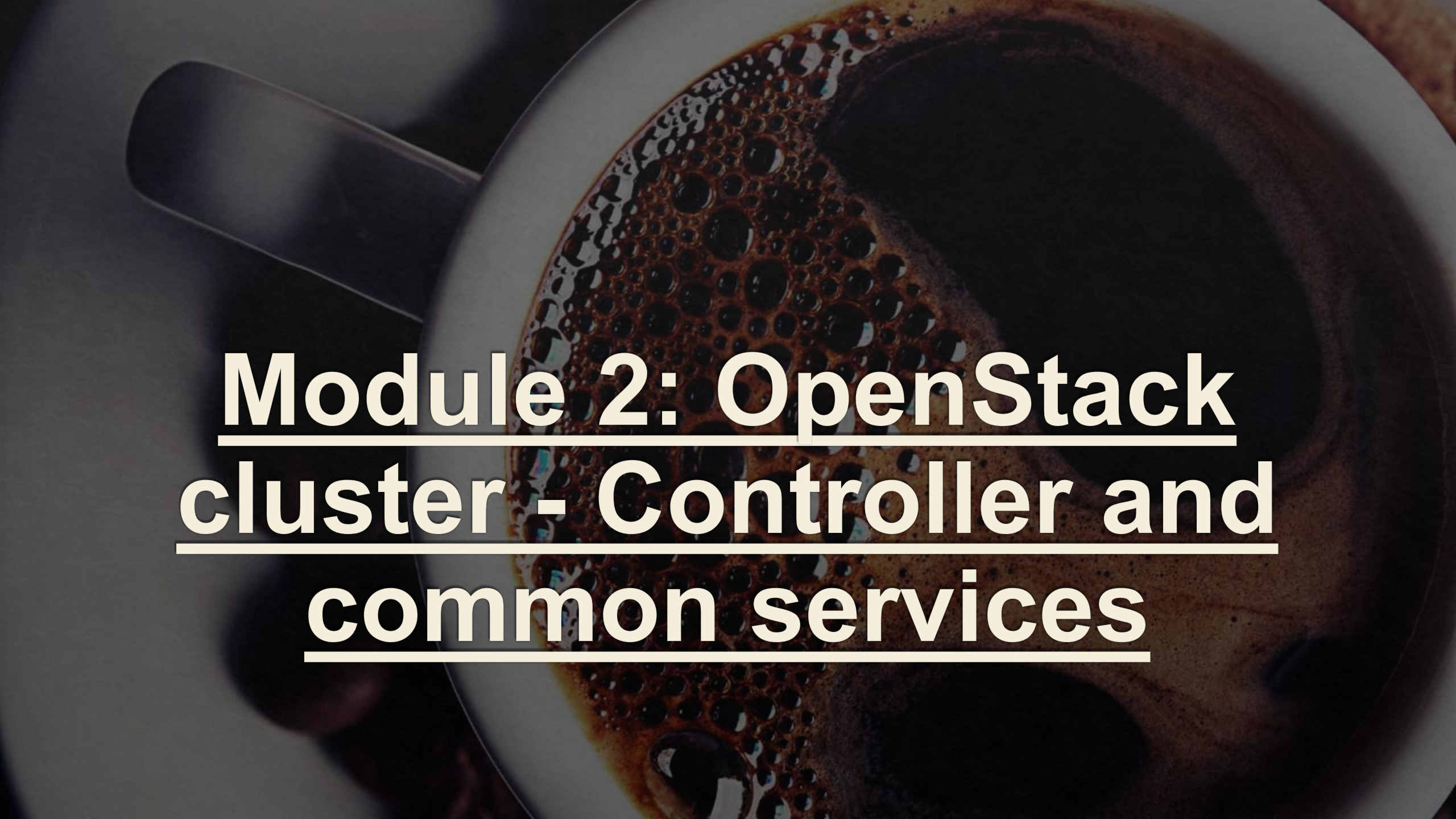


Module 2: OpenStack cluster - Controller and common services

CONTENTS

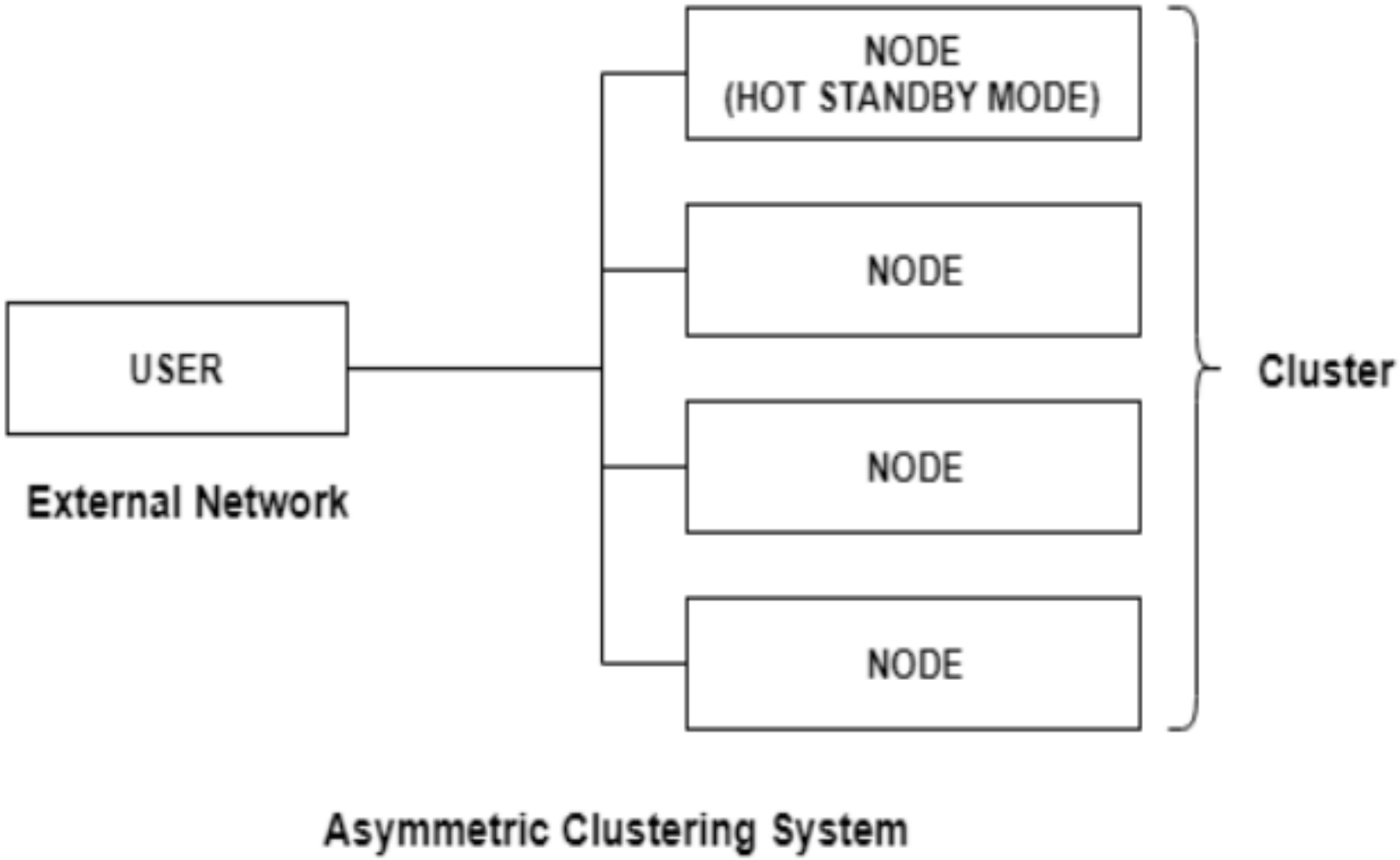
- OpenStack Cluster – The Cloud Controller and Common Services- Asymmetric clustering, Symmetric clustering, The cloud controller - The keystone service.
- The nova-conductor service, The nova-scheduler service, The API services, Image management, The network service, The horizon dashboard, The telemetry services.

Understanding the art of clustering

- Do not be afraid to claim that clustering actually provides high availability in a given infrastructure.
- The aggregation of the capacity of two or more servers is meant to be a server cluster.
- This aggregation will be performed by means of the accumulation of several machines.

Asymmetric clustering

- 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.



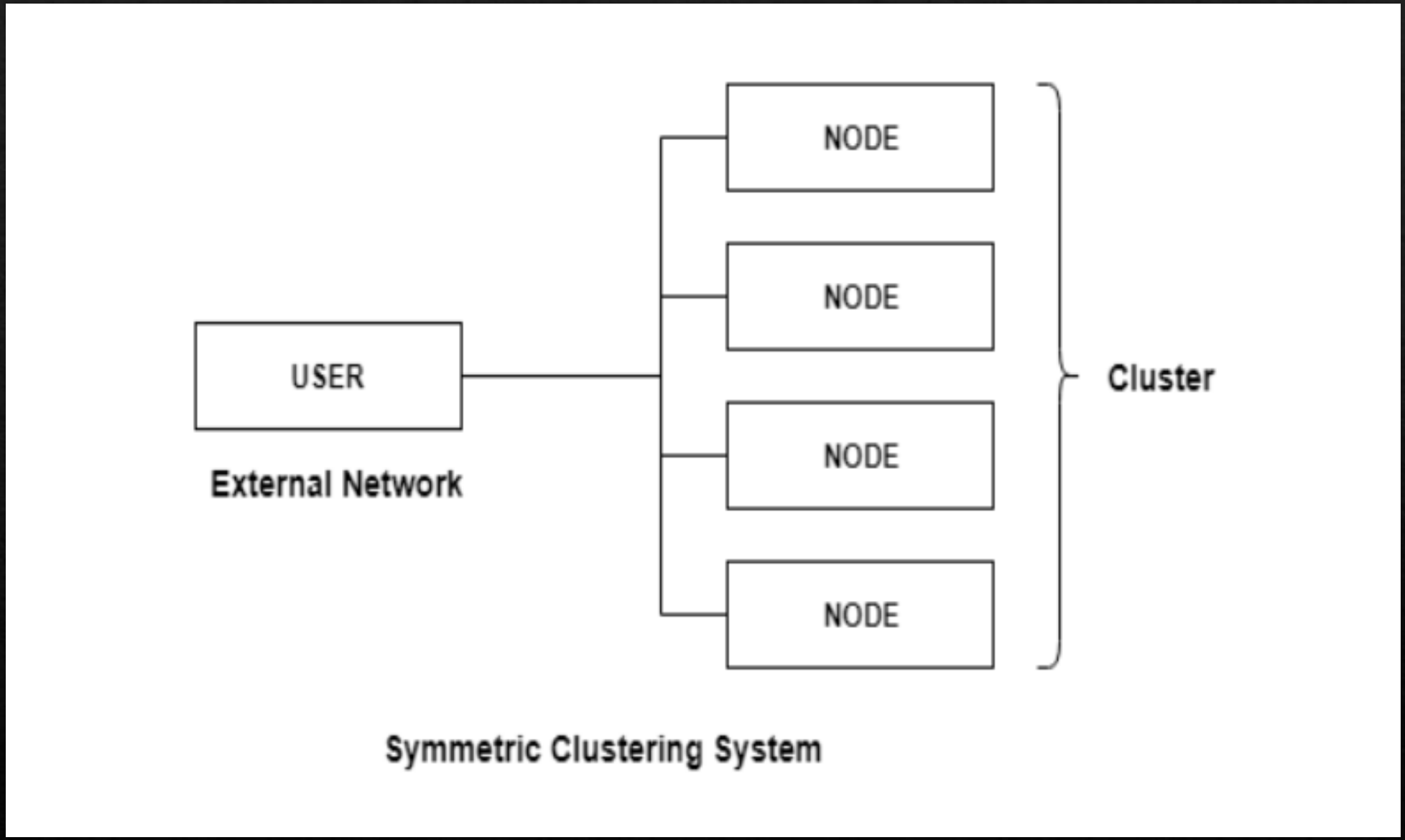
Asymmetric Clustering System

How Asymmetric Clustering Works

- The following steps demonstrate the working of the asymmetric clustering system −
  - 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.
  - A distributed cache is used in asymmetric clustering to improve the performance of the system.
  - Resources such as memory, peripheral devices etc. are divided between the nodes of the asymmetric clustering system at boot time.

Symmetric clustering

- 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 Clustering System

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.

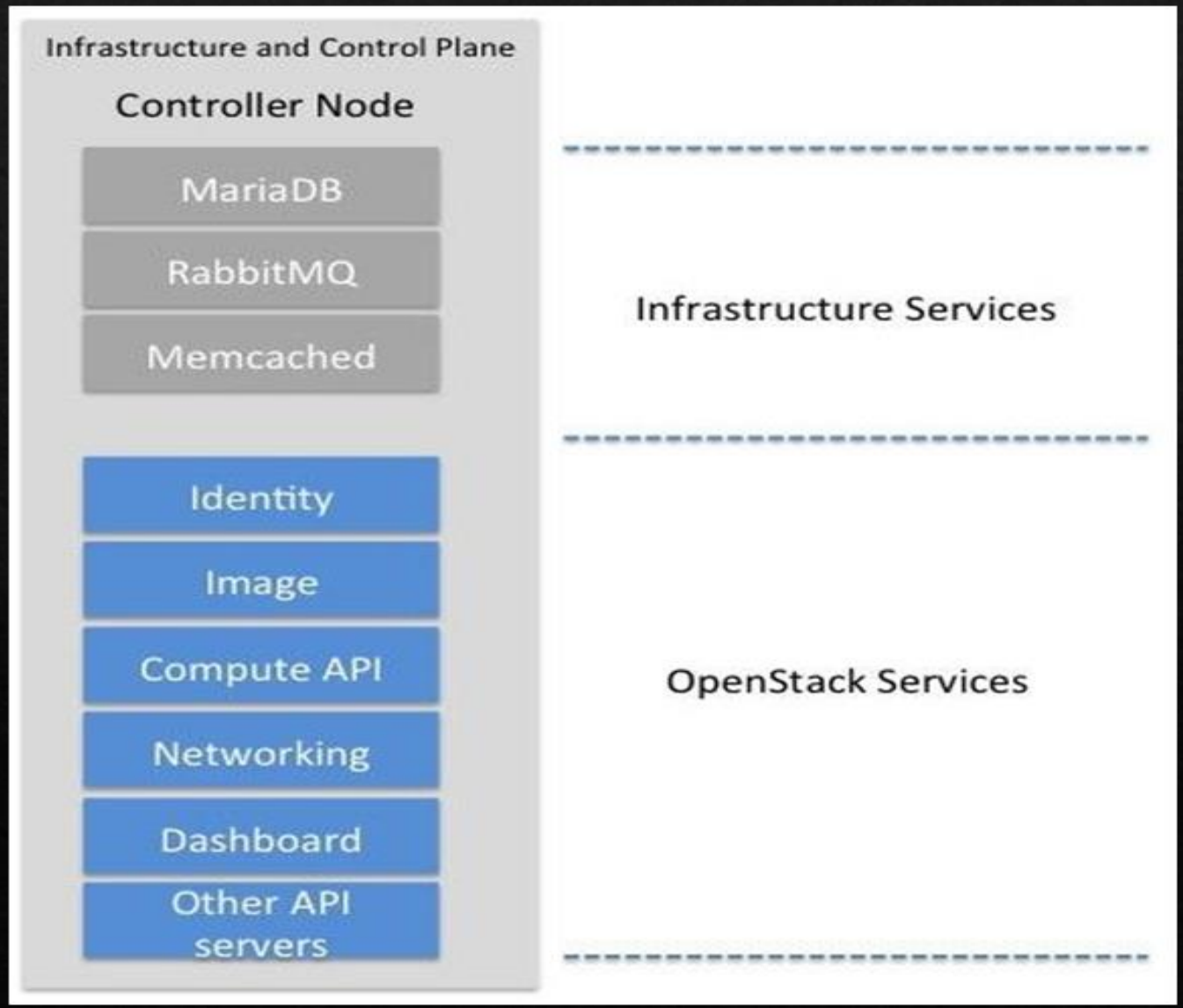
The cloud controller

- 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.

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.

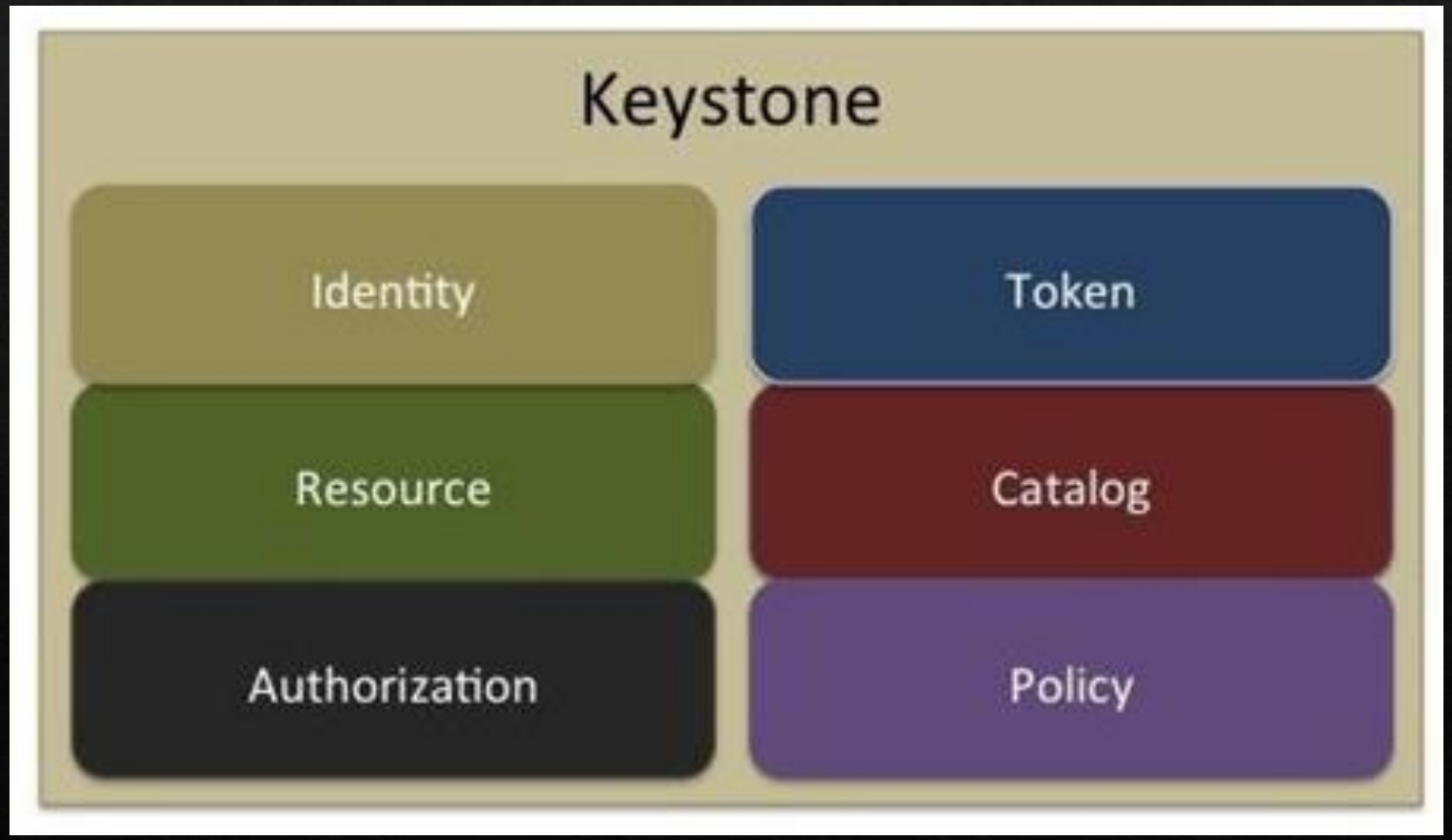
- The controller node aggregates the most critical services for OpenStack. Let's look at the services on the controller node:



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



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(**Lightweight Directory Access Protocol**). 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 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 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 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 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.

Federated Keystone

- 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.
  - 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

- In a nutshell, the steps involved in authenticating users using Federated Identity are as follows:
  - The user tries to access a resource available with the SP.
  - 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.
  - 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.
  - The user then uses the unscoped token to determine a list of accessible domains and projects in the Service Provider's cloud.
  - 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.

The nova-conductor service

- If you have tried to install OpenStack from the Grizzly release onward, while checking Nova services running in your OpenStack node, you may have noticed a new service called nova-conductor.
- This amazing new service has changed the way the nova- compute service accesses the database. It was added to provide security by decoupling direct database access from the compute nodes.
- A vulnerable compute node running the nova-compute service may be attacked and compromised. You can imagine how attacking a virtual machine can bring the compute node under the control of the attacker.

- Even worse, it can compromise the database. Then, you can guess the rest: your entire OpenStack cluster is under attack!
- ***The job of nova-conductor is to carry out database operations on behalf of compute nodes and provide a layer of database access segregation.***
- So, you can assume that nova-conductor compiles a new layer on top of nova-compute. Furthermore, instead of resolving the complexity of the database requests bottleneck, nova- conductor parallelizes the requests from compute nodes.

The nova-scheduler service

- Several workflow scheduling studies and implementations have been recently conducted in cloud computing, generally in order to define the best placement of a resource provisioning.
- In our case, we will decide which compute node will host the virtual machine. It's important to note that there are bunches of scheduling algorithms in OpenStack.

- Nova-scheduler may also influence the performance of the virtual machines. Therefore, OpenStack supports a set of filters that implement a resource availability check on the compute nodes and then runs the filtered list of compute nodes through a weighting mechanism to arrive at the best node to start a virtual machine.
- The scheduler gives you the choice to configure its options based on a certain number of metrics and policy considerations.
- Additionally, nova-scheduler can be thought of as the ***decision-maker box*** in a cloud controller node that ***applies complex algorithms for the efficient usage and placement of virtual machines.***

- Eventually, the scheduler in OpenStack, as you may understand at this stage, will be running in the cloud controller node.
- A point here that needs to be investigated is: what about different schedulers in a high-availability environment? In this case, we exploit the openness of the OpenStack architecture by running multiple instances of each scheduler, as all of them are listening to the same queue for the scheduling requests.
- It is important to know that 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.

The API services

- In a nutshell, we have already covered the nova-api service.
- It might be important to step forward and learn that nova-api is considered the *orchestrator engine* component in cloud controller specifications. Without any doubt, nova-api is assembled in the controller node.
- The nova-api service may also fulfill more complicated requests by passing messages within other daemons by means of writing to the databases and queuing messages.
- As this service is based on the endpoint concept where all API queries are initiated, nova-api provides two different APIs using either the OpenStack API or EC2 API.

- This makes it imperative to decide which API will be used before deploying a cloud controller node that may conduct a real issue, as you may decide to take over both APIs.
- The reason behind this is the *heterogeneity* of the information presentation used by each API;
  - for example, OpenStack uses names and numbers to refer to instances, whereas the EC2 API uses identifiers based on hexadecimal values.
- Additionally, we have brought compute, identity, image, network, and storage APIs to be placed in the controller node, which can also be chosen to run other API services.
- For instance, we satisfy our deployment by gathering the majority of the API services to run in the cloud controller node.
    - `An **Application Programming Interface** (**API**) enables public access to the OpenStack services and offers a way to interact with them. The API access can be performed either through a command line or through the Web. `

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.

The network service

- Just like OpenStack's Nova service provides an API for dynamic requests to compute resources, we adopt the same concept for the network by allowing its API to reside in the cloud controller, which supports extensions to provide advanced network capabilities, such as firewall, routing, load balancing, and so on.
- As was discussed earlier, separating most of the network workers is highly recommended.

- On the other hand, you must also think about the huge amount of traffic that hits a cloud controller with regards to its running multiple services; therefore, you should bear in mind the performance challenges that you may face.
- In this case, clustering best practices come in to help your deployment be more scalable and increase its performance. The previous mentioned techniques are essential but not sufficient. They need basic hardware support with at least 10 GB of **bonded** NICs, for example.
- The NIC bonding technique is used to increase the available bandwidth. Two or more bonded NICs appear to be the same physical device.

The Horizon dashboard

- As the OpenStack dashboard runs behind an Apache web server and is based on the Python Django web application framework, you might consider providing a separate node that is able to reach the API endpoints to run the Horizon dashboard to decrease the load on your cloud controller node.
- Several OpenStack deployments in production run Horizon in the controller node but still leave it up to you to monitor it and take separate decisions.

The telemetry services

- Prior to the **Liberty** release, the telemetry service was composed initially of one component, known as Ceilometer, providing metering of resource utilization in OpenStack. In a multi- user shared infrastructure, keeping track of resource usage is of prime importance.
- Resource utilization data can be used for multiple purposes such as billing, capacity planning, and auto scaling of virtual infrastructure based on demand and throughput.
- Ceilometer was originally designed to be part of the billing solution for OpenStack but, later, numerous other use cases were discovered for its independent existence.

- Since the Liberty release, the telemetry service has been decomposed in two additional projects where it can be optionally enabled to function alongside Ceilometer:
  - Aodh: a sub-project which takes care of generating alarms when resources utilization crosses a certain predefined threshold.
  - Gnocchi: a sub-project for metrics and events storage at scale
- Ceilometer uses agent-based architecture. The service itself is composed of an API server, multiple data collection agents, and the data store. Ceilometer collects data through the REST APIs, listening to notifications, and directly polling resources.

- The following are the data collection agents and their roles:
  - The Ceilometer polling agent provides a plugin framework to collect various resource utilization data. It runs on the controller nodes and provides a flexible means to add data collection agents by implementing them as plugins.
  - The Ceilometer central agent runs on the controller node and uses the REST APIs exposed by other OpenStack services to poll for virtual resources created by tenants. The Central agent can poll for object and block storage, network, and physical hardware using SNMP.
  - The Ceilometer compute agent runs on the compute node and its main purpose is to collect hypervisor statistics.
  - The Ceilometer IPMI agent runs on the monitored compute node. It polls data by using the Intelligent Platform Management Interface (IPMI) on the servers to collect physical resource utilization data. The monitored servers must be equipped with IPMI sensors and installed with the ipmitool utility. It uses the message bus to send the collected data.
  - The Ceilometer notification agent consumes notification data from various OpenStack components on the message bus. It also runs on the controller node.

Alarms

- Apart from the various agents for data collection, Ceilometer also contains alarm notification and evaluation agents. The alarm evaluation agent checks for resource utilization parameters crossing a threshold value and uses the notification agent to emit the corresponding notification. The alarm system can be used to build auto-scaling infrastructure or take other corrective actions. The Aodh sub-project has been designed to decouple the alarming from Ceilometer and can be enabled when deploying a fresh installation of OpenStack

Events

- Events describe the state change of a resource in OpenStack; for example, the creation of a volume, starting a virtual machine instance, and so on. Events can be used to provide data to the billing systems. Ceilometer listens to the notification emitted from various OpenStack services and converts them to events.