






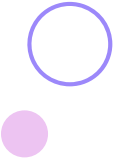
OpenStack Compute & Storage

Module III





Content

- 
- 
- 
- 
- 
1. Openstack Compute
 2. Deciding on the Hypervisor
 3. OpenStack Magnum project
 4. Segregating the compute cloud
 5. Overcommitment considerations
 6. Storing instances' alternatives
 7. Understanding instance booting
 8. Planning for service recovery
- 

01

OPENSTACK COMPUTE



OPENSTACK COMPUTE

- The compute nodes should be separately deployed in the cluster, as it forms the resources part of the OpenStack infrastructure.
- The compute servers are the heart of the cloud computing service, as they provide the resources that are directly used by the end users.
- From a deployment perspective, an OpenStack compute node might not be complicated to install, as it will basically run nova-compute and the network agent for Neutron.
- The cloud controller presents a wide range of services, so we have agreed to use HA(High Availability) clusters and a separate deployment scheme for the controller to crystallize the cloud controller setup. This way, we suffer less from the issue of service downtime.
- On the other hand, a compute node will be the space where the virtual machine will run; in other words, the space on which the end user will focus.
- The end user only wants to push the button and get the application running on the top of your IaaS layer.
- It is your mission to guarantee a satisfactory amount of resources to enable your end user to do this.
- A good design of cloud controller is needed but is not enough; we need to take care over compute nodes as well.

OPENSTACK COMPUTE

- In cloud computing, the term “compute” describes concepts and objects related to software computation. It is a generic term used to reference processing power, memory, networking, storage, and other resources required for the computational success of any program.
- In cloud computing, the term “compute” describes concepts and objects related to software computation. It is a generic term used to reference processing power, memory, networking, storage, and other resources required for the computational success of any program.
- For example, applications that run machine learning algorithms or 3D graphics rendering functions require many gigs of RAM and multiple CPUs to run successfully. In this case, the CPUs, RAM, and Graphic Processing Units required will be called compute resources, and the applications would be compute-intensive applications.

OPENSTACK COMPUTE



- What are compute resources?
- Compute resources are measurable quantities of compute power that can be requested, allocated, and consumed for computing activities. Some examples of compute resources include:

CPU

- The central processing unit (CPU) is the brain of any computer. CPU is measured in units called millicores. Application developers can specify how many allocated CPUs are required for running their application and to process data.

Memory

- Memory is measured in bytes. Applications can make memory requests that are needed to run efficiently.
- If applications are running on a single physical device, they have limited access to the compute resources of that device. But if applications run on the cloud, they can simultaneously access more processing resources from many physical devices.

OPENSTACK COMPUTE




- What are compute service components?
- Components for receiving requests and launching and managing Virtual Machines
- Summary of the various building blocks of the compute service:
- **The nova-api** service interacts with the user API calls that manage the compute instances. It communicates with the other components of the compute service over the message bus.
- **The nova-scheduler** is the service that listens to the new instance request on the message bus. The job of this service is to select the best compute node for the new instance.
- **The nova-compute** service is the process responsible for starting and terminating the virtual machines. This service runs on the compute nodes and listens for new requests over the message bus.





OPENSTACK COMPUTE

- **nova- conductor service**
 - The compute nodes are not provided direct access to the database. This design limits the risk of a compromised compute node providing the attacker complete access to the database. The database access calls from the compute nodes are handled by the nova- conductor service.
 - Nova uses the **metadata service** to provide the virtual machines with configuration data used to initialize and configure the instance.
 - ● **nova-consoleauth** daemon provides authentication for the VNC proxy, such as novncproxy and xvncproxy, access to the console of instances over the VNC protocol.
- 



Compute Services in AWS

1. Amazon Elastic Compute Cloud (EC2)
2. Amazon Elastic Container Registry (ECR)
3. Amazon Elastic Container Service (ECS)
4. Amazon Elastic Kubernetes Service (EKS)
5. AWS Elastic Beanstalk (EBS)
6. AWS Lambda
7. Amazon Lightsail
8. AWS Batch

aws



Amazon EC2



amazon
ECR

AWS BATCH



AWS Lambda



Elastic
Beanstalk



AWS ECS



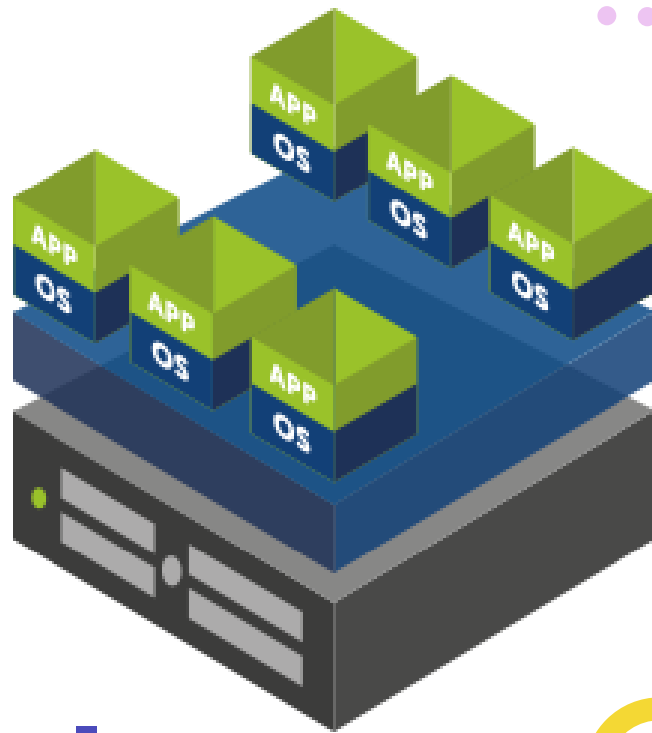
Amazon EKS



Amazon
Lightsail

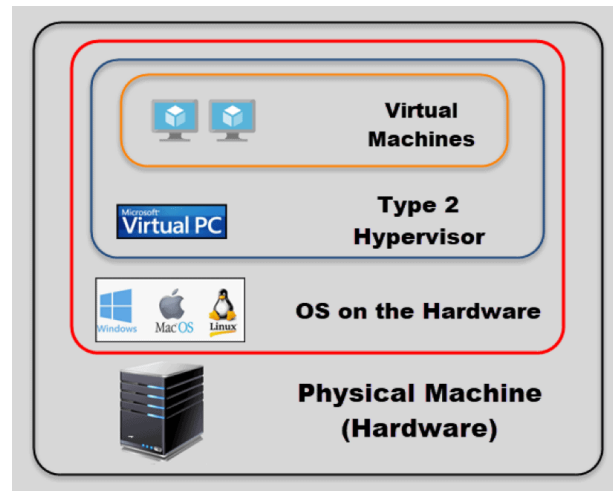
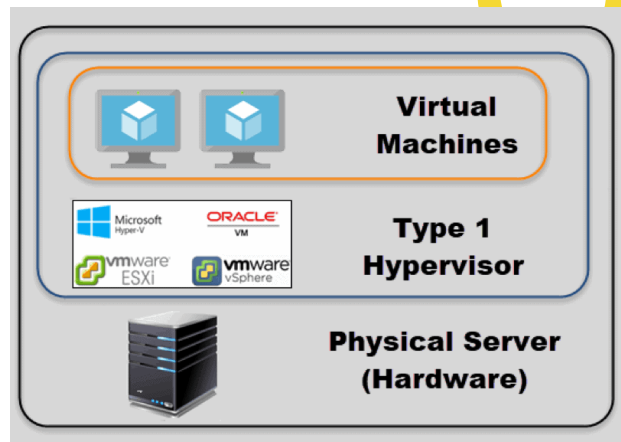
02

Deciding on the Hypervisor

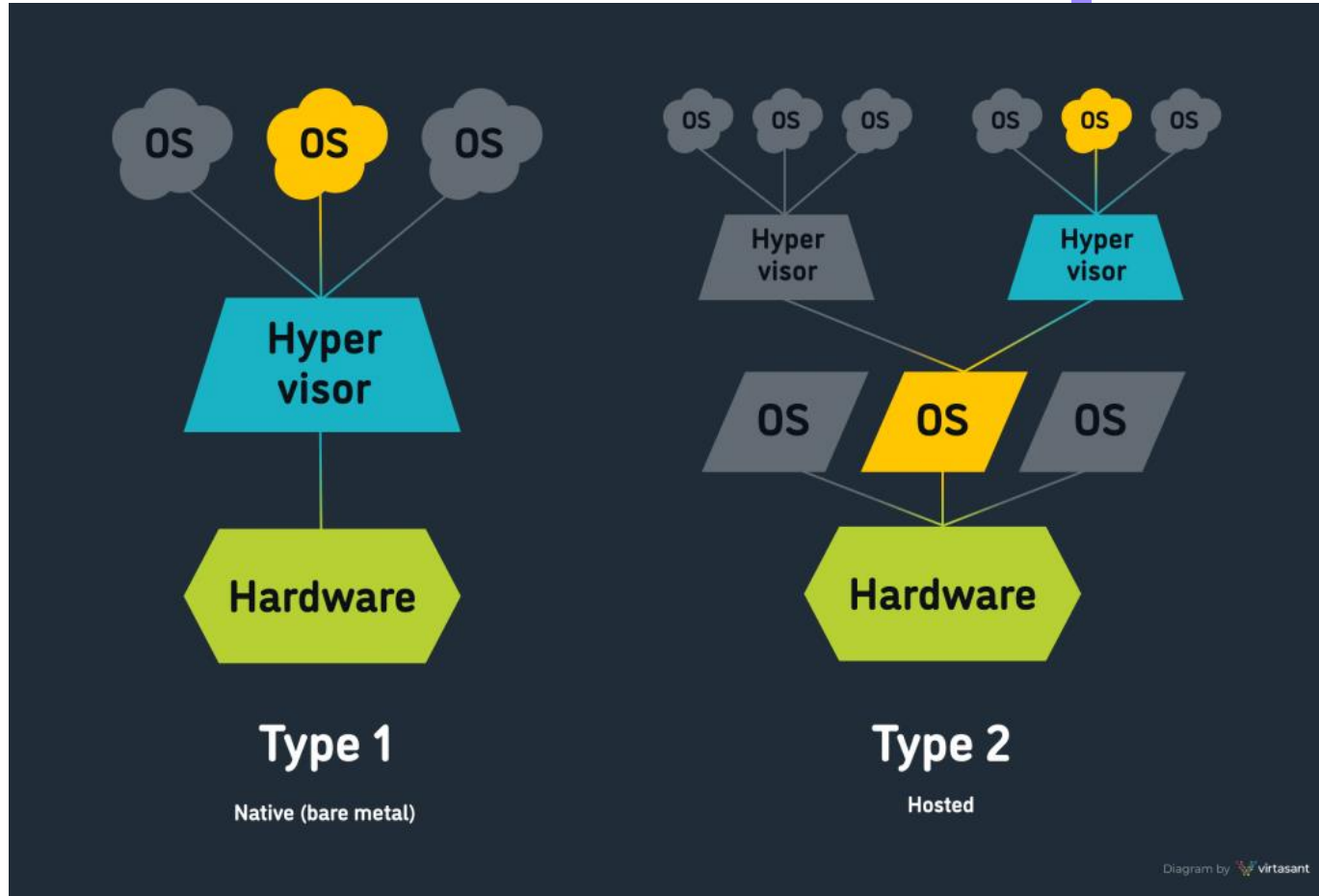


Deciding on the Hypervisor

- A hypervisor is a crucial piece of software that makes virtualization possible. It abstracts guest machines and the operating system they run on, from the actual hardware.
- Also known as Virtual Machine Monitor
- Hypervisors emulate available resources so that guest machines can use them. No matter what operating system you boot up with a virtual machine, it will think that actual physical hardware is at its disposal.
- Type 1 Hypervisor (also called bare metal or native)
- Type 2 Hypervisor (also known as hosted hypervisors)



Deciding on the Hypervisor



Popular Hypervisors



• VMware Hypervisors

- VMware vSphere/ESXi is a type 1 hypervisor for data center server virtualization. vSphere can be used in an on-premise environment or a cloud environment.
- VMware Fusion is a type 2 hypervisor, targeting MacOS users.
- VMware Workstation is also a type 2 hypervisor for Windows and Linux platforms.

• Hyper-V Hypervisor

- Hyper-V, Microsoft's hypervisor designed for Windows systems, is considered type 1 according to Microsoft. It runs on Windows Server Core, but Hyper-V inserts itself below the operating system and runs directly on the physical hardware.

• Citrix Hypervisor

- Formerly known as XenServer, Citrix Hypervisor is a commercial type 1 hypervisor.

• Open Source Hypervisors

- KVM kernel modules and user space tools are available in most Linux distributions through their packaging systems.
- Xen is a type 1 hypervisor that is a project under the Linux Foundation.

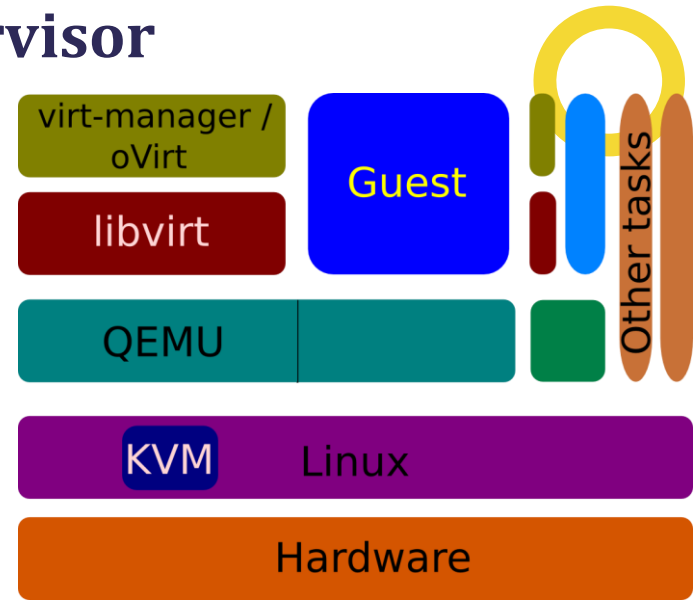


Popular Type 1 Hypervisors



Deciding on the Hypervisor

- KVM is the default hypervisor for OpenStack compute
- Most of the **OpenStack nova-compute** deployments run **KVM** as the main hypervisor.
- The fact is that KVM is best suited for workloads that are natively stateless using **libvirt**.
- **libvirt** is an open-source API, daemon and management tool for managing platform virtualization.
- It can be used to manage KVM, Xen, VMware
- ESXi, QEMU and other virtualization technologies.
- These APIs are widely used in the orchestration layer of hypervisors in the development of a cloud-based solution.



Deciding on the Hypervisor

You can check out your compute node from `/etc/nova/nova.conf` in the following lines:

```
compute_driver=libvirt.LibvirtDriver  
libvirt_type=kvm
```

For proper, error-free hypervisor usage, it is required to first check whether KVM modules are loaded from your compute node:

```
# lsmod | grep kvm  
kvm_intel or kvm_amd
```

Otherwise, you may load the required modules via the following:

```
# modprobe -a kvm
```

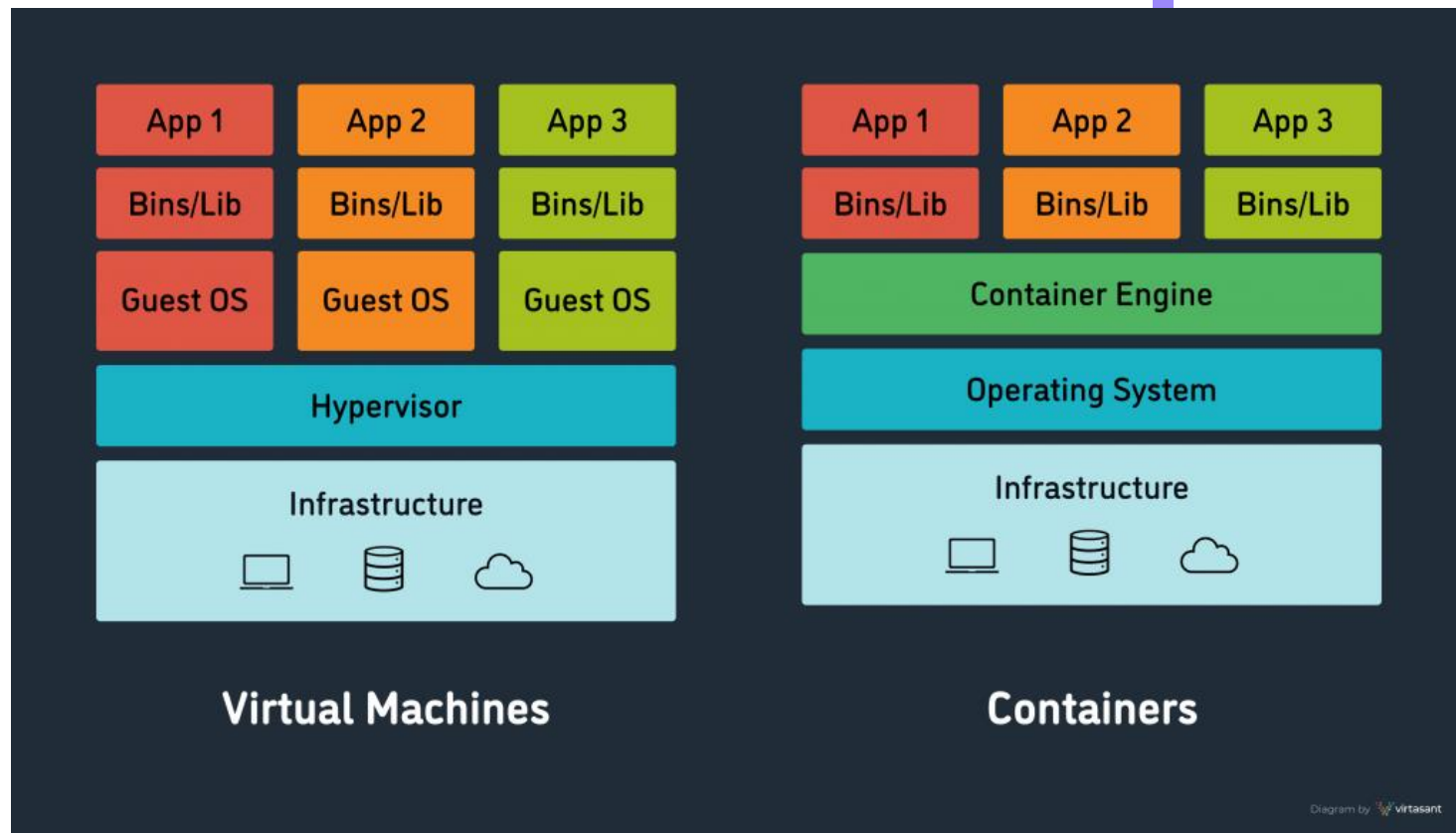
To make your modules persistent at reboot, which is obviously needed, you can add the following lines to the `/etc/modules` file when your compute node is an Intel-based processor:

```
kvm  
kvm-intel
```

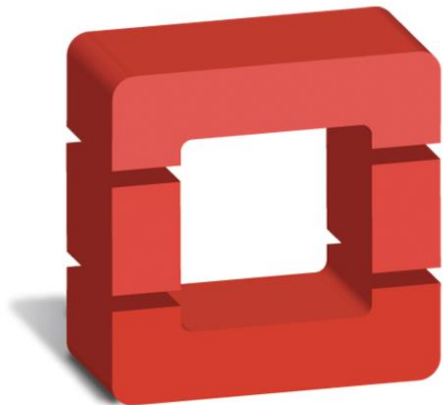

Containers vs Hypervisors

- While they are similar in some ways, containers and hypervisors are not a choice you have to make.
- Indeed, Hypervisors and containers are typically utilized simultaneously.
- Hypervisors allow you to divide a single computer's hardware resources between multiple VMs.
- Containers allow you to split a single computer into segregated logical namespaces.
- Containers are all about isolation, not virtualization. From an application development point of view, they look like the same thing, but they work in different layers.
- Popular containerization tools, like Docker, can create and run multiple containers on the host's Linux kernel. Every container has its specific network stack and its own process space, including all of the underlying dependencies required to run the application. Containers do not contain the operating system, so they are very compact, and start up in milliseconds.
- Containers provide an excellent platform for building and sharing packaged, ready-to-run applications, and micro-services.
- Containers run closer to the application layer, while hypervisors run closer to the hardware layer. In most cases, hypervisors run the VMs, and containers run on those VMs.

Containers vs Hypervisors



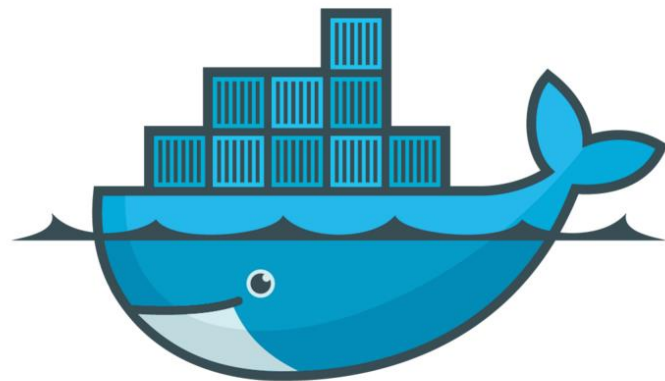
Containers vs Hypervisors



openstack®

CLOUD SOFTWARE

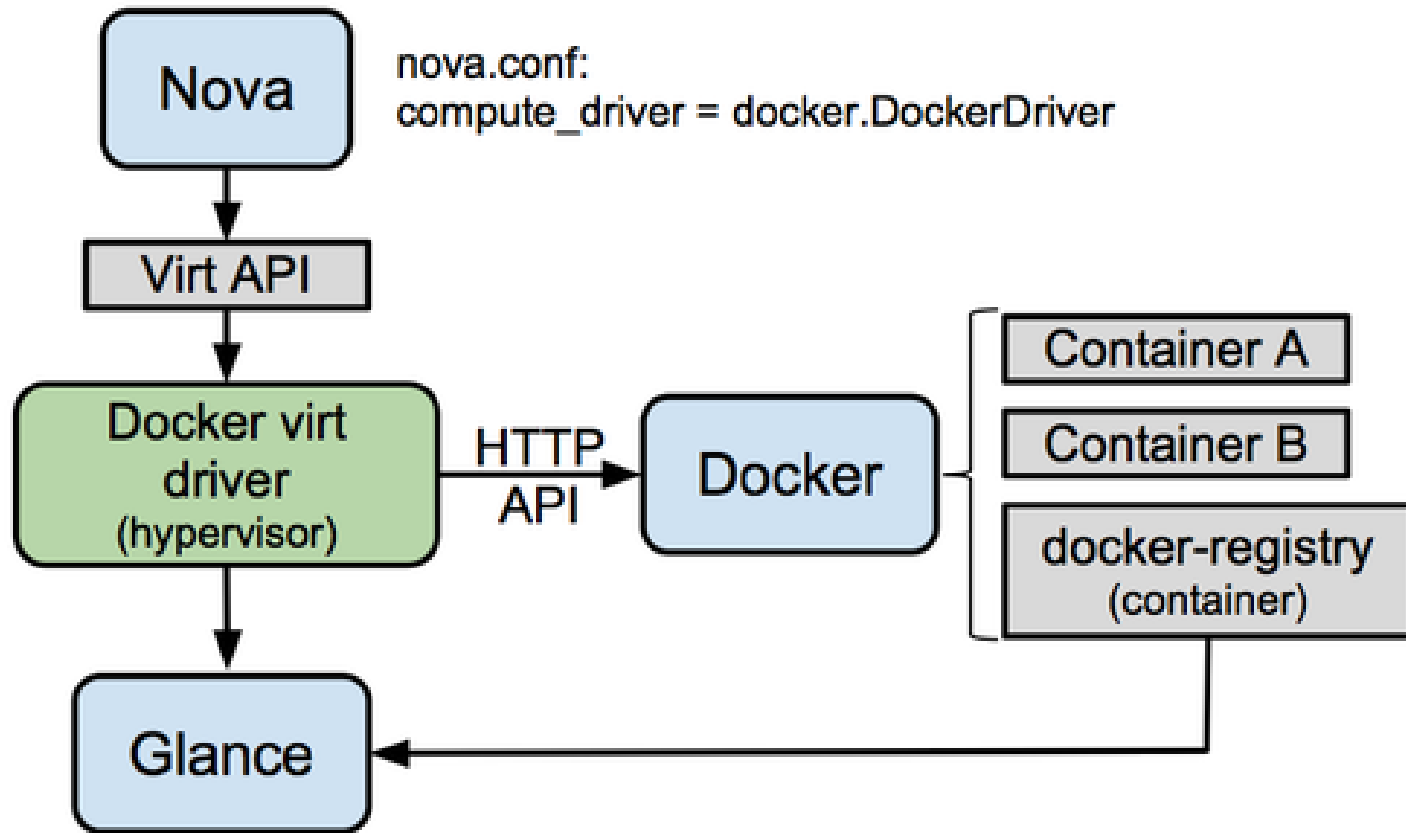
Infrastructure orchestration tool



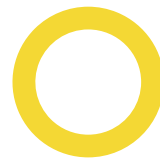
docker

Application provisioning tool

The Docker containers



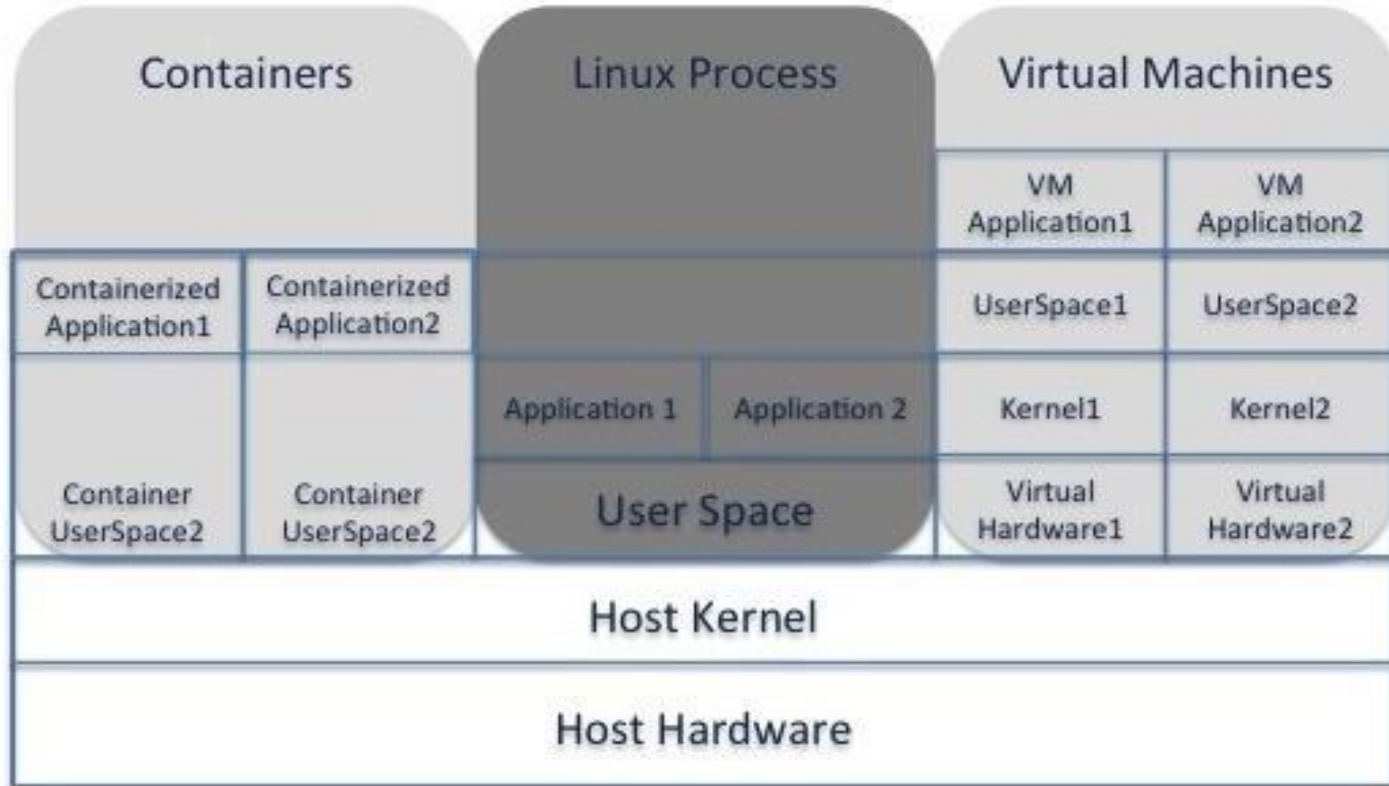
The Docker containers



- Docker driver for OpenStack nova-compute.
- While a virtual machine provides a complete virtual hardware platform on which an operating system can be installed in a conventional way and applications can be deployed, a container, on the other hand, provides an isolated user space to host an application.
- The containers use the same underlying kernel of the host operating system. In a way, containers are providing an encapsulation mechanism that captures the user space configuration and dependencies of an application.
- This encapsulated application runtime environment can be packaged into portable images.
- The advantage of this approach is that an application can be delivered along with its dependency and configuration as a self- contained image



The Docker containers



The Docker containers

- Docker helps enterprises deploy their applications in highly portable and self-sufficient containers, independent of the hardware and hosting provider.
- It brings the software deployment into a secure, automated, and repeatable environment.
- What makes Docker special is its usage of a large number of containers, which can be managed on a single machine.
- Additionally, it becomes more powerful when it is used alongside Nova.
- Docker is based on containers that are not a replacement for virtual machines, but which are very specific to certain deployments.
- Containers are very lightweight and fast, which may be a good option for the development of new applications and even to port older applications faster.
- Imagine an abstraction that can be shared with any application along with its own specific environment and configuration requirements without them interfering with each other.
- This is what Docker brings to the table.
- Docker can save the state of a container as an image that can be shared through a central image registry.
- This makes Docker awesome, as it creates a portable image that can be shared across different cloud environments.

03

OpenStack
Magnum
project



MAGNUM

an OpenStack Community Project

MAGNUM VISION

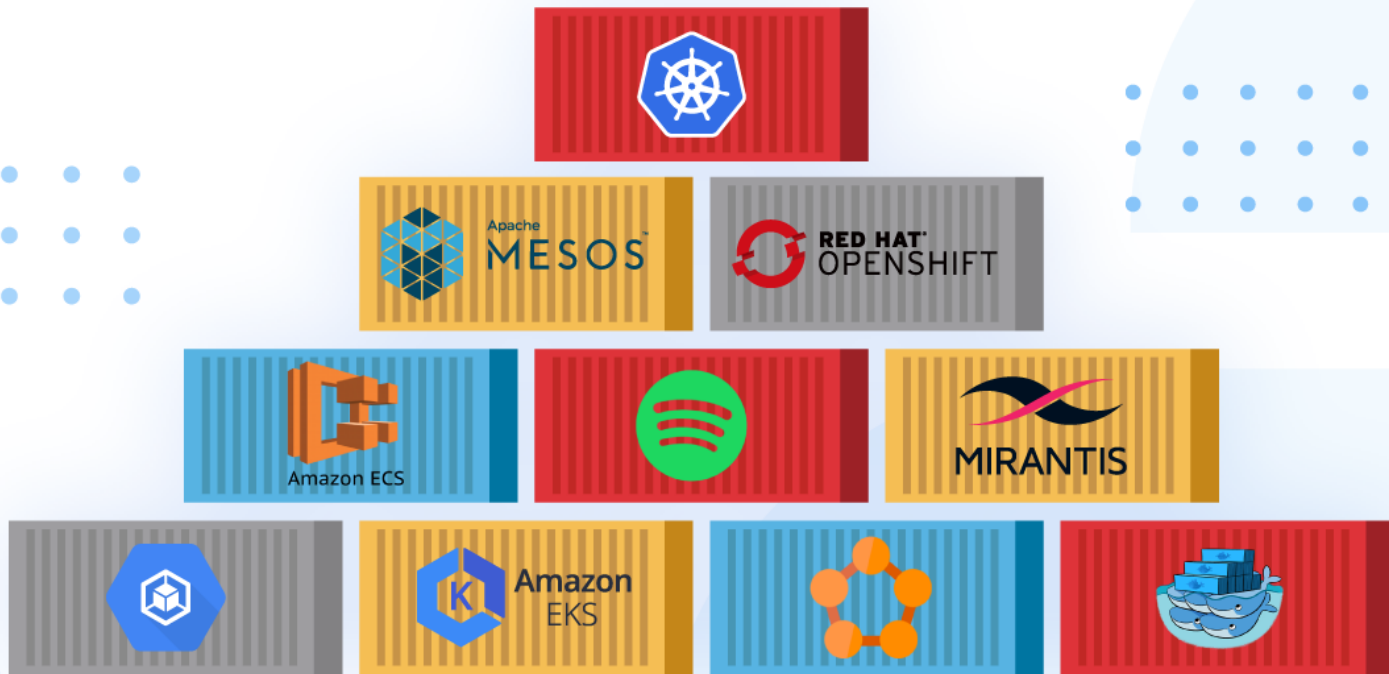
Best infrastructure software combined with the best container software



Container orchestration

- Container orchestration automates the **deployment, management, scaling, and networking of containers.**
- Enterprises that need to deploy and manage hundreds or thousands of Linux® containers and hosts can benefit from container orchestration.
- It can help you to deploy the same application across different environments without needing to redesign it.
- And **microservices** in containers make it easier to orchestrate services, including storage, networking, and security.
- Containers give your microservice-based apps an ideal application deployment unit and self-contained execution environment. They make it possible to run multiple parts of an app independently in microservices, on the same hardware, with much greater control over individual pieces and life cycles.

Top 10 Container Orchestration Tools



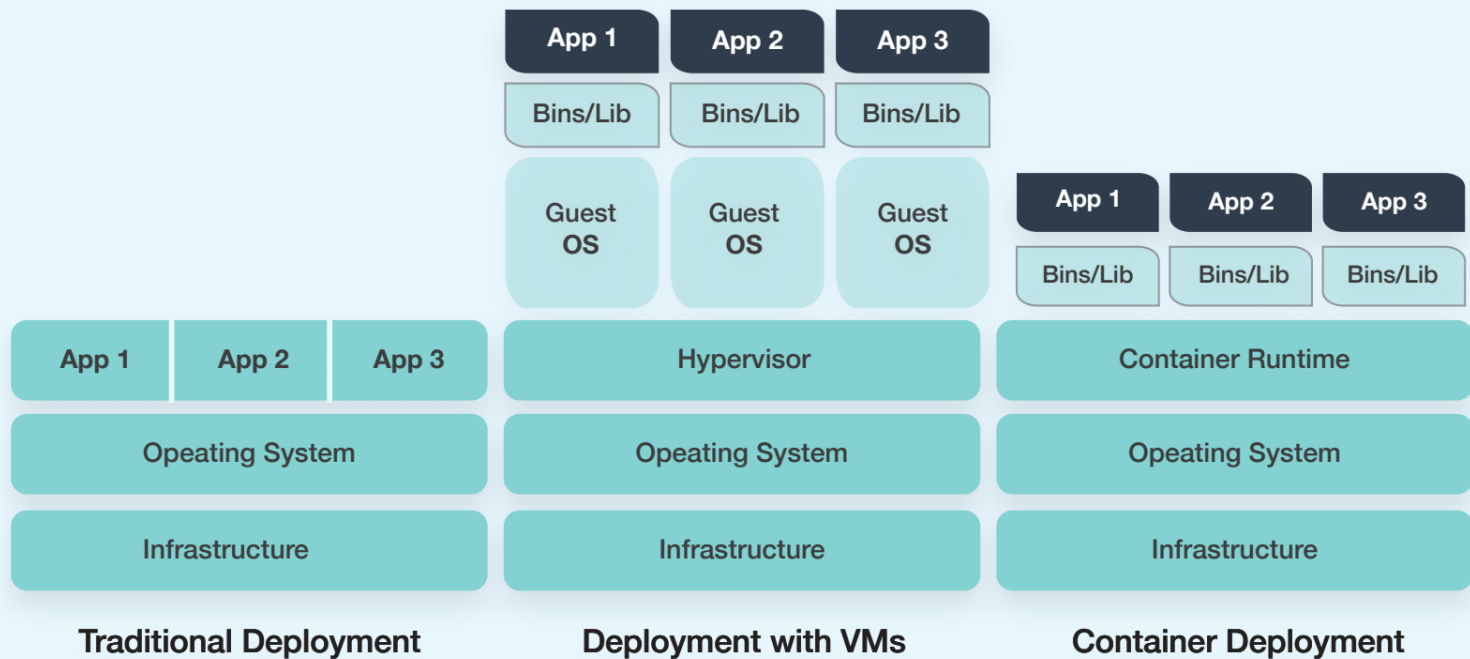
Container orchestration how it works

- When you use a container orchestration tool, such as **Kubernetes**, you will describe the configuration of an application using either a YAML or JSON file.
- The configuration file tells the **configuration management tool** where to find the container images, how to establish a network, and where to store logs.
- When deploying a new container, the **container management tool** automatically schedules the deployment to a cluster and finds the right host, taking into account any defined requirements or restrictions.
- The **orchestration tool** then manages the container's lifecycle based on the specifications that were determined in the compose file.



Self-paced Micro course










OpenStack Magnum Project





- Magnum is an OpenStack API service developed by the OpenStack Containers Team making container orchestration engines (COE) such as Docker Swarm and Kubernetes available as first class resources in OpenStack.
 - Magnum uses Heat to orchestrate an OS image which contains Docker and COE and runs that image in either virtual machines or bare metal in a cluster configuration.
 - Magnum offers complete life-cycle management of COEs in an OpenStack environment, integrated with other OpenStack services for a seamless experience for OpenStack users who wish to run containers in an OpenStack environment.
- 
- 
- 



OpenStack Magnum Project

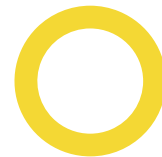


Following are few salient features of Magnum:

- Standard API based complete life-cycle management for Container Clusters
 - Multi-tenancy for container clusters
 - Choice of COE: Kubernetes, Swarm
 - Choice of container cluster deployment model: VM or Bare-metal
 - Keystone-based multi-tenant security and auth management
 - Neutron based multi-tenant network control and isolation
 - Cinder based volume service for containers
 - Integrated with OpenStack: SSO experience for cloud users
 - Secure container cluster access (TLS enabled)
- 
- 



OpenStack Magnum Project








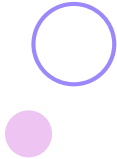
- The application containers, are not like virtual machines. Hosting an application in containers very often means deploying multiple containers, each running just a single process; these containerized processes then collaborate with each other to provide the complete features of the application.
- This means that, unlike virtual machines, containers running a single process would most likely need to be spawned in groups, would require network connectivity for communication between collaborating processes, and have storage requirements too. This is the idea behind the OpenStack Magnum project. Magnum is built to support orchestration of groups of connected containers using a Container Orchestration Engine (COE) such as Kubernetes, Apache Mesos, Docker Swamp.





OpenStack Magnum Project







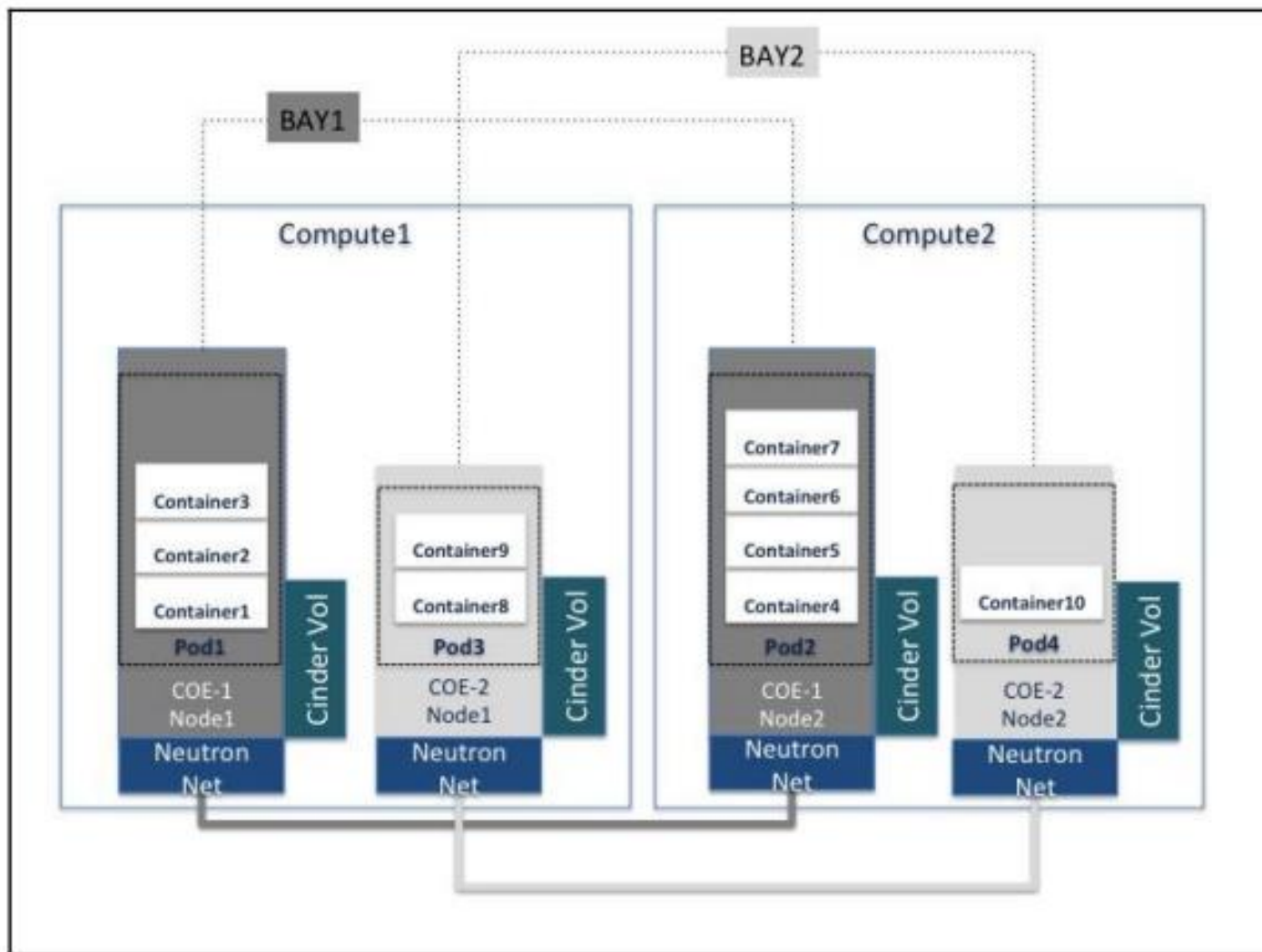
- In contrast to the Nova Docker driver, Magnum works by first deploying the COE nodes and then launching groups of containers for deploying applications on these nodes.
 - The COE system acts as an orchestrator to launch applications across multiple containers
 - Magnum leverages OpenStack services to provide the COE infrastructure.
 - COE Nodes are deployed as Nova instances.
 - It uses the Neutrons networking service to provide network connectivity between the COE nodes, although the connectivity between the application containers is handled by the COE itself.
 - Each of the COE nodes is connected to a Cinder volume that is used to host the application containers. Heat is used to orchestrate the virtual infrastructure for COE.
- 
- 
- 
- 
- 
- 



OpenStack Magnum Project



- Magnum defines the following components:
 - **A Bay** is a group of nodes that run COE software. The nodes can run an API server or minions. The COE architecture consists of an API server that receives the orchestration requests from the user. The API server then interacts with the minion server where the Containers are launched.
 - **A Pod** is a group of containers running on the same node and the concept of Service that consists of one or more Bays that provide to a consumable service. The Service abstraction is required as the bays providing the service may get created and deleted while the service is still available.
 - **A BayModel** is a template that can be used to create a Bay; it is similar to the concept of Nova flavor that is used to create a virtual machine instance.
- 
- 
- 
- 



04


Segregating the compute cloud

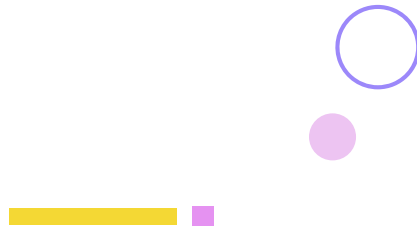
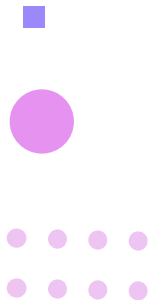




Segregating the compute cloud



- As your cloud infrastructure grows in size, you need to devise strategies to maintain **low latency** in the API services and **redundancy** of your service.
 - To cope with **unplanned downtime** due to natural forces or based on the hypervisor capability itself, the operator must plan for service continuity.
 - OpenStack Nova provides several concepts that help you segregate the cloud resources.
 - Each segregation strategy brings in its own advantages and shortcomings.
- 



Availability zone and Region



Regions

- Regions are separate geographic areas that use to house its infrastructure.

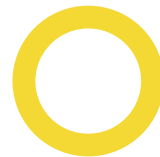
- These are distributed around the world so that customers can choose a region closest to them in order to host their cloud infrastructure there



- Amazon Web Services (AWS) currently has 26 regions in operation and a further 8 under development, meaning that the company will have a total of 34 regions available by the end of 2024.

- Within each AWS region are 3 to 6 isolated, and physically separate locations, known as availability zones, which have independent power, cooling, and physical security, and are connected to each other with a redundant, low-latency, private fiber-optic network.

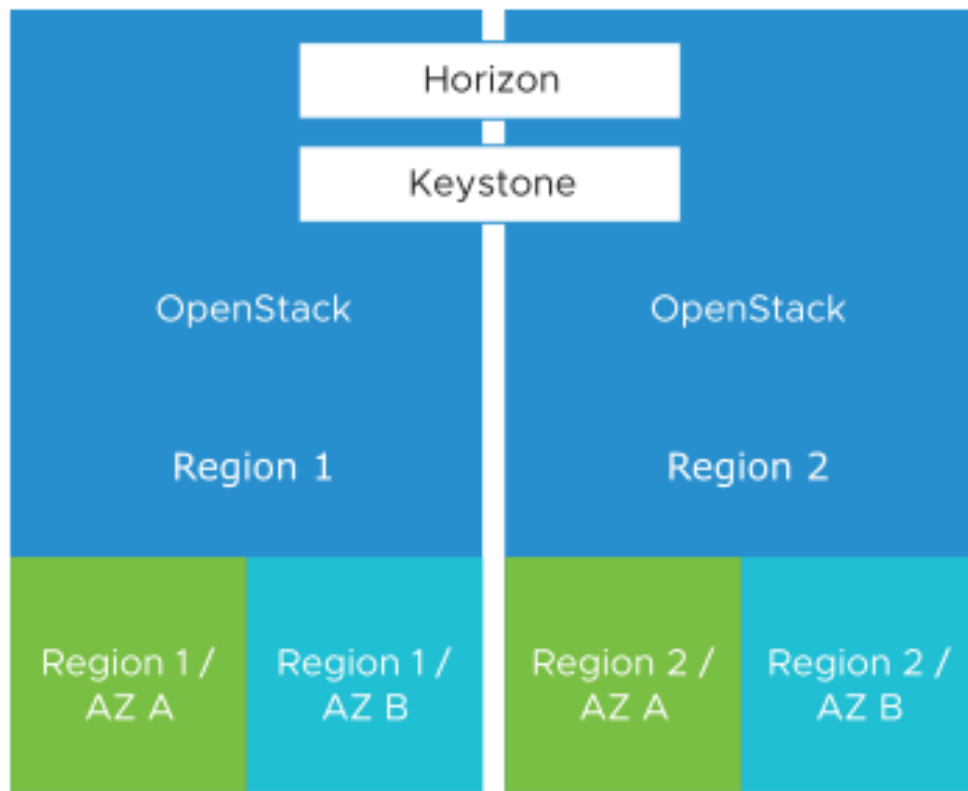
Region



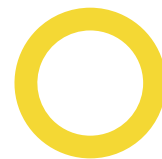
- A Region is full OpenStack deployment, including its own API endpoints, networks and compute resources, excluding the Keystone and Horizon. Each Region shares a single set of Keystone and Horizon services.
- The concept of cells allows extending the compute cloud by segregating compute nodes into groups but maintaining a single Nova API endpoint.
- Nova regions take an orthogonal approach and allow multiple Nova API endpoints to be used to launch virtual machines.
- Each Nova region has a complete Nova installation, with its own set of compute nodes its and own Nova API endpoint.
- Different Nova regions of an OpenStack cloud share the same Keystone service for authentication and advertising the Nova API endpoints.
- The end user will have to select the region where he wants the virtual machines to be launched.
- Another way of thinking about the contrast between cells and regions is that Nova - cells implementation uses RPC calls, while regions use REST APIs to provide segregation.



Region

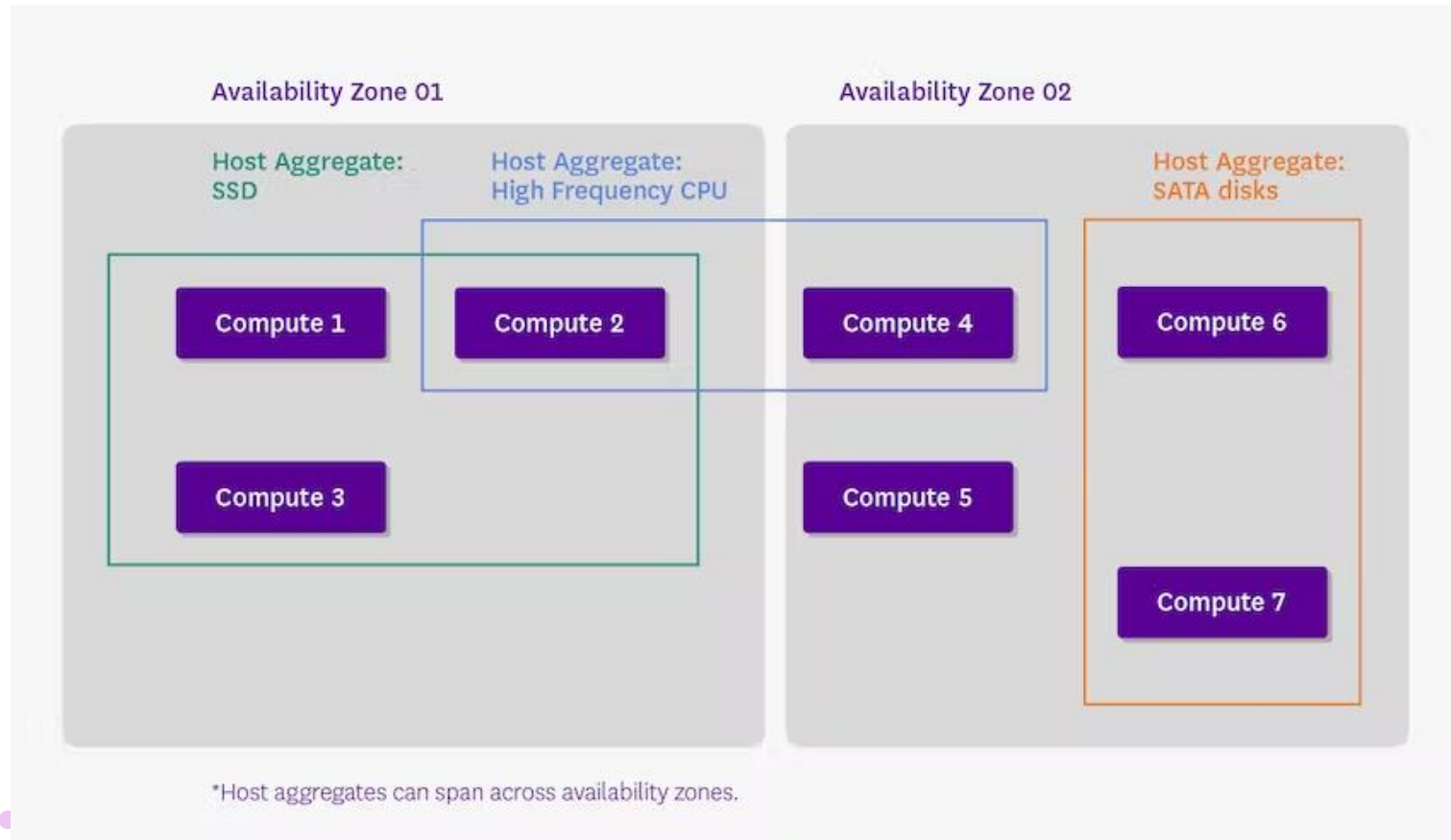


Host Aggregates



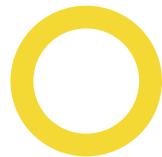
- Host Aggregates are logical groups of compute nodes.
- Each aggregate consists of compute nodes and the relating metadata
- Host aggregates are a mechanism for partitioning hosts in an OpenStack cloud, or a region of an OpenStack cloud, based on **arbitrary characteristics**
- Customers using OpenStack as a service never see host aggregates; administrators use them to group hardware according to various properties.
- Most commonly, host aggregates are used to differentiate between physical host configurations.
- For example, you can have an aggregate composed of machines with 2GB of RAM and another aggregate composed of machines with 64GB of RAM.
- This highlights the typical use case of aggregates: defining static hardware profiles.
- Once an aggregate is created, administrators can then define specific public flavors from which clients can choose to run their virtual machines (the same concept as EC2 instance types on AWS).
- Flavors are used by customers and clients to choose the type of hardware that will host their instance.

Host Aggregates





Host Aggregates



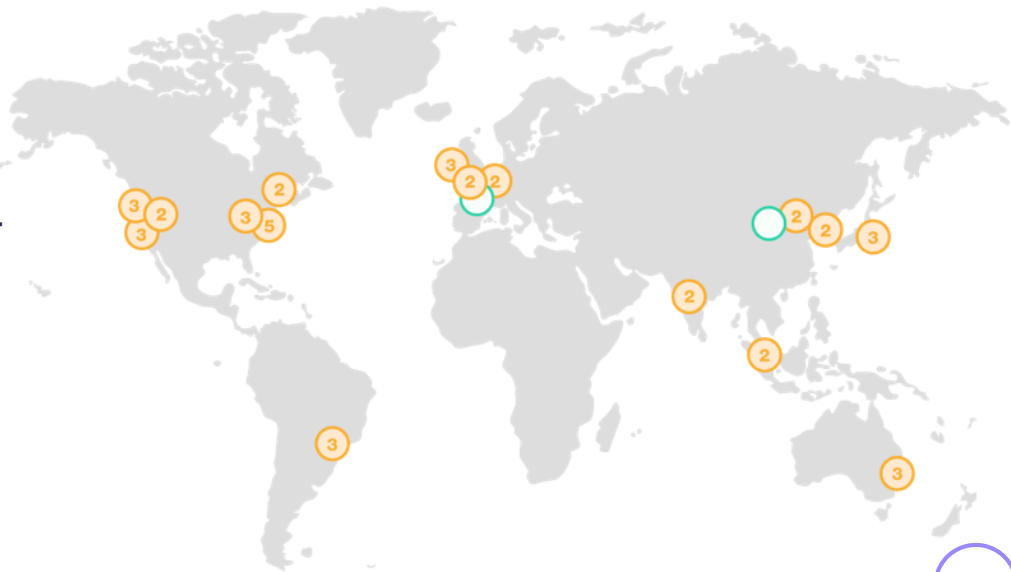
- The Host Aggregate is a strategy of grouping together compute nodes that provides compute resources with specialized features.
- Let's say you have some compute nodes with better processors or better networking capability.
- Then you can make sure that virtual machines of a certain kind that require better physical hardware support are always scheduled on these compute nodes.
- Attaching a set of metadata to the group of hosts can create the Host Aggregates.
- To use a Host Aggregate, the end user needs to use a flavor that has the same metadata attached.



Segregating the compute cloud

Availability Zones

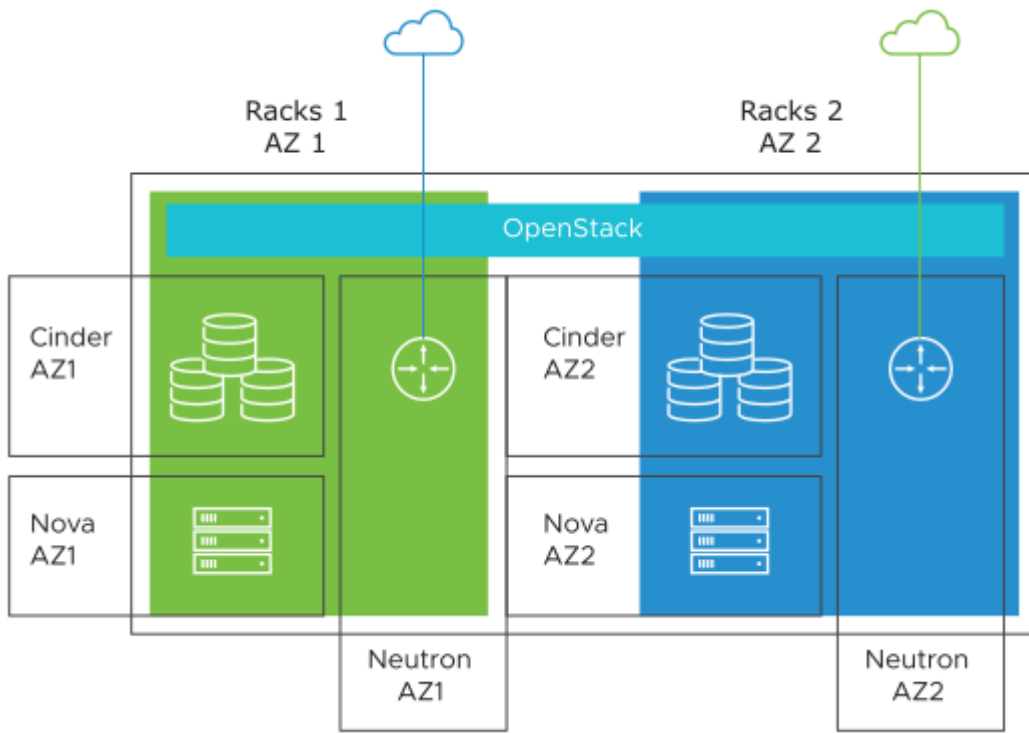
- An availability zone is a logical data center in a region available for use by any customer.
- Each zone in a region has redundant and separate power, networking and connectivity to reduce the likelihood of two zones failing simultaneously.
- A common misconception is that a single zone equals a single data center. In fact, each zone is backed by one or more physical data centers, with the largest backed by five.
- While a single availability zone can span multiple data centers, no two zones share a data center.



Availability zones

- The concept of **Availability Zones (AZ)** in Nova is to group together compute nodes based on fault domains: for example, all compute nodes hosted on a rack in the lab. All the nodes connect to the same **Top-of-Rack (ToR)** switch or are fed from the same Power Distribution Unit (PDU), and can form a fault domain as they depend on a single infrastructure resource.
- The idea of Availability Zones maps to the concept of hardware failure domains.
- Think of a situation when you lost network connectivity to a **ToR** switch or lost power to the rack of compute nodes due to the failure of a **PDU**.
- With Availability Zones configured, the end users can still continue to launch instances just by choosing a different Availability Zone.
- One important thing to keep in mind is that a compute node cannot be part of multiple Availability Zones.
- To configure an Availability Zone for a compute node, edit the `/etc/nova.conf` file on that node and update the `default_availability_zone` value.
- Once updated, the Nova compute service on the node should be restarted.

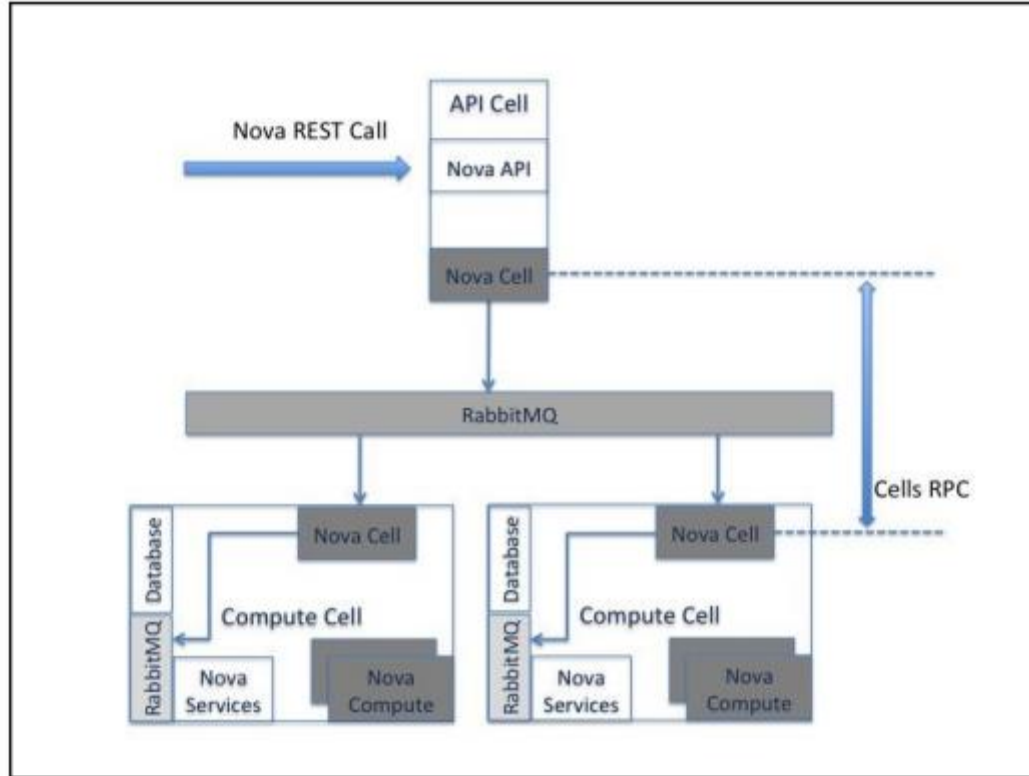
Availability zones



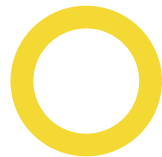
Segregating the compute cloud- Nova cells

- The nova-cells service handles communication between cells and selects cells for new instances. This service is required for every cell. Communication between cells is pluggable, and currently the only option is communication through RPC. Cells scheduling is separate from host scheduling. nova-cells first picks a cell.
- In a conventional OpenStack setup, all the compute nodes need to speak to the message queue and the database server (using nova-conductor).
- This approach creates a heavy load on the message queues and databases. As your cloud grows, a lot of compute servers try to connect to the same infrastructure resources, which can cause a bottleneck.
- This is where the concept of cells in Nova helps scale your compute resources.
- Nova cells are a way of scaling your compute workload by distributing the load on infrastructure resources, such as databases and message queues, to multiple instances.
- The Nova cell architecture creates groups of compute nodes that are arranged as trees, called cells. Each cell has its own database and message queue. The strategy is to constrain the database and message queue communication to be within the individual cells.
- So how does the cell architecture work? Let's look at the components involved in the cells' architecture and their interaction. As mentioned earlier, the cells are arranged as trees. The root of the tree is the API cell and it runs the Nova API service but not the Nova compute service, while the other nodes, called the compute cells, run all Nova services.

Segregating the compute cloud



Segregating the compute cloud



● Nova cells

- The cells' architecture works by decoupling the Nova API service that receives the user input from all other components of Nova compute.
- The interaction between the Nova API and other Nova components is replaced by message-queue-based RPC calls. Once the Nova API receives a call to start a new instance, it uses the cell RPC calls to schedule the instance on one of the available compute cells.
- The compute cells run their own database, message queue, and a complete set of Nova services except the Nova API. The compute cell then launches the instance by scheduling it on a compute node:
- Although cells have been implemented in Nova for quite some time, they have not seen widespread deployment and have been marked as experimental. As of today, the cells are an optional feature, but the Nova project is working on a newer implementation of cell architecture with the vision to make cells the default architecture to implement the compute cloud. In the current cell architecture, scheduling of an instance requires two levels of scheduling. The first level of scheduling is done to select the cell that should host the new virtual machine. Once a cell is selected, the second level of scheduling selects the compute node to host the virtual machine. Among other improvements, the new implementation (V2 API) will remove the need for two levels of scheduling.



Workload segregation



- Workload segregation is a usability feature in OpenStack cloud.
- It impacts how virtual machine instances are placed in the cloud environment.
- Handles the placement of instances relative to each other, ensuring specific requirements are met.
- Handling Multiple Instances:
 - Scenario 1: When you need two virtual machines to be placed on the same compute node.
 - Scenario 2: Ensuring high-availability by not placing instances of critical applications on the same compute node.
- Affinity Policies:
 - Workload segregation is achieved through affinity policies.
 - Example: Affinity policies ensure instances are placed together (affinity) or apart (anti-affinity) based on specific requirements.
- Significance
 - Precision: Allows precise control over the placement of virtual machines, ensuring optimal performance and availability.
 - Flexibility: Offers flexibility in handling complex application architectures within the cloud environment.



Workload segregation

Configure Nova Scheduler:

- Update Nova filter scheduler configuration to include Affinity filters.
- Modify **scheduler_default_filters** to add **ServerGroupAffinityFilter** and **ServerGroupAntiAffinityFilter**.

```
scheduler_default_filters = ServerGroupAffinityFilter, ServerGroupAntiAffinityFilter
```

Create Server Groups:

- Use Nova client to create server groups with affinity or anti-affinity policies.
- Affinity: # nova server-group create svr-grp1 affinity
- Anti-Affinity: # nova server-group create svr-grp2 anti-affinity

Apply Affinity Policies:

- Affinity Policy: Places VMs on the same compute node.
- Anti-Affinity Policy: Forces VMs onto different compute nodes.




Workload segregation



Launch Virtual Machines:

- Start VMs associated with a server group using Nova client commands.
- Include the `--hint group=svr-grp1-uuid` option to specify the server group.



```
# nova boot --image imageID --hint group=svr-grp1-uuid --flavor "Standard" vm1
# nova boot --image imageID --hint group=svr-grp1-uuid --flavor "Standard" vm2
```

Result:

- Virtual machines (vm1 and vm2) are ensured to be placed on the same compute node due to the affinity policy specified during server group creation.
- 





Changing the color of the hypervisor



- **Enable Multi-Hypervisor Support:**

- OpenStack supports multiple hypervisors, providing flexibility for users.

- Users might require different hypervisors for their applications due to platform compatibility and performance reasons.

- Let's take an example and see how such multi-hypervisor capability can be factored in the OpenStack environment.

- If you already have a VMware vSphere running in your infrastructure, this example will be suitable for you if you plan to integrate vSphere with OpenStack.

- **Integration with VMware vSphere:**

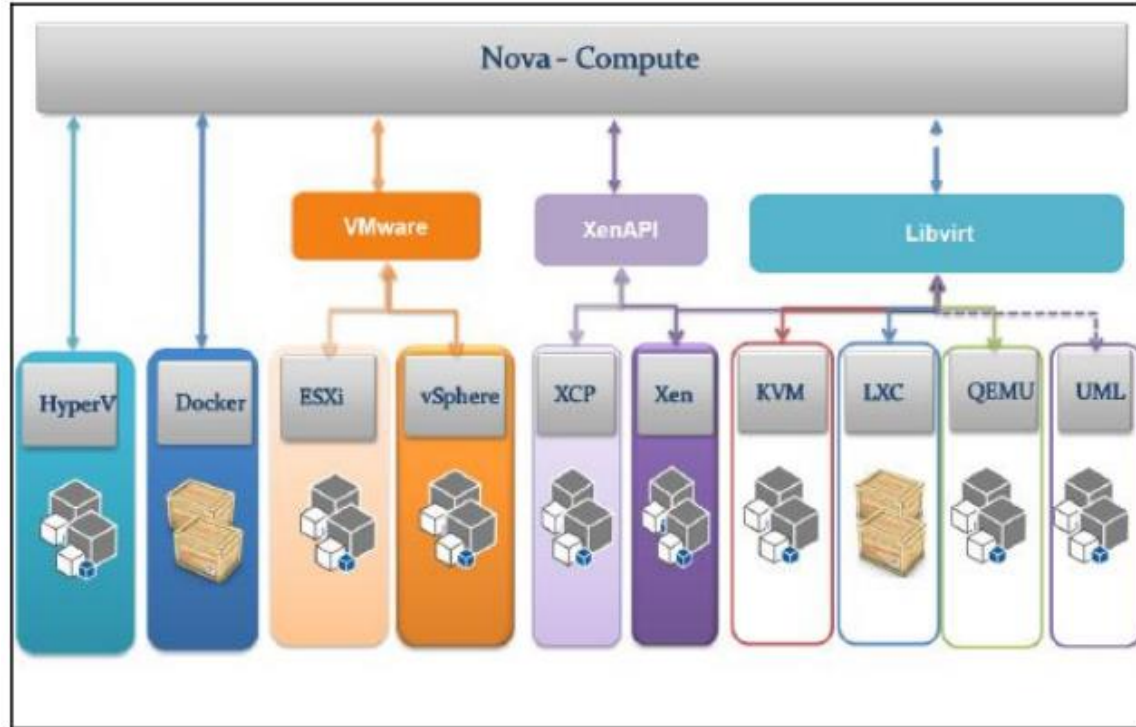
- Integration involves using specific OpenStack drivers for vSphere: **vmwareapi.VMwareESXDriver** and **vmwareapi.VMwareVCDriver**.

- These drivers enable OpenStack to manage **ESXi** hosts and **vCenter** servers in vSphere environments.

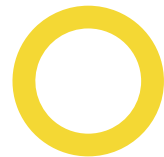


Changing the color of the hypervisor

The following figure depicts the integration between nova-compute and KVM, QEMU, and LXC by means of libvirt tools and XCP through APIs, while vSphere, Xen, or Hyper-V can be managed directly via nova-compute:



Changing the color of the hypervisor



- `vmwareapi.VMwareESXDriver`: This allows nova-compute to reach the ESXi host by means of the vSphere SDK
- `vmwareapi.VMwareVCDriver`: This allows nova-compute to manage multiple clusters by means of a single VMware vCenter server

Benefits of Integration:

- Advanced capabilities like vMotion, high availability, and Dynamic Resource Scheduler (DRS) can be harnessed.
- OpenStack treats each vSphere cluster as a compute node, utilizing aggregate resources of all managed ESXi hosts.

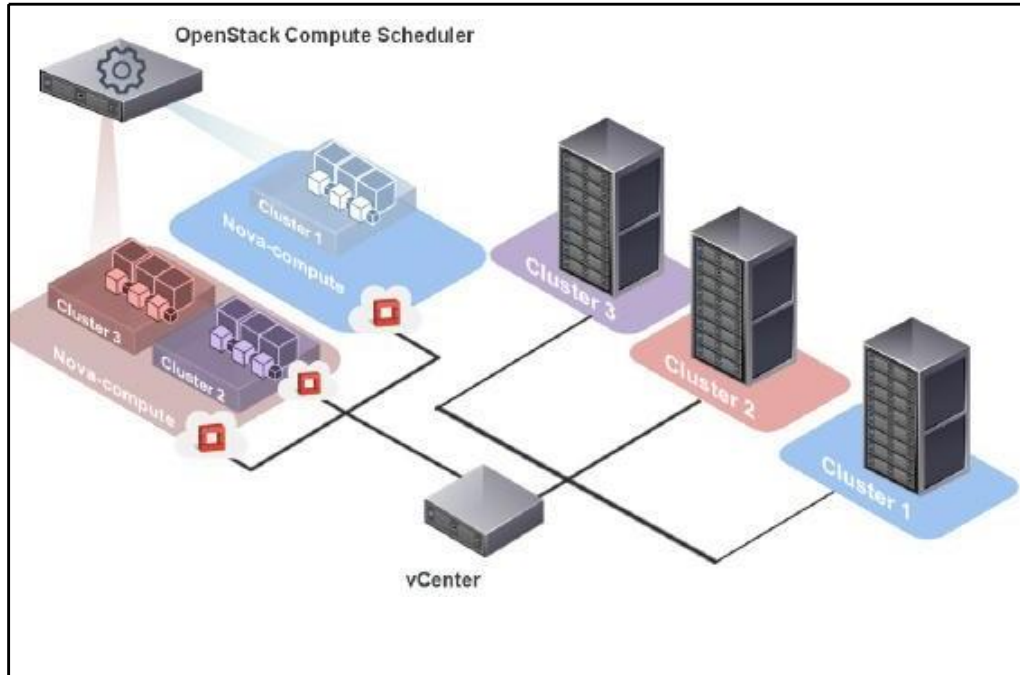
Management Best Practices:

- ● Compute nodes should be placed in a separate management vSphere cluster for HA and DRS benefits.
- vCenter management is restricted to OpenStack compute nodes within this management cluster.



Changing the color of the hypervisor

- Imagine the several functions we will gain from such an integration using the OpenStack driver with which we attempt to harness advanced capabilities, such as vMotion, high availability, and Dynamic Resource Scheduler (DRS).
- It is important to understand how such integration can offers more flexibility:






Changing the color of the hypervisor



● A brief example can be conducted with the following steps:

- Create a new host aggregate; this can be done through Horizon.
- Select Admin project. Point to the Admin tab and open System Panel. Click on the Host Aggregates category and create new host named vSphere- Cluster_01.
- Assign the compute nodes managing the vSphere clusters within the newly created host aggregate.
- Create a new instance flavor and name it vSphere.extra, with particular VM resource specifications.
- Map the new flavor to the vSphere host aggregate.


● This is amazing because any user requesting an instance with the vSphere.extra flavor will be forwarded only to the compute nodes in the vSphere-Cluster_01 host aggregate.

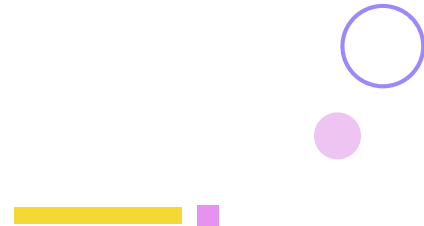
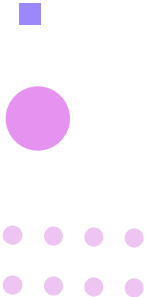




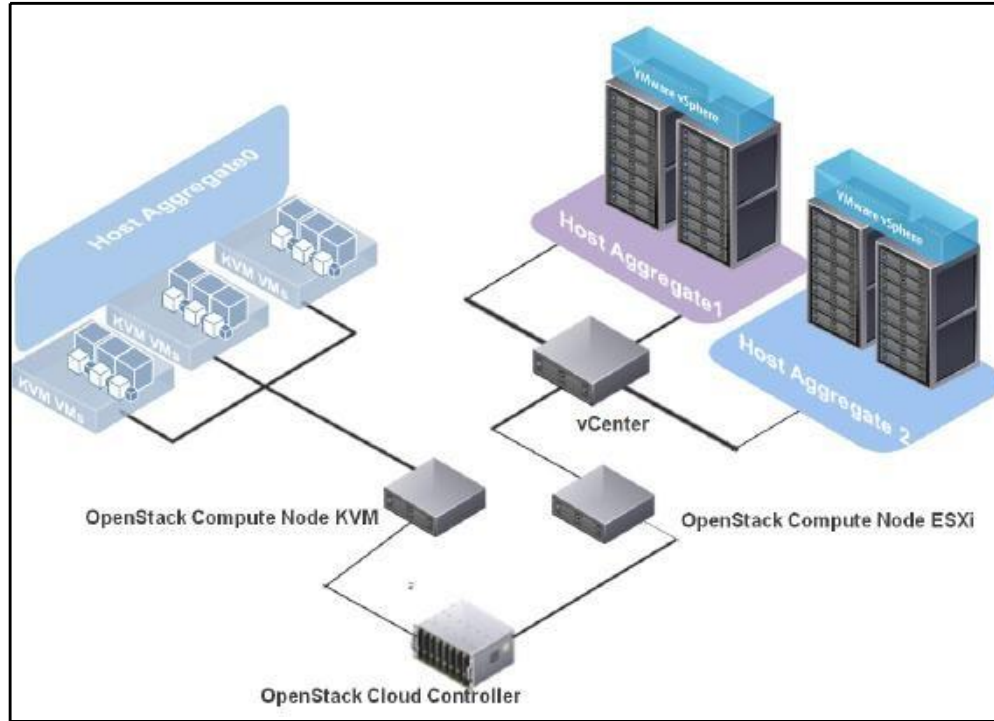
Changing the color of the hypervisor



- Result:
 - Instances requested with the **vSphere.extra** flavor are placed exclusively on compute nodes within the vSphere-Cluster_01 host aggregate.
 - Provides users with the choice to deploy VMs on specific vSphere clusters, ensuring tailored hardware requirements are met.
- 



Changing the color of the hypervisor



05

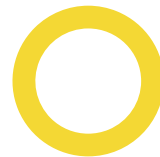


Overcommitment considerations

Overcommitment considerations

- In OpenStack, overcommitment refers to the practice of allocating more virtualized computing resources (such as CPU and RAM) to virtual machines (VMs) than the physical host actually has.
- This is possible due to the dynamic and variable nature of resource usage in most cloud environments.
- Overcommitment is a way to make more efficient use of the available resources and improve the overall utilization of the cloud infrastructure.
- Hyper-threading, doubling existing cores, is recommended for CPU per compute node.
- Many purchased physical nodes might be more powerful than necessary, leading to resource wastage.
- ● Sizing compute nodes is crucial to avoid wastage.

Overcommitment considerations

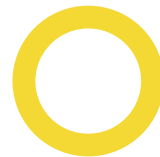


Steps for Overcommitment:

- Estimate CPU and RAM size using sample calculations.
- Utilize OpenStack resources' overcommitment effectively.
- Gather resources' usage statistics periodically.
- CPU Overcommitment: OpenStack allows for overcommitting CPU resources by running more virtual CPU cores (vCPUs) on a hypervisor host than the physical CPU cores available. This is based on the understanding that not all VMs use 100% of their allocated CPU all the time.
- RAM Overcommitment: RAM overcommitment involves allocating more virtual memory (RAM) to VMs than the physical host has. Similar to CPU overcommitment, this assumes that VMs often don't use all their allocated RAM simultaneously.



Overcommitment considerations



Memory and CPU Overcommitment:

- Overcommitment is a hypervisor feature, allowing VMs to use more resources than the host has.
- For instance, a server with 4 GB RAM can run eight VMs, each allocated 1 GB memory space.
- Dynamic relocation of unused resources is essential, but it must be used judiciously to avoid server crashes.

Overcommitment Ratios in OpenStack:

- OpenStack allows overcommitment of CPU and RAM resources by changing default limits in local configurations.
- Compute nodes use ratios to determine VMs per hardware thread/core and memory association. Defaults are 16:1 for CPU and 1.5:1 for RAM.
- Before setting ratios, it's recommended to collect measurement results for specific hardware vendors from sources like spec.org.

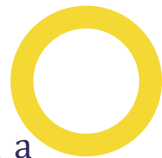
Cautionary Note:

- Overcommitment is a useful feature but must be used carefully to prevent resource exhaustion and server crashes.

Overcommitment considerations

- **Default CPU Allocation Ratio:** The default ratio in OpenStack is 16:1, meaning 16 virtual CPU cores for every physical CPU core.
- **Example:** If a physical node has 24 cores, there are $24 * 16 = 384$ available virtual cores. Allocating 4 virtual cores per instance allows for 96 instances on each node.
- Overcommitting CPU is suitable for non-CPU-intensive workloads. If workloads are very CPU-heavy, it's advisable to limit the overcommitment ratio.
- **Misuse Warning:** Changing the ratio to 1:1 eliminates CPU overcommitment, limiting vCPUs to the physical CPU cores. However, you can still run more VMs than the number of physical CPU cores on the compute node.

CPU allocation ratio



- **Calculation Formula:** The formula to determine how many virtual instances can run on a compute node is:

Number of Virtual Instances



*= (CPU overcommitment ratio * Number of physical cores) / Number of virtual cores per instance*

- CPU Overcommitment Ratio: This is the ratio indicating how many virtual CPU cores can be allocated per physical CPU core. For example, if the ratio is 16:1, it means 16 virtual CPU cores can be allocated for every 1 physical CPU core.
- Number of Physical Cores: This refers to the total number of physical CPU cores available on the compute node.
- Number of Virtual Cores per Instance: This represents how many virtual CPU cores are allocated to each virtual machine instance.



RAM allocation ratio: 1.5:1




- The default 1.5:1 memory allocation ratio means instances can be allocated to compute nodes if total instance memory usage is less than 1.5 times the physical memory available.
 - For example, a node with 96 GB RAM can run instances totaling 144 GB (36 virtual machines with 4 GB RAM each).
 - *Flavors in OpenStack: Flavors are hardware templates defining RAM, disk space, and CPU cores per CPU for virtual machines.*
 - Changing Default Settings: Use `cpu_allocation_ratio` and `ram_allocation_ratio` directives in `/etc/nova/nova.conf` to modify default allocation settings.
- 
- 





RAM allocation ratio: 1.5:1

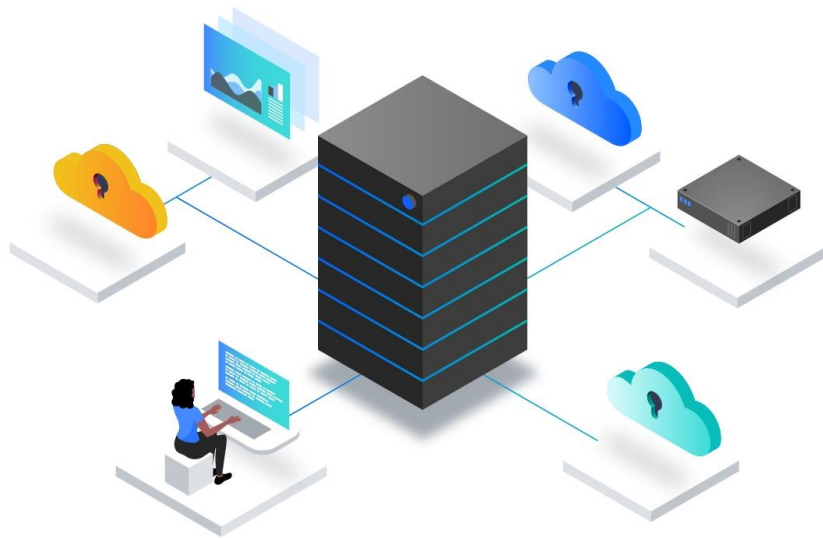


- Overcommitment is used judiciously based on active monitoring of hardware usage. Peak times and varying resource needs of users' virtual instances must be considered.
 - Balancing resource assignments during peak times is crucial. Understanding system virtualization and gathering information helps in dynamically optimizing resource handling.
 - Choosing the right hypervisor is essential for efficiently managing varied resource requirements in a dynamic environment.
- 



06

Storing instances' alternatives



Storing instances' alternatives

- When considering storage for instances, it's important to think about disk space capacity, in addition to CPU and RAM.
- Basically, there are many approaches to doing this, but it might expose other trade-offs: **capacity and performance.**

External shared file storage

- The disks of running instances are hosted externally and do not reside in compute nodes. This will have many advantages, such as the following:
 - Ease of instance recovery in the case of compute - node failure Shared external storage for other installation purposes
- On the other hand, it might present a few drawbacks, such as the following:
 - Heavy I/O disk usage affecting the neighboring VM Performance degradation due to network latency

Storing instances' alternatives




- **Internal non-shared file storage**
- In this case, compute nodes can satisfy each instance with enough disk space. This has two main advantages:
 - Unlike the first approach, heavy I/O won't affect other instances running in different compute nodes
 - Performance increase due to direct access to the disk I/O
- However, some further disadvantages can be seen, such as the following:
 - Inability to scale when additional storage is needed
 - Difficulties in migrating instances from one compute node to another
 - Failure of compute nodes automatically leading to instance loss

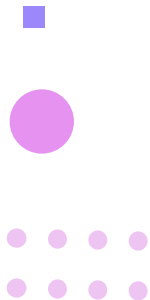




Storing instances' alternatives

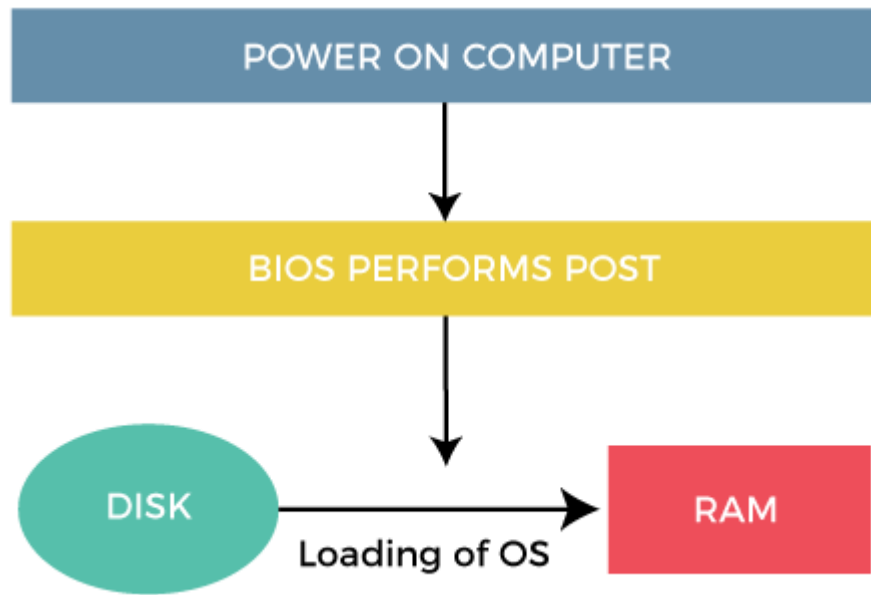


- In all cases, we might have more concerns for reliability and scalability.
 - Thus, adopting the external shared file storage would be more convenient for our OpenStack deployment.
 - Although there are some caveats to the external instances' disk storage that must be considered, performance can be improved by reducing network latency.
- 



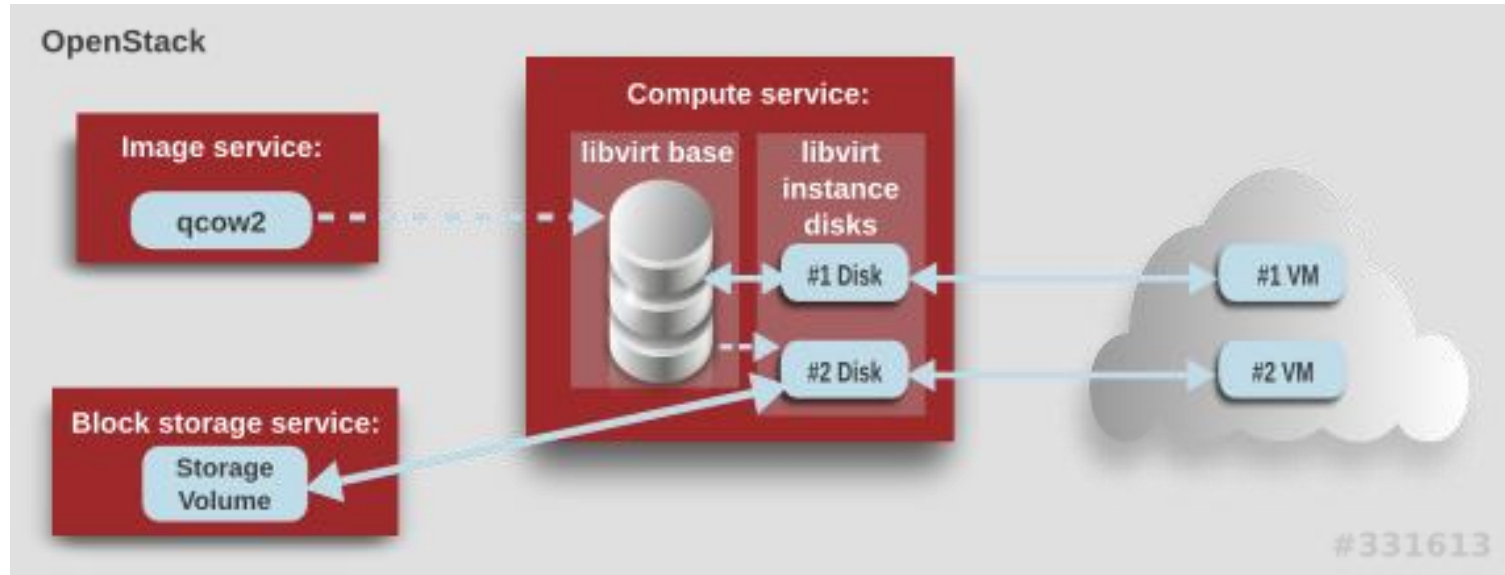
07

Understanding instance booting

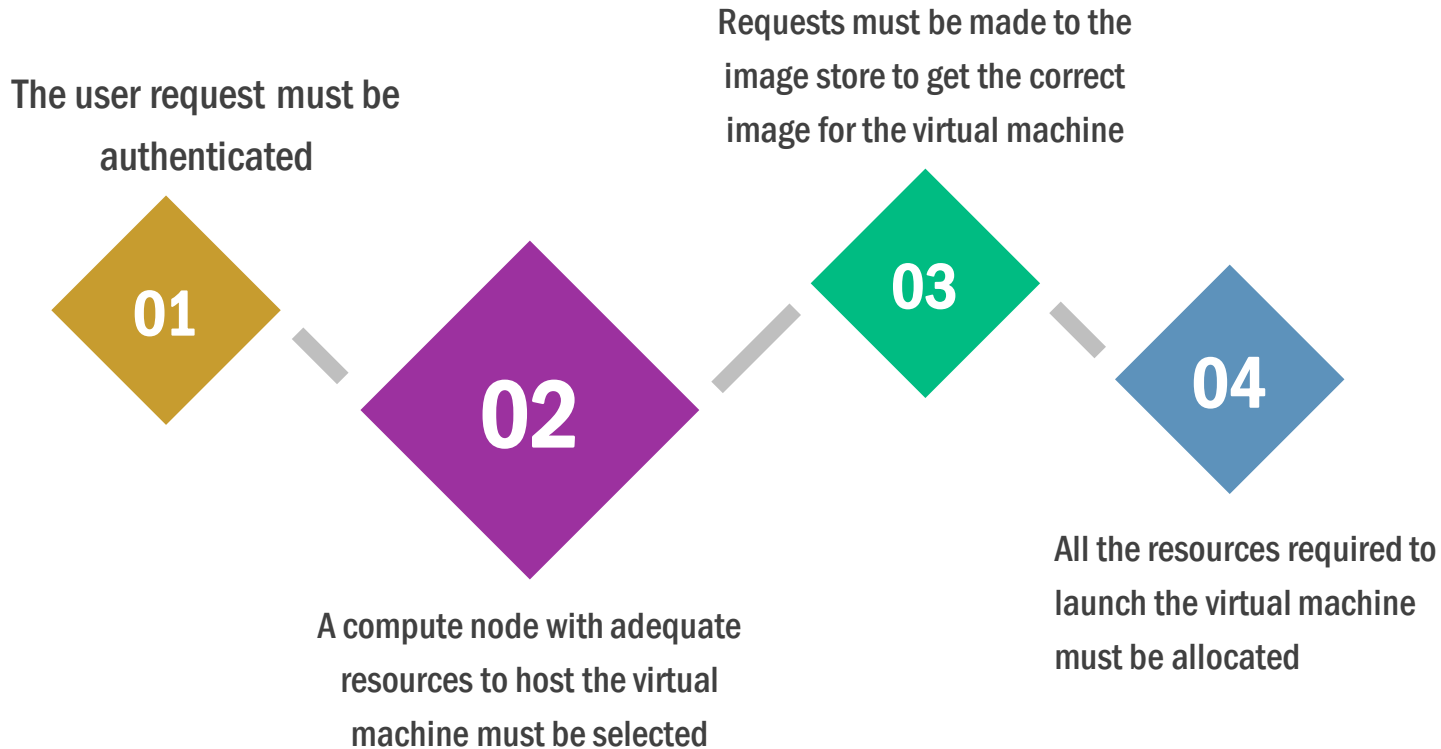


Understanding instance booting

Launching an instance on your [OpenStack](#) cloud requires interaction with multiple services.



When a user requests a new virtual machine, behind the scenes



When a user requests a new virtual machine, behind the scenes

1. Authentication:

When a user requests a new virtual machine, their request is authenticated to ensure authorized access to OpenStack services.

2. Nova Scheduling Process:

Nova, OpenStack's compute service, performs the scheduling process.

It selects an appropriate compute node with available resources (CPU, RAM, disk) to host the virtual machine.

3. Image Selection:

Requests are made to the image store to retrieve the correct image for the virtual machine.

The image contains the operating system and other necessary software configurations.

4. Resource Allocation:

All required resources for the virtual machine are allocated, including network connectivity and storage volume

When a user requests a new virtual machine, behind the scenes

5. Booting from Image:

The virtual machine boots from the selected image, initializing the operating system and applications.

6. Getting Instance Metadata:

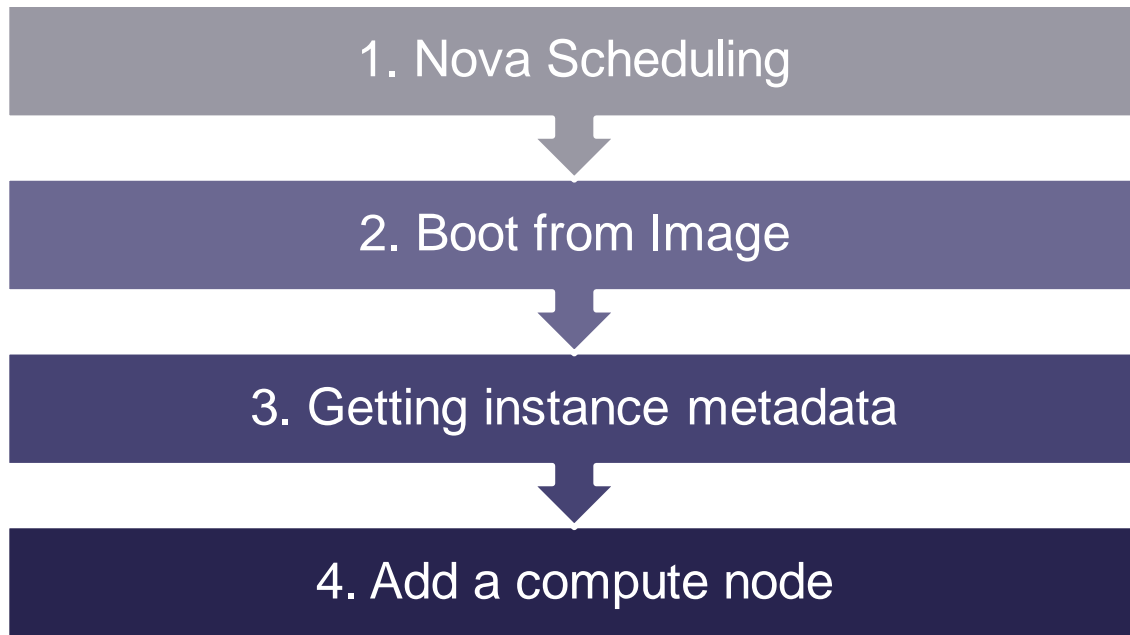
Metadata associated with the instance, such as IP addresses and configuration details, is retrieved.

This metadata is essential for networking and application setup within the virtual machine.

7. Adding a Compute Node (Optional):

If necessary, additional compute nodes can be added to the OpenStack environment to scale the capacity for hosting more virtual machines.

Steps of instance booting



■ Step 1: Understanding the Nova scheduling process

■ Selecting the Right Compute Node:

- This is a critical step in launching a virtual machine, as it determines where the VM will run.
- OpenStack uses a default scheduler called the "**filter scheduler**" for this purpose.
- The **VM flavor** defines the necessary resources that the hosting compute node must provide.

■ Filtering Candidates:

- All potential compute nodes undergo a filtering process to ensure they meet the VM's resource needs.
- Nodes that don't meet these requirements are filtered out of consideration.

■ Step 1: Understanding the Nova scheduling process

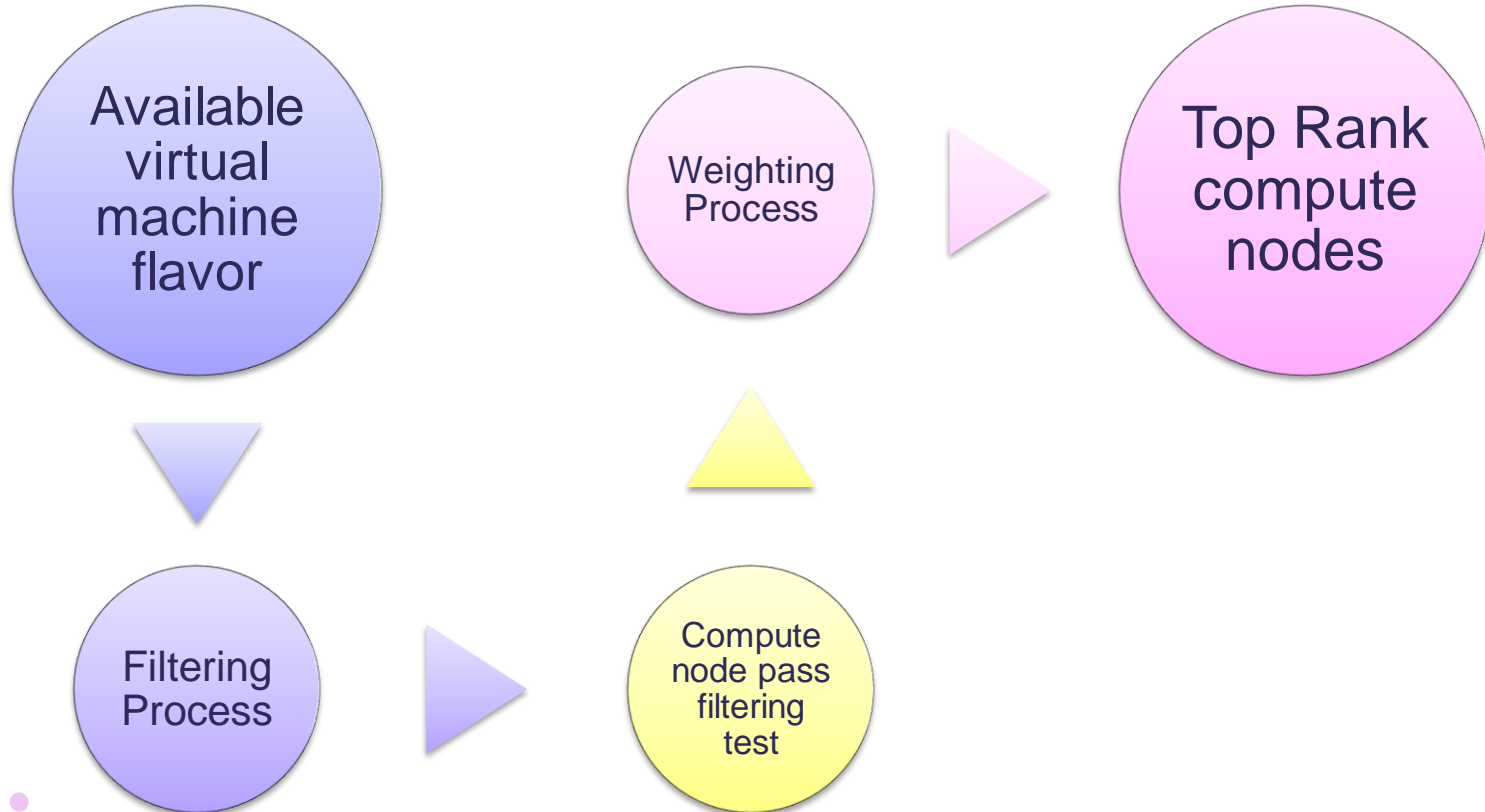
Weighting Compute Nodes:

- After filtering, the remaining compute nodes are ranked based on their resource availability.
- This ranking is done using a combination of **pluggable filters and weights**.

Customizing Scheduler Behavior:

- The scheduler's behavior can be customized by modifying the list of filters or weights used in the ranking process.
 - You can set the value of **scheduler_default_filters** to make these changes.
 - Understanding this process is essential for efficiently assigning virtual machines to compute nodes in your OpenStack environment. The scheduler
- ensures that VMs are placed on suitable nodes with adequate resources, enhancing performance and resource utilization.

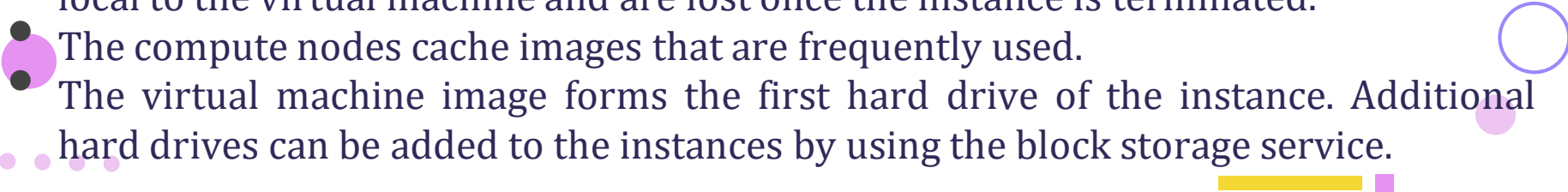
Step 1: Understanding the Nova scheduling process





Step 2: Booting from image



- To launch a virtual machine, the user must **select the image that will be loaded on the virtual machine** and the **hardware characteristics of the instance**, such as the *memory, processor, and disk space*.
 - The **hardware requirements** can be selected by choosing the **correct machine flavor**.
 - Flavors provide the hardware definition of a virtual machine.
 - New flavors can be added to provide custom hardware definitions.
 - To boot the virtual machine, the compute node must download the image that needs to be loaded on the instance.
 - It should be noted that the same image could be used to launch multiple virtual machines.
 - The image is always copied to the hypervisor. Any changes made to the image are local to the virtual machine and are lost once the instance is terminated.
 - The compute nodes cache images that are frequently used.
 - The virtual machine image forms the first hard drive of the instance. Additional hard drives can be added to the instances by using the block storage service.
- 

Step 3: Getting the instance metadata

- As virtual machines are launched on the OpenStack environment, it must be provided with **initialization data that will be used to configure the instance**.
- This early initialization data configures the instance with information such as *hostname, local language, user SSH keys, and so on.*
- It can be used to write out files such as **repository configuration** or **set up automation tools** such as Puppet, Chef, or keys for Ansible-based deployment.
- This initialization data can be metadata associated with the instance or user-provided configuration options.
- **The cloud images are packaged with an instance initialization daemon called cloud-init.**
- The **cloud-init** daemon looks at various data sources to get configuration data associated with a virtual machine.
- *The most commonly used data sources are the EC2 and Config Drive.*

Step 3: Getting the instance metadata

- Example Request for SSH Public Key:
 - Use curl command to download **SSH public key**:
 - `# curl http://169.254.169.254/latest/meta-data/public-keys/0/openssh-key -O`
- User-Data Injection:
 - User-data provides customized information (shell scripts, environment variables) during instance boot.
 - Inject user-data during instance creation using Nova command line.
- Retrieving User-Data:
 - Query metadata service endpoint through OpenStack metadata API to retrieve user data.
 - Example curl request: `# curl http://169.254.169.254/openstack/latest/user-data`
- Config Drive Implementation:
 - Original Config Drive used for network configuration.
 - Later cloud-init versions allow Config Drives as a complete data source.

Step 3: Getting the instance metadata


Here are the possible instance information hierarchically organized, accessible by sending a GET request to the metadata endpoint.

reservation-id	public-keys/	security-groups	public-ipv4	ami-manifest-path
instance-type	instance-id	local-ipv4	local-hostname	placement/
ami-launch-index	public-hostname	hostname	ami-id	instance-action



Step 4: Add a compute node



- Using OpenStack Ansible (OSA), adding a compute node is much simpler than understanding the resource requirements needed for a node. Basically, the compute node will run nova-compute together with the networking plugin agent.
 - What you should understand at this stage of automated deployment is how to make the new compute node communicate with the controller and network nodes:
- 

Adjust Configuration:

- Add new compute node details to openstack_user_config.yml file.
- Specify IP address for the new node (e.g., cn-01: ip: 172.47.0.20).



Define Node Settings:

- Set hypervisor type, CPU allocation ratio, RAM allocation ratio, and max instances in user_variables.yml file.
- 
- 
- 





Step 4: Add a compute node



Install Containers:

- Run setup-hosts.yml Playbook to install containers on the new node.
- Use --limit option to deploy only on the new host (e.g., --limit cn-01).




Optional Monitoring:

- Optionally, configure telemetry service for monitoring the new node in ceilometer.yml file.
- 

Refined Update (Optional):

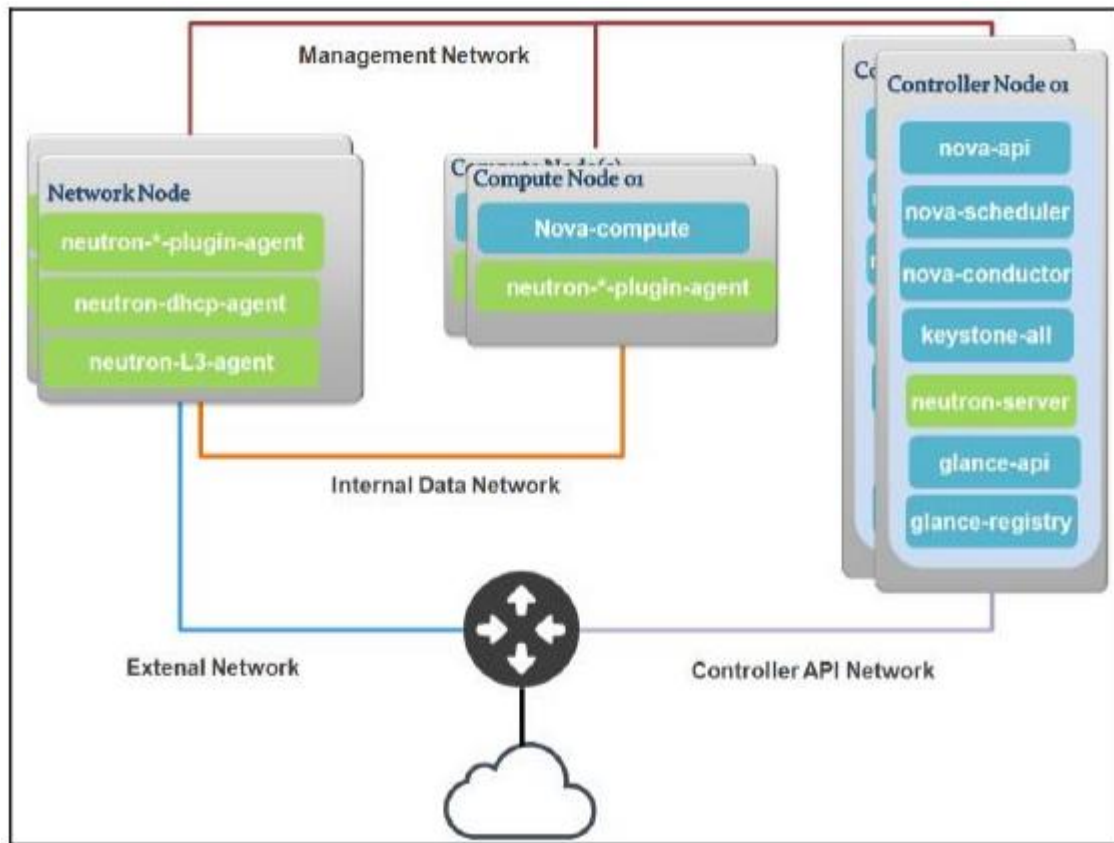
- Deploy the new service only on compute nodes group (compute_hosts) specified in openstack_user_config.yml.
- Use Ansible commands with specific tags to update the compute nodes.

Verification:

- Verify the new compute node's status and connectivity.
 - Access the node using SSH command line and check /etc/hosts for the new host entry.
- 
- 
- 



Step 4: Add a compute node



08





Planning for service recovery





Planning for service recovery







- Planning a backup is vital for system administrators and cloud operators.
 - Starting production without disaster recovery plans is highly risky.
 - OpenStack lacks built-in backup tools, unlike some proprietary solutions.
 - OpenStack is a collection of components; backup strategy needs to cover each component.
 - Develop backup strategies for each component. Strategies should be easy, efficient, and support auto-recovery.
 - Determine what needs to be backed up. Consider configuration files, databases, and other critical data.
 - Initial focus often centers on the cloud controller. Backup should cover configuration files and databases related to the cloud controller.
 - Need for Detailed Strategy:
 - Develop a detailed backup strategy for the cloud controller and other components.
 - Ensure the strategy accounts for possible failure scenarios and quick recovery methods.
- 
- 
- 
- 





Planning for service recovery









1. Exploration of Backup Solutions:
 - Explore available open-source backup solutions compatible with OpenStack.
 - Consider tools that align with the cloud environment's specific needs.
 2. Regular Testing and Review:
 - Regularly test backup and recovery procedures to ensure effectiveness.
 - Review and update backup strategies based on changes in the cloud infrastructure.
 3. Collaboration and Documentation:
 - Collaborate with other team members to develop comprehensive backup plans.
 - Document backup procedures and recovery steps for reference and training.
 4. Continuous Monitoring:
 - Implement continuous monitoring to detect backup failures promptly.
 - Monitor backup performance and adjust strategies based on monitoring data.
- 
- 
- 
- 





Backup with backup-manager










- Choosing the Right Backup Tool:
 - Various backup methods are available, making it essential to select the appropriate tool for your system's needs.
 - Using Backup-Manager:
 - Backup-Manager is a straightforward command-line backup tool widely compatible with Linux distributions.
 - Installation on nodes is easy, with configuration centralized in the `/etc/backup-manager.conf` file.
 - Configuring Backup Parameters:
 - Specify directories and files for backup, defining them in the `BM_TARBALL_DIRECTORIES` section.
 - Choose backup methods like `tarball` and `mysql` and set the corresponding directories and databases to back up.
 - Define the backup repository's location, typically `/var/backups/`.
 - Optionally, compress files using `gzip` and configure SSH account details for remote uploads.
- 
- 
- 
- 
- 
- 



Backup with backup-manager



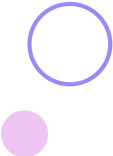



- Securing MySQL Database Backups:
 - For MySQL databases, traditional mysqldump method is used.
 - Configure databases to back up and provide the necessary root password securely in /root/.my.cnf.
 - Considerations for Compute Nodes:
 - Compute nodes utilize the /var/lib/nova/ folder for backups, excluding the live KVM instances.
 - Backup of instances is feasible through Horizon snapshots or by installing a backup tool within the instances themselves.
 - Note on Security:
 - Ensure sensitive data such as database passwords is protected; restrict permissions for /etc/backup-manager.conf and root user access.
 - Backup-Manager simplifies the backup process, allowing configuration flexibility and secure management of crucial data within the OpenStack environment.
- 
- 
- 
- 
- 
- 
- 



Simple recovery steps




- **Stop Services:**
 - Halt all intended services for recovery (e.g., Glance services on the cloud controller).
 - Commands: **\$ stop glance-api and \$ stop glance-registry.**
 - **Import Database:**
 - Restore the backed-up Glance database using MySQL import command:
 - Command: **\$ mysql glance < glance.sql.**
 - **Restore Directories:**
 - Copy backed-up Glance directories to the appropriate location:
 - Command: **\$ cp -a /var/backups/glance /glance/.**
 - **Restart Services:**
 - Restart Glance services and MySQL:
 - Commands: **\$ service start mysql, \$ glance-api start, and \$ glance-registry start.**
- 
- 



Data protection as a service



- Data Protection as a Service:
 - Some third-party solutions enable tenants to back up entire OpenStack clusters using external storage like NetApp SolidFire.
 - Implementations involve using object storage for file and archive backups, requiring regular snapshots of instances uploaded to the image store.
 - Block storage volumes can be snapshotted and uploaded to object storage.
 - Challenges include the absence of incremental backups and managing diverse storage backup workloads efficiently.
- 





Previous Year University Question Paper 2021

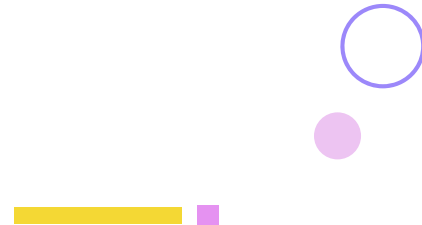


PART A

- 
1. Write a short note on Cinder block storage service and its components.
 2. What are the approaches available for segregating cloud services?

PART B

1. Write a comparison about Nova Docker driver and OpenStack Magnum project for hosting an application.
2. Describe Swift architecture.









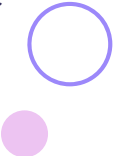
Previous Year University Question Paper 2022



PART A

- 
1. Summarize on docker containers
 2. Describe on the nova- conductor service

PART B

- 
- 
1. Summarize on the following tools available in segregating the compute cloud
 - a. Availability zones
 - b. Host Aggregates
 - c. Nova cells
 2. Illustrate the Swift architecture with its physical design considerations using suitable diagram
- 
- 
- 

Thanks!

Do you have any questions?

navyamolkt@amaljyothi.ac.in
me@AJCE

