OpenStack cluster - Controller and common services

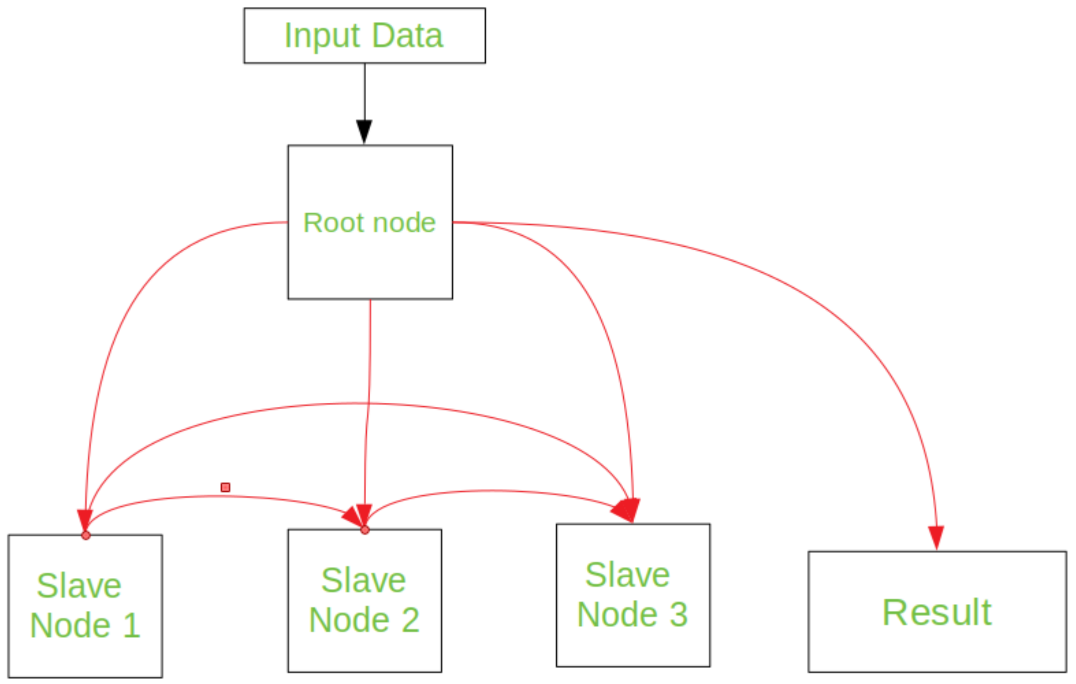
Module II

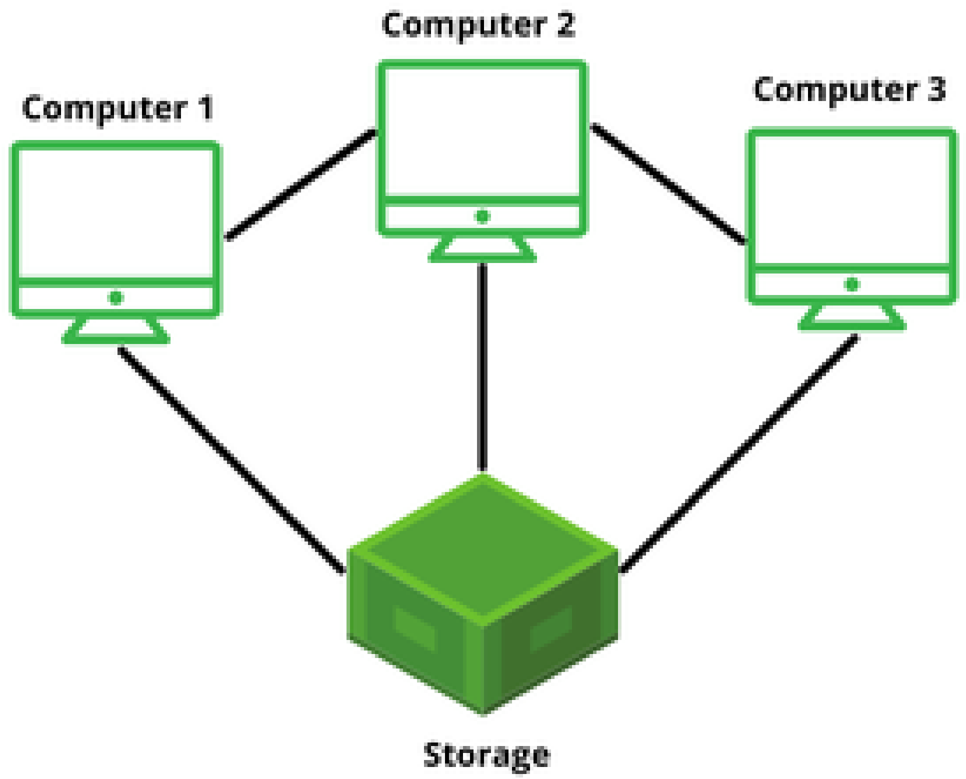
Content

OpenStack Cluster

Cluster

- A cluster is a set of loosely or tightly connected computers working together as a unified computing resource that can create the illusion of being one machine. Computer clusters have each node set to perform the same task, controlled and produced by the software.
- Group of nodes hosted on virtual machines and connected within a virtual private cloud
- Cluster computing is a collection of tightly or loosely connected computers that work together so that they act as a single entity. The connected computers execute operations all together thus creating the idea of a single system. The clusters are generally connected through fast local area networks (LANs)
- A cluster is defined as a collection of homogeneous objects. The “homogeneous” here means that the objects managed (aka. Nodes) have to be instantiated from the same “profile type”.
- A cluster can contain zero or more nodes.





Hardware Cluster helps in enable high-performance disk sharing between systems, while the **Software Cluster** allows all systems to work together.

OpenStack Cluster

- Clusters are first-class citizens in Senlin service design. A cluster is defined as a collection of homogeneous objects. The “homogeneous” here means that the objects managed (aka. Nodes) have to be instantiated from the same “profile type”.
- A cluster can contain zero or more nodes. Senlin provides REST APIs for users to create, retrieve, update, delete clusters. Using these APIs, a user can manage the node membership of a cluster.
- A cluster is owned by a user (the owner), and it is accessible from within the Keystone project (tenant) which is the default project of the user.
- A cluster has the following timestamps when instantiated:

• init_at: the timestamp when a cluster object is initialized in the Senlin database, but the actual cluster creation has not yet started;
• created_at: the timestamp when the cluster object is created, i.e. the CLUSTER_CREATE action has completed;
• updated_at: the timestamp when the cluster was last updated.

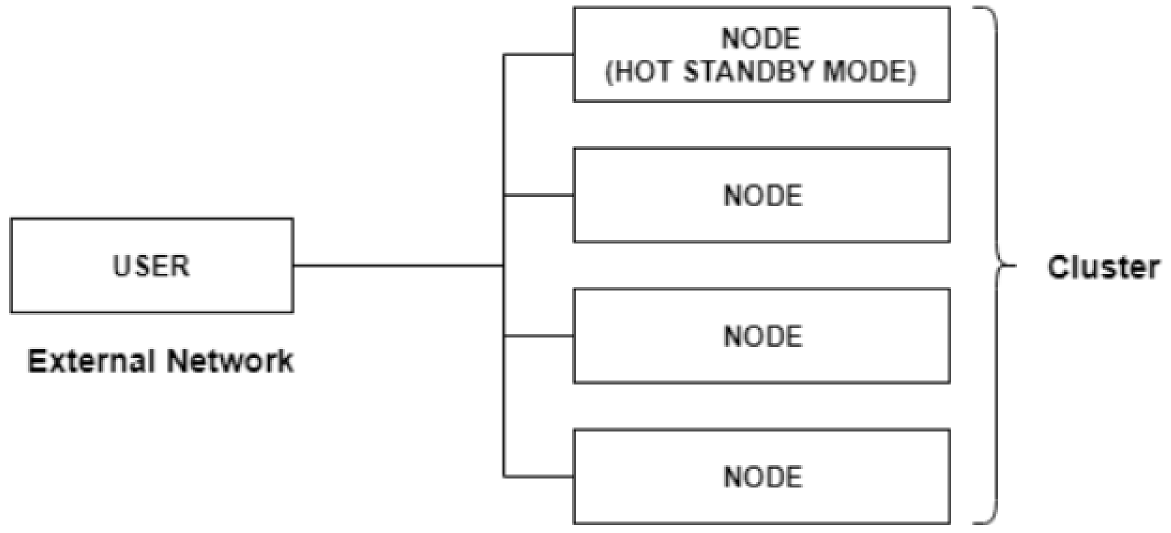
Cluster Statuses

- A cluster can have one of the following statuses during its lifecycle:

•INIT: the cluster object has been initialized, but not created yet;
•ACTIVE: the cluster is created and providing service;
•CREATING: the cluster creation action is still on going;
•ERROR: the cluster is still providing services, but there are things going wrong that needs human intervention;
•CRITICAL: the cluster is not operational, it may or may not be providing services as expected. Senlin cannot recover it from its current status. The best way to deal with this cluster is to delete it and then re-create it if needed.
•DELETING: the cluster deletion is ongoing;
•WARNING: the cluster is operational, but there are some warnings detected during past operations. In this case, human involvement is suggested but not required.
•UPDATING: the cluster is being updated.

Types of Cluster Systems:

- Primarily, there are two types of Cluster Systems:
  - **Asymmetric Cluster**
  - **Symmetric Cluster**
- **Asymmetric Cluster**
- Asymmetric clustering is mostly used for high availability purposes as well as for the scalability of read/write operations in databases, messaging systems, or files.
- In such systems, a standby server is involved to take over only if the other server is facing an event of failure. We may call the passive server, the sleepy watcher, where it can include the configuration of a failover.



Asymmetric Clustering System

Types of Cluster Systems:

- In this type of clustering, all the nodes run the required applications, and one node is in hot standby mode. The Hot standby node is used for monitoring the server till it fails, when it fails then it takes its place.
- How Asymmetric Clustering Works
- The following steps demonstrate the working of the asymmetric clustering system

1. There is a master node in asymmetric clustering that directs all the slaves nodes to perform the tasks required. The requests are delegated by the master node.
2. A distributed cache is used in asymmetric clustering to improve the performance of the system.
3. Resources such as memory, peripheral devices etc. are divided between the nodes of the asymmetric clustering system at boot time.

Asymmetric Cluster

- Advantages of Asymmetric Clustering
- The different advantages of asymmetric clustering are −

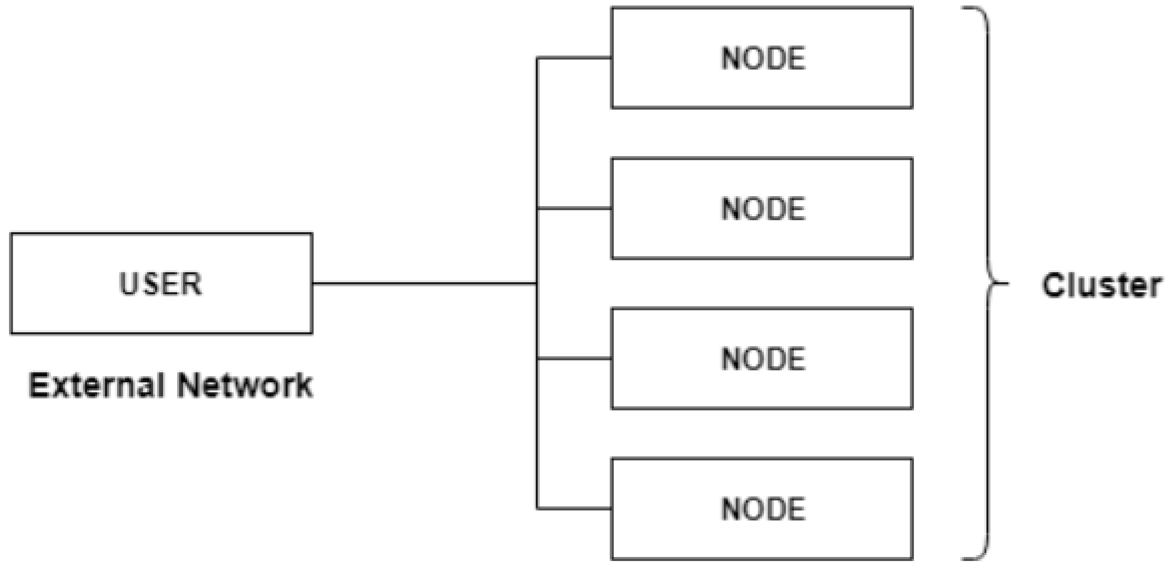
1. Since many computer systems are connected together and the processors work in parallel, this reduces the cost of having separate peripheral devices and memory.
2. The asymmetric clustering system has increased reliability as even if one nodes fails, the others can pick up the slack.
3. All the nodes in the asymmetric clustering system have their own kernels and can operate on different operating systems.
4. All the processors in the asymmetric clustering system are independent and only share memory for inter process communications.

Symmetric Cluster

- In symmetric clustering, all nodes are active and all participators handle the process of requests. This setup might be cost-effective by serving active applications and users.
- A failed node can be discarded from the cluster, while others take over its workload and continue to handle transactions.
- Symmetric clustering can be thought of as being similar to a load-balancing cluster situation, where all nodes share the workload by increasing the performance and scalability of services running in the cloud infrastructure.

Symmetric Cluster

- In this type of clustering, all the nodes run applications and monitor other nodes at the same time. This clustering is more efficient than Asymmetric clustering as it doesn’t have any hot standby key.



Symmetric Clustering System

Symmetric Cluster

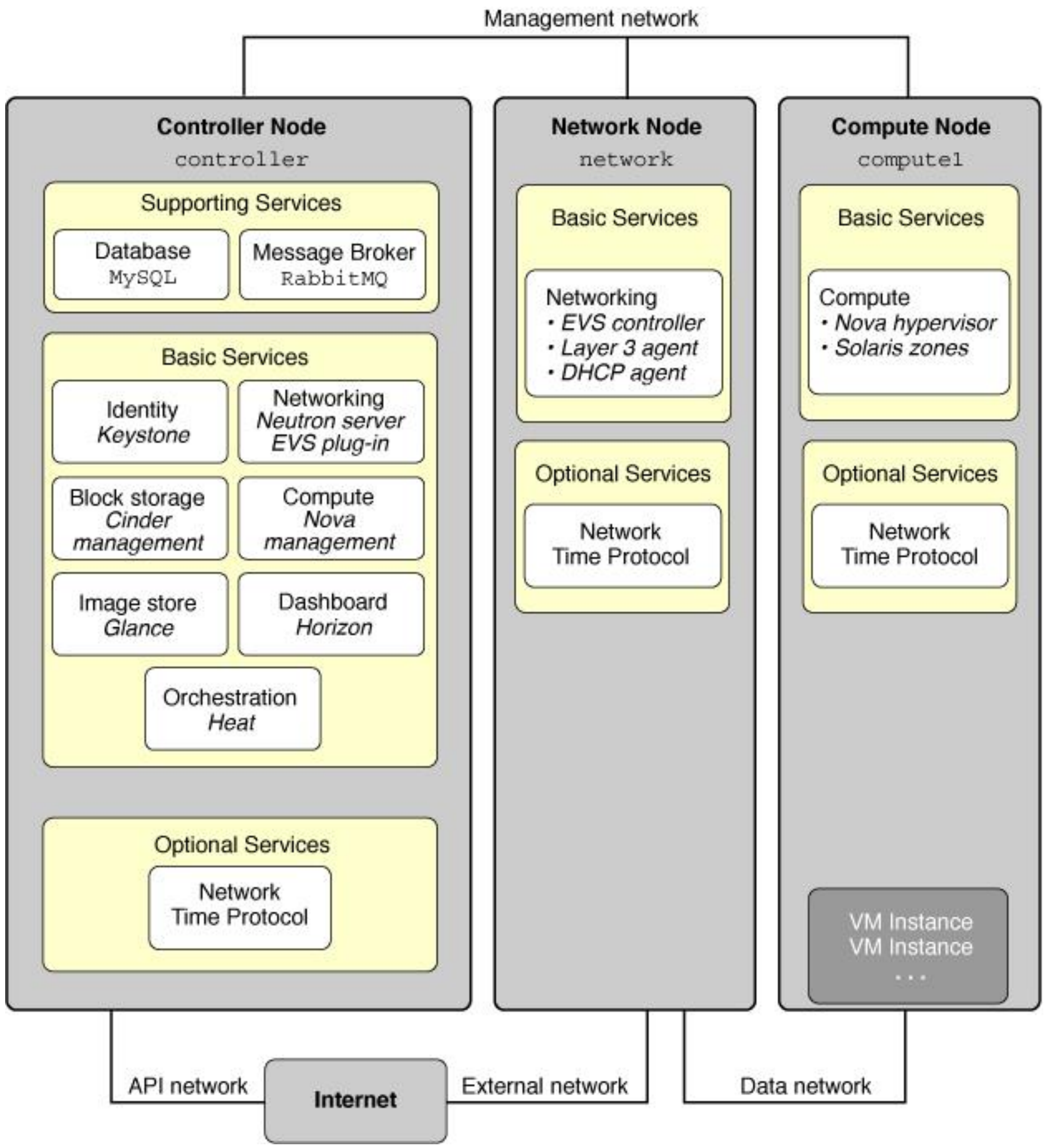
- Advantages of Symmetric Clustering System
- The different advantages of symmetric clustering are −

1. The symmetric clustering system is quite reliable. This is because if one nodes fails, the others can pick up the slack. This will not result in the failure of the system but will lead to graceful degradation.
2. Many computer systems are connected together and work in parallel in the symmetric clustering system. This reduces the overall cost as there is shared peripheral devices and memory.
3. Symmetric clustering systems are quite scalable as it is easy to add a new node to the system. There is no need to take the entire cluster down to add a new node.

Classification of clusters

- Computer clusters are managed to support various purposes, from general-purpose business requirements like web-service support to computation-intensive scientific calculations. There are various classifications of clusters. Some of them are as follows:
- 1. Fail Over Clusters
- The function of switching applications and data resources over from a failed system to an alternative system in the cluster is referred to as fail-over. These types are used to cluster database of critical mission, mail, file, and application servers.
- 2. Load Balancing Cluster
- The cluster requires better load balancing abilities amongst all available computer systems. All nodes in this type of cluster can share their computing workload with other nodes, resulting in better overall performance. For example, a web-based cluster can allot various web queries to various nodes, so it helps to improve the system speed. When it comes to grabbing requests, only a few cluster systems use the round-robin method. For example- a high-performance cluster used for scientific calculation would balance the load from different algorithms from the web-server cluster, which may just use a round-robin method by assigning each new request to a different node. This type of cluster is used on farms of Web servers (web farm).
- 3. High Availability Clusters
- These are also referred to as "HA clusters". They provide a high probability that all resources will be available. If a failure occurs, such as a system failure or the loss of a disk volume, the queries in the process are lost. If a lost query is retried, it will be handled by a different cluster computer. It is widely used in news, email, FTP servers, and the web.

Three-Node Architecture Overview



Divide and conquer

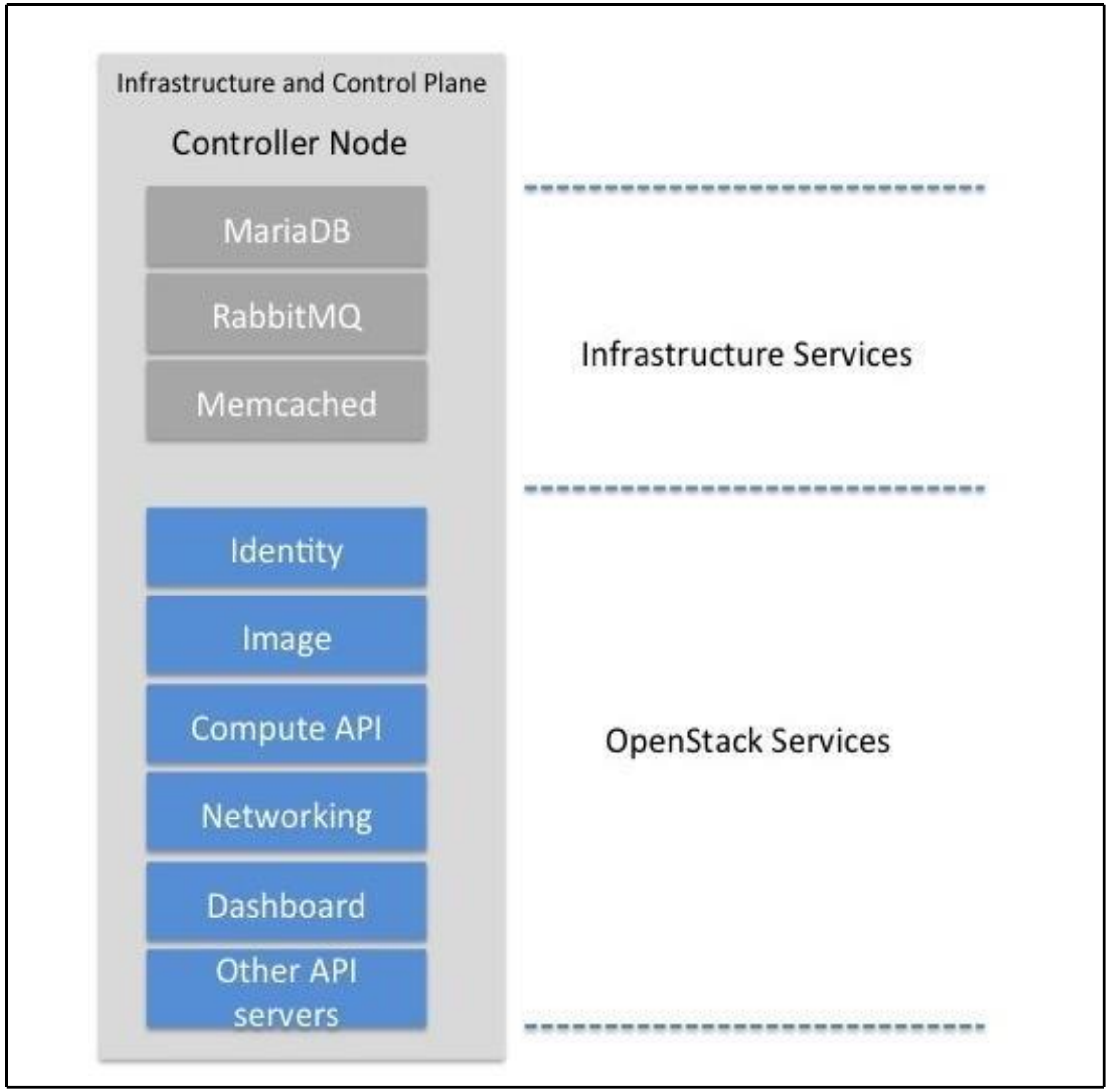
- OpenStack was designed to be horizontally scalable; we have already seen how its functions have been widely distributed into multiple services.
- The services themselves are composed of the API server, schedulers, workers, and agents.
- While the OpenStack controller runs the API services, the compute, storage, and network nodes run the agents and workers.

Cloud Controllers

Cloud Controllers

- The concept of cloud controllers aims to provide **central management** and **control** over your OpenStack deployments. We can, for example, consider that the cloud controller is managing all **API calls** and **messaging transactions**.
- Considering a medium- or large-scale infrastructure, we will need, with no doubt, more than a single node.
- For an OpenStack cloud operator, controllers can be thought of as **service aggregators**, where the majority of management services needed to operate OpenStack are running.
- Let's see what a cloud controller mainly handles:
- It presents a **gateway** for access to cloud management and services consumption. It provides the API services in order to make different OpenStack components talk to each other and provides a service interface to the end user.
- It provides mechanisms for **highly available integrated services** by the means of clustering and load-balancing utilities.
- It provides **critical infrastructure services**, such as a **database** and **message queue.**
- It exposes the **persistent storage**, which might be backed onto separate storage nodes

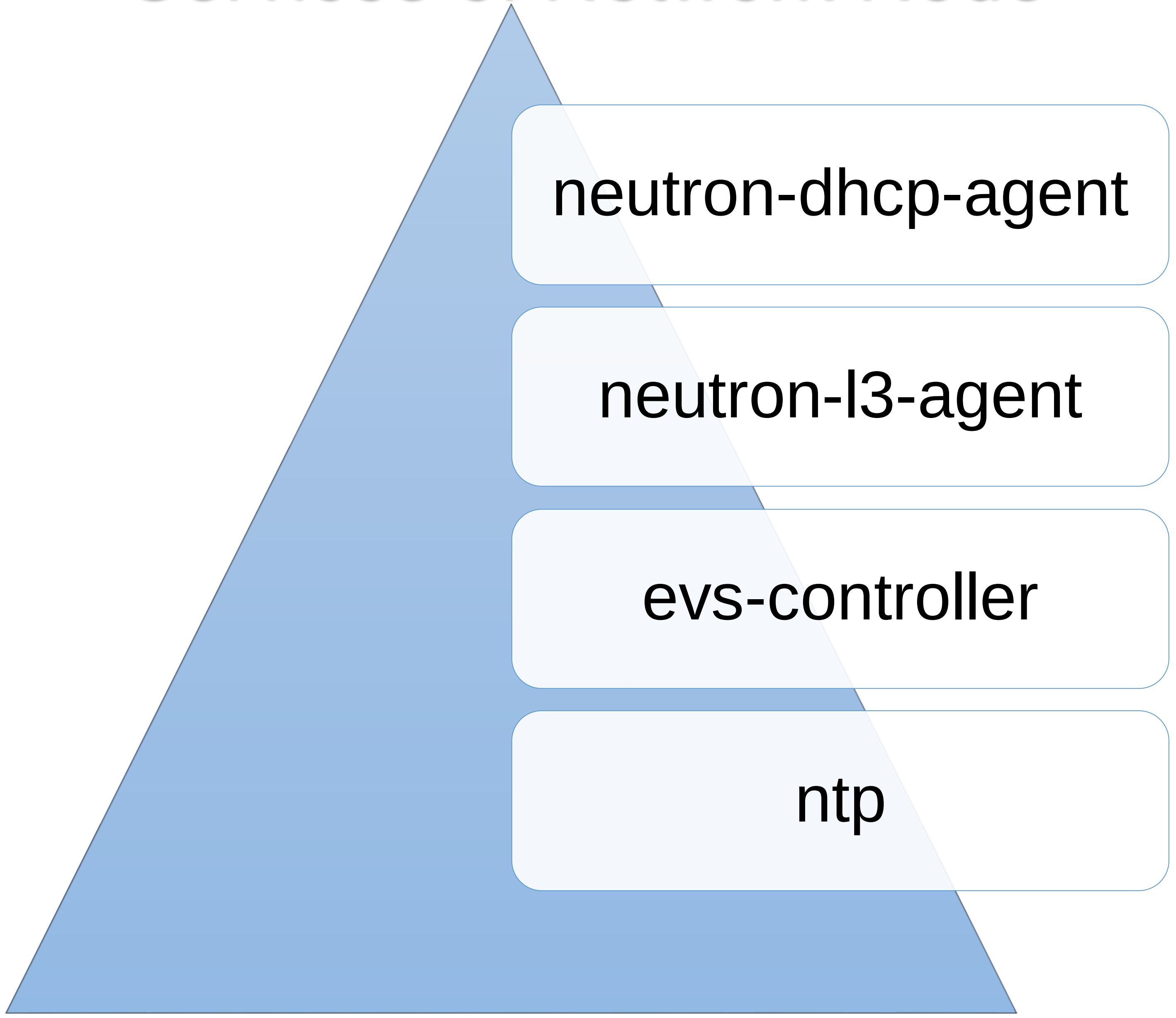
Services in Cloud Controller



Services of Controller Node

mysql	rabbitmq	keystone	cinder-api	cinder-db	cinder-db
cinder-scheduler	cinder-volume:default	cinder-volume:setup	glance-api	glance-db	glance-registry
glance-scrubber	neutron-server	evs	nova-api-ec2	nova-api-osapi-compute	nova-cert
nova-conductor	nova-objectstore	nova-scheduler	http	ntp	heat-api
	heat-db	heat-api-cfn	heat-api-cloudwatch	heat-engine	

Services of Network Node



Services of Compute Node

- nova-compute
- ntp

02 The cloud controller

- ✓The keystone service.
- ✓The nova-conductor service
- ✓The nova-scheduler service
- ✓The API services
- ✓Image management
- ✓The network service
- ✓The horizon dashboard
- ✓The telemetry services.

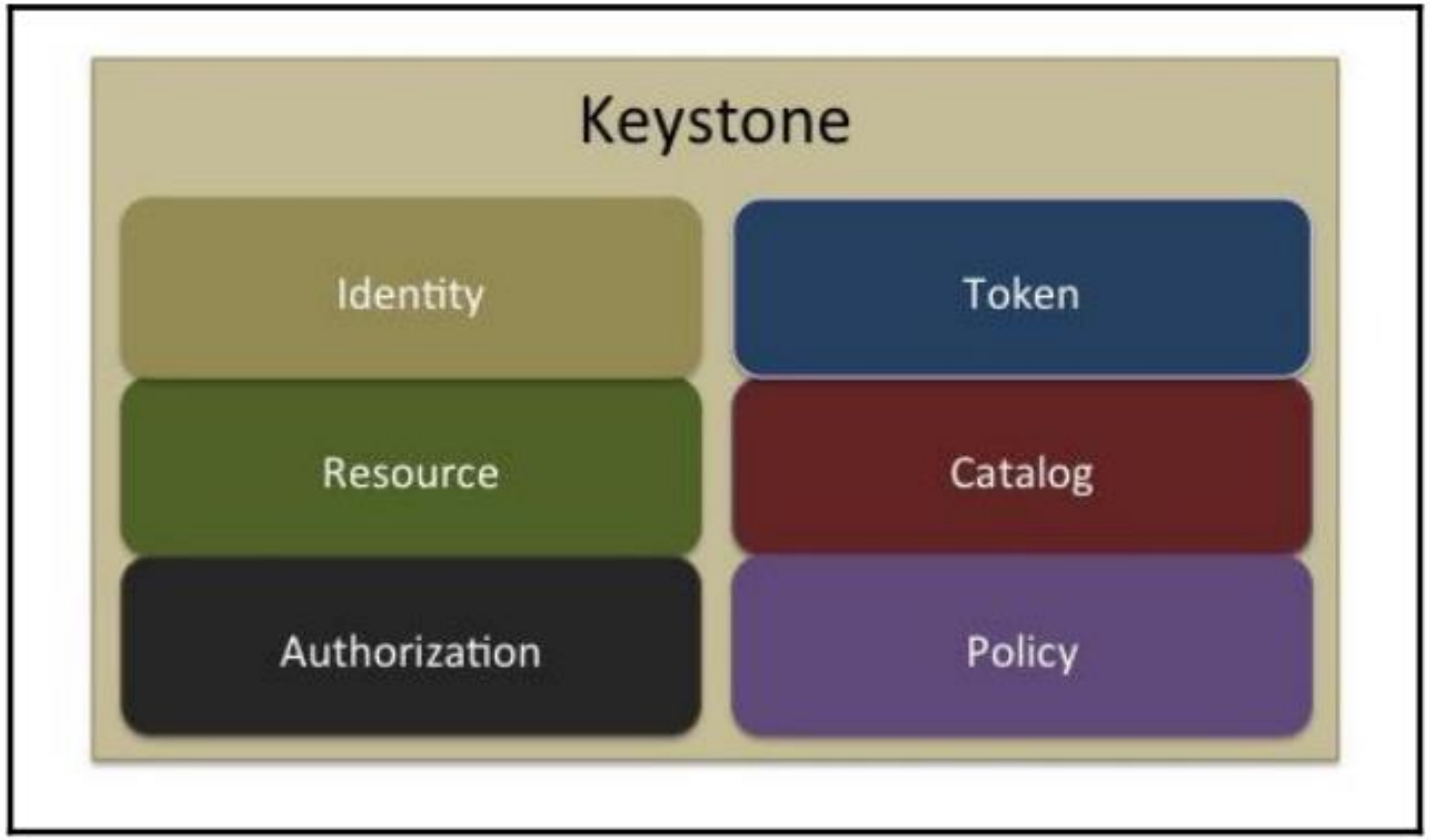
1. KEYSTONE SERVICE

The Keystone Service

- The Keystone service provides **identity and service cataloging** in OpenStack.
- All other services in OpenStack **must register** with Keystone with their **API endpoints**.
- Keystone thus keeps a **catalog** of various services running in your OpenStack cloud that can be queried using the Keystone REST APIs.
- Keystone also maintains a **policy engine** which provides **rule-based access and authorization** of services.
- The Keystone service itself is **composed of multiple providers** that work in conjunction with each other.
- Each of these providers implements a concept in the Keystone architecture:

The Keystone Services

- Identity
- Resource
- Authorization
- Token
- Catalog
- Policy



The Keystone Services

- The identity provider
- The identity provider validates user and group credentials.
- OpenStack Keystone provides a built-in identity provider that can be used to create and manage user and group credentials.
- Keystone can also integrate with external identity providers such as LDAP.
- The OpenStack Ansible project provides the playbooks to integrate the LDAP service with Keystone as an external identity provider.
- Keystone supports various user types to manage access levels to OpenStack services.

The Keystone Service

- The identity provider
- The user can be one of the following:
  - A service user who is associated with a service running in OpenStack
  - An administrative user who has administrative access to services and resources created
  - An end user who has no extra access rights and is a consumer of OpenStack resources

The Keystone Service

- The resource provider
- The resource provider implements the concept of projects and a domain in Keystone.
- The concept of a domain in Keystone provides a container for Keystone entities such as users, groups, and projects.
- Think of a domain as a company or a service provider.

The authorization provider

- The concept of authorization provides a relationship between OpenStack users and groups to a list of roles.
- The roles are used to manage access to services running in OpenStack.
- The Keystone policy provider enforces rules based on the group and role a user belongs to.

The Keystone Service

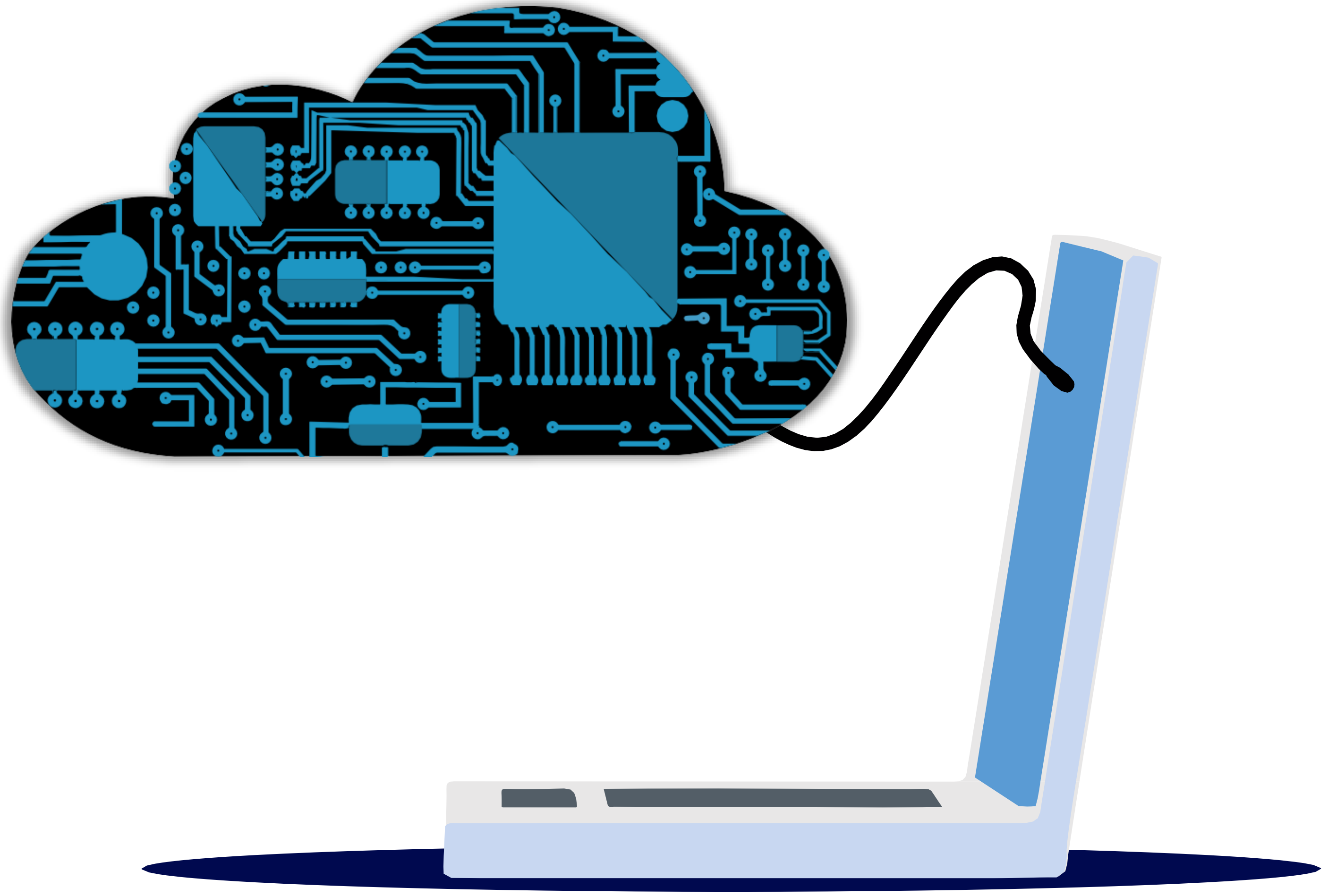
- The token provider
- To access the services in OpenStack, the user must authenticate with the identity provider.
- Once the authentication of the user is established, the token provider generates a token that provides the user with authorization to access the OpenStack services.
- The token is valid for a limited period of time before it expires. A valid token is required to access the OpenStack services.
- The catalog provider
- As mentioned earlier, all the OpenStack services must register with the Keystone service.
- The catalog provider maintains a registry of services and associated endpoints running in the OpenStack cloud.
- In a way, Keystone provides an entry point for discovering services running in your OpenStack cluster.

The Keystone Service

- The policy provider
- The policy provider associates rules to allow access to the Keystone resources.
- The policies are composed of individual rules that define which users and roles are allowed to access which resources.
- For example, the ability to add users to a group is provided only to users with an admin role.

The Keystone Service

- Advanced features provided by the Keystone service:
- Federated Keystone
- Fernet tokens



1. Federated Identity

- Federated Identity is a mechanism to use the identity service provided by an external Identity Provider (IdP) to access resources available with the Service Providers (SP).
- In the case of OpenStack, the identity provider can be a third party while OpenStack acts as the SP with resources such as virtual storage, network, and compute instances.
- Using Federated Identity has some advantages over maintaining an authentication system in OpenStack:
- An organization can use an already existing identity source, such as LDAP or Active Directory, to provide user authentication in OpenStack. It removes the need to manage separate identity sources for OpenStack users.

1. Federated Identity

- Imagine you have a magic key that can open many doors in different places. This magic key is your identity, like your username and password. Now, sometimes, you want to use this key in new places without creating a new key every time.
- Federated Identity is like having a special friend who vouches for you. Instead of making a new key for every place, your friend (Identity Provider) says, "Hey, I know this person!" to the security guard (Service Provider) at different places (like websites or services).

Federated Identity

- Using an identity service can help in integrating with different cloud services.
- It provides a single source of truth and user management. So, if a user is part of the identity service, he can be very easily provided access to the OpenStack cloud infrastructure.
- It lowers the security risk by not running yet another identity system for OpenStack users.
- It helps in providing better security to OpenStack users and a single sign on.
- So, the obvious question to ask is: How does Federated Identity work? To understand this, let's look at how the user interacts with the SP and the Identity Provider services.

1. Federated Identity

- The steps involved in authenticating users using Federated Identity are as follows:

1. The user tries to access a resource available with the SP.
2. The SP checks to see whether the user already has an authenticated session. If the user does not have a session, he is redirected to the authentication URL of the Identity Provider.
3. The Identity Provider prompts the user for the identity credentials such as username and password, which it validates, and issues an unscoped token. The unscoped token contains, among other information, the list of groups the authenticated user belongs to.
4. The user then uses the unscoped token to determine a list of accessible domains and projects in the Service Provider's cloud.
5. The user then uses the unscoped token to get a scoped token for the project and domain of his interest and can start using resources on the cloud.

1. Federated Identity

- An interesting use case of Federated Identity is the use of the Keystone service from two different cloud infrastructures that act as a Service Provider and Identity Provider (also known as the K2K or Keystone to Keystone use case), thus allowing the user from the first Keystone instance acting as the Identity Provider to access the resources on the other cloud infrastructure that is acting as the Service Provider.

2. Fernet Token

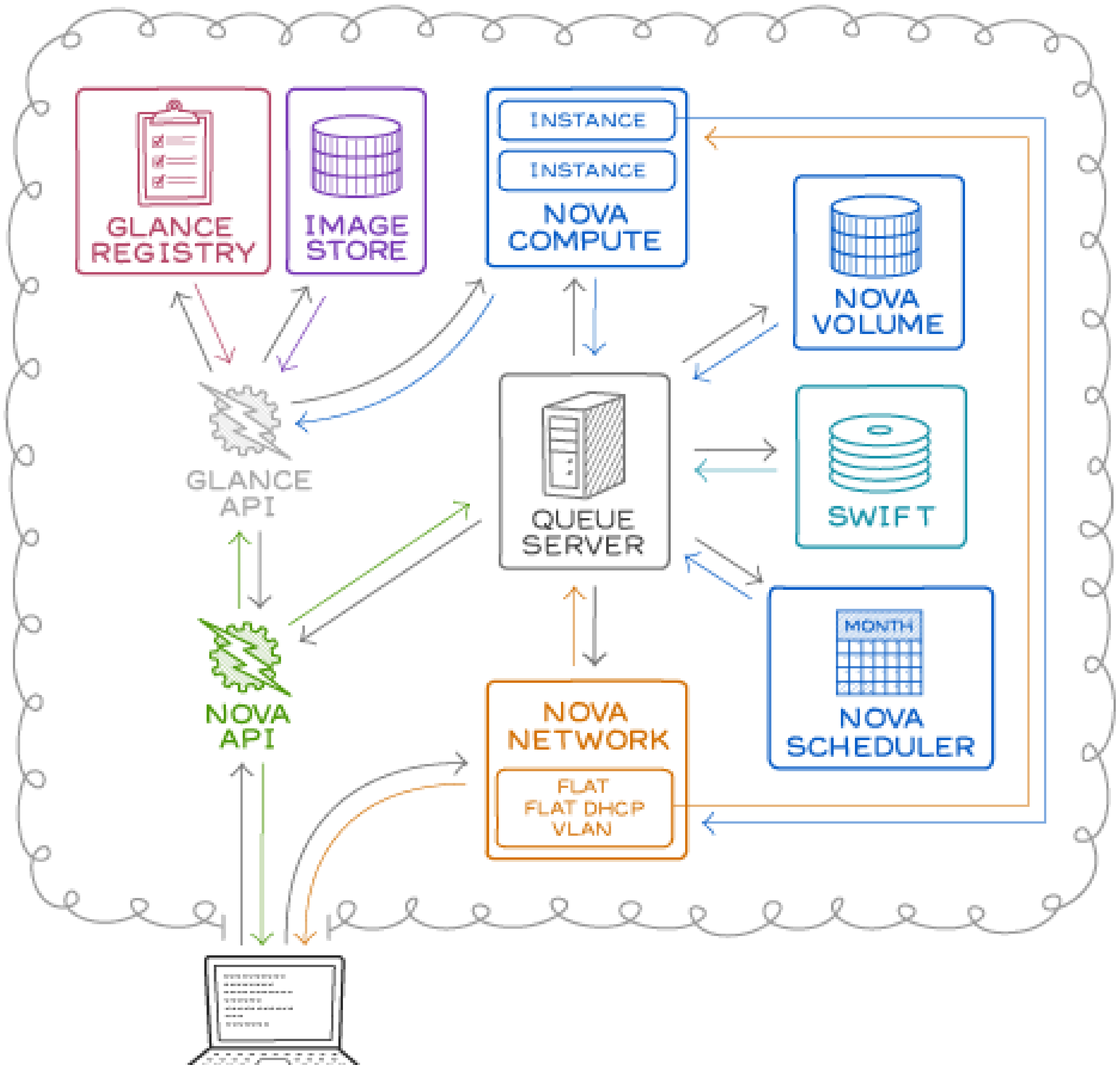
- Traditionally, Keystone issues Public Key Infrastructure (PKI) -based tokens to authenticated users.
- The user accesses various OpenStack services using these tokens.
- The PKI tokens encapsulate the user's identity and the authorization context in JSON format.
- This makes the PKI tokens large in size. In addition, these tokens need to be stored in the database to allow token revocation.
- PKIZ tokens are an enhancement to the PKI tokens in that they use compression to reduce the size of the tokens but still create considerable stress on the database for holding all the issued tokens.

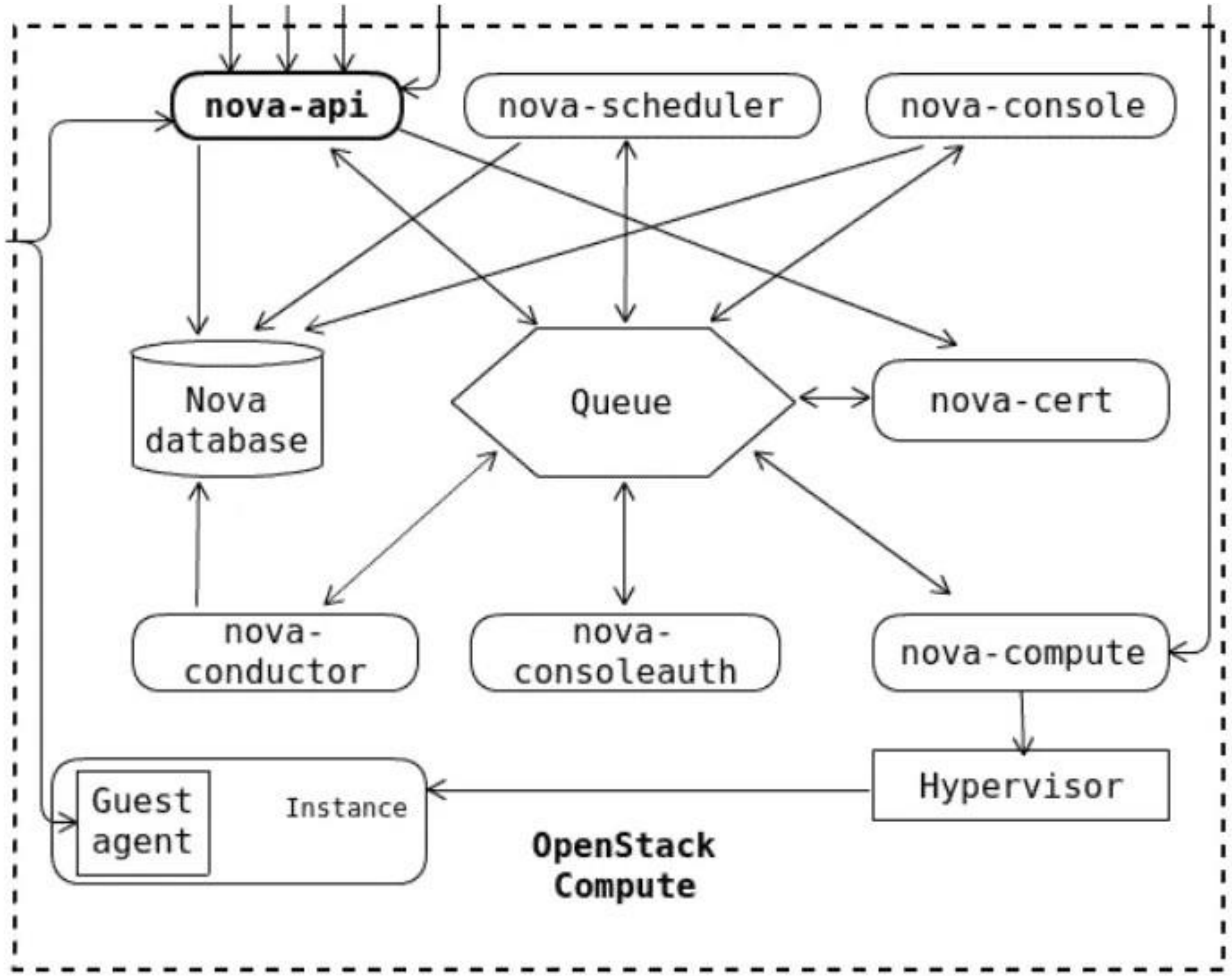
2. Fernet Token

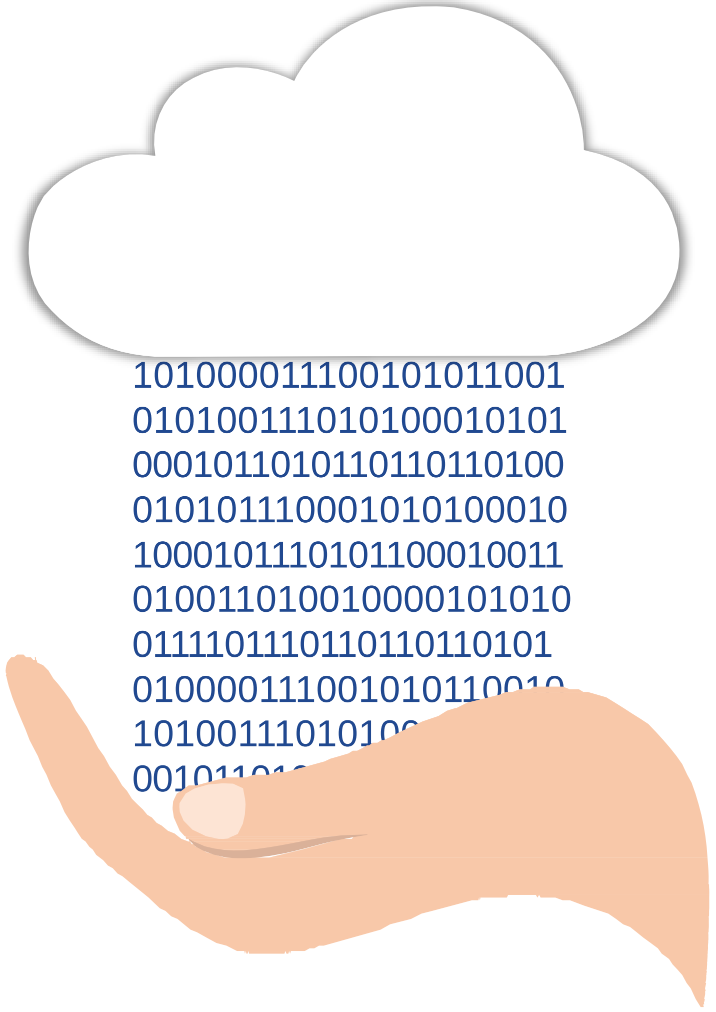
- Fernet tokens were introduced in Keystone to counter the shortcomings of size and database load and are on the way to becoming the default token format.
- The Fernet tokens are comparatively smaller in size. The major advantage of using Fernet tokens is that they need not be stored in the database and they are considerably faster in token creation as there is no database interaction involved.
- The Fernet token contains the user identity, scope of authorization over a project, and an expiration time.
- Encrypting the identity and authorization with keys generates Fernet tokens. They are considerably smaller and faster than the PKI tokens.

NOVA – OpenStack Compute

- This is the fundamental computing engine of OpenStack.
- It manages a large number of Virtual machines and other instances, which handle computing tasks.
- In a cloud computing environment, it acts as a controller, which manages all the resources in a virtual environment
- **Functionality**
- The nova-API handles the requests and responses from and to the end-user.
- The nova-compute creates and destroys the instances as and when a request is made.
- Nova-scheduler schedules the tasks to the nova-compute
- The glace registry stores the details of the image along with its metadata.
- The Image store stores the images predefined by the admin/user.
- The nova network ensures network connectivity and routing.


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


nova-api
nova-scheduler
nova-console
Nova database
Queue
nova-cert
nova-conductor
nova-consoleauth
nova-compute
Hypervisor
Guest agent
Instance
OpenStack Compute



2. NOVA-CONDUCTOR SERVICE

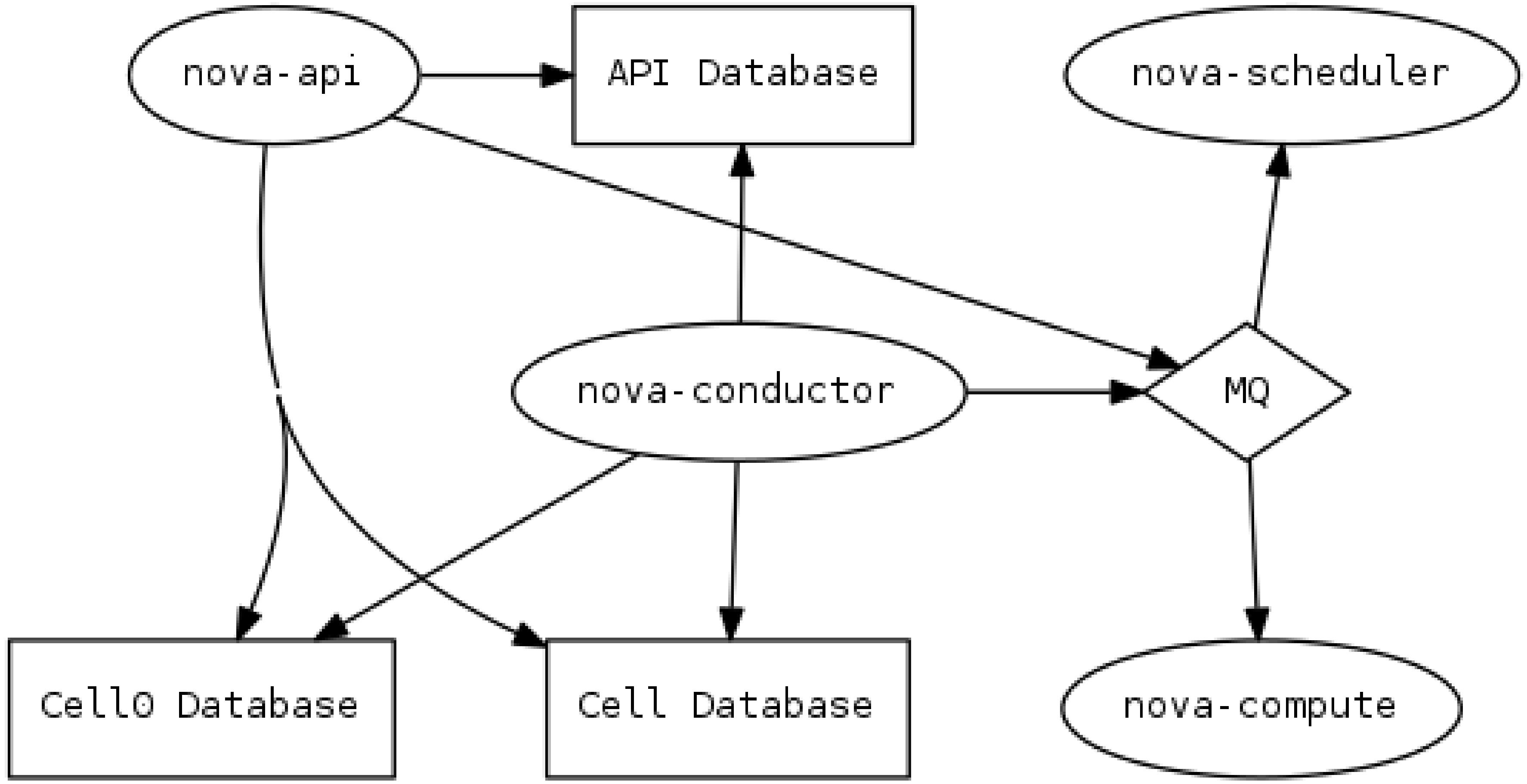
The nova-conductor service

- **What is Nova-Conductor?**
  - Nova-Conductor is a part of the Nova project in OpenStack.
  - It handles tasks usually performed by Nova compute nodes, like database access, scheduling, and resource tracking.
- **What is the "Nova-Conductor Service"?**
  - "Nova-Conductor service" refers to the active instance of Nova-Conductor in an OpenStack setup.
  - It fundamentally changes how the Nova-Compute service accesses the database.
- **Why was it Added?**
  - Enhancing Security: It enhances security by separating direct database access from compute nodes.
  - Preventing Attacks: Compute nodes running Nova-Compute service can be vulnerable to attacks and compromise.
  - Database Access Segregation: Nova-Conductor acts as an intermediary, carrying out database operations on behalf of compute nodes. It adds a layer of segregation for database access.
- **Key Function:**
  - Database Operations: Nova-Conductor performs database operations for compute nodes.
  - Access Segregation: It provides a layer of separation, ensuring secure database access.
  - New Layer: It creates a new layer on top of Nova-Compute, improving overall system security.

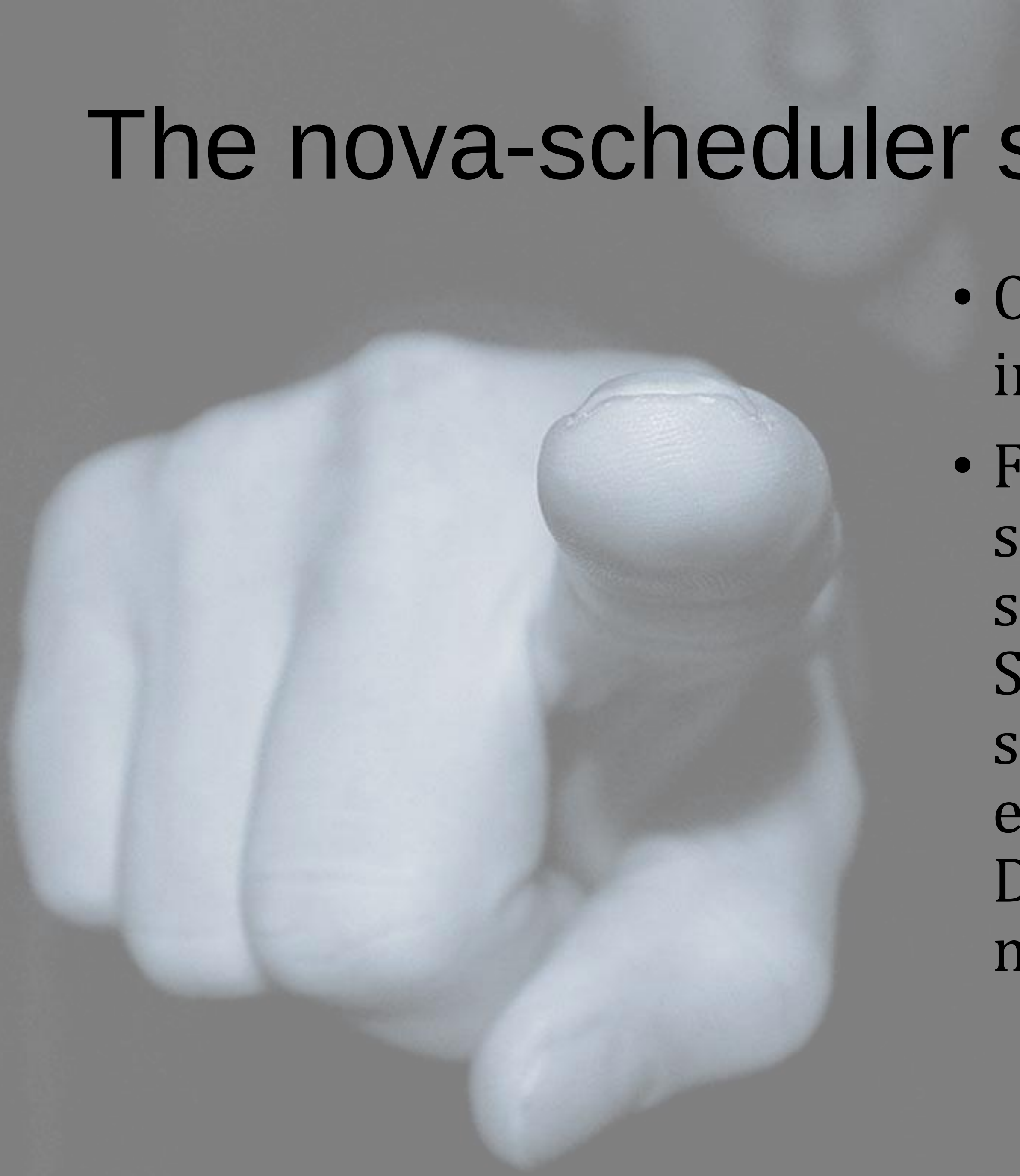
3. NOVA-SCHEDULER SERVICE

The nova-scheduler service

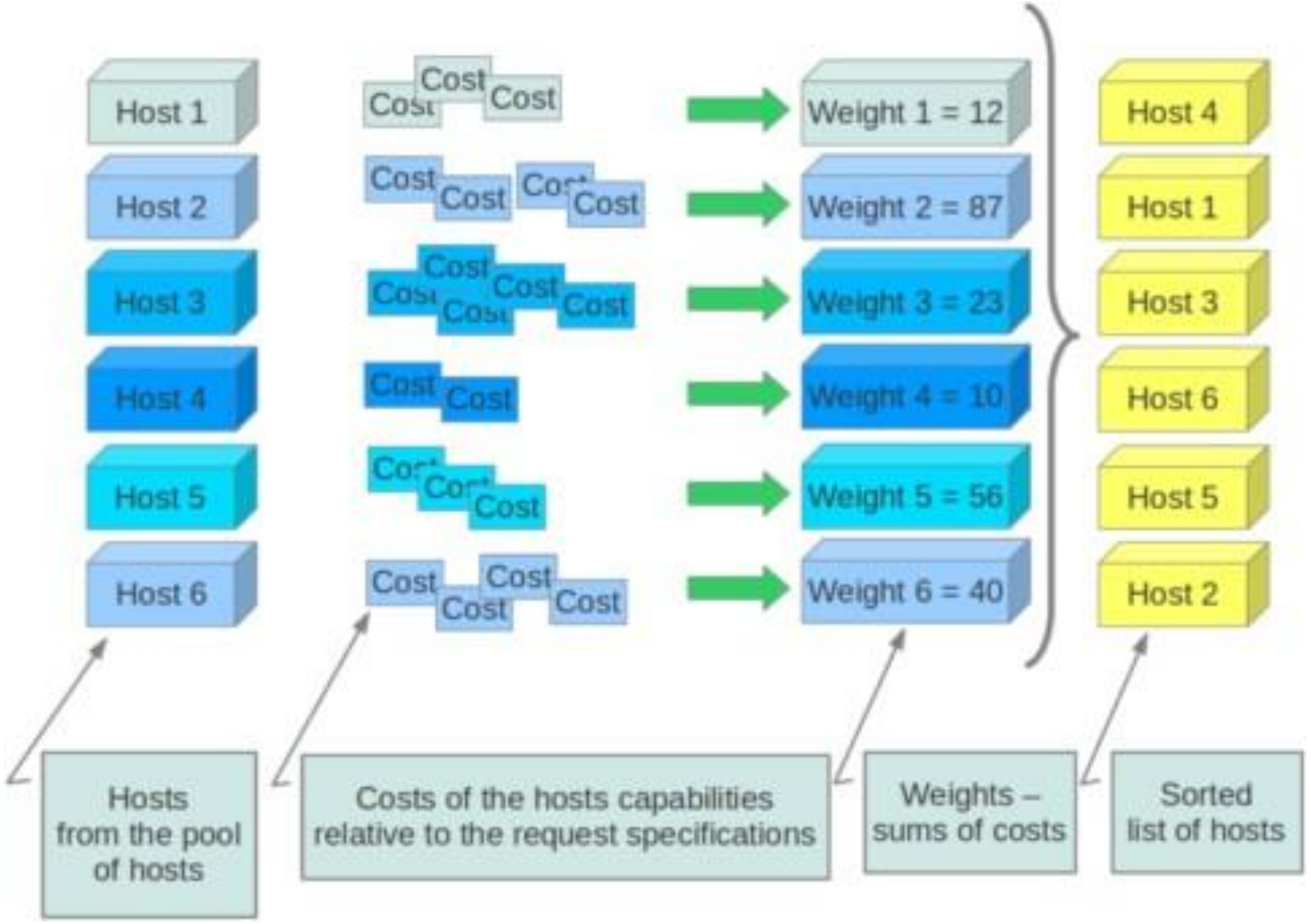
- **Purpose:**
  - Nova Scheduler is a core part of OpenStack Nova, focusing on determining where virtual instances run within the cloud infrastructure.
- **Functions:**
  - *Instance Placement*: Decides which compute node hosts a specific virtual machine instance.
  - *Resource Optimization*: Ensures efficient use of available resources by distributing instances wisely.
  - *Load Balancing*: Prevents overloading nodes by evenly distributing workloads.
- **Features:**
  - *Filtering & Weighting:* Uses filters to eliminate unsuitable hosts and assigns weights based on criteria like CPU and memory for optimal placements.
  - *Customizable Algorithms*: Allows customization of scheduling algorithms to match specific workload requirements.
  - *Affinity Rules:* Supports rules for instances to run together (affinity) or apart (anti-affinity) on nodes.

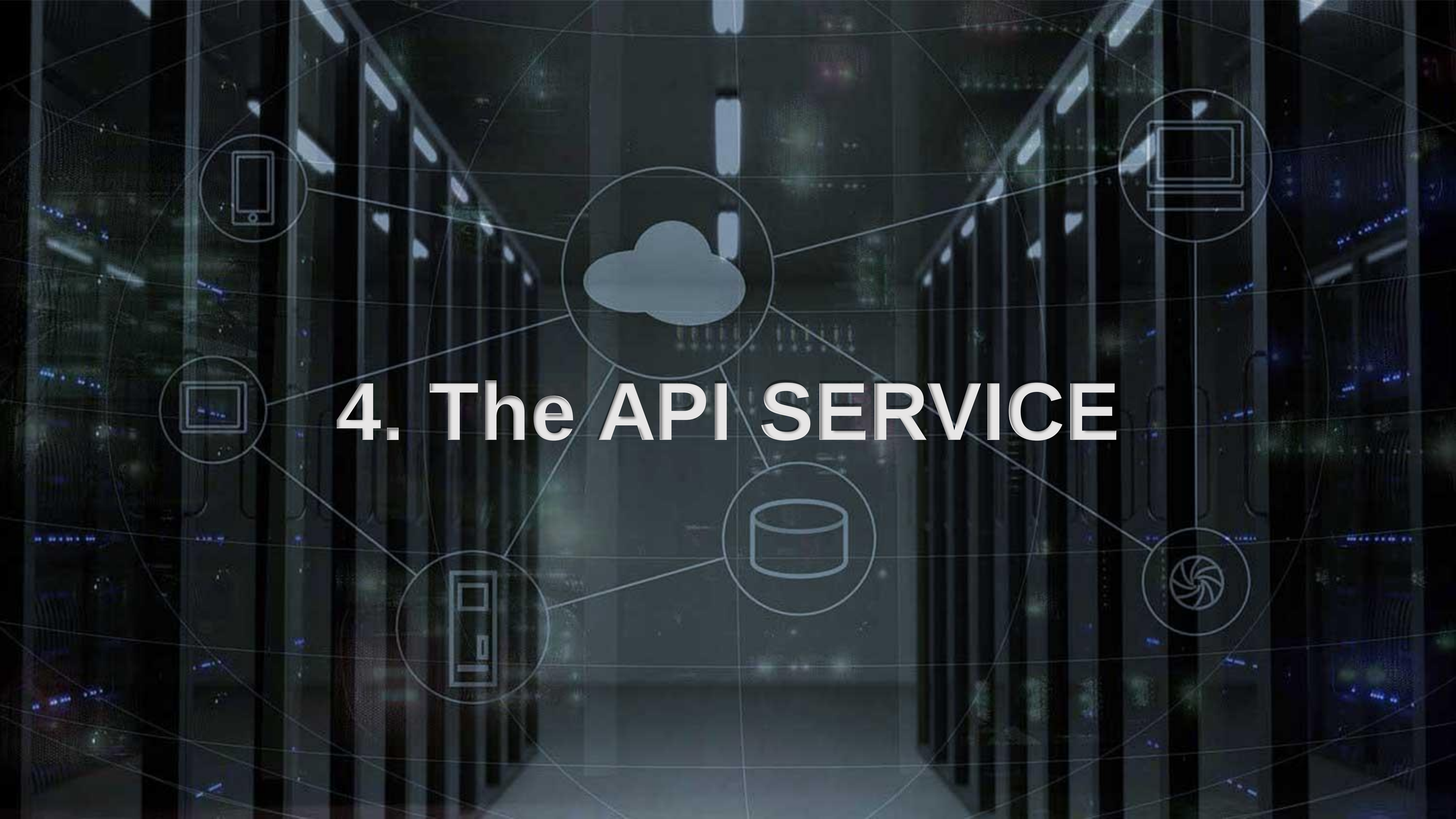

nova-api
API Database
nova-scheduler
nova-conductor
MQ
Cell0 Database
Cell Database
nova-compute

The nova-scheduler service



- Other OpenStack services also implement schedulers.
- For example, cinder-scheduler is a scheduling service for block storage volumes in OpenStack. Similarly, Neutron implements a scheduler to distribute network elements such as routers and DHCP servers among the network nodes.


Host 1
Host 2
Host 3
Host 4
Host 5
Host 6
Cost
Weight 1 = 12
Weight 2 = 87
Weight 3 = 23
Weight 4 = 10
Weight 5 = 56
Weight 6 = 40
Host 4
Host 1
Host 3
Host 6
Host 5
Host 2
Hosts from the pool of hosts
Costs of the hosts capabilities relative to the request specifications
Weights – sums of costs
Sorted list of hosts



4. The API SERVICE

The API services

- API Basics:
  - API is a way to interact with OpenStack services.
  - Accessible via command line or web interface.
- Nova-Api Service:
  - Handles user requests for the compute service.
  - Creates and manages virtual machines.
  - Acts as an orchestrator engine in the cloud controller.
- API Types:
  - Supports two APIs: OpenStack API and EC2 API.
  - Choice depends on data format preferences; OpenStack uses names and numbers, while EC2 API uses hexadecimal identifiers.

The API services

- Important Consideration:
  - Decision on API usage crucial before deploying the cloud controller node.
  - Information presentation varies between APIs.
- Key Role:
  - Initiates and processes API queries.
  - Orchestrates tasks like provisioning virtual machines.
- Learn More:
- For detailed information, refer to OpenStack API Documentation.

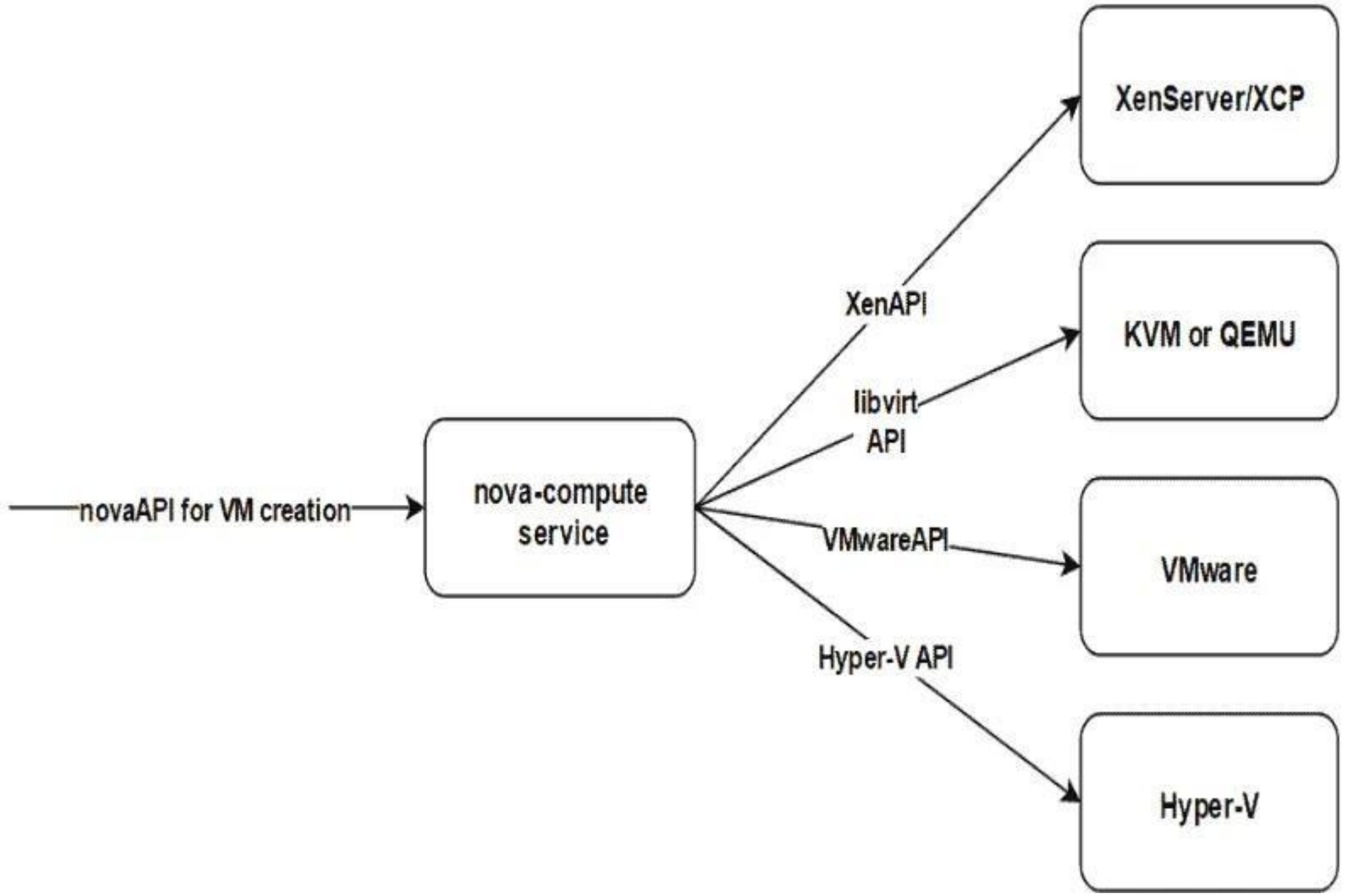
5. NOVA-COMPUTE SERVICE

The nova-compute service

- User Request: When a user requests a new virtual machine via the OpenStack dashboard, the request is sent to the nova-api service.
- Handoff to Nova-Compute: nova-api processes the request and hands it over to nova-compute.
- Hypervisor Communication: nova-compute understands the specific API supported by the underlying hypervisor (like KVM).
- Virtual Machine Creation: Using the appropriate API (like libvirt for KVM), nova-compute instructs the hypervisor to create the new virtual machine.
- Result: The KVM hypervisor processes the request and generates the new virtual machine as per the user's initial request.

The nova-compute service

- Underneath, the entire lifecycle of the virtual machine is managed by the hypervisors. Whenever the end user submits the instance creation API call to the nova-api service, the nova-api service processes it and passes the request to the nova-compute service.
- The nova-compute service processes the nova-api call for new instance creation and triggers the appropriate API request for virtual machine creation in a way that the underlying hypervisor can understand.
- For example, if we choose to use the KVM hypervisor in the OpenStack setup, when the end user submits the virtual machine creation request via the OpenStack dashboard, the nova-api calls will get sent to the nova-api service. The nova-api service will pass the APIs for instance creation to the nova-compute service. The nova-compute service knows what API the underlying KVM hypervisor will support. Now, pointing to the underlying KVM hypervisor, the nova-compute will trigger the libvirt-api for virtual machine creation. Then, the KVM hypervisor processes the libvirt-api request and creates a new virtual machine.


novaAPI for VM creation
nova-compute service
XenAPI
libvirt API
VMwareAPI
Hyper-V API
XenServer/XCP
KVM or QEMU
VMware
Hyper-V

6. IMAGE MANAGEMENT SERVICE

Image management

- The cloud controller will also be hosting the Glance service for image management that is responsible for the storage, listing, and retrieval of images using the glance-api and glance-registry.
- The Glance API provides an external REST interface to query virtual machine images and associated metadata.
- The Glance registry stores the image metadata in the database while the storage backend stores the actual image.
- While designing the image, a store decision can be made about which backend will be used to launch the controller in the cloud.
- The glance-api supports several backend options to store images. Swift is a good alternative and allows storing images as objects, providing a scalable placement for image storage.
- Other alternatives are also possible such as filesystem backend, Amazon S3, and HTTP

7. NETWORK SERVICE

The network service

- **API for Network Control:**
  - Similar to how OpenStack's Nova service provides an interface (API) to request computing resources, a similar mechanism exists for network resources.
  - This API is located within the cloud controller and allows users to make dynamic requests for various network functionalities, such as firewall configurations, routing rules, and load balancing settings.
  - The API supports extensions, enabling the implementation of advanced network features beyond the basic functionalities.
- **Separation of Network Workers:**
  - In complex network setups, different tasks are performed by different network workers.
  - It is recommended to separate these workers to enhance the efficiency and security of the network system.
  - Separation might involve categorizing tasks and assigning specific workers to handle each category. For instance, one set of workers might handle routing, while another deals with load balancing.

The network service

- **Performance Challenges:**
  - Cloud controllers run multiple services simultaneously, leading to a significant volume of network traffic.
  - Handling this high traffic load can pose performance challenges, potentially leading to slowdowns or delays in processing requests.
  - Addressing these challenges requires careful planning and implementation of strategies to optimize the performance of the cloud controller and the associated network services.
- **Clustering for Scalability:**
  - Clustering refers to the practice of connecting multiple hardware or software entities so that they work together as a single system.
  - Clustering enhances scalability by allowing the network to handle a larger number of requests and users. It also improves fault tolerance, ensuring that the network remains operational even if some components fail.

The network service

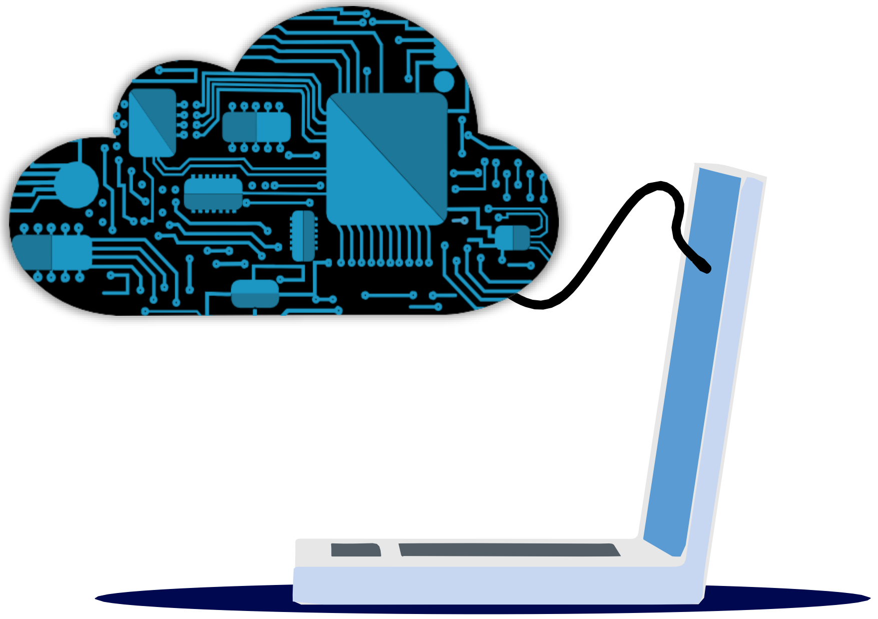
- **NIC Bonding (Network Interface Card):**
  - NIC bonding is a technique used to increase the available bandwidth and provide fault tolerance.
  - It involves combining two or more network interfaces into a single bonded interface.
  - From the perspective of the system and network, these bonded NICs appear as a single, higher-bandwidth, and more reliable physical device.

8. HORIZON SERVICE

The Horizon dashboard

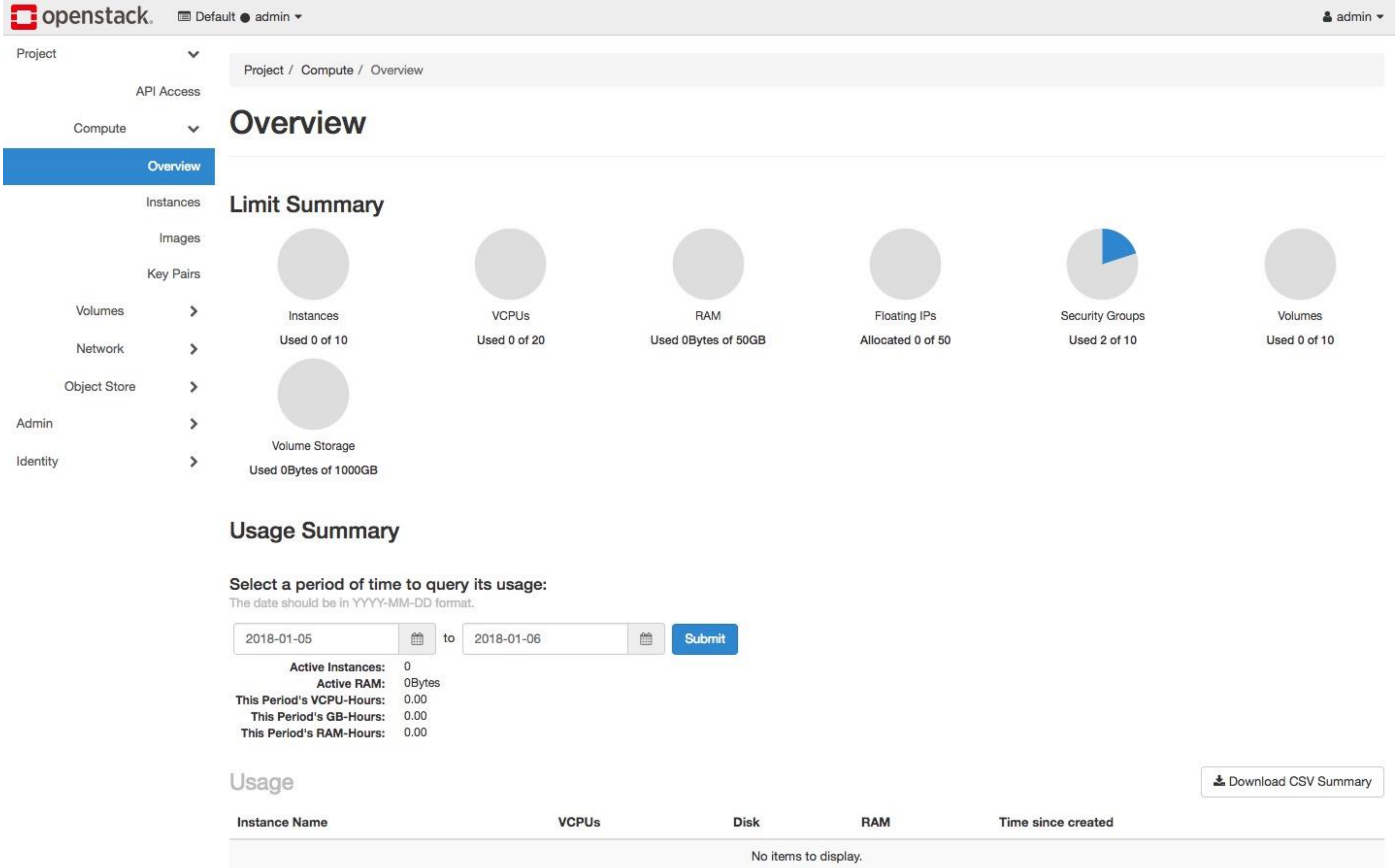
- Dashboard Configuration:
  - OpenStack dashboard runs on Apache server with Python Django framework.
- Load Optimization:
  - Set up a separate node for Horizon dashboard to reduce load on the main controller.
- User Flexibility:
  - Users can choose to run Horizon on the controller node, making monitoring decisions themselves.

HORIZON
an OpenStack Community Project



The Horizon dashboard

- **Modularity and Extensibility:**
  - Horizon is designed with modularity in mind, allowing developers to add or remove components easily. It supports a pluggable architecture, enabling the integration of additional panels and features.
- **Role-Based Access Control:**
  - Horizon supports role-based access control, enabling administrators to define specific roles and permissions for users. This feature ensures that users only have access to the functionalities and resources they are authorized to use.
- **Integration with OpenStack Services:**
  - Horizon seamlessly integrates with various OpenStack services, including Nova (compute), Neutron (networking), Cinder (block storage), and Keystone (identity), providing a unified interface to manage these services.
- **Interactive Data Visualization:**
  - Horizon offers interactive data visualization features, such as graphs and charts, allowing users to monitor resource usage, performance metrics, and other relevant data in real-time.



Overview

Limit Summary

Instances	VCPUs	RAM	Floating IPs	Security Groups	Volumes	Volume Storage
Used 0 of 10	Used 0 of 20	Used 0Bytes of 50GB	Allocated 0 of 50	Used 2 of 10	Used 0 of 10	Used 0Bytes of 1000GB

Usage Summary

Select a period of time to query its usage:
The date should be in YYYY-MM-DD format.

2018-01-05 to 2018-01-06 Submit

Active Instances:	0
Active RAM:	0Bytes
This Period's VCPU-Hours:	0.00
This Period's GB-Hours:	0.00
This Period's RAM-Hours:	0.00

Usage

Download CSV Summary

Instance Name	VCPUs	Disk	RAM	Time since created
No items to display.				

9. TELEMETRY SERVICE

The telemetry services

- Telemetry Service in OpenStack:
  - Before the Liberty release, the telemetry service was called Ceilometer, used for tracking resource usage in multi-user shared infrastructures.
- Importance of Resource Usage Tracking:
  - Tracking resource usage is vital for billing, capacity planning, and auto scaling in shared infrastructures, ensuring efficient resource management.
- Post-Liberty Release Changes:
  - After Liberty, the telemetry service was split into two additional projects that can work with Ceilometer if enabled.

The telemetry services

- **Aodh:**
  - Aodh generates alarms when resource utilization exceeds predefined thresholds, enhancing real-time monitoring and response capabilities.
  - Example Scenario: Aodh can generate an alarm when CPU usage in a virtual machine surpasses 90%, allowing administrators to take immediate corrective measures.
- **Gnocchi:**
  - Gnocchi is a sub-project focusing on scalable storage of metrics and events, allowing efficient handling of large amounts of data generated by resource usage tracking.
  - Example Scenario: Gnocchi can store and manage historical data related to CPU, memory, and network usage for all instances in an OpenStack cloud, enabling administrators to analyze usage patterns over months or years.

The telemetry services

- **Architecture:**
  - Ceilometer employs an agent-based architecture, consisting of components such as the API server, data collection agents, and a data store.

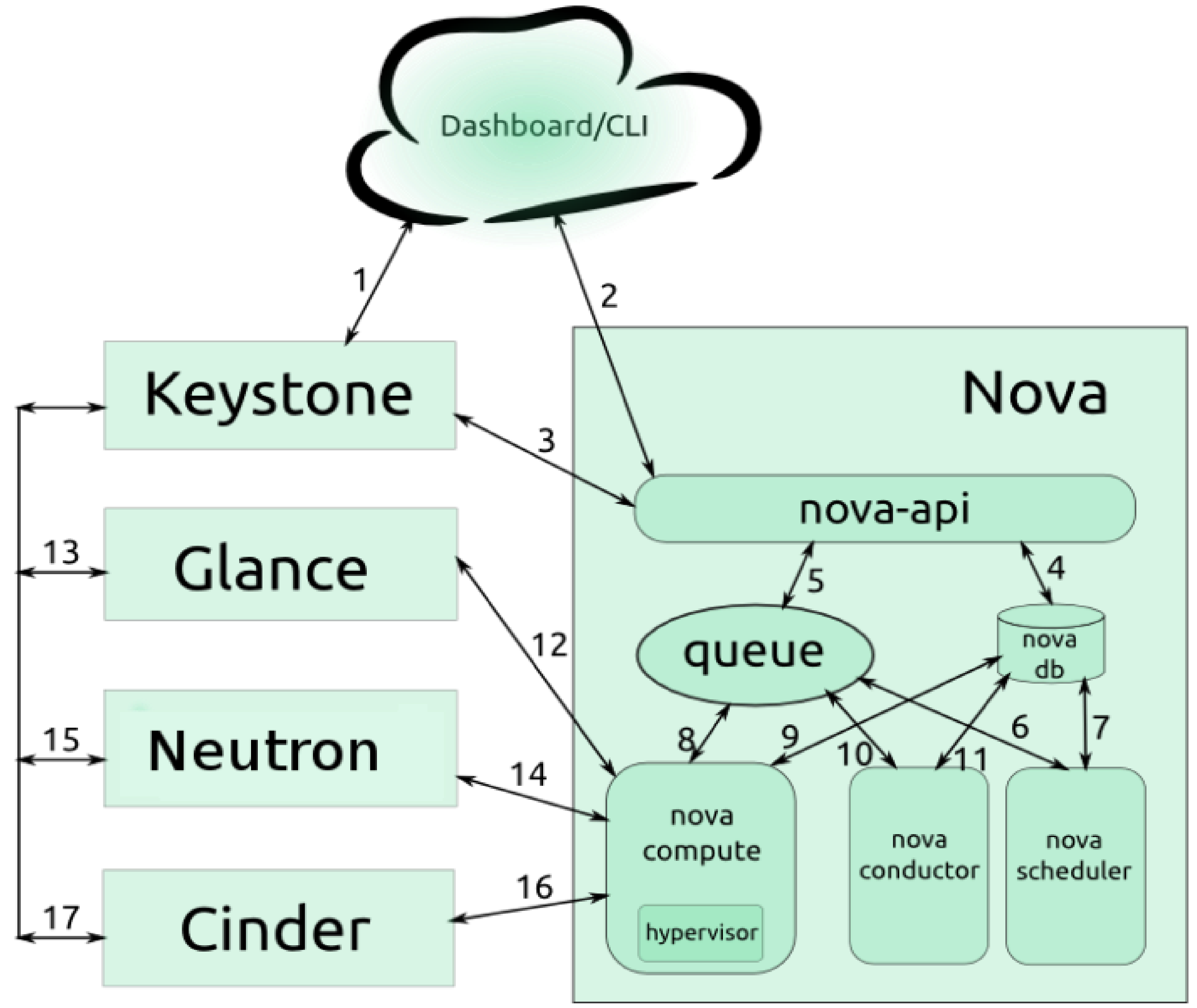
1. **Data Sources:**
   - Ceilometer gathers data from various sources:
   - **REST APIs:** It collects data by querying OpenStack services through their REST APIs, allowing it to obtain real-time information about resource usage.
   - **Notifications:** Ceilometer listens to notifications sent out by OpenStack services. These notifications contain important events, allowing Ceilometer to react promptly to changes in the system.
   - **Direct Polling:** The system directly polls resources, meaning it actively queries instances and other elements to collect usage data.

The telemetry services

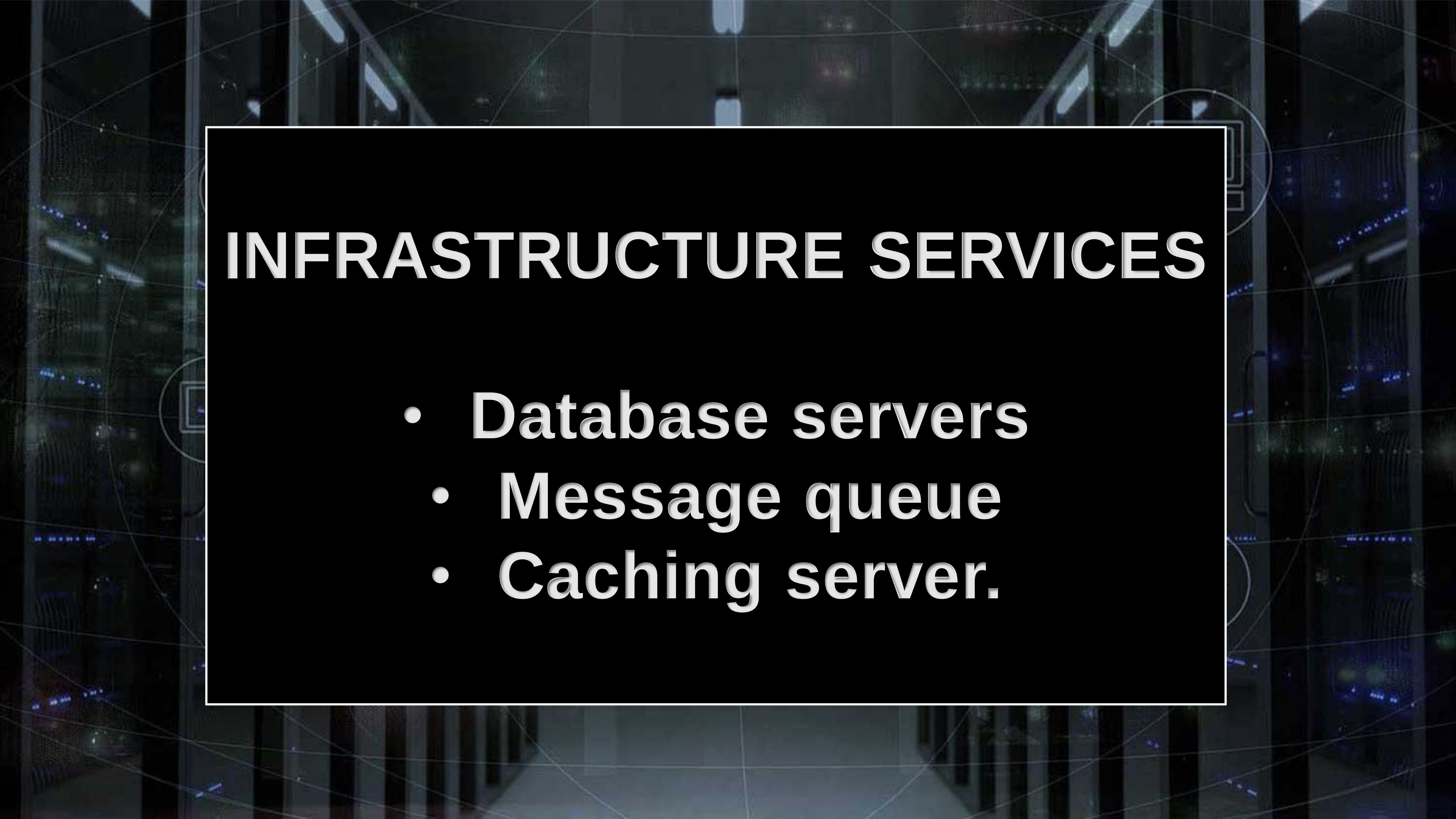
2. **Data Collection Agents:**
   - **Polling Agent:**
     - Located on controller nodes, the polling agent employs a plugin framework to collect diverse resource utilization data.
   - **Central Agent:**
     - The central agent operates on the controller node as well.
     - This agent can gather data from object and block storage, network components, and even physical hardware using SNMP (Simple Network Management Protocol).

The telemetry services

3. **Ceilometer Compute Agent:**
   - **Location**: Operates on the compute node.
   - **Purpose**: Gathers hypervisor statistics, providing insights into the performance and usage of the compute resources.
4. **Ceilometer IPMI Agent:**
   - **Location**: Runs on the monitored compute node.
   - **Data Collection**: Polls physical resource usage data using IPMI on servers equipped with IPMI sensors and the ipmitool utility.
   - **Data Transmission**: Uses the message bus to send the collected data.
5. **Ceilometer Notification Agent:**
   - **Location**: Operates on the controller node.
   - **Function**: Consumes notification data from various OpenStack components on the message bus, aiding in real-time monitoring and response.



VM provisioning in-depth



INFRASTRUCTURE SERVICES

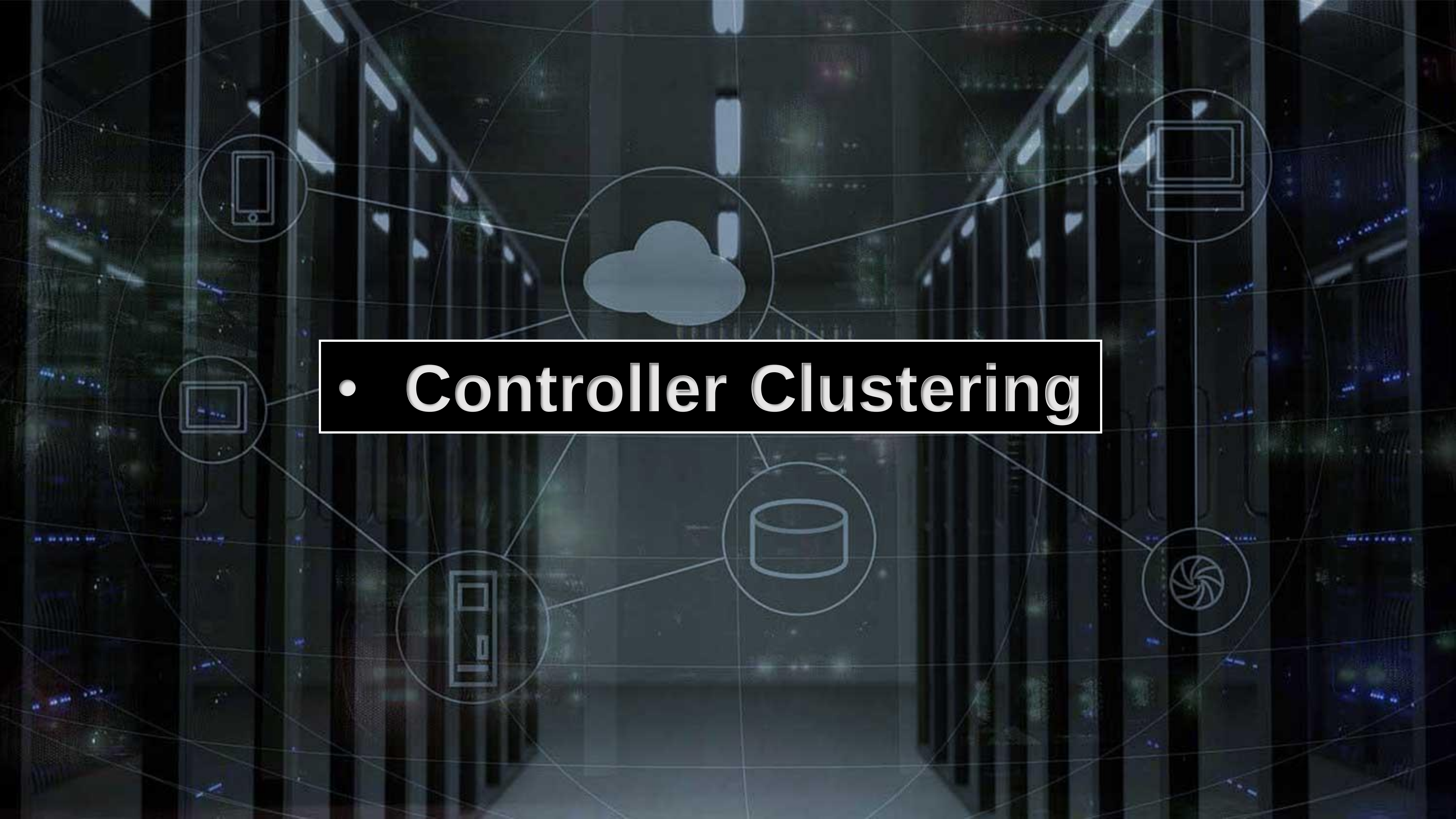
- Database servers
- Message queue
- Caching server.

INFRASTRUCTURE SERVICES

- **1. Database Servers (e.g., MySQL, PostgreSQL):**
- Database servers store and manage structured data, crucial for applications to read, write, and manipulate information.
- Example: In a web application, a database server could store user profiles, product details, and transaction records. For instance, an online store's database might have tables for customers, products, orders, and payments.
- **2. Message Queue (e.g., RabbitMQ, ZeroMQ):**
- Message queues facilitate communication between different parts of a system, allowing asynchronous processing of tasks and improving system reliability.
- Example: In a distributed application, a message queue could be used to handle user requests. For instance, when a user places an order, a message is sent to the queue. Workers in the system pick up these messages and process them independently, ensuring orders are processed even if one part of the system experiences high load.

INFRASTRUCTURE SERVICES

- **3. Caching Server (e.g., Memcached):**
- Caching servers store frequently accessed data in a high-speed, easily retrievable form, reducing the need to fetch data from slower, long-term storage.
- Example: In a content-heavy website, a caching server could store images, stylesheets, or database query results. When a user accesses the site, these elements are retrieved from the cache rather than generating them from scratch or fetching them from a database. This significantly speeds up the website's loading time..



- Controller Clustering

Previous Year University Question Paper 2021

Part A

- Explain clustering and its types used in OpenStack. (3)
- What is the service provided for image management in OpenStack? Explain it.(3)

Part B

- Describe Keystone service and its service providers. (6)
- Explain the working of Ansible playbooks. (6)

Previous Year University Question Paper 2022

Part A

- Explain asymmetric clustering and symmetric clustering. (3)
- List out functionalities handled by the cloud controller.(3)

Part B

- Explain keystone architecture. (6)
- Describe on
    a) Nova – scheduler service
    b) Horizon dashboard
    c) Telemetry services

Further Reading

- https://docs.openstack.org/project-deploy-guide/openstack-ansible/latest/deploymenthost.html
- https://www.openstack.org/videos/summits/vancouver-2015/deploying-openstack-with-ansible
- https://github.com/openstack/openstack-ansible

Behind The Scene

- **Nova-compute**
- In the next few rows we try to briefly explain what happens when a new request for starting an OpenStack instance is done. Note that this is very high level description.
- Authentication is performed either by the web interface **horizon** or **nova** command line tool:
  - keystone is contacted and authentication is performed
  - a *token* is saved in the database and returned to the client (horizon or nova cli) to be used with later interactions with OpenStack services for this request.
- **nova-api** is contacted and a new request is created:
  - it checks via *keystone* the validity of the token
  - checks the authorization of the user
  - validates parameters and create a new request in the database
  - calls the scheduler via queue

Behind The Scene

- **nova-scheduler** find an appropriate host
  - reads the request
  - find an appropriate host via filtering and weighting
  - calls the choosen *nova-compute* host via queue
- **nova-compute** read the request and start an instance:
  - generates a proper configuration for the hypervisor
  - get image URI via image id
  - download the image
  - request to allocate network via queue
- **nova-compute** requests creation of a neutron *port*
- **neutron** allocate the port:
  - allocates a valid private ip
  - instructs the plugin agent to implement the port and wire it to the network interface to the VM[#]_

Behind The Scene

- **nova-api** contacts *cinder* to provision the volume
  - gets connection parameters from cinder
  - uses iscsi to make the volume available on the local machine
  - asks the hypervisor to provision the local volume as virtual volume of the specified virtual machine
- **horizon** or **nova** poll the status of the request by contacting **nova-api** until it is ready.

- THANK YOU
- You can find me @navyamolkt@amaljyothi.ac.in