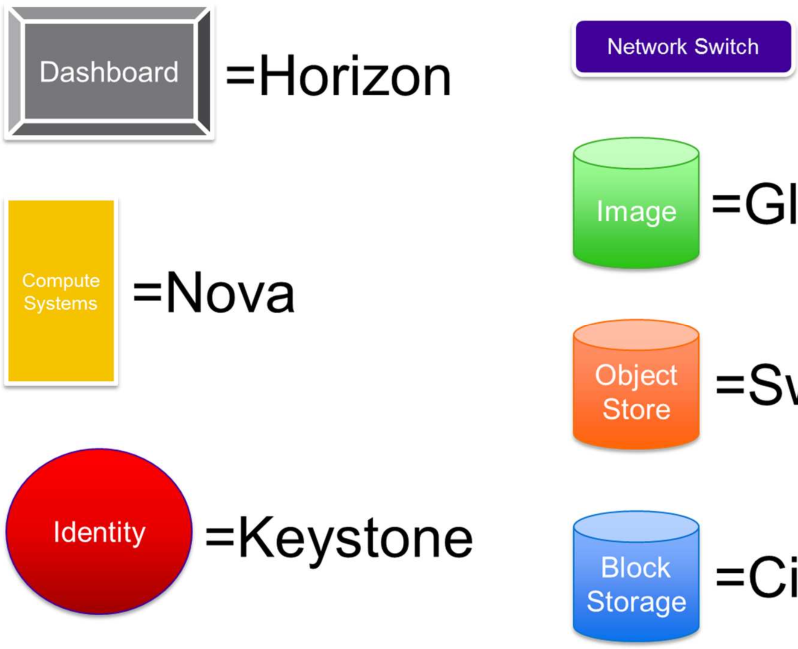
OpenStack services



Heat

Ceilometer

OpenStack main components



=Horizon

=Quantum / Neutron

=Glance

=Nova

=Swift

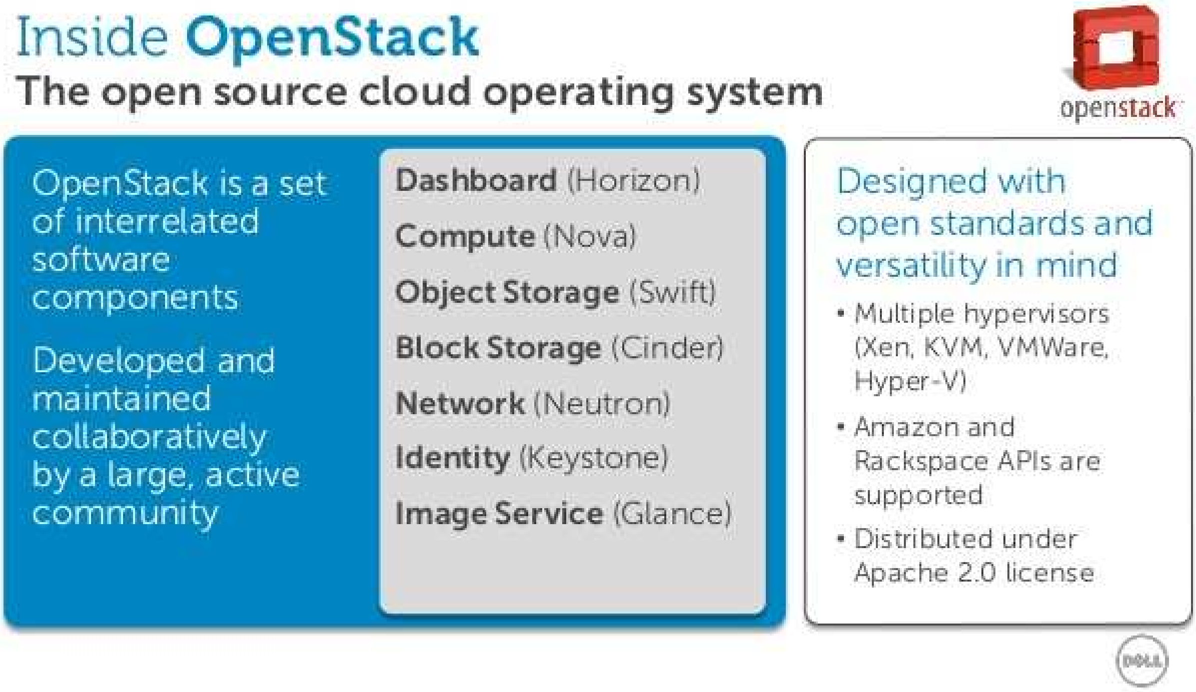
=Keystone

=Cinder

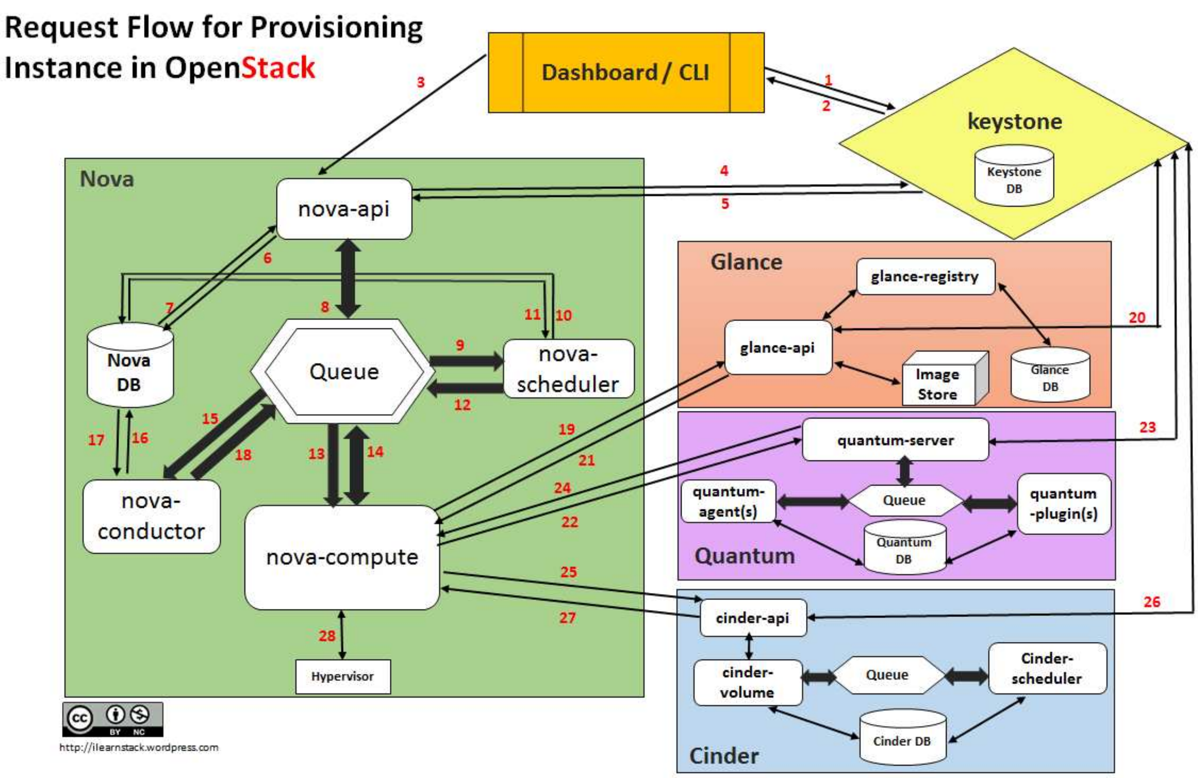
Ceilometer

Heat

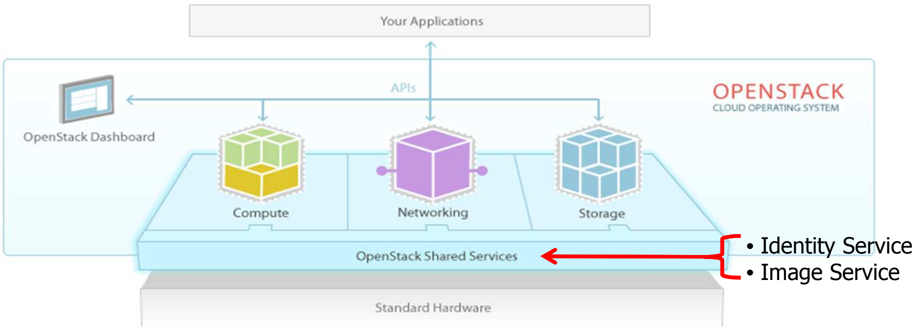
OpenStack main components



OpenStack main worflow



OpenStack services (detailed)



- **Dashboard**: Web application used by administrators and users to manage cloud resources
- **Identity**: provides unified authentication across the whole system
- **Object Storage**: redundant and highly scalable object storage platform
- **Image Service**: component to save, recover, discover, register and deliver VM images
- **Compute**: component to provision and manage large sets of VMs
- **Networking**: component to manage networks in a pluggable, scalable, and API-driven fashion

OpenStack Services: Design Guidelines

All OpenStack services share the same internal architecture organization that follow a few clear design and implementation guidelines:

- **Scalability and elasticity:** gained mainly through ***horizontal scalability***
- **Reliability:** ***minimal dependencies*** between different services and ***replication*** of core components
- **Shared nothing between different services:** each service ***stores all needed information internally***
- **Loosely coupled asynchronous interactions:** internally, ***completely decoupled pub/sub communications*** between core components/services are preferred, even to realize high-level synch RPC-like operations

OpenStack Services: Main Components

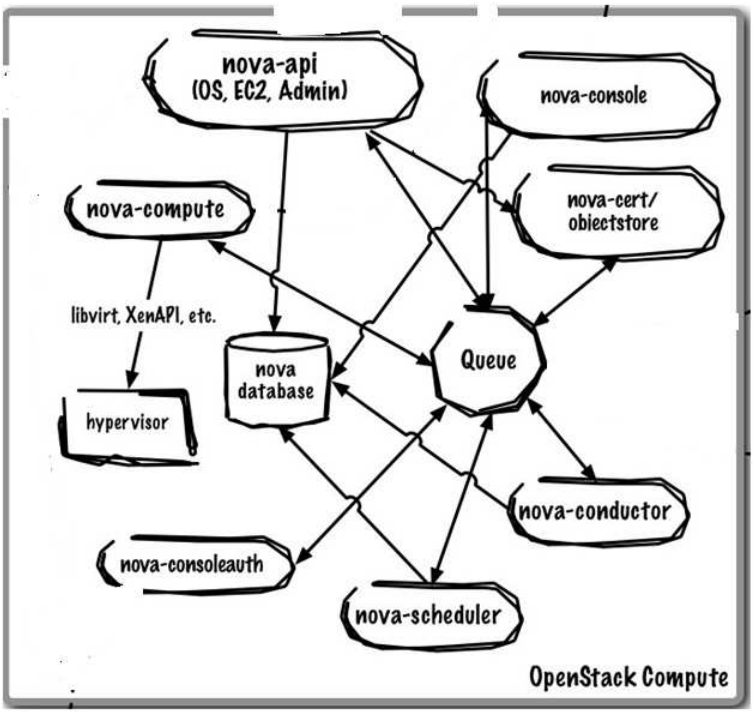
Deriving from the guidelines, every service consists of the following core components:

- **pub/sub messaging service:** Advanced Message Queuing Protocol (***AMQP***) standard and RabbitMQ default implementation
- **one/more internal core components:** realizing the service application logic
- **an API component:** acting as a service front-end to export sevice functionalities via interoperable ***RESTful APIs***
- **a local database component:** ***storing internal service state*** adopting existing solutions, and making different technological choices depending on service requirements (ranging from MySQL to highly scalable MongoDB, SQLAlchemy, and HBase)

Nova - Compute

- Provides **on-demand virtual servers**
- Provides and manages **large networks** of virtual machines (functionality moving to Neutron)
- Modular architecture designed to horizontally scale on standard hardware
- Supports several hypervisor (i.e. KVM, XenServer, etc.)
- Developers can access computational resources through **APIs**
- Administrators and users can access computational resources through **Web interfaces** or **CLI**

Nova – Components (a good OpenStack service example)



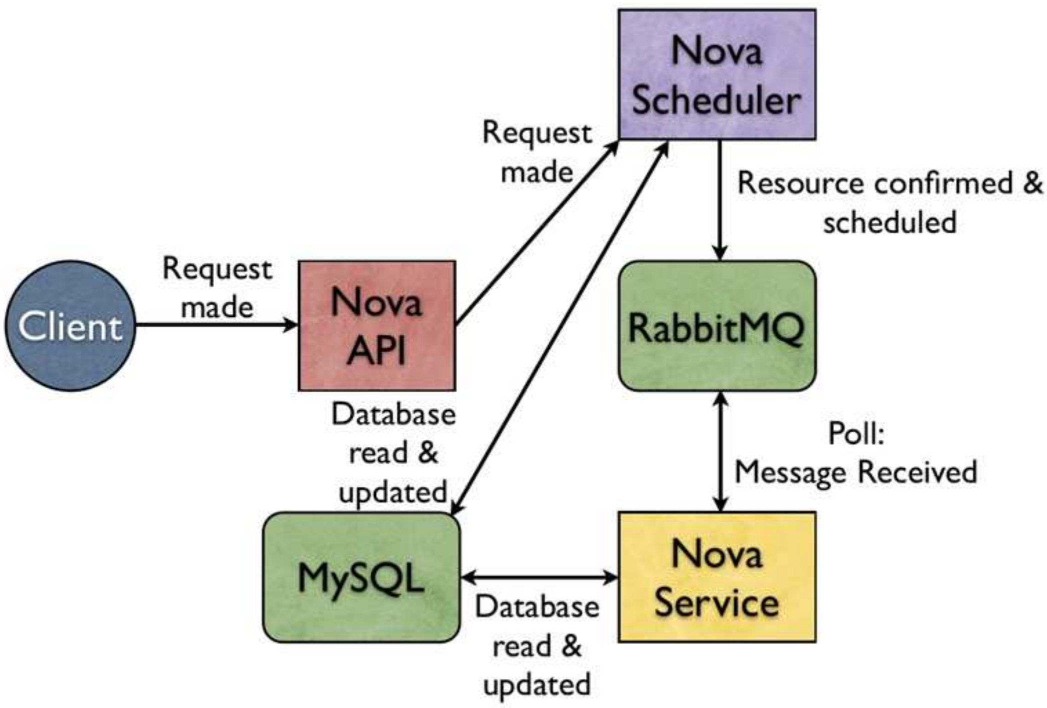
Nova – Components (1)

- **nova-API**: RESTful API web service used to send commands to interact with OpenStack. It is also possible to use CLI clients to make OpenStack API calls
- **nova-compute**: hosts and manages VM instances by communicating with the underlying hypervisor
- **nova-scheduler**: coordinates all services and determines placement of new requested resources
- **nova database**: stores build-time and run-time states of Cloud infrastructure
- **queue**: handles interactions between other Nova services By default, it is implemented by RabbitMQ, but also Qpid can be used

Nova – Components (2)

- **nova-console**, **nova-novncproxy** e **nova-consoleauth**: provides, through a proxy, user access to the consoles of virtual instances
- **nova-network**: accepts requests coming from the queue and executes tasks to configure networks (i.e., changing IPtables rules, creating bridging interfaces, … These functionalities are now moved to Neutron service.
- **nova-volume**: handles persistent volumes creation and their de/attachment from/to virtual instances These functionalities are now moved to Cinder services

Nova General interaction scheme



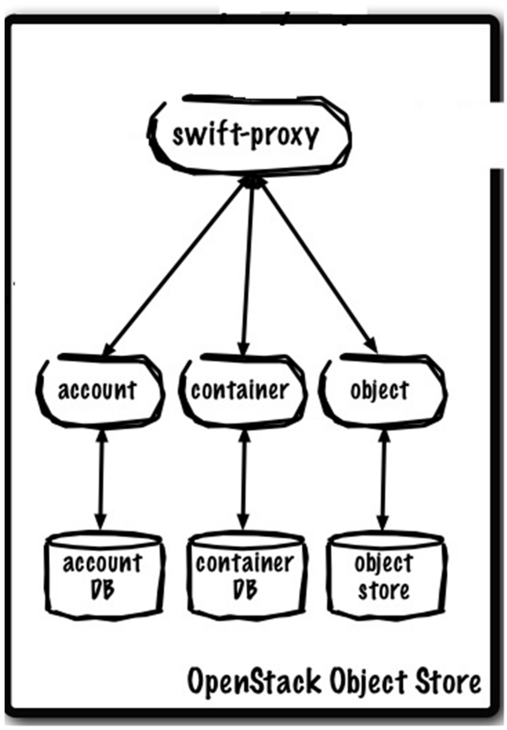
Swift - Storage

Swift allows to store and recover files

- Provides a **completely distributed storage** platform that can be accessed by APIs and integrated inside applications or used to store and backup data
- **It is not** a **traditional filesystem**, but rather a distributed storage system for static data such as virtual machine images, photo storage, email storage, backups and archives
- It doesn't have a central point of control, thus providing properties like **scalability**, **redundancy**, and **durability**

Swift - Components

- **Proxy Server:** handles incoming requests such as files to upload, modifications to metadata or container creation
- **Accounts server:** manages accounts defined through the object storage service
- **Container server:** maps containers inside the object storage service
- **Object server:** manages files that are stored on various storage nodes



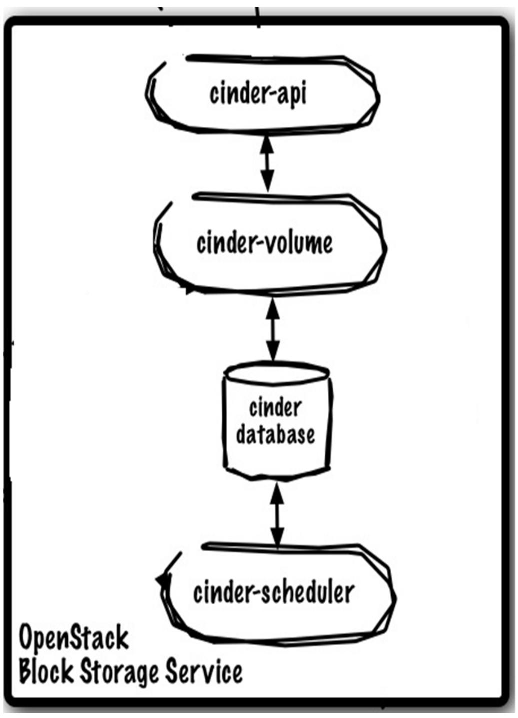
Cinder – Block Storage

Cinder handles **storage devices** that can be attached to **VM instances**

- Handles the **creation**, **attachment** and **detachment** of volumes to/from instances
- Supports iSCSI, NFS, FC, RBD, GlusterFS protocols
- Supports several storage platforms like Ceph, NetApp, Nexenta, SolidFire, and Zadara
- Allows to **create snapshots** to backup data stored in volumes. Snapshots can be restored or used to create a new volume

Cinder – Block Storage

- **cinder-API:** accepts user requests and redirects them to cinder-volume in order to be processed
- **cinder-volume:** handles requests by reading/writing from/to cinder database, in order to maintain the system in a consistent state
 Interacts with the other components through a message queue
- **cinder-scheduler:** selects the best storage device where to create the volume
- **cinder database:** maintains volumes' state



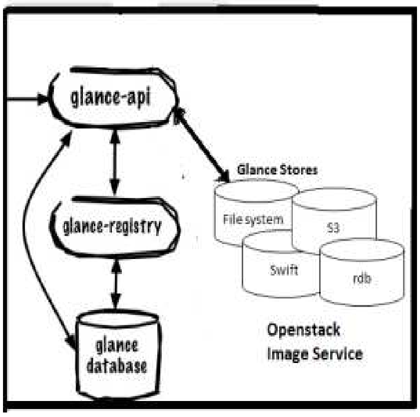
Glance – Image Service

Glance handles the **discovery**, **registration**, and **delivery** of disk and virtual server **images**

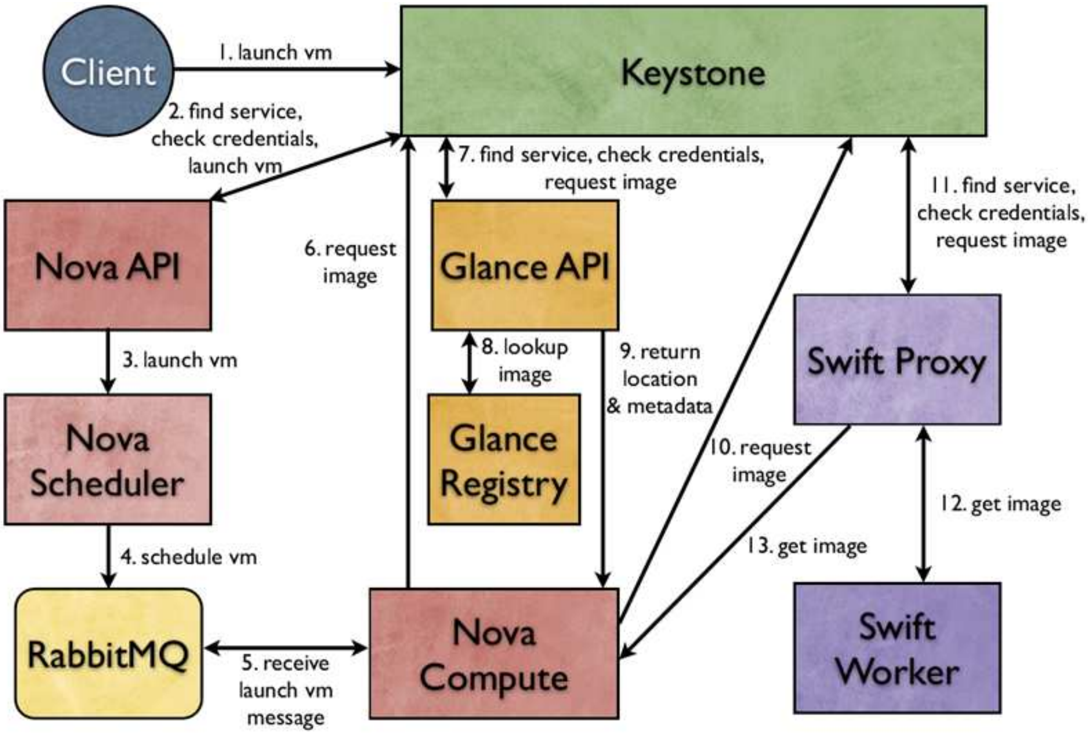
- Allows to store images on **different storage systems**, i.e., Swift
- Supports **several disk formats** (i.e. Raw, qcow2, VMDK, etc.)

Glance – Components

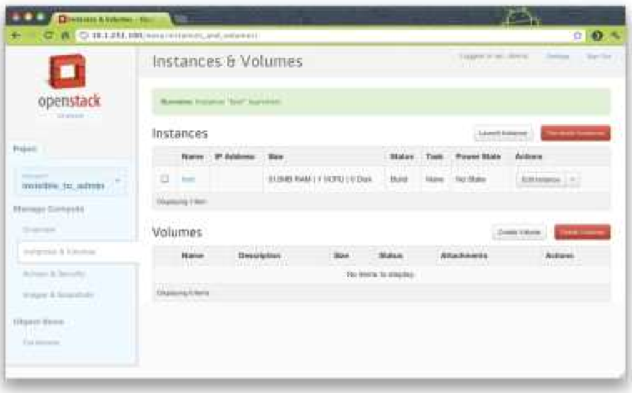
- **glance-API:** handles API requests to discover, store and deliver images
- **glance-registry:** stores, processes and retrieves image metadata (dimension, format,...).
- **glance database:** database containing image metadata
- Glance uses an external **repository** to store images Currently supported repositories include filesystems, Swift, Amazon S3, and HTTP



Nova – Launching a VM



Horizon - Dashboard



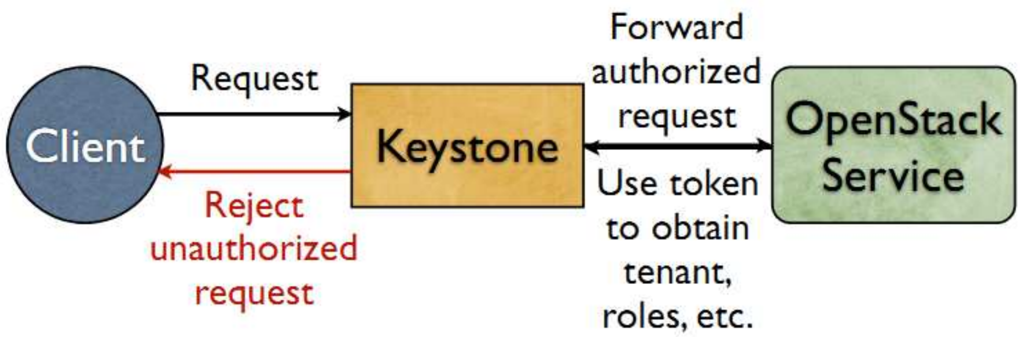
Provides a modular web-based user interface to access other OpenStack services

Through the dashboard it is possible to perform actions like launch an instance, to assign IP addresses, to upload VM images, to define access and security policies, etc.

Keystone – Authentication and Authorization

- Keystone is a framework for the authentication and authorization for all the other OpenStack services
- Creates **users** and groups (also called **tenants**), adds/removes users to/from groups, and defines **permissions** for cloud resources using role-based access control features. Permissions include the possibility to launch or terminate instances
- Provides 4 primary services:
  - **Identity**: user information authentication
  - **Token**: after logged-in, replaces password authentication
  - **Catalog**: maintains an endpoint registry used to discovery OpenStack services endpoints
  - **Policy**: provides a rule-based authorization engine

Keystone

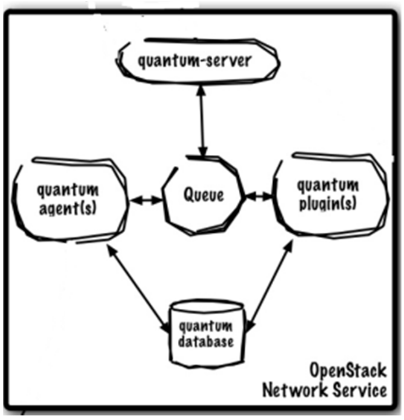


Neutron Networking

- **Pluggable**, **scalable** e **API-driven** support to manage networks and IP addresses.
- **NaaS** "**Network as a Service**"
  Users can create their own networks and plug virtual network interface into them
- **Multitenancy:** isolation, abstraction and full control over virtual networks
- **Technology-agnostic:** APIs specify service, while vendor provides his own implementation. Extensions for vendor-specific features
- **Loose coupling:** standalone service, not exclusive to OpenStack

Neutron – Components

- **neutron-server:** accept request sent through APIs e and forwards them to the specific plugin
- **Plugins and Agents:** executes real actions, such as dis/connecting ports, creating networks and subnets, creating routers, etc.
- **message queue:** delivers messages between quantum-server and various agents
- **neutron database:** maintains network state for some plugins



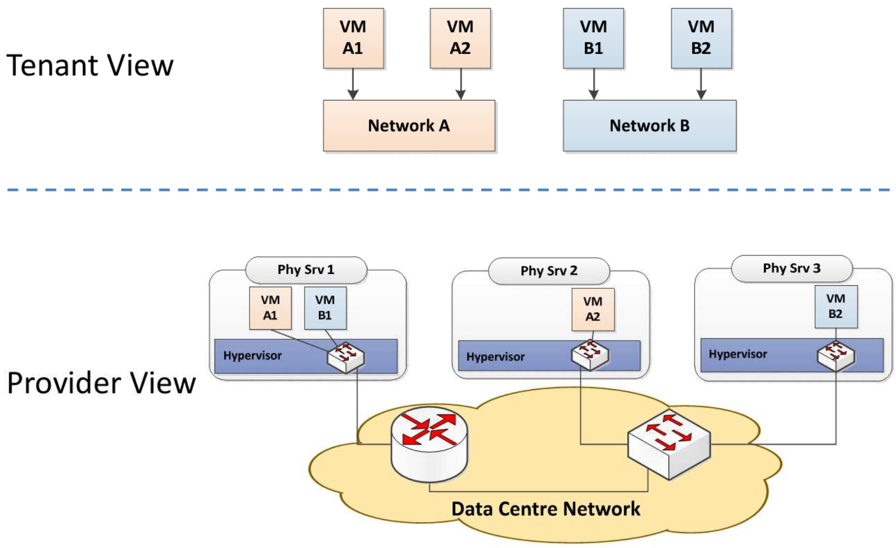
Neutron – Agents

- **dhcp agent:** provides DHCP functionalities to virtual networks
- **plugin agent:** runs on each hypervisor to perform local vSwitch configuration. The agent that runs, depends on the used plug-in (e.g. OpenVSwitch, Cisco, Brocade, etc.).
- **L3 agent:** provides L3/NAT forwarding to provide external network access for VMs

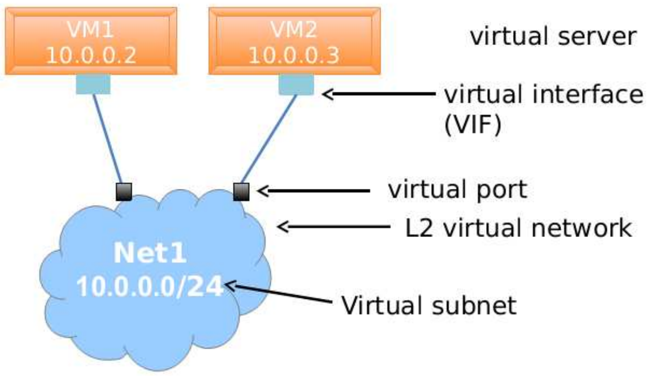
Neutron logical view vs. physical view

Neutron **decouples** the logical view of the network from the physical view

It provides APIs to define, manage and connect virtual networks



Neutron - logical view



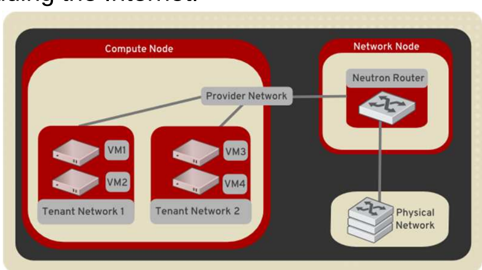
- **Network**: represents an isolated virtual Layer-2 domains; a network can also be regarded as a logical switch;
- **Subnet**: represents IPv4 or IPv6 address blocks that can be assigned to VMs or router on a given network;
- **Ports**: represent logical switch ports on a given network that can be attached to the interfaces of VMs. A logical port also defines the MAC address and the IP addresses to be assigned to the interfaces plugged into them. When IP addresses are associated to a port, this also implies the port is associated with a subnet, as the IP address was taken from the allocation pool for a specific subnet.

Neutron - tenant networks

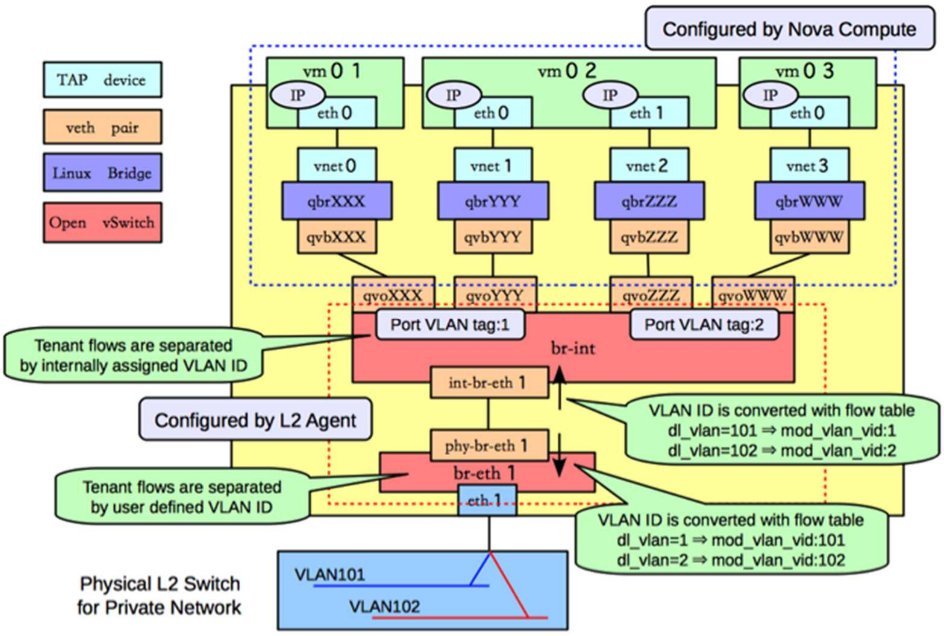
Tenant networks can be created by users to provide connectivity within tenants. Each tenant network is fully isolated and not shared with other tenants.

Neutron supports different types of tenant networks:

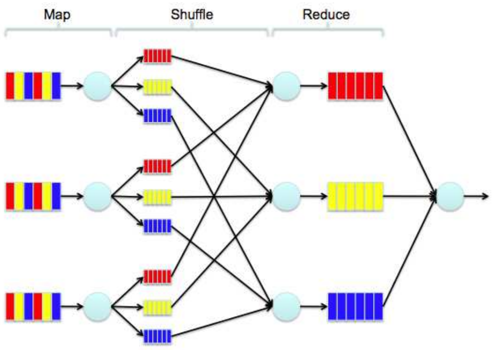
- **Flat**: no tenant support. Every instance resides on the same network, which can also be shared with the hosts. No VLAN tagging or other network segregation takes place;
- **Local**: instances reside on the local compute host and are effectively isolated from any external networks;
- **VLAN**: each tenant network uses VLAN IDs (802.1Q tagged) corresponding to VLANs present in the physical network. This allows instances to communicate with each other across the environment, other than with dedicated servers, firewalls, load balancers and other networking infrastructure on the same layer 2 VLAN. Switch must support 802.1Q standard in order to provide connectivity between two VMs on different hosts;
- **VXLAN and GRE**: tenant networks use network overlays to support private communication between instances. A Networking router is required to enable traffic to traverse outside of the tenant network. A router is also required to connect directly-connected tenant networks with external networks, including the Internet.



Neutron – VLAN tenant network



Apache Hadoop



Open source project for the development of **distributed** and **concurrent** applications

- based on Google **MapReduce**
- designed for distributed processing of large data sets across **very large clusters** of computers
- highly **fault tolerant**
- relies on **Hadoop Distributed File System (HDFS)** for the distribution of data

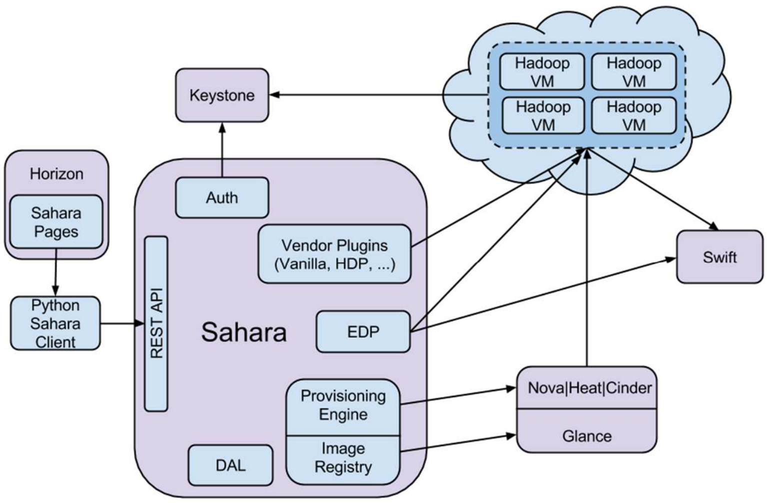
Hadoop for OpenStack

Hadoop can exploit the virtualization provided by **OpenStack** in order to obtain more flexible clusters and a better resource utilization

OpenStack service **Sahara** can be used to **deploy** and **configure Hadoop clusters** in a Cloud environment:

- cluster scaling functionalities
- Analytics as a Service (AaaS) functionalities
- accessible via OpenStack dashboard, CLI or RESTful API

Sahara components



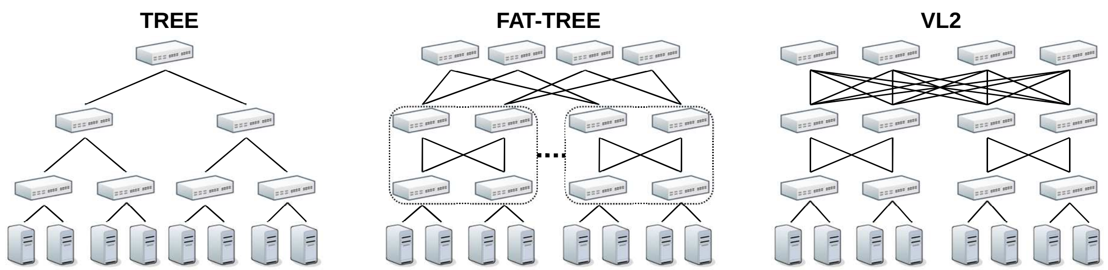
Cloud-based Context Data Handling

- A smart city features **thousands of sensors**
  - **State-of-the-art mobile devices** are equipped with **several onboard sensors**, e.g., camera, accelerometers, GPS, etc.
  - **Physical environments** will include additional sensors, e.g., temperature and lighting sensors, feeding new data directly into the system
- Cloud technologies enable the **rapid provisioning of scalable services** through **distributed** and **virtualized resources**
  - **On-demand computing resources** and **pay-per-use model**
  - **Dynamic resource scaling** depending on user requests
- **Cloud solutions for CDDI**
  - **High scalability** to address **context data storage and processing**
  - **Dynamic resource scaling** lets the CDDI require new computing resources when the data to be processed increase (due to conference events, etc.)

Cloud solutions need VM placement algorithms to decide which VMs should be co-located on the same physical host

Network-aware VM Placement

- **Modern Data Centers (DCs) for Cloud computing**
  - large-scale scale systems with **hundreds of servers** and **thousands of Virtual Machines (VMs)**
  - **virtualization solutions** to improve **resource utilization** by **VM consolidation**
- **VM placement must consider network requirements and constraints**
  - useful to **prevent reduced network performance**
  - hard to enforce at run-time due to **time-varying traffic demands**
  - difficult to apply due to many entangled aspects, including **network architecture**, **competing traffic demands**, and so forth



Network-aware VM Placement

- **Virtualized DC**
  - **Co-location choices greatly affect the final traffic** in the DC network
  - **In-memory message passing mechanisms** much faster than real network communications

- **Network fabric** of modern DC employs **graph-based topology** and **dynamic multi-path routing**
  - **No knowledge** of which **traffic flows will be routed on a specific link**
  - **Limited networking resources** have to be considered to avoid unfeasible solutions

- **Network-aware VM placement** addressed in the past with different objectives, but
  - **Traffic demands are time-varying** and **traffic bursts can mine network performance**
  - **Placement stability** has to be increased by ensuring spare capacities