CLOUD COMPUTING

20MCA265

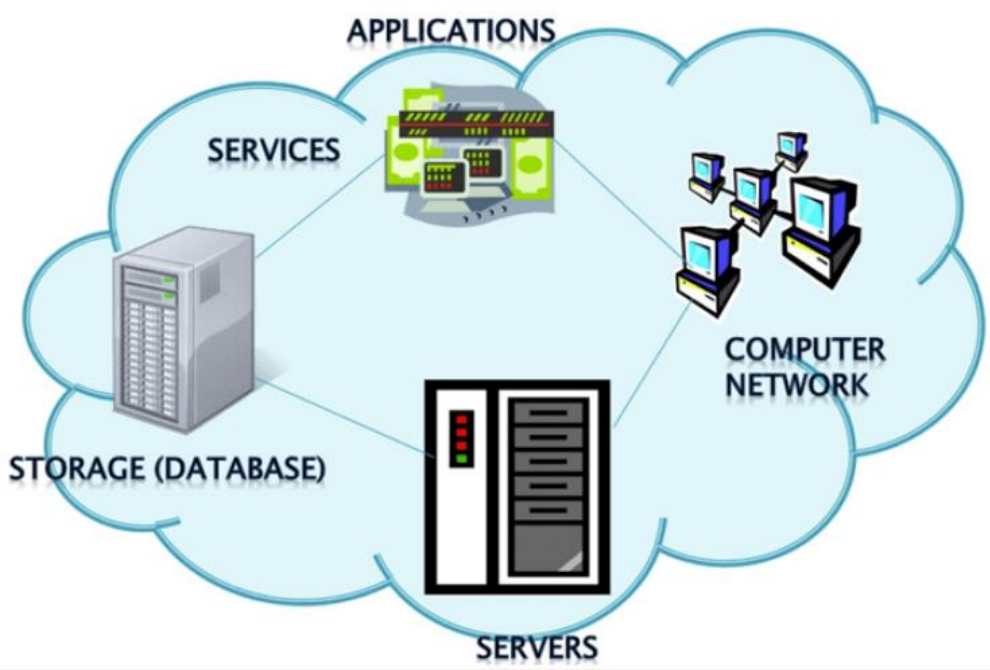
MODULE 1 - CONTENTS

- Introduction to cloud computing, private cloud, public cloud, hybrid cloud architecture.
- Cloud Services - Infrastructure as a Service, Platform as a Service, Storage as a Service.
- Designing OpenStack Cloud Architectural Consideration - OpenStack - The new data centre paradigm - OpenStack logical architecture - Nova - Compute Service-Neutron - Networking services - Gathering the pieces and building a picture - A sample architecture setup.

INTRODUCTION TO CLOUD COMPUTING

- Cloud computing is the ***on-demand delivery of IT resources over the Internet*** with pay-as-you-go pricing. Instead of buying, owning, and maintaining physical data centers and servers, you can access technology services, such as computing power, storage, and databases, on ***an as-needed basis from a cloud provider*** like Amazon Web Services (AWS).

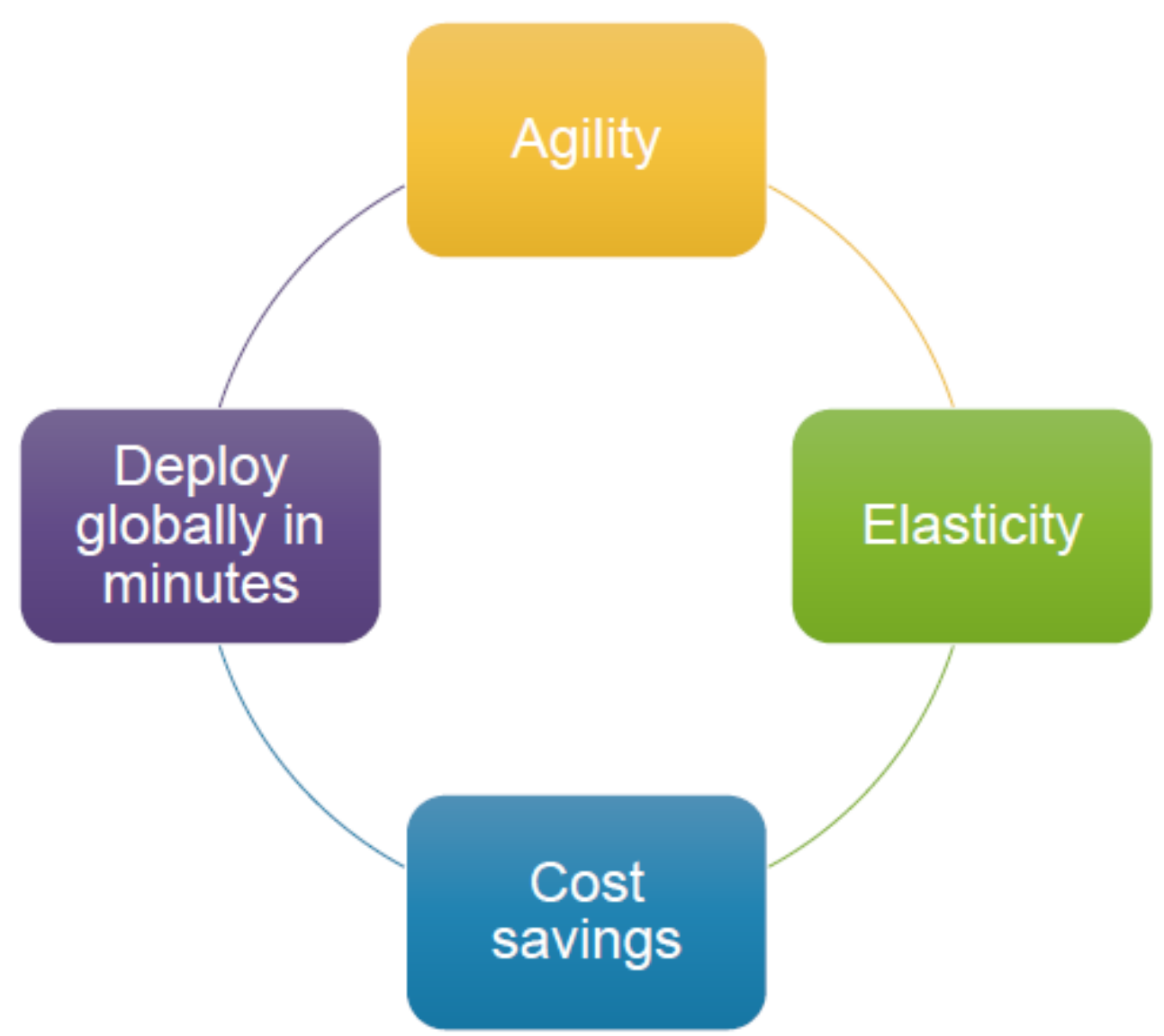
- In other words, we can say cloud computing means the **delivery of computing services**, such as servers, software, networking, analytics, intelligence, and databases over the internet or the cloud. It offers flexible resources, faster innovation, and economics of scale.



WHO IS USING CLOUD COMPUTING?

- Organizations of every type, size, and industry are using the cloud for a wide variety of use cases, such as ***data backup, disaster recovery, email, virtual desktops, software development and testing, big data analytics, and customer-facing web applications***.
  - For example, healthcare companies are using the cloud to develop more personalized treatments for patients.
  - Financial services companies are using the cloud to power real-time fraud detection and prevention.
  - And video game makers are using the cloud to deliver online games to millions of players around the world.

BENEFITS OF CLOUD COMPUTING



BENEFITS OF CLOUD COMPUTING

- **Agility**
  - The cloud gives you easy access to a broad range of technologies so that you can innovate faster and build nearly anything that you can imagine. You can quickly spin up resources as you need them–from infrastructure services, such as compute, storage, and databases, to Internet of Things, machine learning, data lakes and analytics, and much more.

BENEFITS OF CLOUD COMPUTING

- **Elasticity**
  - With cloud computing, you don’t have to over-provision resources up front to handle peak levels of business activity in the future. Instead, you provision the amount of resources that you actually need. You can scale these resources up or down to instantly grow and shrink capacity as your business needs change.

BENEFITS OF CLOUD COMPUTING

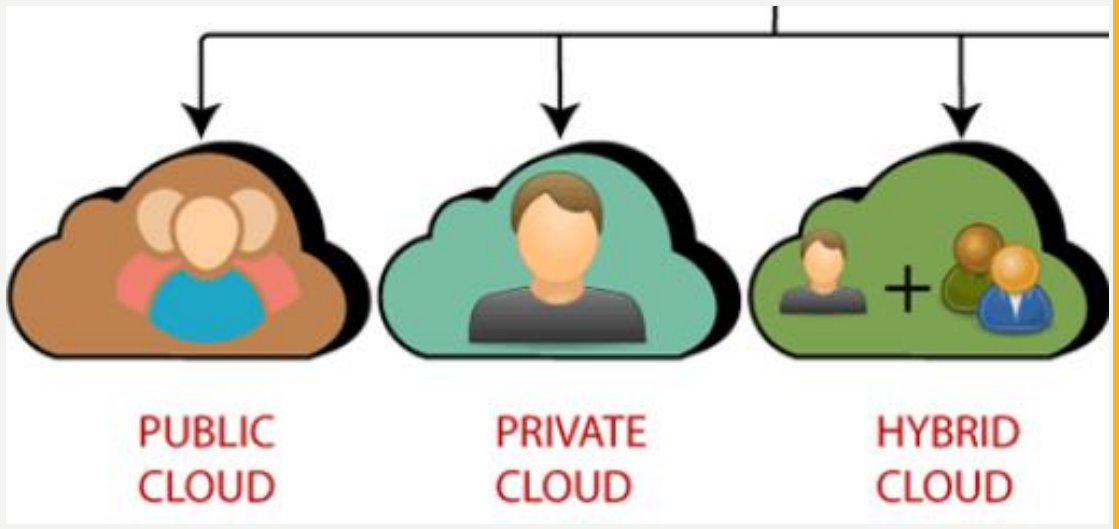
- **Cost savings**
  - The cloud allows you to trade capital expenses (such as data centers and physical servers) for variable expenses, and only pay for IT as you consume it. Plus, the variable expenses are much lower than what you would pay to do it yourself because of the economies of scale.

BENEFITS OF CLOUD COMPUTING

- **Deploy globally in minutes**
  - With the cloud, you can expand to new geographic regions and deploy globally in minutes. For example, AWS has infrastructure all over the world, so you can deploy your application in multiple physical locations with just a few clicks. Putting applications in closer proximity to end users reduces latency and improves their experience.

TYPES OF CLOUD

- Private cloud
  - Public cloud
    - Hybrid cloud



PRIVATE CLOUD

- Private cloud is a cloud computing environment dedicated to a single customer.
- *Private cloud* (also known as an internal cloud or corporate cloud) is a cloud computing environment in which all hardware and software resources are dedicated exclusively to, and accessible only by, a single customer.
- Private cloud combines many of the benefits of cloud computing—including elasticity, scalability, and ease of service delivery—with the access control, security, and resource customization of on-premises infrastructure.

PRIVATE CLOUD

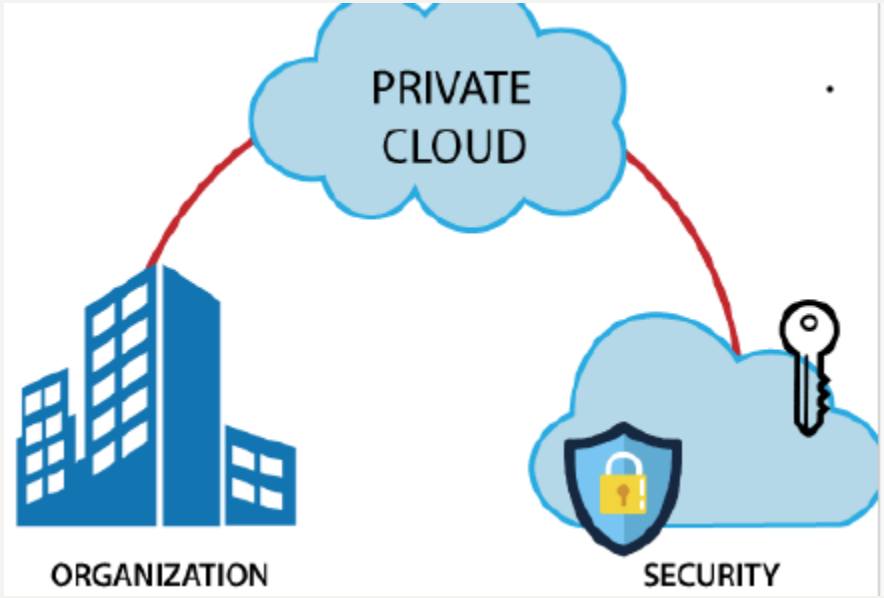
- Many companies choose private cloud over *public cloud* (cloud computing services delivered over infrastructure shared by multiple customers) because private cloud is an easier way (or the only way) to meet their regulatory compliance requirements.
- Others choose private cloud because their workloads deal with confidential documents, intellectual property, personally identifiable information (PII), medical records, financial data, or other sensitive data.

PRIVATE CLOUD

- By building private cloud architecture according to cloud native principles, an organization gives itself the ***flexibility to easily move workloads to public cloud*** or run them within a *hybrid cloud* (mixed public and private cloud) environment whenever they're ready.
- Private cloud is a ***single-tenant*** **environment**, meaning all resources are accessible to one customer only—this is referred to as ***isolated access***.
- Private clouds are typically hosted on-premises in the customer's data center. But, private clouds can also be hosted on an independent cloud provider's infrastructure or built on rented infrastructure housed in an offsite data center.

PRIVATE CLOUD ARCHITECTURE

- Single-tenant design aside, private cloud is based on the same technologies as other clouds—technologies that enable the customer to provision and configure virtual servers and computing resources on demand in order to quickly and easily (or even automatically) scale in response to spikes in usage and traffic, to implement redundancy for high availability, and to optimize utilization of resources overall.



PRIVATE CLOUD ARCHITECTURE

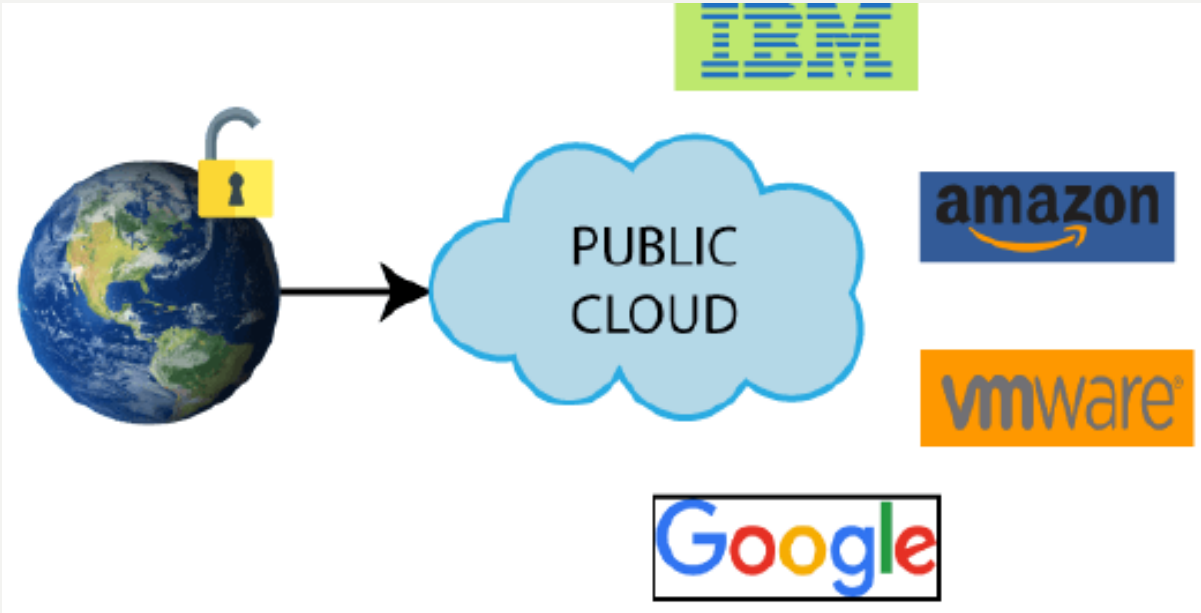
- These technologies include the following:
  - **Virtualization,** which enables IT resources to be abstracted from their underlying physical hardware and pooled into unbounded resource pools of computing, storage, memory, and networking capacity that can then portioned among multiple virtual machines (VMs), containers, or other virtualized IT infrastructure elements.
  - **Management software** gives administrators centralized control over the infrastructure and applications running on it. This makes it possible to optimize security, availability, and resource utilization in the private cloud environment.
  - **Automation** speeds tasks—such as server provisioning and integrations—that would otherwise need to be performed manually and repeatedly. Automation reduces the need for human intervention, making self-service resource delivery possible.

BENEFITS OF PRIVATE CLOUD

- **Full control over hardware and software choices.** Private cloud customers are free to purchase the hardware and software they prefer, vs. the hardware and software the cloud provider offers.
- **Freedom to customize hardware and software in any way.** Private cloud customers can customize servers in any way they want and can customize software as needed with add-ons or through custom development.
- **Greater visibility into security and access control**, because all workloads run behind the customers' own firewall.
- **Fully enforced compliance with regulatory standards**. Private cloud customers aren't forced to rely on the industry and regulatory compliance offered by the cloud service provider.

PUBLIC CLOUD

- The public cloud is defined as computing services offered by third-party providers over the public Internet, making them available to anyone who wants to use or purchase them.
- Public cloud services may be free or offered through a variety of subscription or on-demand pricing schemes, including a pay-per-usage model.



- The main benefits of the public cloud are as follows:
  - a reduced need for organizations to invest in and maintain their own on-premises IT resources;
  - scalability to meet workload and user demands; and
  - fewer wasted resources because customers only pay for what they use.

- Public cloud is an alternative application development approach to traditional on-premises IT architectures. In the basic public cloud computing model, a third-party provider hosts scalable, on-demand IT resources and delivers them to users over a network connection, either over the public internet or a dedicated network.
- The public cloud model encompasses many different technologies, capabilities and features. At its core, however, a public cloud consists of the following key characteristics:

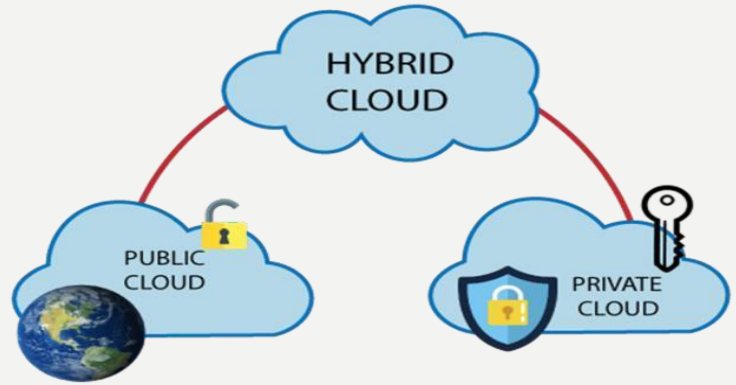
  - on-demand computing and self-service provisioning;
  - resource pooling;
  - scalability and rapid elasticity;
  - pay-per use pricing;
  - measured service(*reference to services where the cloud provider measures or monitors the provision of services for various reasons, including billing, effective use of resources, or overall predictive planning.)*
  - resiliency and availability;
  - security; and
  - broad network access (*People need to be able to get the information they need from anywhere, and from any device.).*
- The public cloud provider supplies the infrastructure needed to host and deploy workloads in the cloud. It also offers tools and services to help customers manage cloud applications, such as data storage, security and monitoring.

- Unlike private clouds, public clouds can save companies from the expensive costs of having to purchase, manage and maintain on-premises hardware and application infrastructure - the cloud service provider is held responsible for all management and maintenance of the system.
- Public clouds can also be deployed faster than on-premises infrastructures and with an almost infinitely scalable platform.
- When selecting a provider, organizations can opt for a large, general-use provider -- such as AWS, Microsoft Azure or Google Cloud Platform (GCP) -- or a smaller provider. General cloud providers offer broad availability and integration options and are desirable for multipurpose cloud needs.

HYBRID CLOUD

- *Hybrid cloud refers to a mixed computing, storage, and services environment made up of* ***on-premises infrastructure, private cloud services, and a public cloud****—such as Amazon Web Services (AWS) or Microsoft Azure—with orchestration among the various platforms. Using a combination of public clouds, on-premises computing, and private clouds in your data center means that you have a hybrid cloud infrastructure.*

HYBRID CLOUD



- Hybrid cloud is a combination of **public and private** clouds. **Hybrid cloud = public cloud + private cloud**
- The main aim to combine these cloud (Public and Private) is to create a unified, automated, and well-managed computing environment.
- In the Hybrid cloud, **non-critical activities** are performed by the **public cloud** and **critical activities** are performed by the **private cloud**.
- Mainly, a hybrid cloud is used in finance, healthcare, and Universities.
- The best hybrid cloud provider companies are **Amazon, Microsoft, Google, Cisco,** and **NetApp**.

ADVANTAGES OF HYBRID CLOUD

- There are the following advantages of Hybrid Cloud -

1) Flexible and secure

- It provides flexible resources because of the public cloud and secure resources because of the private cloud.

2) Cost effective

- Hybrid cloud costs less than the private cloud. It helps organizations to save costs for both infrastructure and application support.
- It offers the features of both the public as well as the private cloud. A hybrid cloud is capable of adapting to the demands that each company needs for space, memory, and system.

DISADVANTAGES OF HYBRID CLOUD

1) Networking issues

– In the Hybrid Cloud, networking becomes complex because of the private and the public cloud.

2) Infrastructure Compatibility

– Infrastructure compatibility is the major issue in a hybrid cloud. With dual-levels of infrastructure, a private cloud controls the company, and a public cloud does not, so there is a possibility that they are running in separate stacks.

3) Reliability

– The reliability of the services depends on cloud service providers.

HYBRID CLOUD SCENARIOS

- **Dynamic or frequently changing workloads.** Use an easily scalable public cloud for your dynamic workloads, while leaving less volatile, or more sensitive, workloads to a private cloud or on-premises data center.
- **Separating critical workloads from less-sensitive workloads.** You might store sensitive financial or customer information on your private cloud, and use a public cloud to run the rest of your enterprise applications.
- **Big data processing.** It's unlikely that you process big data continuously at a near-constant volume. Instead, you could run some of your big data analytics using highly scalable public cloud resources, while also using a private cloud to ensure data security and keep sensitive big data behind your firewall.

- **Moving to the cloud incrementally, at your own pace.** Put some of your workloads on a public cloud or on a small-scale private cloud. See what works for your enterprise, and continue expanding your cloud presence as needed—on public clouds, private clouds, or a mixture of the two.
- **Temporary processing capacity needs.** A hybrid cloud lets you allocate public cloud resources for short-term projects, at a lower cost than if you used your own data center's IT infrastructure. That way, you don't overinvest in equipment you'll need only temporarily.

- **Flexibility for the future.** No matter how well you plan to meet today's needs, unless you have a crystal ball, you won't know how your needs might change next month or next year. A hybrid cloud approach lets you match your actual data management requirements to the public cloud, private cloud, or on-premises resources that are best able to handle them.
- **Best of both worlds.** Unless you have clear-cut needs fulfilled by only a public cloud solution or only a private cloud solution, why limit your options? Choose a hybrid cloud approach, and you can tap the advantages of both worlds simultaneously.

- **Public cloud** is cloud computing that's delivered via the internet and shared across organizations.
- **Private cloud** is cloud computing that is dedicated solely to your organization.
- **Hybrid cloud** is any environment that uses both public and private clouds.

TYPES OF CLOUD COMPUTING

- Three major forms of cloud computing exist. These are:
  - **Infrastructure as a Service (IaaS):** It is the basic cloud service that offers networking services, load balancers, virtual machines, and firewalls services.
  - It includes a method of providing everything from OS to servers and storage via IP-based networking as part of an on-demand service.
  - Some common examples of IaaS are IBM cloud, AWS, and Microsoft Azure.

- **Platform as a Service (PaaS):** A cloud computing service that offers an on-demand platform for software application development, management, testing, and distribution.
- If you use PaaS services, then you don't have to worry about setting up or maintaining the underlying server, network, storage, and database infrastructure required for the development.
- Example of PaaS is Google App Engine, Salesforce.com, etc.

- **Software as a Service (SaaS):** It is a distribution model. Through this service, computer applications (web services) are distributed over the Internet. Users may use a computer or mobile device that has internet connectivity to access SaaS services.
- The most common example of a SaaS is Microsoft Office 365, which provides productivity and email services.

What is the difference between IaaS, PaaS and SaaS?

■ USER'S RESPONSIBILITY ■ SERVICE PROVIDER'S RESPONSIBILITY

	EXAMPLES	APPLICATIONS	MIDDLEWARE	VIRTUALIZATION	DATA	O/S	NETWORKING	RUNTIME	SERVERS	STORAGE
SaaS	Dropbox, Salesforce CRM, Zoom, Office 365, Google Apps	■	■	■	■	■	■	■	■	■
PaaS	Microsoft Azure App Service, AWS Elastic Beanstalk, Google Kubernetes Engine, Red Hat OpenShift	■	■	■	■	■	■	■	■	■
IaaS	Microsoft Azure, Amazon Web Services (AWS), Google Compute Engine (GCE)	■	■	■	■	■	■	■	■	■

INTRODUCTION TO OPENSTACK

- OpenStack is a ***cloud OS*** that is used to **control the large pools of computing, storage, and networking resources within a data center**. OpenStack is an open-source and free software platform. This is essentially used and ***implemented as an IaaS*** for cloud computing.

DESIGNING OPENSTACK CLOUD ARCHITECTURAL CONSIDERATION

- The adoption of cloud technology has changed the way enterprises run their IT services. By leveraging new approaches on how resources are being used, several cloud solutions came into play with different categories: private, public, hybrid, and community.
- Whatever cloud category is used, this trend was felt by many organizations, which needs to introduce an orchestration engine to their infrastructure to embrace elasticity, scalability, and achieve a unique user experience to a certain extent.

- Nowadays, a remarkable orchestration solution, which falls into the private cloud category, has brought thousands of enterprises to the next era of data center generation: **OpenStack**.
- Within every new release, OpenStack brings more great features, which makes it a glorious solution for organizations seeking to invest in it, with returns in operational workloads and flexible infrastructure.

Series	Status	Initial Release Date	Next Phase	EOL Date
Yoga	Development	2022-03-30 *estimated* (schedule)	Maintained *estimated 2022-03-30*	
Xena	Maintained	2021-10-06	Extended Maintenance *estimated 2023-04-06*	
Wallaby	Maintained	2021-04-14	Extended Maintenance *estimated 2022-10-14*	
Victoria	Maintained	2020-10-14	Extended Maintenance *estimated 2022-04-27*	
Ussuri	Maintained	2020-05-13	Extended Maintenance *estimated 2021-11-12*	
Train	Extended Maintenance (see note below)	2019-10-16	Unmaintained *TBD*	
Stein	Extended Maintenance (see note below)	2019-04-10	Unmaintained *TBD*	
Rocky	Extended Maintenance (see note below)	2018-08-30	Unmaintained *TBD*	

Queens	Extended Maintenance (see note below)	2018-02-28	Unmaintained *TBD*	
Pike	Extended Maintenance (see note below)	2017-08-30	Unmaintained *TBD*	
Ocata	Extended Maintenance (see note below)	2017-02-22	Unmaintained *estimated 2020-06-04*	
Newton	End Of Life	2016-10-06		2017-10-25
Mitaka	End Of Life	2016-04-07		2017-04-10
Liberty	End Of Life	2015-10-15		2016-11-17
Kilo	End Of Life	2015-04-30		2016-05-02
Juno	End Of Life	2014-10-16		2015-12-07
Icehouse	End Of Life	2014-04-17		2015-07-02
Havana	End Of Life	2013-10-17		2014-09-30
Grizzly	End Of Life	2013-04-04		2014-03-29
Folsom	End Of Life	2012-09-27		2013-11-19

OPENSTACK - THE NEW DATA CENTER PARADIGM

- Cloud computing is about providing various types of infrastructural services, such as **Software as a Service** (**SaaS**), **Platform as a Service** (**PaaS**), and **Infrastructure as a Service** (**IaaS**).
- The challenge, which has been set by the public cloud is about agility, speed, and self-service.
- Most companies have expensive IT systems, which they have developed and deployed over the years, but they are siloed and need human intervention.
- In many cases, IT systems are struggling to respond to the agility and speed of the public cloud services.

- The traditional data center model and siloed infrastructure might become unsustainable in today's agile service delivery environment. In fact, today's enterprise data center must focus on speed, flexibility, and automation for delivering services to get to the level of next- generation data center efficiency.

- The big move to a software infrastructure has allowed administrators and operators to deliver a fully automated infrastructure within a minute.
- The next-generation data center reduces the infrastructure to a single, big, agile, scalable, and automated unit.
- The end result is a programmable, scalable, and multi-tenant-aware infrastructure.
- This is where OpenStack comes into the picture: it promises the features of a next-generation data center operating system.

INTRODUCING THE OPENSTACK LOGICAL ARCHITECTURE

Figure: *OpenStack Architecture*

INTRODUCING THE OPENSTACK LOGICAL ARCHITECTURE

- Before delving into the OpenStack architecture , we need to refresh or fill gaps and learn more about the basic concepts and usage of each core component.
- In order to get a better understanding on how it works, it will be beneficial to first briefly parse the things, which make it work.
- Various OpenStack services, which work together to provide the cloud experience to the end user. Despite the different services catering to different needs, they follow a common theme in their design that can be summarized as follows:

- Most OpenStack services are ***developed in Python***, which aids rapid development.
- All OpenStack services ***provide REST APIs***. These APIs are the main external communication interfaces for services and are used by the other services or end users.
- The OpenStack service itself may be implemented as different components. The components of a service communicate with each other over the message queue.

Key components and services of OpenStack include:

Nova: Nova is the compute service in OpenStack and is responsible for managing and provisioning virtual machines (VMs). It provides an API for creating, scheduling, and managing instances.

Neutron: Neutron is the networking service that offers network connectivity as a service. It allows users to create and manage networks, routers, subnets, and security groups for their cloud infrastructure.

Cinder: Cinder provides block storage services. It allows users to attach and detach block storage volumes to instances, making it useful for storing data that requires high-performance access.

Swift: Swift is the object storage service, designed for scalable and redundant storage of data and objects. It's suitable for storing unstructured data and can be used for backup and archival purposes.

Glance: Glance is the image service used for storing and managing virtual machine images. It allows users to create, discover, and retrieve VM images, making it easier to deploy instances.

Horizon: Horizon is the web-based dashboard for OpenStack. It provides a user-friendly interface for administrators and users to manage and monitor their cloud resources.

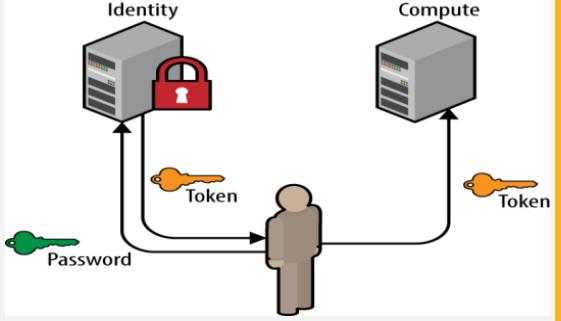
Keystone: Keystone is the identity service that handles authentication and authorization within OpenStack. It manages users, roles, and permissions, ensuring secure access to resources.

Heat: Heat is the orchestration service in OpenStack. It allows users to define templates for provisioning and managing multiple cloud resources as a single stack, making it easier to deploy complex applications.

Ceilometer: Ceilometer provides telemetry and metering services, helping users collect and analyze data related to resource utilization, which is important for billing and monitoring.

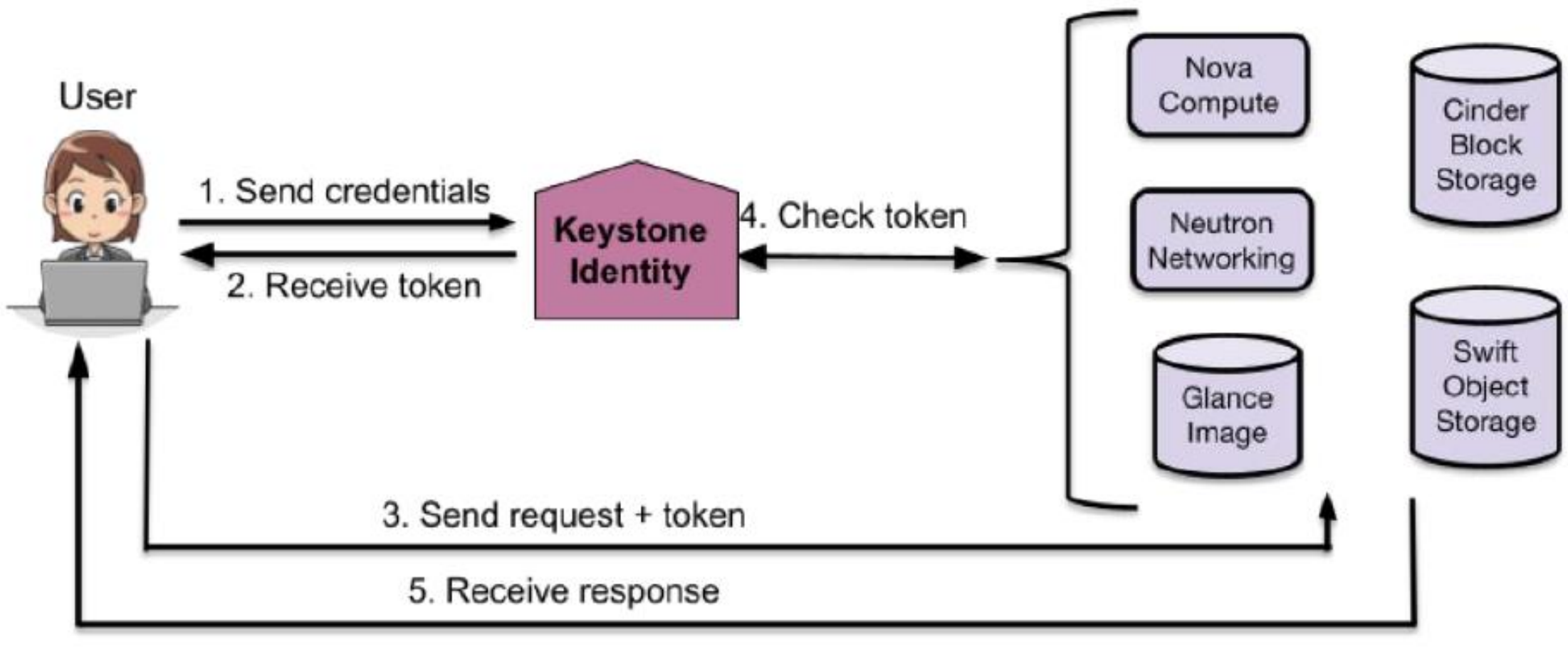
Trove: Trove is the database as a service (DBaaS) component of OpenStack. It simplifies the management of database instances and allows users to easily deploy and scale database services.

Magnum: Magnum is a container orchestration service in OpenStack. It allows users to deploy and manage container clusters using popular container orchestration engines like Docker Swarm, Kubernetes, and Apache Mesos.



Keystone - identity management

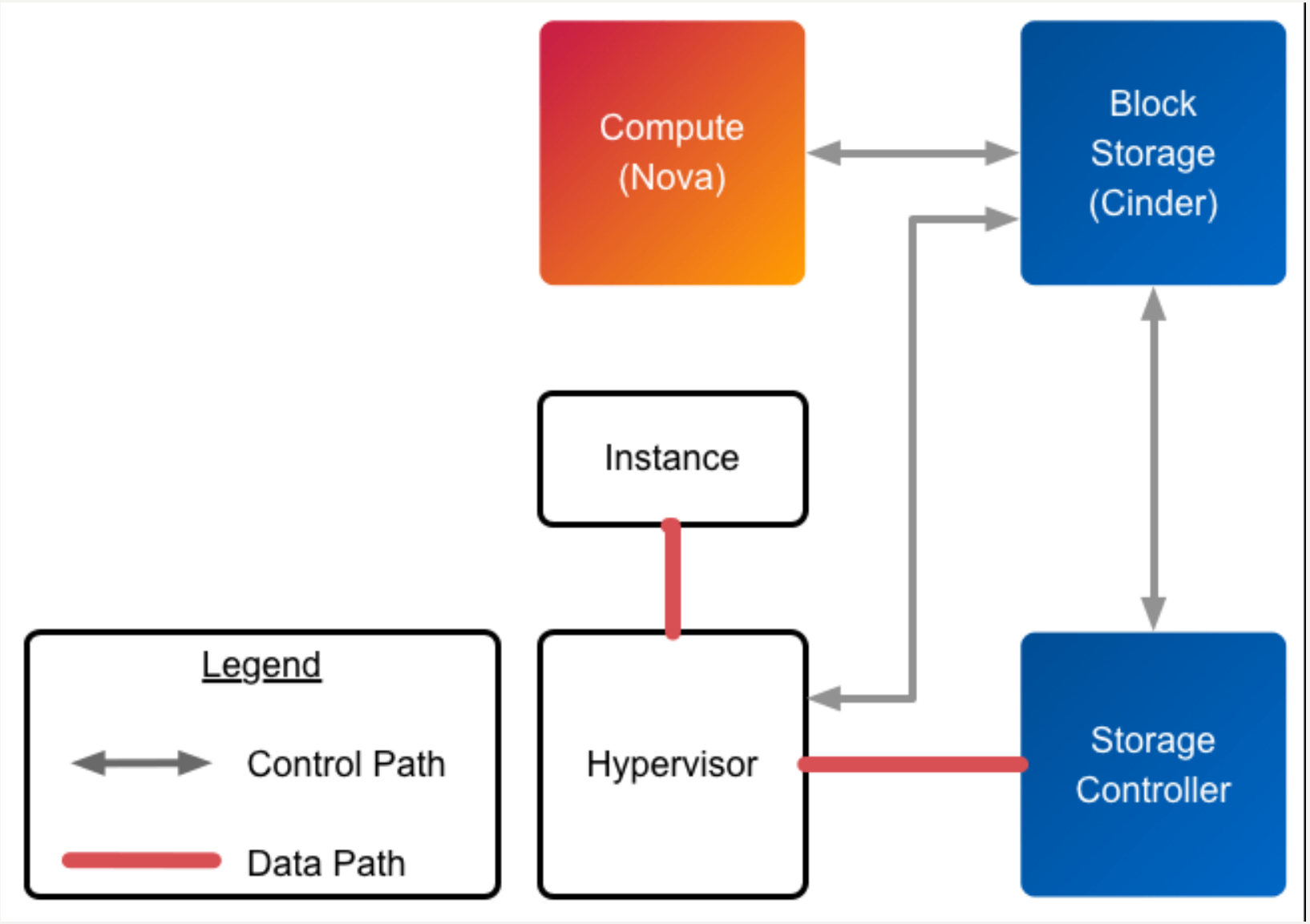
- From an architectural perspective, **Keystone** presents the simplest service in the OpenStack composition.
- It is the ***core component and provides an identity service*** comprising authentication and authorization of tenants in OpenStack.
- Communications between different OpenStack services are authorized by Keystone to ensure that the right user or service is able to utilize the requested OpenStack service.
- Keystone integrates with numerous authentication mechanisms such as username/password and token/authentication-based systems.

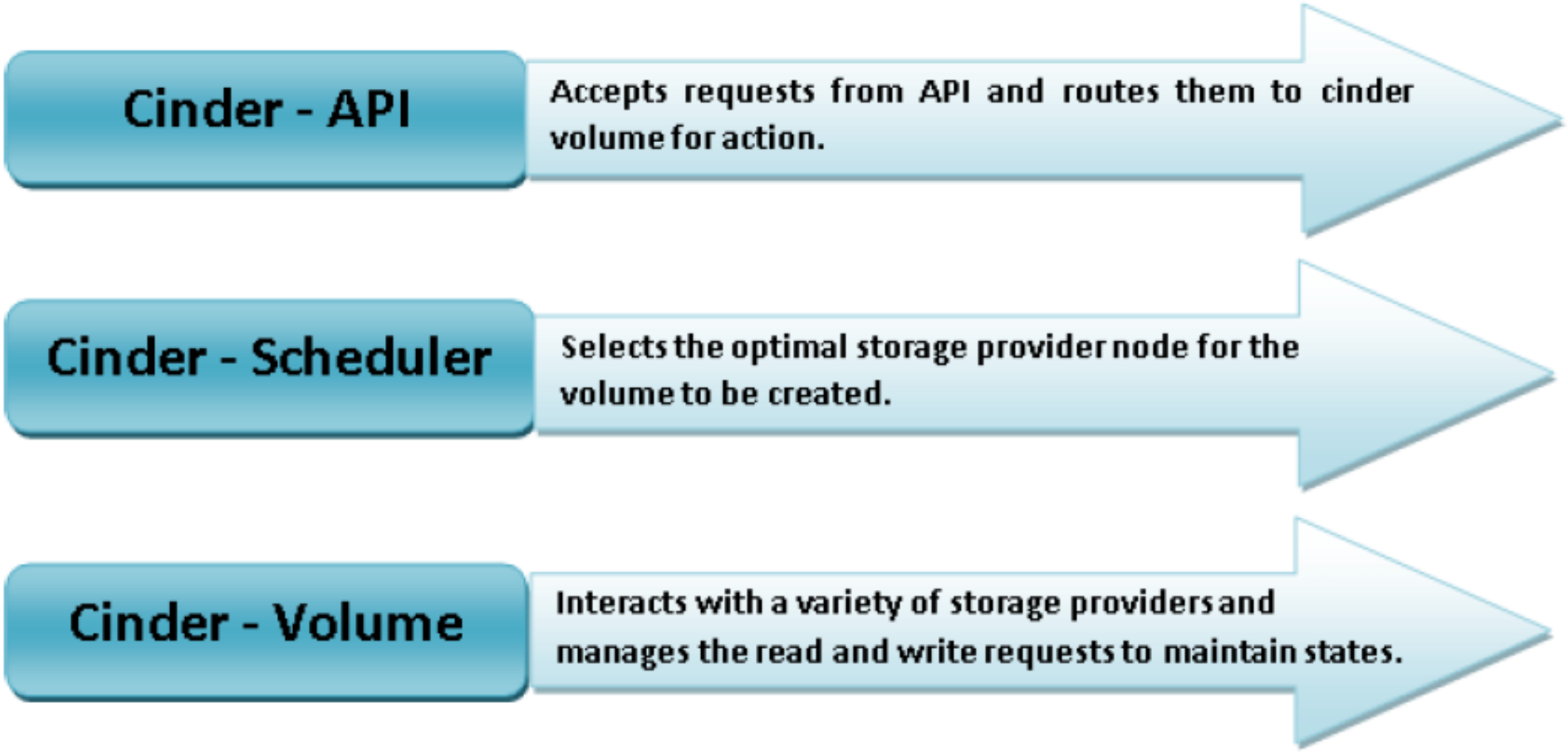

User
1. Send credentials
2. Receive token
Keystone Identity
4. Check token
Nova Compute
Neutron Networking
Glance Image
Cinder Block Storage
Swift Object Storage
3. Send request + token
5. Receive response

- **Swift - object storage**
  - **Swift** is one of the storage services available to OpenStack users. It provides an object-based storage service and is accessible through REST APIs.
  - Compared to traditional storage solutions, file shares, or block-based access, an Object-Storage takes the approach of dealing with stored data as objects that can be stored and retrieved from the Object-Store.
  - A very high-level overview of Object Storage goes like this. To store the data, the Object-Store splits it into smaller chunks and stores it in separate containers.
  - These containers are maintained in redundant copies spread across a cluster of storage nodes to provide high availability, auto-recovery, and horizontal scalability.

- **Swift** has a number of benefits:
  - It has no central brain, and indicates **no Single Point Of Failure (SPOF)**
  - It is curative, and indicates **auto-recovery** in the case of failure.
  - It is **highly scalable** for large petabytes of storage access by scaling horizontally It has a **better performance**, which is achieved by spreading the load over the storage nodes.
  - It has **inexpensive hardware** that can be used for redundant storage clusters.

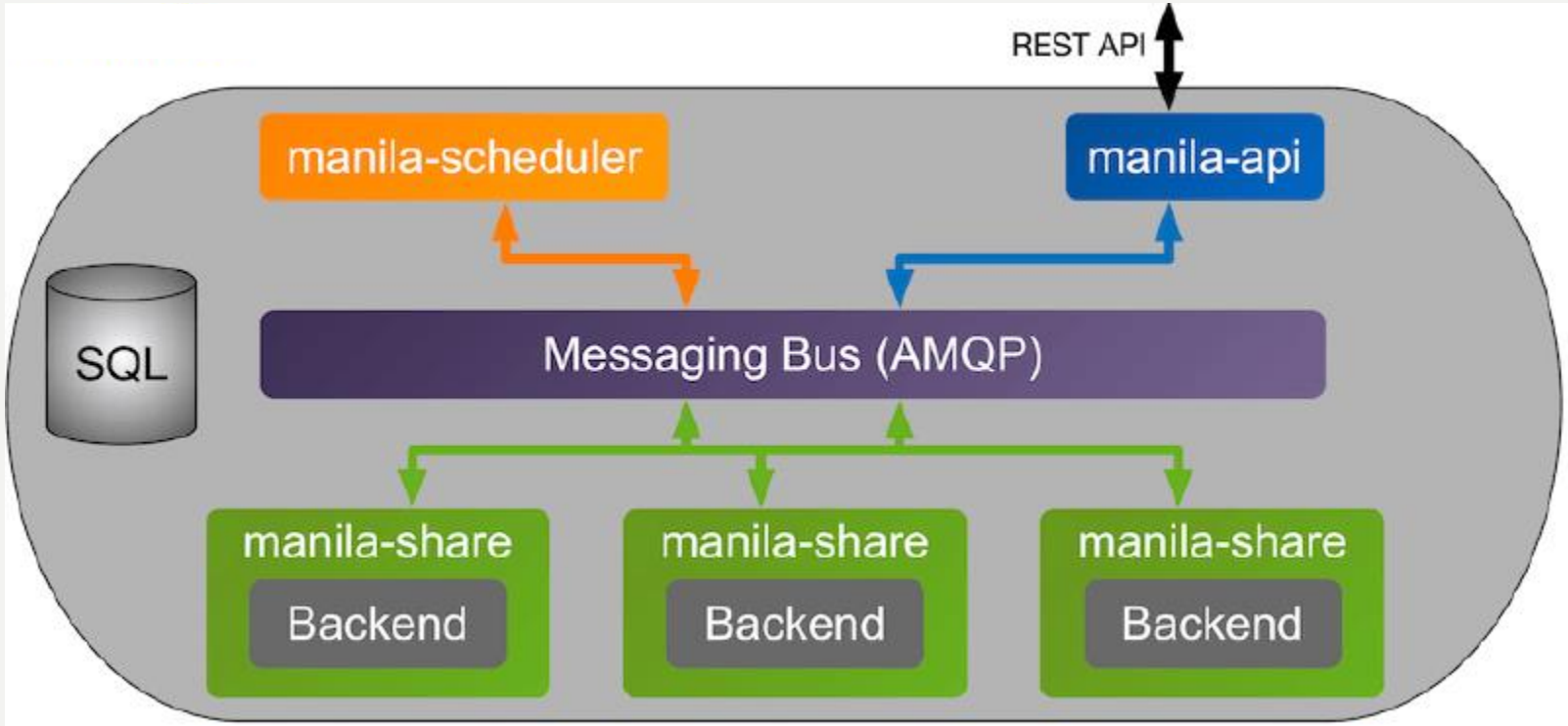
- **Cinder - block storage**
  - The management of the persistent block storage is available in OpenStack by using the Cinder service. Its main capability is to provide block-level storage to the virtual machine.
  - Cinder provides raw volumes that can be used as hard disks in virtual machines.
  - Some of the features that Cinder offers are as follows:
    - **Volume management**: This allows the creation or deletion of a volume.
    - **Snapshot management**: This allows the creation or deletion of a snapshot of volumes.
    - Attaching or detaching volumes from instances Cloning volumes
    - Creating volumes from snapshots
    - Copy of images to volumes and vice versa


Compute
(Nova)
Block
Storage
(Cinder)
Instance
Legend
Control Path
Data Path
Hypervisor
Storage
Controller



Cinder Components are:

- **Manila - File share**
  - Since the **Juno** release, OpenStack has also had a file-share-based storage service called **Manila**.
  - It provides storage as a remote file system. In operation, it resembles the **Network File System** (**NFS)** or **SAMBA** storage service that we are used on Linux.

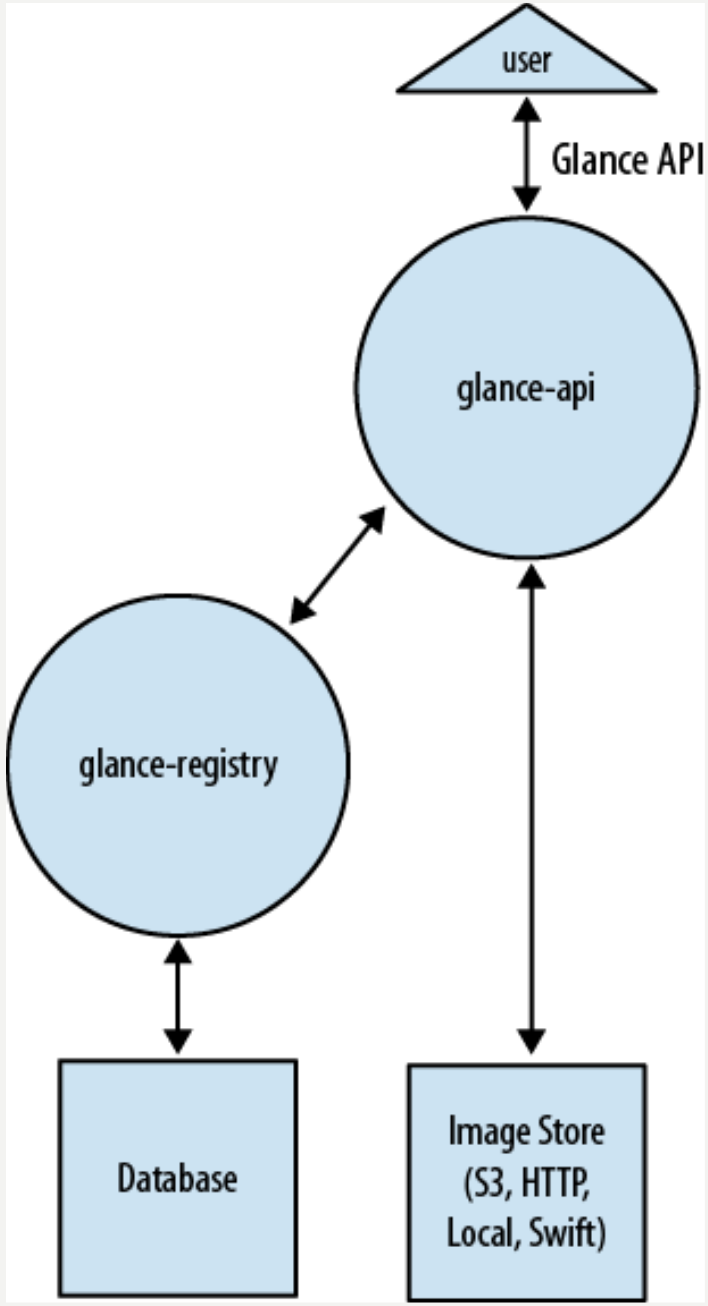


Each storage solution in OpenStack has been designed for a specific set of purposes and implemented for different targets. Before taking any architectural design decisions, it is crucial to understand the difference between existing storage options in OpenStack today, as outlined in the following table:

Specification	**Storage Type**		
	Swift	**Cinder**	**Manila**
Access mode	Objects through REST API	As block devices.	File-based access
Multi-access	OK	No, can only be used by one client	OK
Persistence	OK	OK	OK
Accessibility	Anywhere	Within single VM	Within multiple VMs
Performance	OK	OK	OK

- **Glance - Image registry**
  - **The Glance** service provides a registry of images and metadata that the OpenStack user can launch as a virtual machine. Various image formats are supported and can be used based on the choice of hypervisor. Glance supports images for KVM/Qemu, XEN, VMware, Docker, and so on.
  - What is the difference between Glance and Swift?
    - ***Swift is a storage system, whereas Glance is an image registry***. The difference between the two is that Glance is a service that keeps track of virtual machine images and metadata associated with the images. Metadata can be information such as a kernel, disk images, disk format, and so on. Glance makes this information available to OpenStack users over REST APIs. Glance can use a variety of backends for storing images.
    - Swift, on the other hand, is a storage system. It is designed for object-storage where you can keep data such as virtual disks, images, backup archiving, and so on.

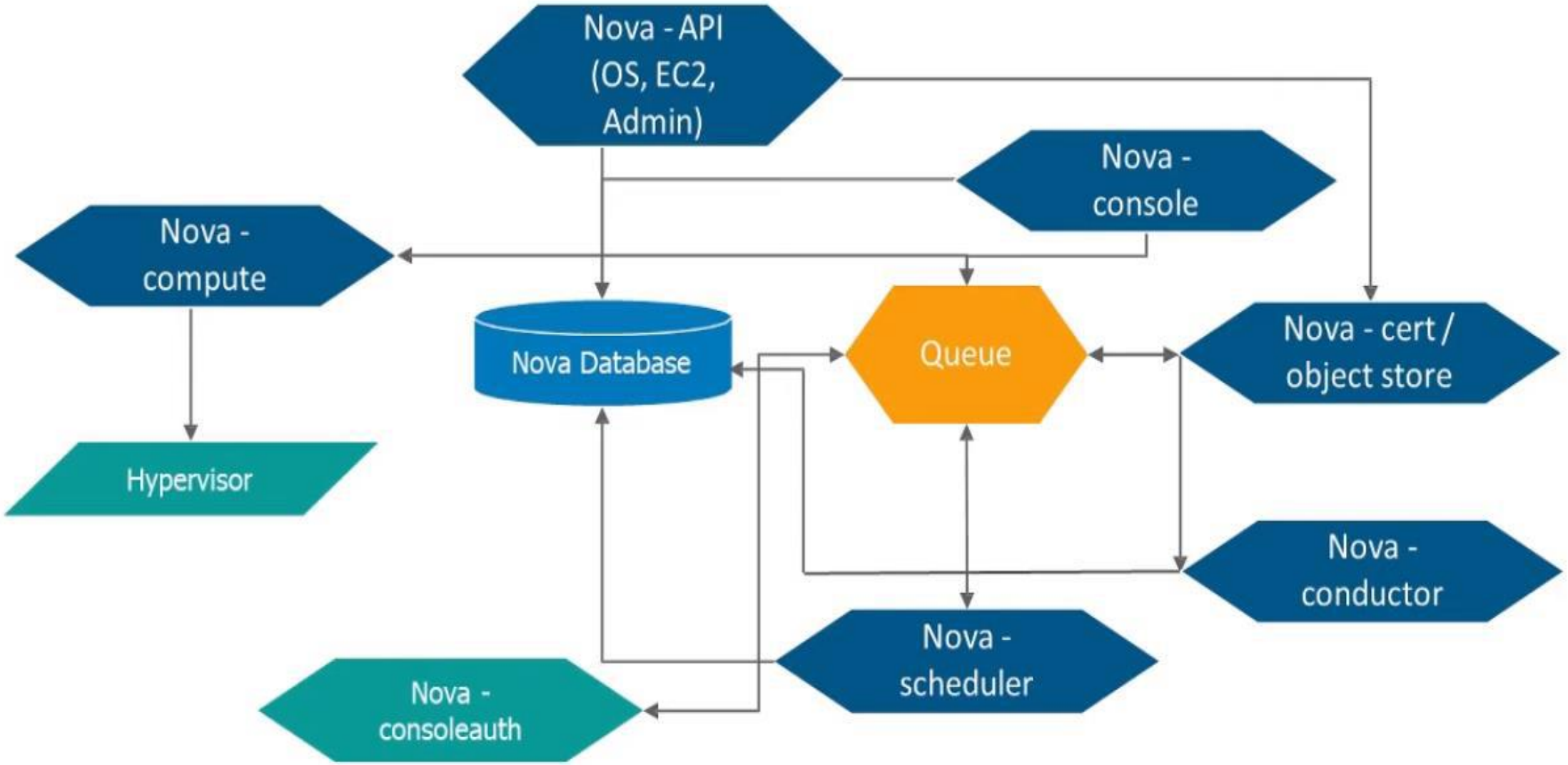
- The mission of Glance is **to be an image registry**. From an architectural point of view, the goal of Glance is to focus on advanced ways **to store and query image information via the Image Service API**. A typical use case for Glance is to allow a client (which can be a user or an external service) to register a new virtual disk image, while a storage system focuses on providing a highly scalable and redundant data store.



NOVA-COMPUTE SERVICE

- **Nova** is the original core component of OpenStack.
- From an architectural level, it is considered one of the most complicated components of OpenStack.
- Nova provides the compute service in OpenStack and manages virtual machines in response to service requests made by OpenStack users.
- What makes Nova complex is its interaction with a large number of other OpenStack services and internal components, which it must collaborate with to respond to user requests for running a VM.
    - ***Let's break down the Nova service itself and look at its architecture as a distributed application that needs orchestration between different components to carry out tasks.***

Nova Architecture



Nova Components

Nova-API

Manages API HTTP requests and is used to interact with Nova

Nova-database

Nova database stores current state of all objects in the compute cluster

Nova (message) Queue

The messaging channel that passes messages between Nova components

Nova-scheduler

It's a daemon which determines on which compute host the instance should run on

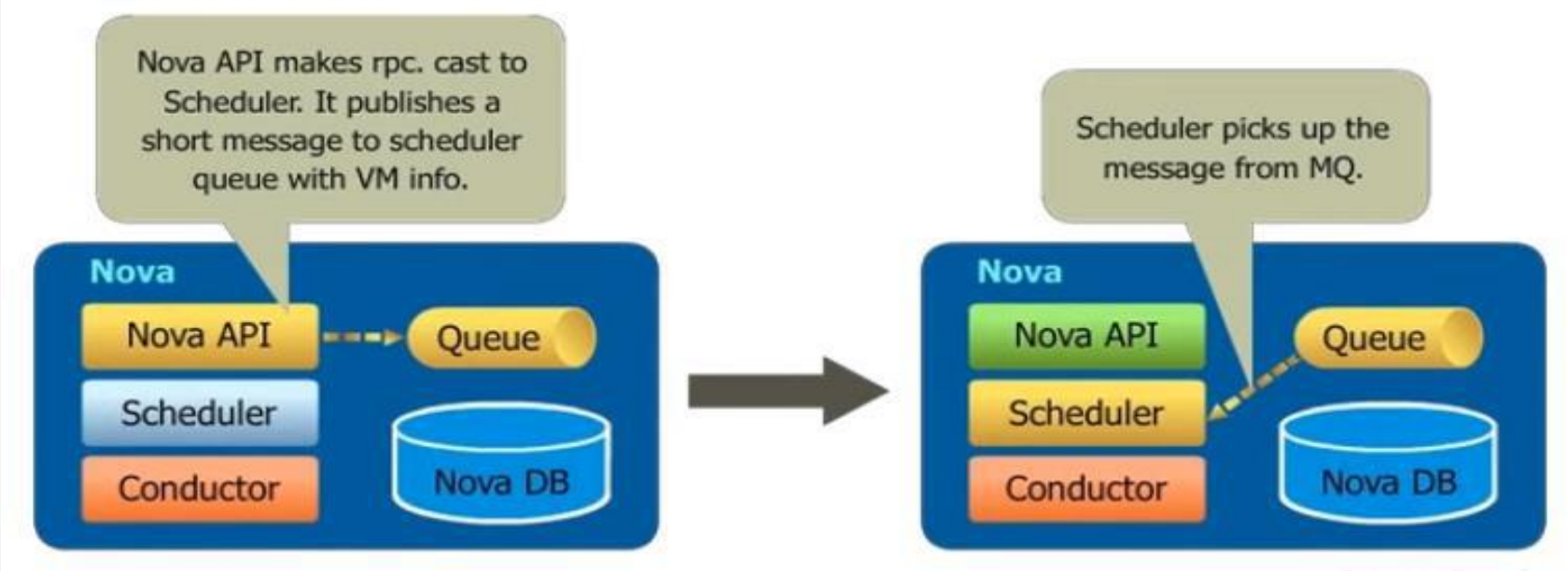
Nova-compute

It is the work daemon which creates and terminates VMs via Hypervisor API

Nova-conductor

It is a service that conducts a no-db-compute function

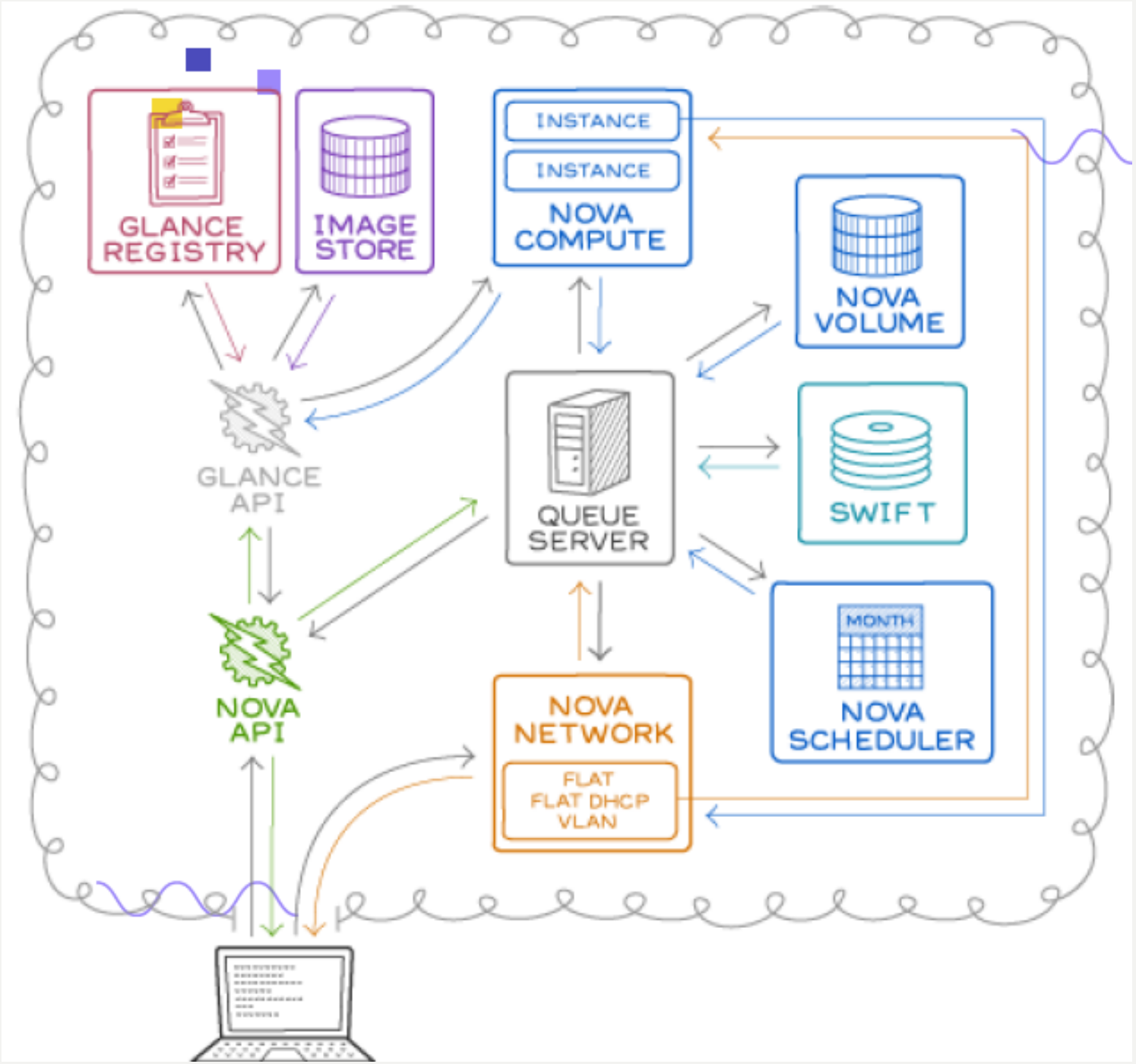
1. **nova-api**
   - The **nova-api** component accepts and responds to the end user and computes API calls.
   - The end users or other components communicate with the OpenStack nova-api interface to create instances via the OpenStack API or EC2 API.
   - The nova-api initiates most orchestrating activities such as the running of an instance or the enforcement of some particular policies.



2. **nova-compute**
   - The **nova-compute** component is primarily a worker daemon that creates and terminates VM instances via the hypervisor's APIs (XenAPI for XenServer, Libvirt KVM, and the VMware API for VMware).
3. **nova-network**
   - The **nova-network** component accepts networking tasks from the queue and then performs these tasks to manipulate the network (such as setting up bridging interfaces or changing IP table rules).
   - **Neutron** is a replacement for the nova-network service.

4. **nova-scheduler**
   - The **nova-scheduler** component takes a VM instance's request from the queue and determines where it should run (specifically which compute host it should run on).
   - At an application architecture level, the term scheduling or scheduler invokes a systematic search for the best outfit for a given infrastructure to improve its performance.
5. **nova-conductor**
   - The **nova-conductor** service provides database access to compute nodes. The idea behind this service is to prevent direct database access from the compute nodes, thus enhancing database security in case one of the compute nodes gets compromised.

- *By zooming out of the general components of OpenStack, we find that Nova interacts with several services such as Keystone for authentication, Glance for images, and Horizon for the web interface. For example, the Glance interaction is central; the API process can upload any query to Glance, while nova-compute will download images to launch instances.*


GLANCE REGISTRY
IMAGE STORE
INSTANCE
INSTANCE
NOVA COMPUTE
NOVA VOLUME
GLANCE API
QUEUE SERVER
SWIFT
NOVA API
MONTH
NOVA SCHEDULER
NOVA NETWORK
FLAT
FLAT DHCP
VLAN

NEUTRON - NETWORKING SERVICES

- Neutron provides a real **Network as a Service** (**NaaS**) capability between interface devices that are managed by OpenStack services such as Nova. There are various characteristics that should be considered for Neutron:
  - It allows users to create their own networks and then attaches server interfaces to them.
  - Its pluggable backend architecture lets users take advantage of vendor-supported equipment.
  - It provides extensions to allow additional network services to be integrated.

- Neutron has many core network features that are constantly growing and maturing. Some of these features are useful for routers, virtual switches, and SDN networking controllers.
- Neutron introduces the following core resources:
  - **Ports**: Ports in Neutron refer to the virtual switch connections. These connections are where instances and network services are attached to networks. When attached to subnets, the defined MAC and IP addresses of the interfaces are plugged into them.

- **Networks**: Neutron defines networks as isolated Layer 2 network segments. Operators will see networks as logical switches that are implemented by the Linux bridging tools, Open vSwitch, or some other virtual switch software.
- **Subnet**: Subnets in Neutron represent a block of IP addresses associated with a network. IP addresses from this block are allocated to the ports.

- Neutron provides additional resources as extensions. The following are some of the commonly used extensions:
  - **Routers**: Routers provide gateways between various networks.
  - **Private IPs**: Neutron defines two types of networks. They are as follows:
    - **Tenant networks**: Tenant networks use private IP addresses. Private IP addresses are visible within the instance and this allows the tenant's instances to communicate while maintaining isolation from the other tenant's traffic. Private IP addresses are not visible to the Internet.

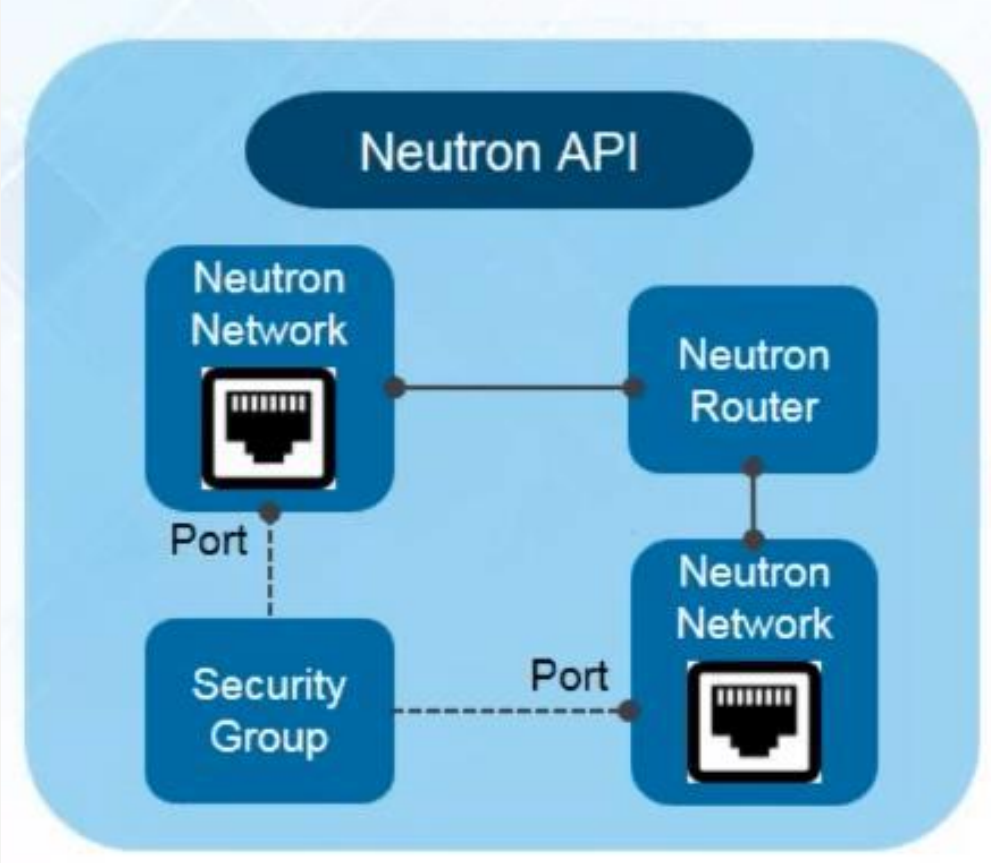
- **External networks**: External networks are visible and routable from the Internet. They must use routable subnet blocks.
- **Floating IPs**: A floating IP is an IP address allocated on an external network that Neutron maps to the private IP of an instance.
- Floating IP addresses are assigned to an instance so that they can connect to external networks and access the Internet. Neutron achieves the mapping of floating IPs to the private IP of the instance by using Network Address Translation (NAT).

- Neutron also provides advanced services to rule additional network OpenStack capabilities as follows:
  - **Load Balancing as a Service (LBaaS)** to distribute the traffic among multiple compute node instances.
  - **Firewall as a Service (FWaaS)** to secure layer 3 and 4 network perimeter access.
  - **Virtual Private Network as a Service (VPNaaS)** to build secured tunnels between instances or hosts.

The Neutron architecture

- The three main components of the Neutron architecture are:
  - **Neutron server**: It accepts API requests and routes them to the appropriate Neutron plugin for action.
  - **Neutron plugins**: They perform the actual work for the orchestration of backend devices such as the plugging in or unplugging ports, creating networks and subnets, or IP addressing.
  - **Neutron agents**: Neutron agents run on the compute and network nodes. The agents receive commands from the plugins on the Neutron server and bring the changes into effect on the individual compute or network nodes. Different types of Neutron agents implement different functionality. (***Eg: L2 and L3 agents***)

- Neutron is a service that manages network connectivity between the OpenStack instances. It ensures that the network will not be turned into a bottleneck or limiting factor in a cloud deployment and gives users real self-service, even over their network configurations.



- Since OpenStack provides infrastructure as a service (IaaS) to end clients, it's necessary to be able **to meter its performance and utilization for billing, benchmarking, scalability, and statistics purposes**.

CEILOMETER, AODH, AND GNOCCHI - TELEMETRY

- Ceilometer, Aodh, and Gnocchi are three related components within the OpenStack ecosystem that work together to provide comprehensive telemetry and metering capabilities. They are designed to collect, store, and analyze data related to resource utilization and performance metrics within an OpenStack cloud environment.

Ceilometer

Role: Ceilometer is the core telemetry component in OpenStack. Its primary purpose is to collect data from various OpenStack services and resources, such as virtual machines, storage volumes, and network resources, to monitor and meter their usage.

Data Collection: Ceilometer collects data through polling mechanisms, event notifications, and other methods. It captures metrics related to CPU usage, memory consumption, network bandwidth, and more.

Data Storage: Ceilometer stores the collected data in a database, typically using backend like MongoDB. This data can then be accessed for analysis, reporting, billing, and other purposes.

Aodh

Role: Aodh is the alarming service in OpenStack, and it works in conjunction with Ceilometer. Its primary function is to define and manage alarms based on the metrics collected by Ceilometer.

Alarm Definition: Aodh allows users to define alarm rules based on threshold conditions or other criteria. For example, an alarm can be set to trigger when CPU usage exceeds a certain percentage for a specified duration.

Alerting: When an alarm condition is met, Aodh can trigger actions, such as sending notifications, executing scripts, or scaling resources up or down automatically.

Integration: Aodh integrates with Ceilometer to utilize the telemetry data collected by Ceilometer for alarm creation and evaluation.

Gnocchi

Role: Gnocchi is the metric storage and retrieval component in OpenStack. It specializes in efficiently storing and querying time-series data, making it well-suited for telemetry use cases.

Data Storage: Gnocchi stores metric data in a more optimized manner compared to traditional databases. It uses a time-series database structure, making it suitable for storing large volumes of metrics efficiently.

Scalability: Gnocchi is designed to be highly scalable, allowing it to handle the massive amount of telemetry data generated in large OpenStack deployments.

Integration: Gnocchi integrates with Ceilometer, serving as the backend for storing and querying the metrics collected by Ceilometer.

Horizon - Dashboard

- Horizon is the name of the web-based graphical user interface (GUI) for the OpenStack cloud platform.
- For example, administrators can use Horizon to **launch virtual machine instances**, view the size and current state of their OpenStack cloud deployment, manage networks, and set limits on the cloud resources available to users. For end users, Horizon acts as a self-service portal to provision cloud resources.
- Horizon is based on a series of modules called **panels** that define the interaction of each service. Its modules can be enabled or disabled, depending on the service availability of the particular cloud. In addition to this functional flexibility, Horizon is easy to style with **Cascading Style Sheets** (**CSS**).

Message Queue

- **Message Queue** provides a central hub to pass messages between different components of a service. This is where information is shared between different daemons by facilitating the communication between discrete processes in an asynchronous way.
- One major advantage of the queuing system is that it can buffer requests and provide unicast and group-based communication services to subscribers.

The database

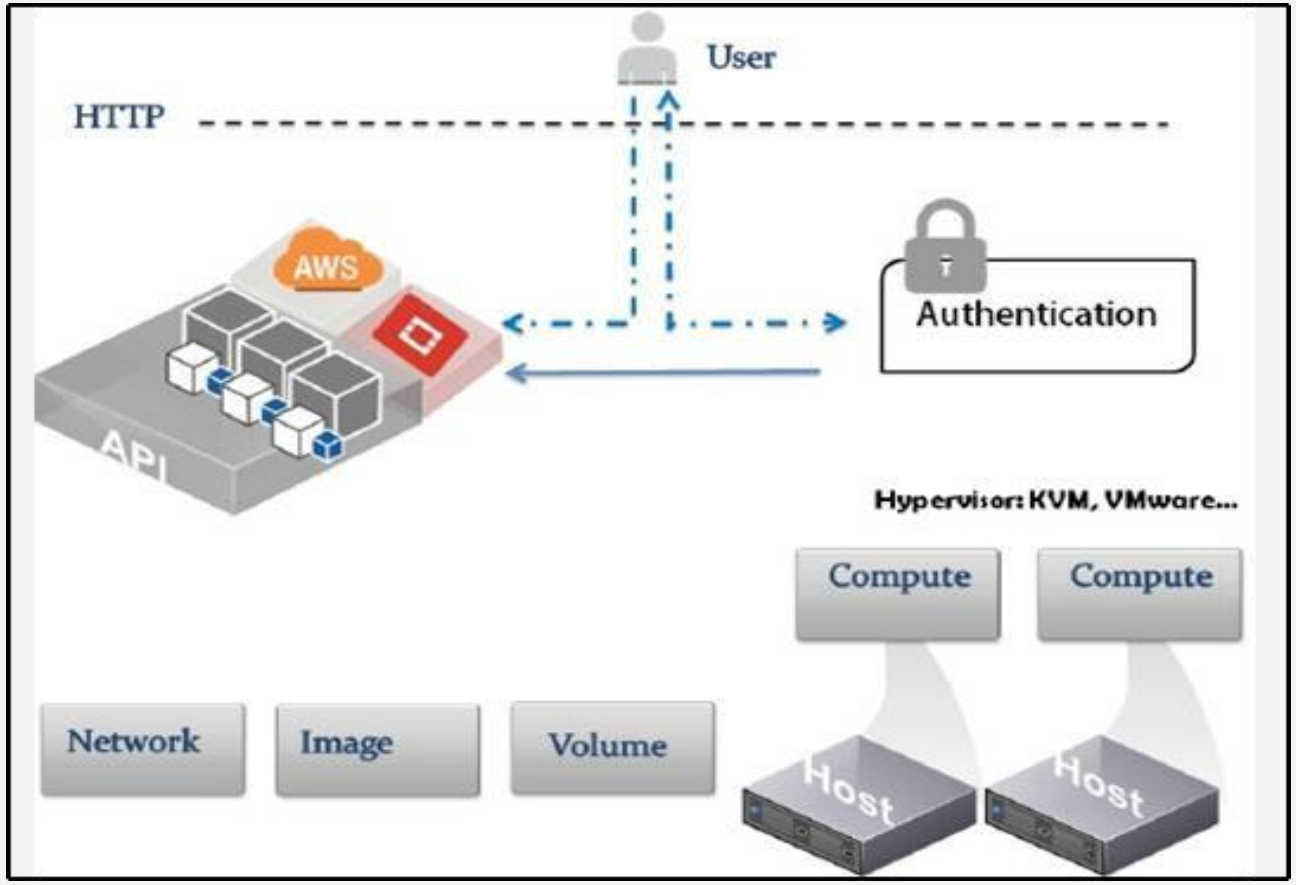
- Its database stores most of the build-time and run-time states for the cloud infrastructure, including instance types that are available for use, instances in use, available networks, and projects. It provides a persistent storage for preserving the state of the cloud infrastructure. It is the second essential piece of sharing information in all OpenStack components.

GATHERING THE PIECES AND BUILDING A PICTURE

- Let's try to see how OpenStack works by chaining all the service cores covered in the previous sections in a series of steps:
  1. Authentication is the first action performed. This is where Keystone comes into the picture. Keystone authenticates the user based on credentials such as the username and password.
  2. The service catalog is then provided by Keystone. This contains information about the OpenStack services and the API endpoints.
  3. You can use the Openstack CLIto get the catalog:
     - **$ oqenstack catalog list**
     - The service catalog is a JSON structure that exposes the resources available on a token request.

4. Typically, once authenticated, you can talk to an API node. There are different APIs in the OpenStack ecosystem (the OpenStack API and EC2 API):

- The following figure shows a high-level view of how OpenStack works:



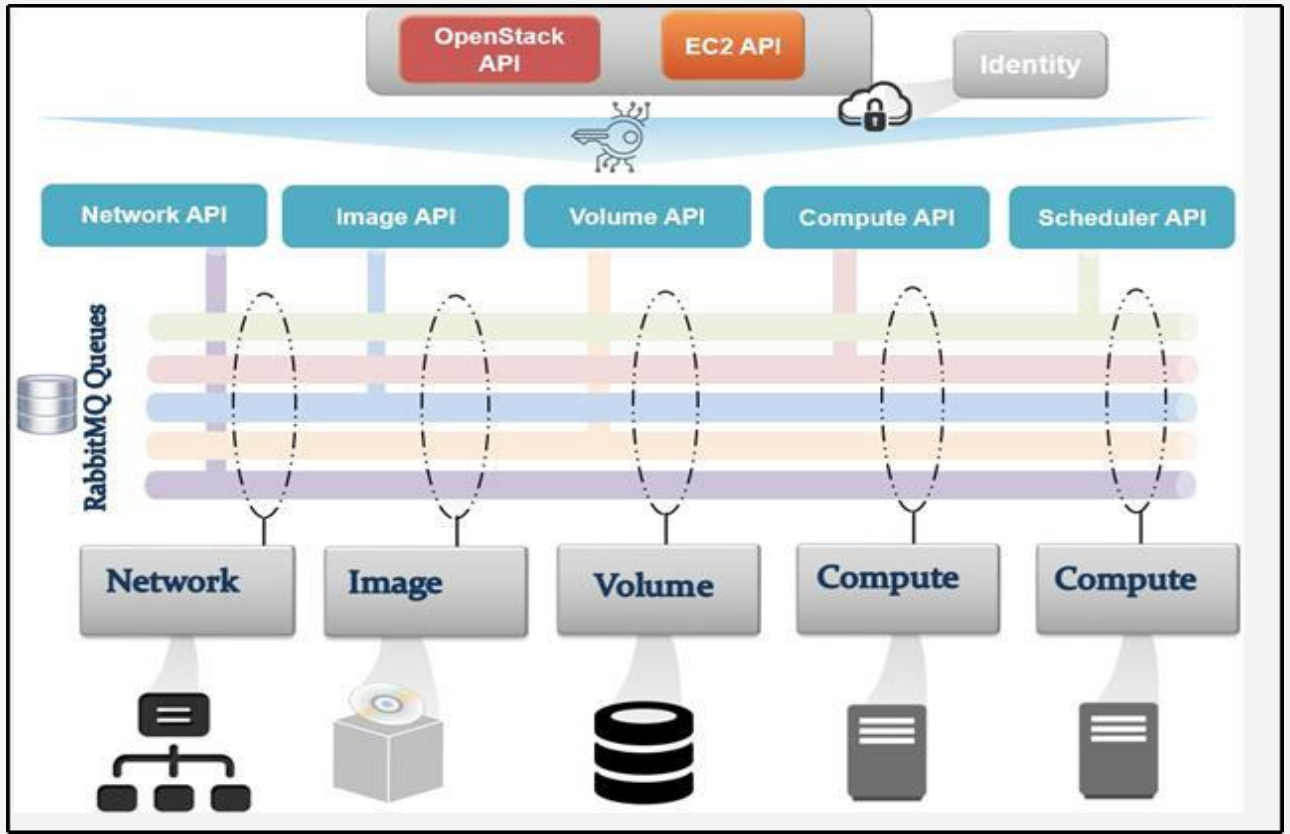
5. Another element in the architecture is the ***instance scheduler***. Schedulers are implemented by OpenStack services that are architected around worker daemons. The worker daemons manage the launching of instances on individual nodes and keep track of resources available to the physical nodes on which they run.

   The scheduler in an OpenStack service looks at the state of the resources on a physical node (provided by the worker daemons) and decides the best candidate node to launch a virtual instance on.

   An example of this architecture is nova- scheduler. This selects the compute node to run a virtual machine or Neutron L3 scheduler, which decides which L3 network node will host a virtual router.

PROVISIONING A VM UNDER THE HOOD

- It is important to understand how different services in OpenStack work together, leading to a running virtual machine. We have already seen how a request is processed in OpenStack via APIs.
- Let's figure out how things work by referring to the following simple architecture diagram:



- The process of launching a virtual machine involves the interaction of the main OpenStack services that form the building blocks of an instance including compute, network, storage, and the base image.
- As shown in the previous diagram, OpenStack services interact with each other via a message bus to submit and retrieve RPC calls. The information of each step of the provisioning process is verified and passed by different OpenStack services via the message bus.
- From an architecture perspective, sub system calls are defined and treated in OpenStack API endpoints involving: Nova, Glance, Cinder, and Neutron.
- On the other hand, the inter-communication of APIs within OpenStack requires an authentication mechanism to be trusted, which involves Keystone.

Provisioning workflow based on API calls in OpenStack:

1. Calling the identity service for authentication
2. Generating a token to be used for subsequent calls
3. Contacting the image service to list and retrieve a base image
4. Processing the request to the compute service API
5. Processing compute service calls to determine security groups and keys
6. Calling the network service API to determine available networks
7. Choosing the hypervisor node by the compute scheduler service
8. Calling the block storage service API to allocate volume to the instance
9. Spinning up the instance in the hypervisor via the compute service API call
10. Calling the network service API to allocate network resources to the instance

- It is important to keep in mind that handling tokens in OpenStack on every API call and service request is a time limited operation. One of the major causes of a failed provisioning operation in OpenStack is the expiration of the token during subsequent API calls.
- Additionally, the management of tokens has faced a few changes within different OpenStack releases. This includes two different approaches used in OpenStack prior to the Liberty release including:
  - Universally Unique Identifier (UUID): Within Keystone version 2, an UUID token will be generated and passed along every API call between client services and back to Keystone for validation. This version has proven performance degradation of the identity service.
  - Public Key Infrastructure (PKI): Within Keystone version 3, tokens are no longer validated at each API call by Keystone. API endpoints can verify the token by checking the Keystone signature added when initially generating the token.

- Starting from the Kilo release, handling tokens in Keystone has progressed by introducing more sophisticated cryptographic authentication token methods, such as ***Fernet.***
- The new implementation will help to tackle the token performance issue noticed in UUID and PKI tokens.
- Fernet is fully supported in the Mitaka release and the community is pushing to adopt it as the default. On the other hand, PKI tokens are deprecated in favor of Fernet tokens in further releases of Kilo OpenStack.

- **Benefits of Fernet tokens**
- Because Fernet tokens are ephemeral, you have the following immediate benefits:
- Tokens do not need to be replicated to other instances of Keystone in your controller cluster
- Storage is not affected, as these tokens are not stored
- The end-result offers increased performance overall. This was the design imperative of Fernet tokens, and the OpenStack community has more than delivered.

A SAMPLE ARCHITECTURE SETUP

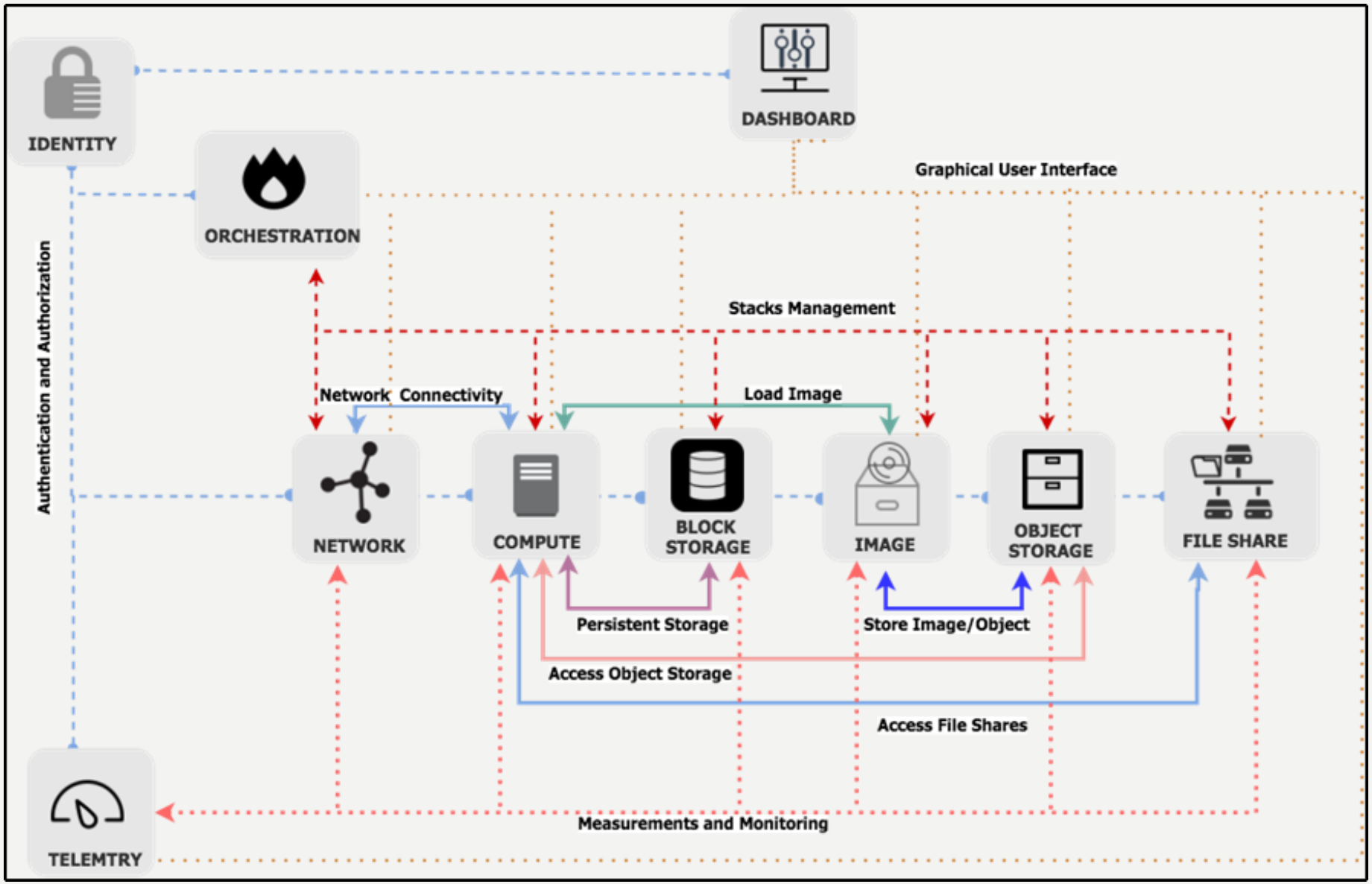
- Many enterprises have successfully designed their OpenStack environments by going through three phases of design: ***designing a conceptual model, designing a logical model, and finally, realizing the physical design.***
- It's obvious that complexity increases from the conceptual to the logical design and from the logical to the physical design.

The conceptual model design

- As the first conceptual phase, we will have our high-level reflection on what we will need from certain generic classes from the OpenStack architecture:

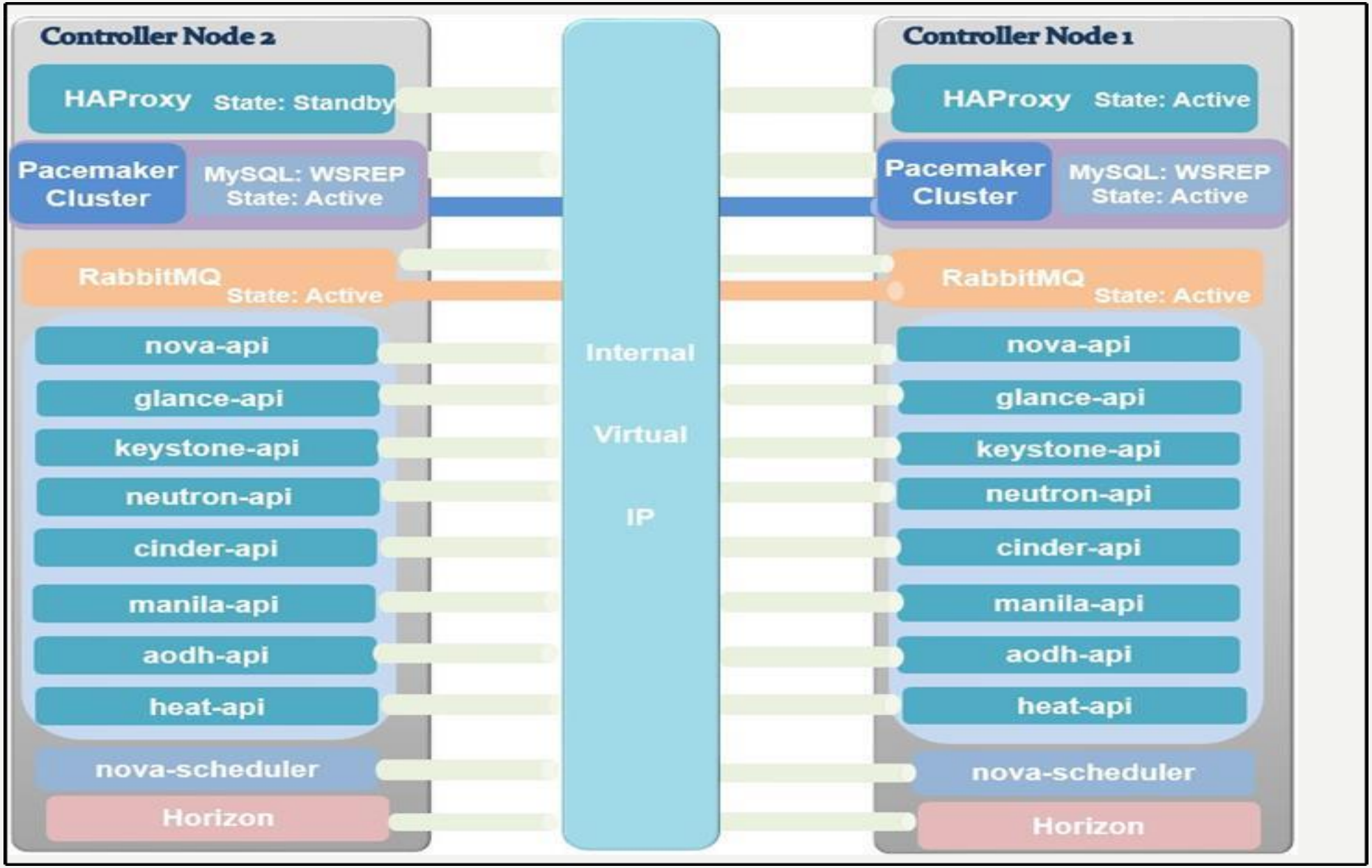
Class	Role
Compute	Stores virtual machine images Provides a user interface
Image	Stores disk files Provides a user interface
Object storage	Stores objects Provides a user interface
Block storage	Provides volumes Provides a user interface
Network	Provides network connectivity Provides a user interface
Telemetry	Provides measurements, metrics, and alerts Provides a user interface
File Share	Provides a scale-out file share system for OpenStack Provides a user interface
Identity	Provides authentication
Dashboard	Provides a graphical user interface
Orchestration	Provides orchestration engine for stack creation Provides a user interface

- Let's map the generic basic classes in the following simplified diagram:



The logical model design

- Based on the conceptual reflection design, most probably you will have a good idea about different OpenStack core components, which will lay the formulation of the logical design.
- We will start by identifying nodes to run an OpenStack service: the cloud controller, network nodes, and the compute node.
- Thus, the physical model design will be elaborated based on the previous theoretical phases by assigning parameters and values to our design. Let's start with our first logical iteration:
- Obviously, in a highly available setup, we should achieve a degree of redundancy in each service within OpenStack.


Controller Node 2
HAProxy State: Standby
Pacemaker Cluster
MySQL: WSREP State: Active
RabbitMQ State: Active
nova-api
glance-api
keystone-api
neutron-api
cinder-api
manila-api
aodh-api
heat-api
nova-scheduler
Horizon
Internal Virtual IP
Controller Node 1
HAProxy State: Active
Pacemaker Cluster
MySQL: WSREP State: Active
RabbitMQ State: Active
nova-api
glance-api
keystone-api
neutron-api
cinder-api
manila-api
aodh-api
heat-api
nova-scheduler
Horizon

- What about high availability?
  - The previous figure includes several essential solutions for a highly- scalable and redundant OpenStack environment such as virtual IP (VIP), HAProxy, and Pacemaker.
- Compute nodes are relatively simple as they are intended just to run the virtual machine's workload. In order to manage the VMs, the nova-compute service can be assigned for each compute node.
- Besides, we should not forget that the compute nodes will not be isolated; a Neutron agent and an optional Ceilometer compute agent may run these nodes.
- Network nodes will run Neutron agents for DHCP, and L3 connectivity.

What about storage..?

- You will have to ask yourself questions such as: *How much data do I need to store? Will my future use cases result in a wide range of applications that run heavy-analysis data? What are my storage requirements for incrementally backing up a virtual machine's snapshots?*
- *Do I really need control over the filesystem on the storage or is just a file share enough? Do I need a shared storage between VMs?*
- Many will ask the following question: *If one can be satisfied by ephemeral storage, why offer block/share storage?* To answer this question, you can think about ephemeral storage as the place where the end user will not be able to access the virtual disk associated with its VM when it is terminated. Ephemeral storage should mainly be used in production when the VM state is non-critical, where users or application don't store data on the VM. If you need your data to be persistent, you must plan for a storage service such as Cinder or Manila.

- Remember that the current design applies for medium to large infrastructures. Ephemeral storage can also be a choice for certain users;
  - for example, when they consider building a test environment.
  - Considering the same case for Swift, we claimed previously that object storage might be used to store machine images, but when do we use such a solution?
    - Simply put, when you have a sufficient volume of critical data in your cloud environment and start to feel the need for replication and redundancy.

The logical networking design

- OpenStack has moved from simplistic network features to more complicated ones, but of course the reason is that it offers more flexibility! This is why OpenStack is here. It brings as much flexibility as it can! Without taking any random network-related decisions, let's see which network modes are available. We will keep on filtering until we hit the first correct target topology:
- The preceding table shows a simple differentiation between two different logical network designs for OpenStack. Every mode shows its own requirements: this is very important and should be taken into consideration before the deployment.
- Arguing about our example choice, since we aim to deploy a very flexible, large-scale environment we will toggle the Neutron choice for networking management instead of nova-network.
- Note that it is also possible to keep on going with nova-network, but you have to worry about any **Single Point Of Failure** (**SPOF**) in the infrastructure. The choice was made for Neutron, since we started from a basic network deployment.

Network mode	**Network Characteristics**	**Implementation**
nova-network	Flat network design without tenant traffic isolation	nova-network Flat DHCP
	Isolated tenants traffic and predefined fixed private IP space size Limited number of tenant networks (4K VLANs limit)	nova-network VLANManager
Neutron	Isolated tenants traffic Limited number of tenant networks (4K VLANs limit)	Neutron VLAN
	Increased number of tenant networks Increased packet size Lower performance	Neutron tunneled networking (GRE, VXLAN, and so on)

- **Physical network layout**
  - We will start by looking at the physical networking requirements of the cloud.
    - **The tenant data network**
      - The main feature of a data network that it provides the physical path for the virtual networks created by the OpenStack tenants. It separates the tenant data traffic from the infrastructure communication path required for the communication between the OpenStack component itself.
    - **Management and the API network**
      - In a smaller deployment, the traffic for management and communication between the OpenStack components can be on the same physical link. This physical network provides a path for communication between the various OpenStack components such as REST API access and DB traffic, as well as for managing the OpenStack nodes.
      - For a production environment, the network can be further subdivided to provide better isolation of traffic and contain the load on the individual networks.

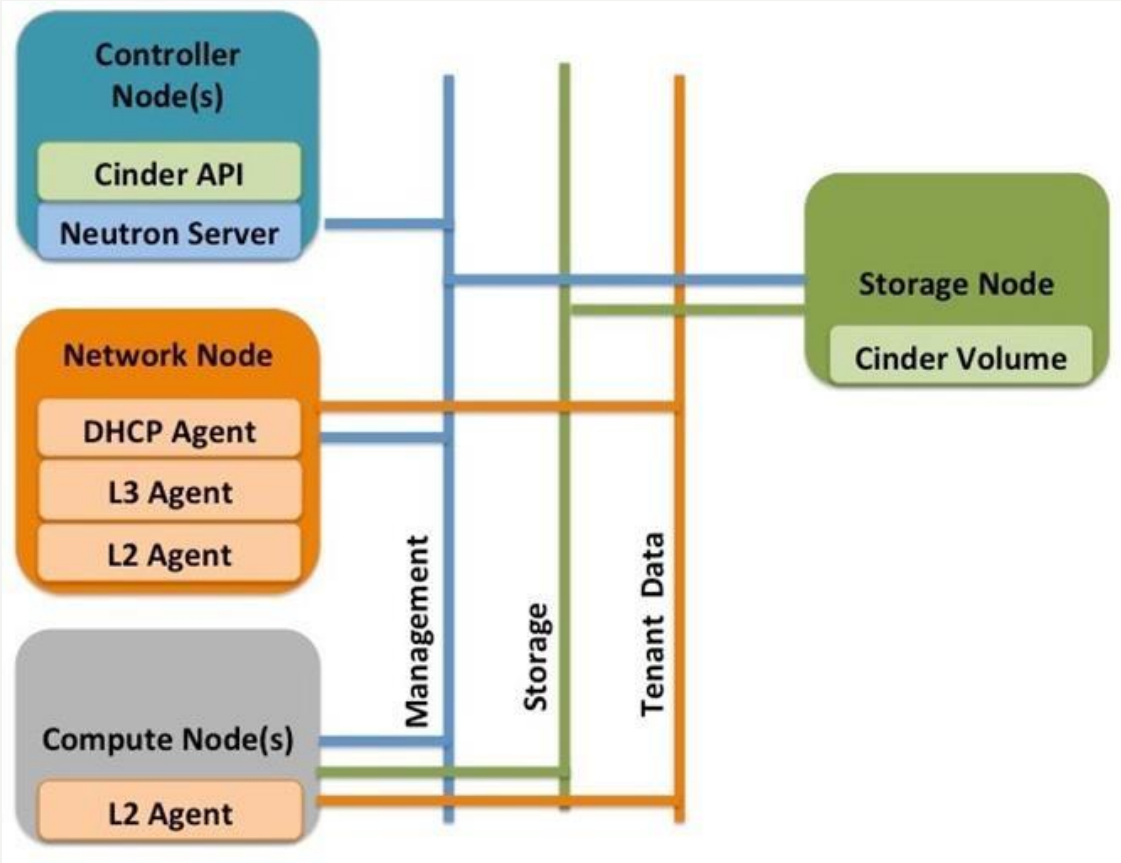
- **The Storage network**
  - The storage network provides physical connectivity and isolation for storage-related traffic between the VMs and the storage servers. As the traffic load for the storage network is quite high, it is a good idea to isolate the storage network load from the management and tenant traffic.

- **Virtual Network types**
  - Let's now look at the virtual network types and their features.
    - The external network
      - The features of an external or a public network are as follows:
      - It provides global connectivity and uses routable IP addressing
      - It is used by the virtual router to perform SNAT from the VM instances and provide external access to traffic originating from the VM and going to the Internet [**SNAT** refers to **Source Network Address Translation**. It allows traffic from a private network to go out to the Internet. OpenStack supports SNAT through its Neutron APIs for routers.]
      - It is used to provide a DNAT service for traffic from the Internet to reach a service running on the VM instance

The tenant networks

- The features of the tenant network are as follows:
- It provides a private network between virtual machines It uses private IP space
- It provides isolation of tenant traffic and allows multi-tenancy requirements for networking services

The next step is to validate our network design in a simple diagram: ➔



The physical model design

- You have to consider the fact that hardware commodity selection will accomplish the mission of our *massive scalable architecture*.
- ***Estimating the hardware capabilities***
  - Since the architecture is being designed to scale horizontally, we can add more servers to the setup. We will start by using commodity class, cost-effective hardware.
  - In order to expect our infrastructure economy, it would be great to make some basic hardware calculations for the first estimation of our exact requirements.
  - Considering the possibility of experiencing contentions for resources such as CPU, RAM, network, and disk, you cannot wait for a particular physical component to fail before you take corrective action, which might be more complicated.
  - **Mean Time To Repair (MTTR)**

CPU calculations

- The following are the calculation-related assumptions:
  - 200 virtual machines
  - No CPU oversubscribing
  - GHz per physical core = 2.6 GHz
  - Physical core hyper-threading support = use factor 2
  - GHz per VM (AVG compute units) = 2 GHz
  - GHz per VM (MAX compute units) = 16 GHz
  - Intel Xeon E5-2648L v2 core CPU = 10
  - CPU sockets per server = 2

- The formula for calculating the total number of CPU cores is as follows:
  - **(number of VMs x number of GHz per VM) / number of GHz per core**
  - **(200 * 2) / 2.6 = 153.846**
  - We have 153 CPU cores for 200 VMs.
- The formula for calculating the number of core CPU sockets is as follows:
  - **Total number of sockets / number of core CPU**
  - **153 / 10 = 15.3**
  - We will need 15 sockets
- The formula for calculating the number of socket servers is as follows:
  - **Total number of sockets / Number of sockets per server**
  - **15 / 2 = 7.5**
  - You will need around seven to eight dual socket servers.
- The number of virtual machines per server with eight dual socket servers is calculated as follows:
  - **We can deploy 25 virtual machines per server**
  - ***200 / 8 = 25***
  - **Number of virtual machines / number of servers**

Memory calculations

- Based on the previous example, 25 VMs can be deployed per compute node. Memory sizing is also important to avoid making unreasonable resource allocations.
- Let's make an assumption list (keep in mind that it always depends on your budget and needs):
  - **2 GB RAM per VM**
  - **8 GB RAM maximum dynamic allocations per VM**
  - **Compute nodes supporting slots of: 2, 4, 8, and 16 GB sticks**
  - RAM available per compute node:
    - **8 ´ 25 = 200 GB**
- Considering the number of sticks supported by your server, you will need around 256 GB installed. Therefore, the total number of RAM sticks installed can be calculated in the following way:
  - **Total available RAM / MAX Available RAM-Stick size**
  - **256 / 16 = 16**

Network calculations

- To fulfill the plans that were drawn for reference, let's have a look at our assumptions:
  - **200 Mbits/second is needed per VM**
  - **Minimum network latency**
- To do this, it might be possible to serve our VMs by using a 10 GB link for each server, which will give:
- **10,000 Mbits/second / 25VMs =400 Mbits/second**

Storage calculations

- Considering the previous example, you need to plan for an initial storage capacity per server that will serve 25 VMs each.
- A simple calculation, assuming 100 GB ephemeral storage per VM, will require a space of
- **25*100 = 2.5 TB of local storage on each compute node.**
- You can assign 250 GB of persistent storage per VM to have 25*250 = 5 TB of persistent storage per compute node.
- Most probably, you have an idea about the replication of object storage in OpenStack, which implies the usage of three times the required space for replication.
- In other words, if you are planning for X TB for object storage, your storage requirement will be 3X.
- Other considerations, such as the best storage performance using SSD, can be useful for a better throughput where you can invest more boxes to get an increased IOPS.
- For example, working with SSD with 20K IOPS installed in a server with eight slot drives will bring you:
- **(20K * 8) / 25 = 6.4 K Read IOPS and 3.2K Write IOPS**