OpenStack Compute & Storage

Module III

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

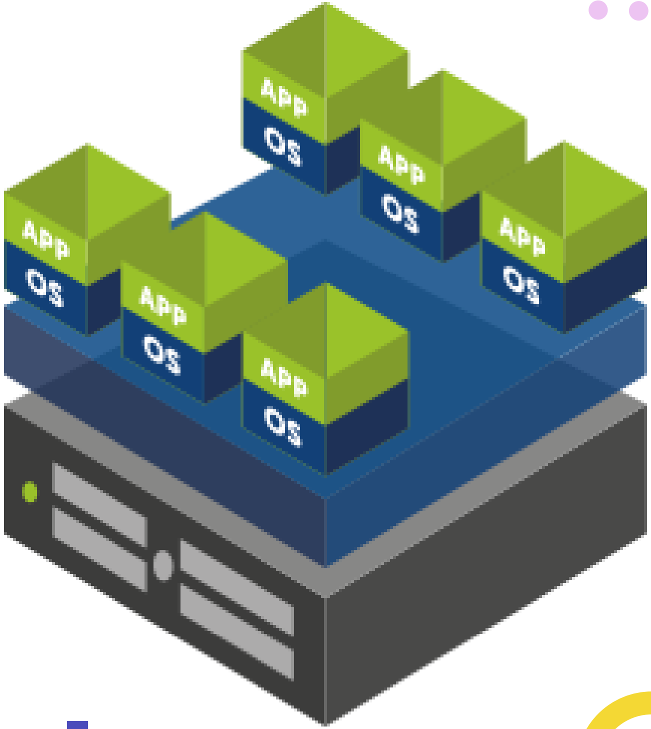
- **nova- conductor service**
- The compute nodes are not provided direct access to the database. This design limits the risk of a compromised compute node providing the attacker complete access to the database. The database access calls from the compute nodes are handled by the nova- conductor service.
- 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

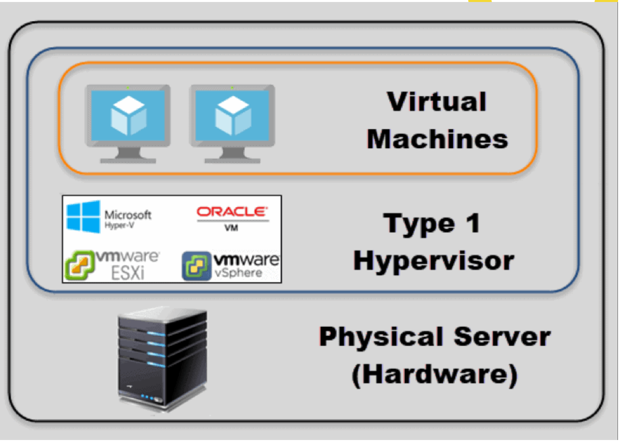
02

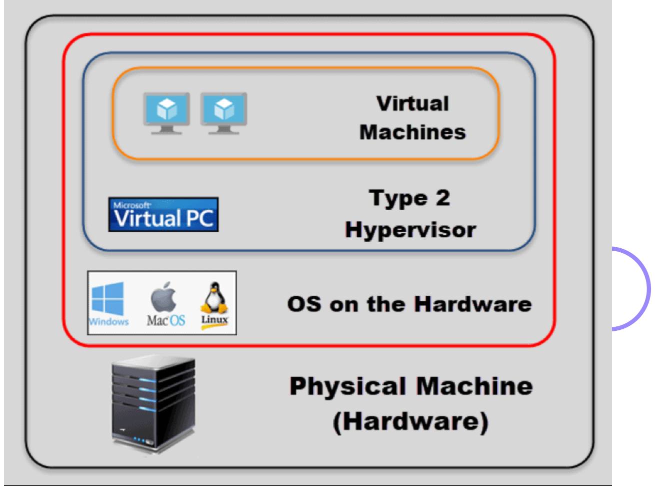
Deciding on the Hypervisor



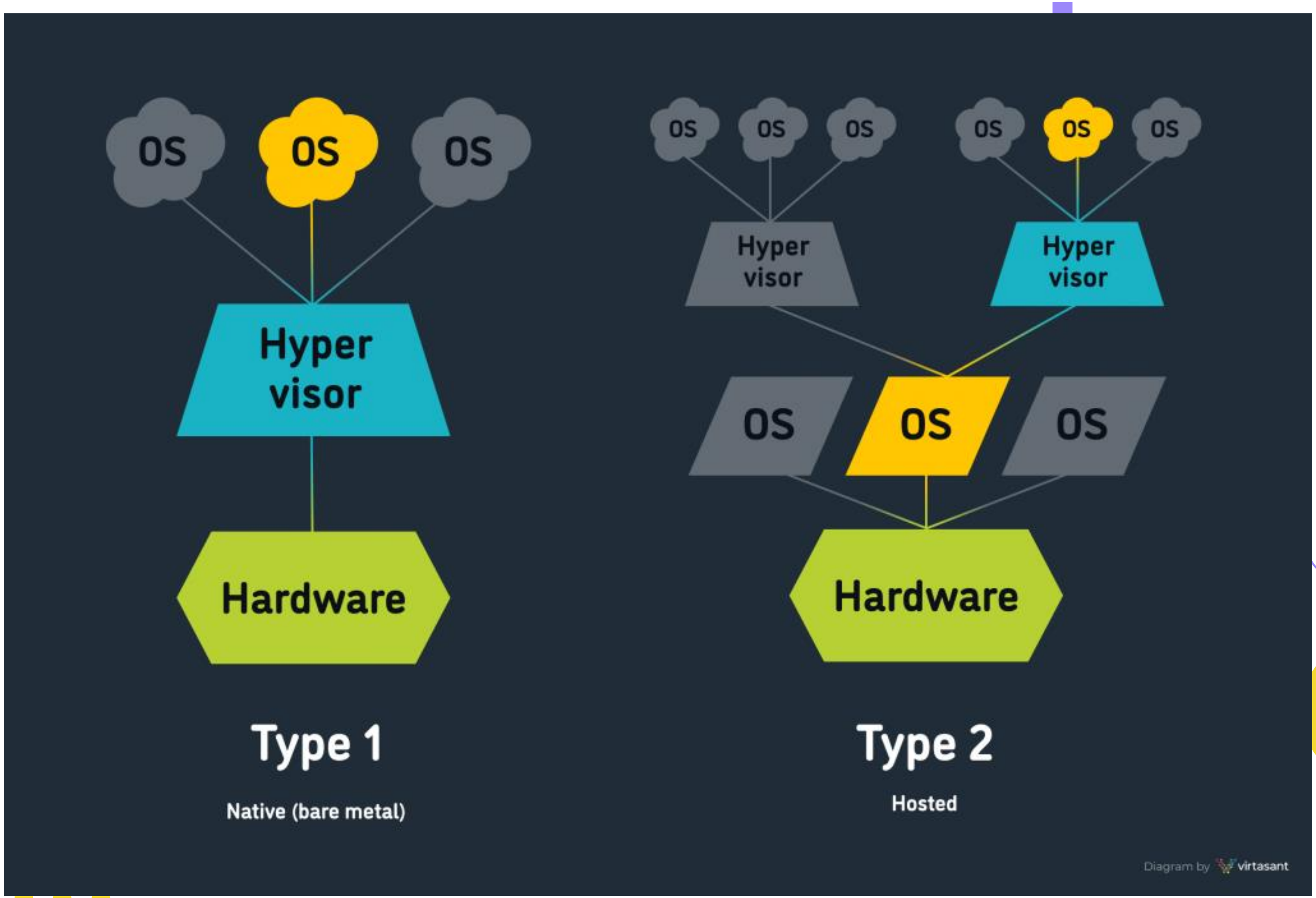
Deciding on the Hypervisor

- 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.

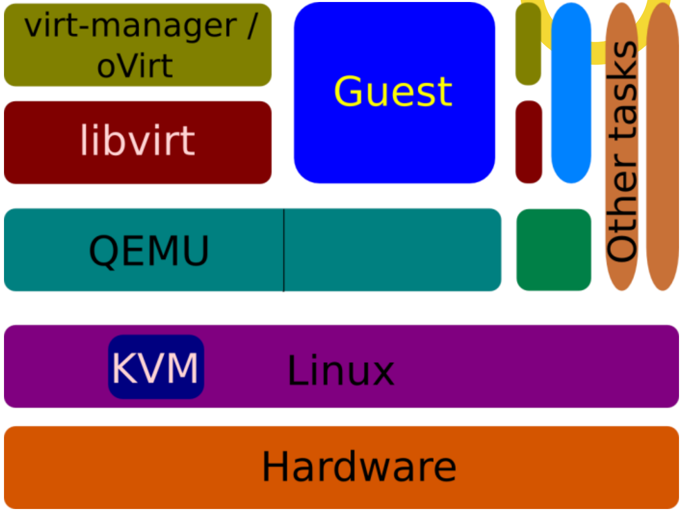
- Citrix Hypervisor
  - Formerly known as XenServer, Citrix Hypervisor is a commercial type 1 hypervisor.

- 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 | greq kvm
kvm_intel or kvm_amd
```

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

```
# modqrobe –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

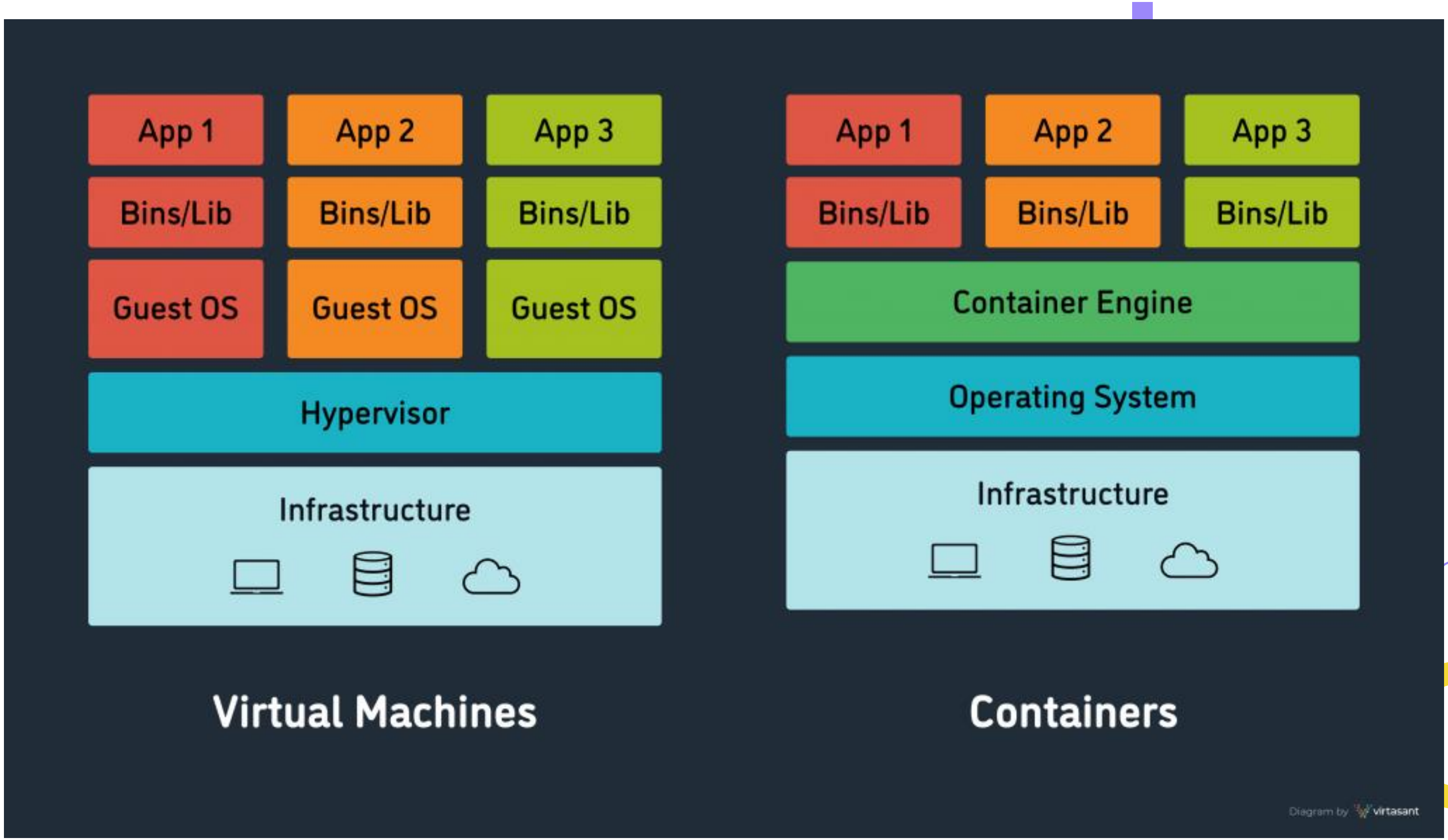
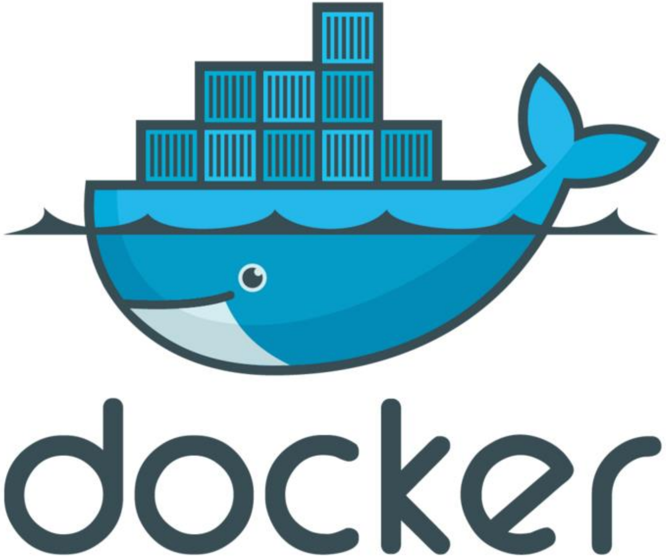


Diagram by virtasant

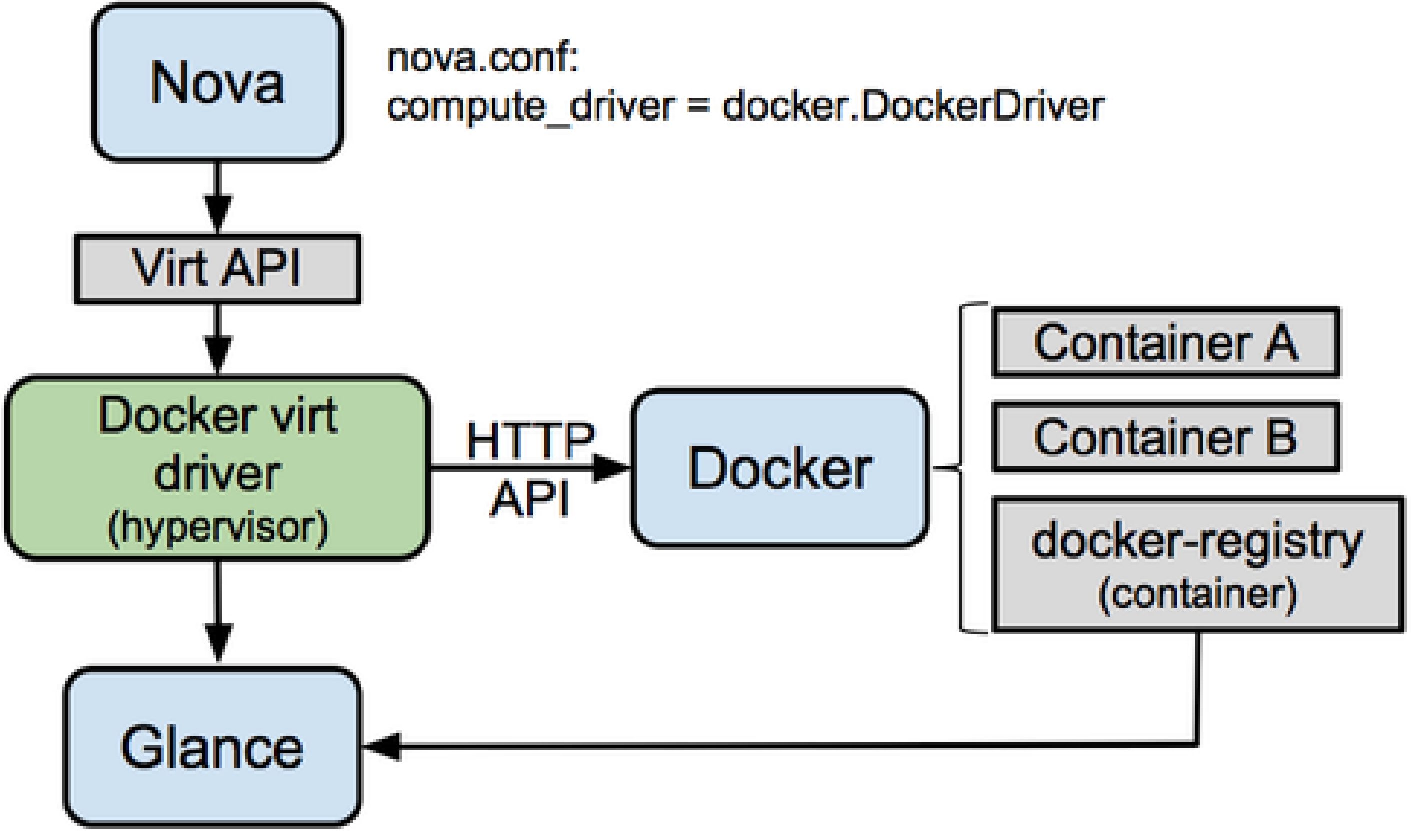
Containers vs Hypervisors

Infrastructure orchestration tool



Application provisioning tool

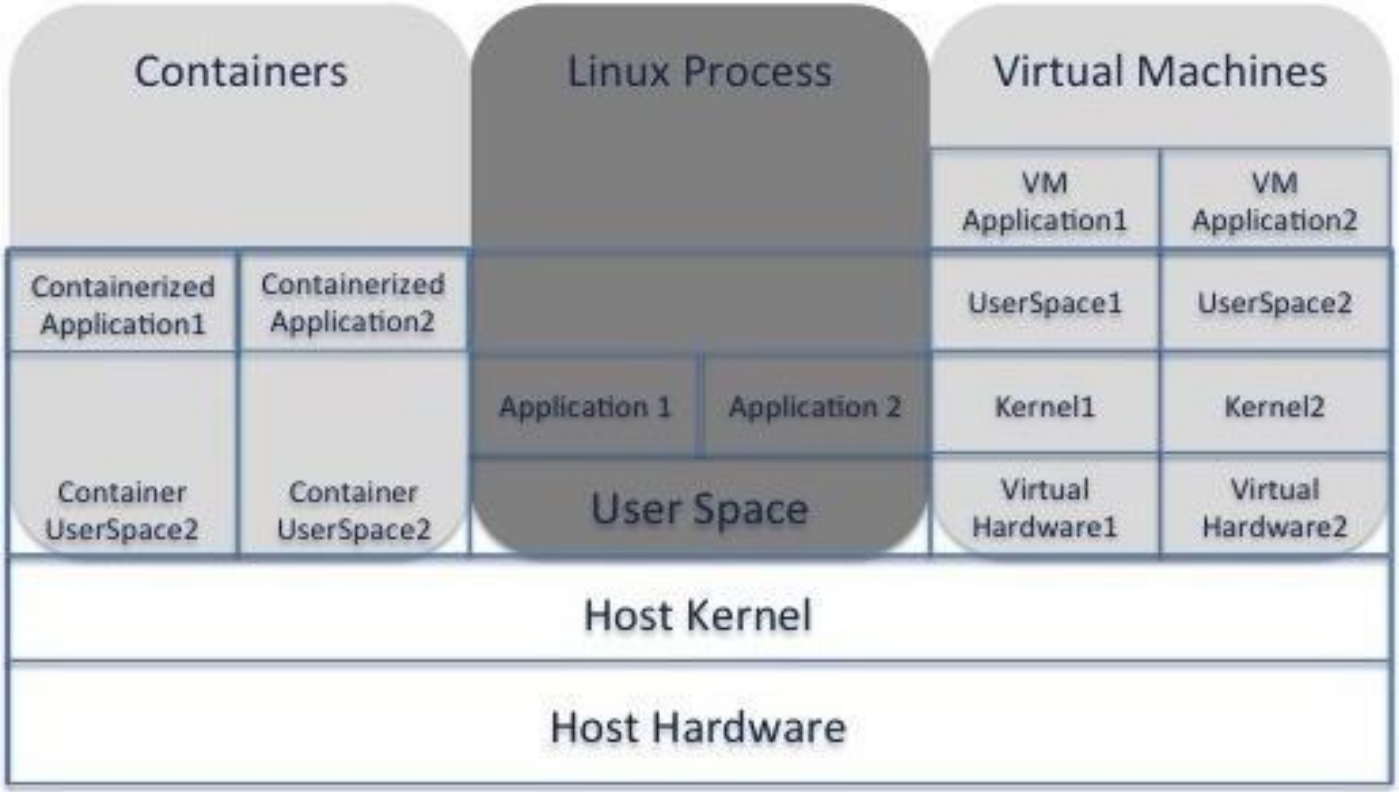
The Docker containers


Nova
nova.conf:
compute_driver = docker.DockerDriver
Virt API
Docker virt driver (hypervisor)
HTTP API
Docker
Container A
Container B
docker-registry (container)
Glance

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



Containers

Linux Process

Virtual Machines

Containerized Application1

Containerized Application2

Container UserSpace2

Container UserSpace2

Application 1

Application 2

User Space

VM Application1

VM Application2

UserSpace1

UserSpace2

Kernel1

Kernel2

Virtual Hardware1

Virtual Hardware2

Host Kernel

Host Hardware

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

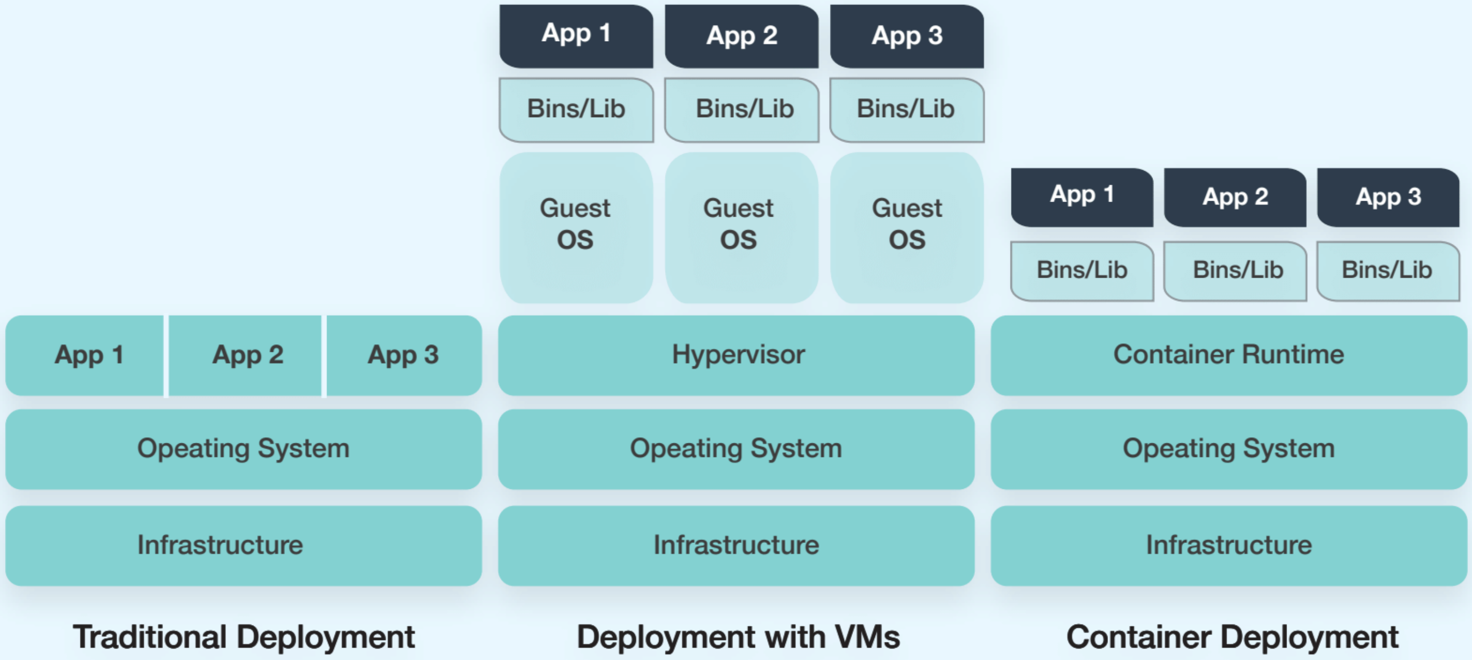
appfleet

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

KubeAcademy
from VMware



App 1 | App 2 | App 3

Opeating System

Infrastructure

Traditional Deployment

App 1 | App 2 | App 3

Bins/Lib | Bins/Lib | Bins/Lib

Guest OS | Guest OS | Guest OS

Hypervisor

Opeating System

Infrastructure

Deployment with VMs

App 1 | App 2 | App 3

Bins/Lib | Bins/Lib | Bins/Lib

Container Runtime

Opeating System

Infrastructure

Container Deployment

SIMFORM

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.

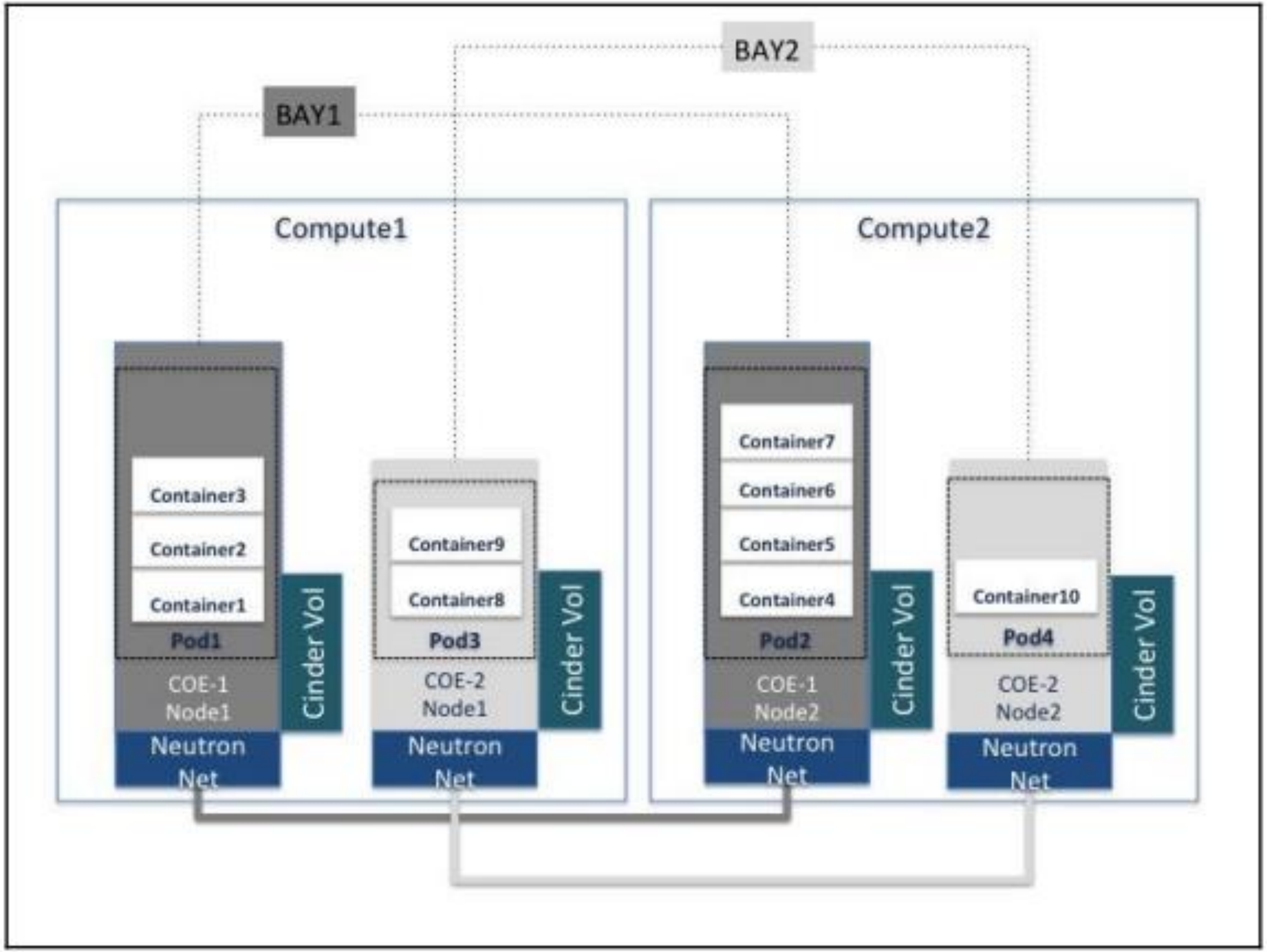
OpenStack Magnum Project

- 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.


BAY2
BAY1
Compute1
Compute2
Container3
Container2
Container1
Pod1
COE-1
Node1
Cinder Vol
Neutron
Net
Container9
Container8
Pod3
COE-2
Node1
Cinder Vol
Neutron
Net
Container7
Container6
Container5
Container4
Pod2
COE-1
Node2
Cinder Vol
Neutron
Net
Container10
Pod4
COE-2
Node2
Cinder Vol
Neutron
Net

04

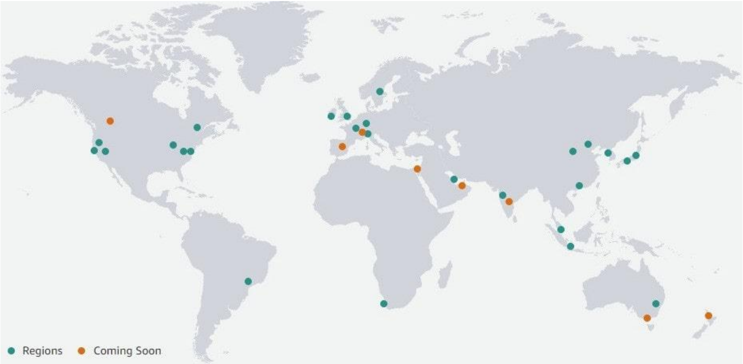
Segregating the compute cloud

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 uses 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

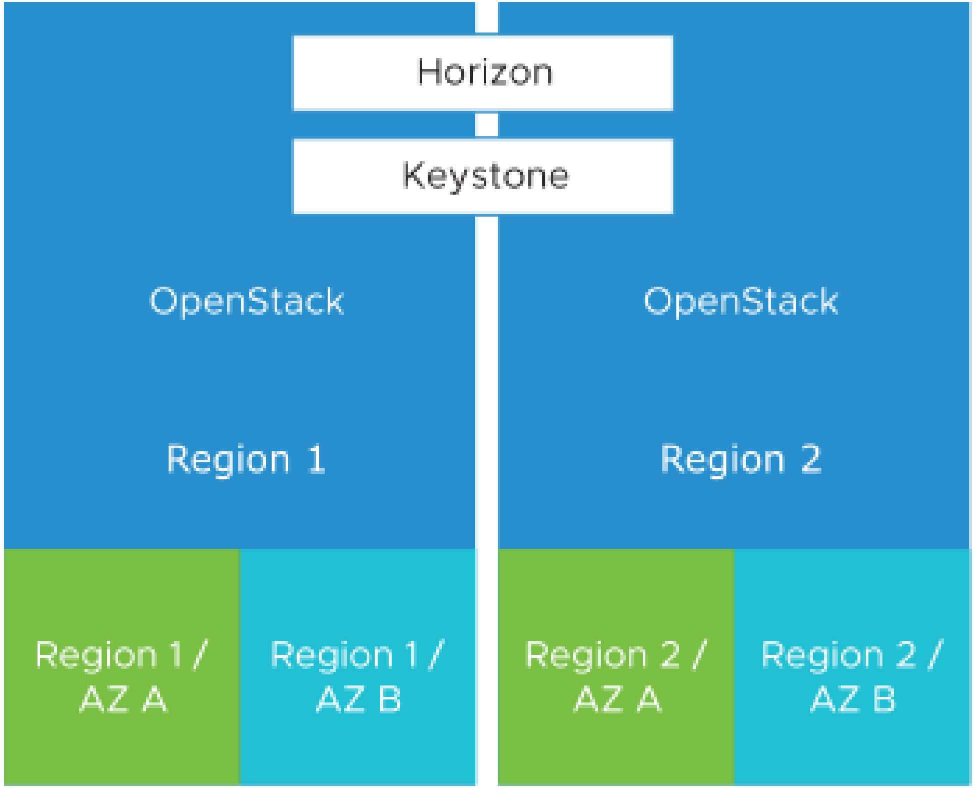


- Amazon Web Services (AWS) currently has 26 regions in operation and a further 8 under development, meaning that the company will have a total of 34 regions available by the end of 2024.
- Within each AWS region are 3 to 6 isolated, and physically separate locations, known as availability zones, which 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



Horizon

Keystone

OpenStack

Region 1

OpenStack

Region 2

Region 1 / AZ A

Region 1 / AZ B

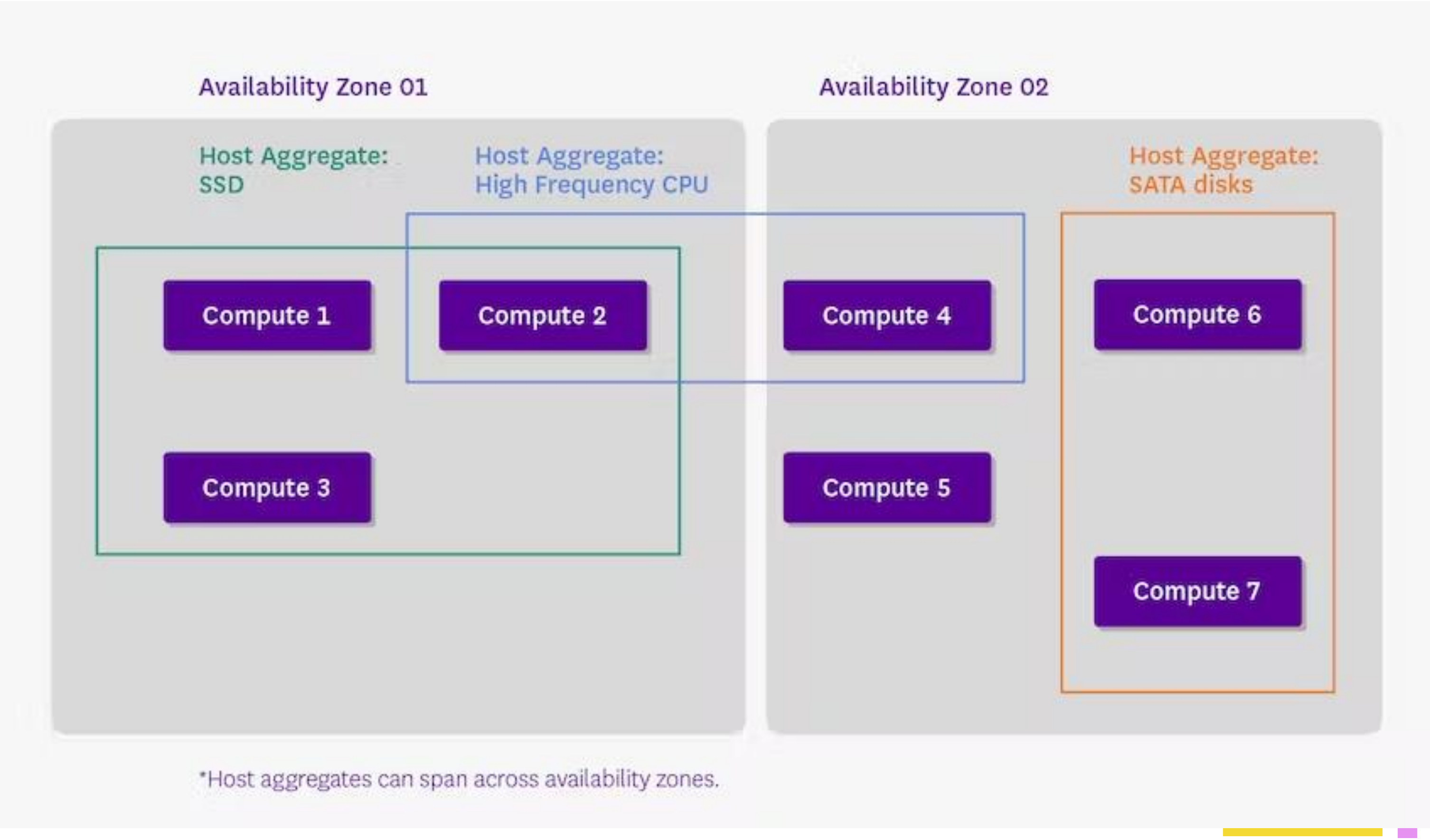
Region 2 / AZ A

Region 2 / AZ B

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



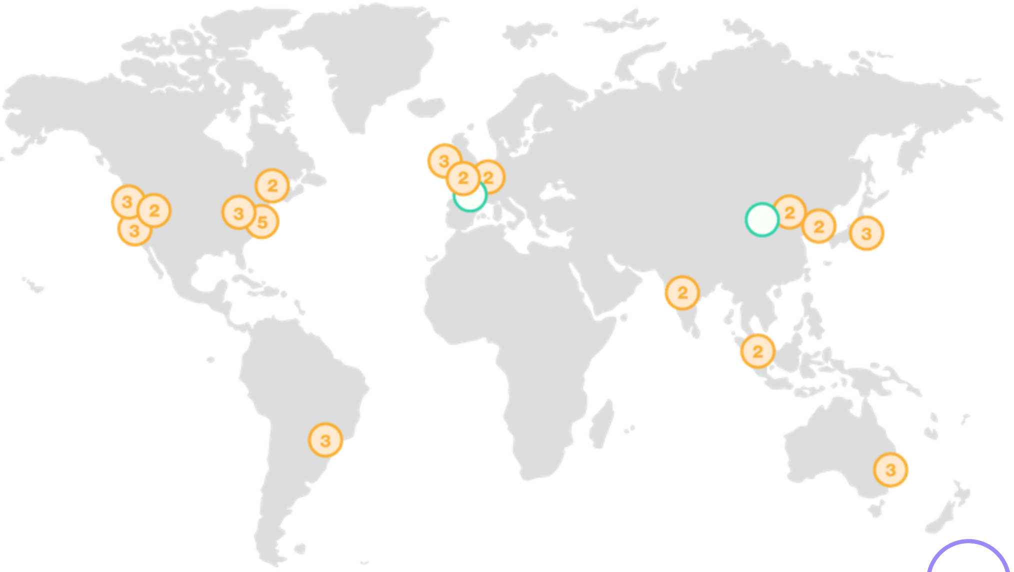
*Host aggregates can span across availability zones.

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

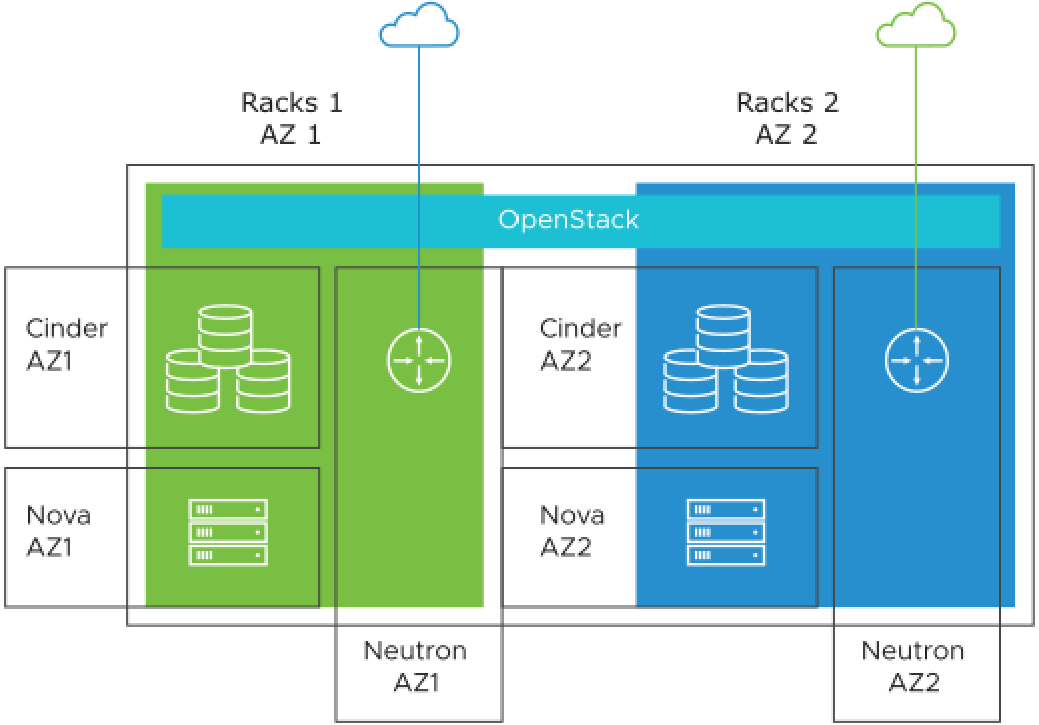
- **Availability Zones**
- An availability zone is a logical data center in a region available for use by any customer.
- Each zone in a region has redundant and separate power, networking and connectivity to reduce the likelihood of two zones failing simultaneously.
- A common misconception is that 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.

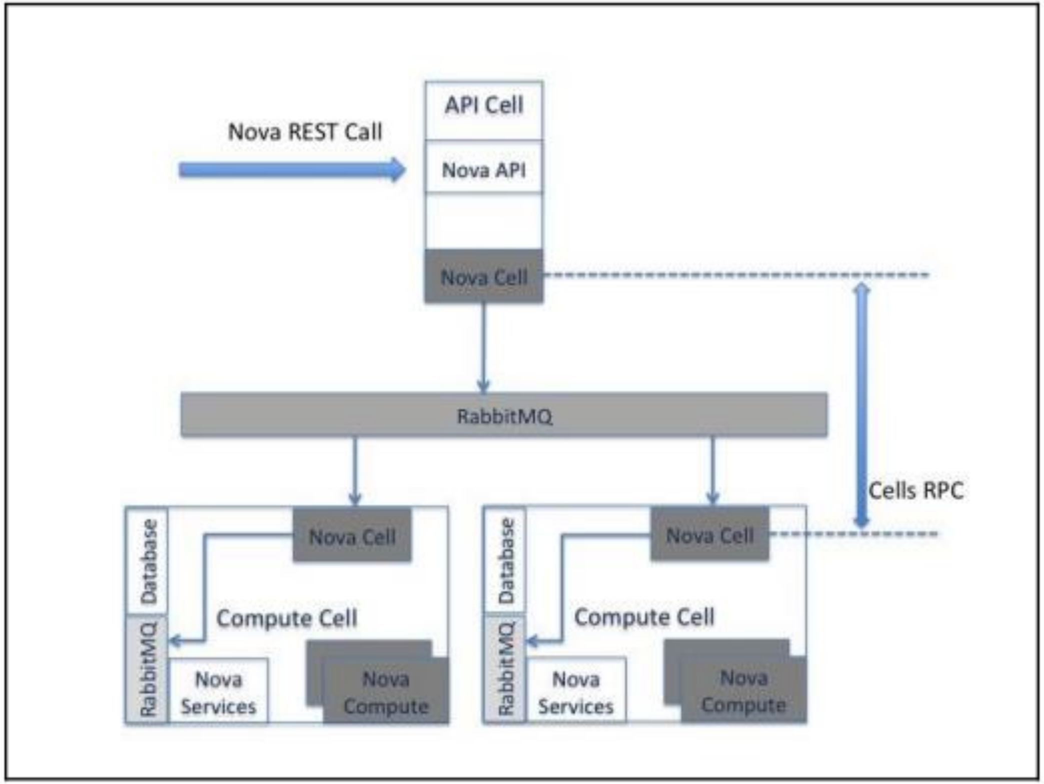
Availability zones



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

- Although the workload segregation is more of a usability feature of OpenStack cloud, it is worth mentioning in a discussion on cloud segregation.
- In the previous sections, we discussed Availability Zones and Host Aggregates that impact the way virtual machine instances are placed in an OpenStack cloud.
- The approach discussed till now handled the instance scheduling by handling a single virtual machine at a time, but what happens if you need to place your instances relative to each other?
- This use case is handled with workload segregation with affinity policy.
- To make the situation a bit clearer, let's take the example of when you have two virtual machines and you want them to be placed on the same compute node.
- Another example is when you want to have virtual machines running your application in a high-availability mode.
- Obviously, you don't want to place the instances providing the HA application on the same compute node.

Workload segregation

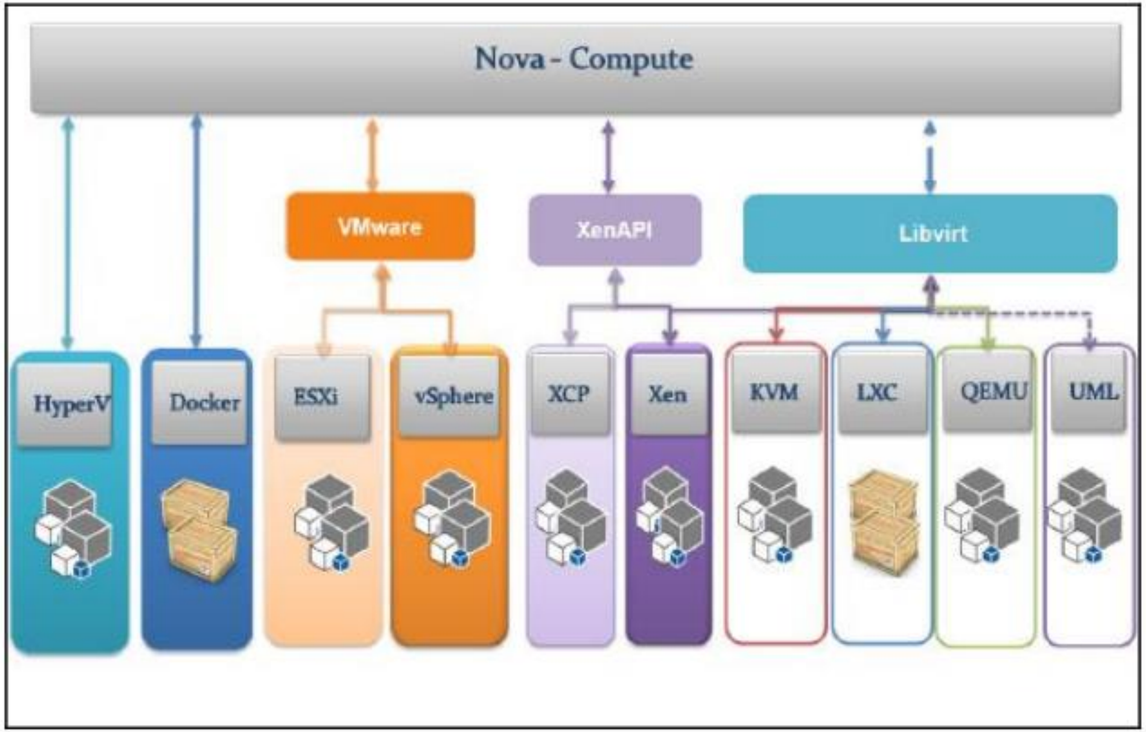
- To use workload segregation, the Nova filter scheduler must be configured with Affinity filters. Add ServerGroupAffinityFilter and ServerGroupAntiAffinityFilter to the list of scheduler filters:
- *scheduler_default_filters = ServerGroupAffinityFilter, ServerGroupAntiAffinityFilter*
- Use the Nova client to create server groups. The server group can be created with an affinity or anti-affinity-based policy as shown here:
- *# nova server–grouq svr–grqJ affinity*
- *# nova server–grouq svr–grq2 anti–affinity*
- The affinity policy places the virtual machines on the same compute node while the anti-affinity policy forces the virtual machines onto different compute nodes.
- To start the virtual machines associated with a server group, use the --hint group=svr- grp1-uuid command with the Nova client:
- *# nova boot ––image imageJ ––hint grouq=svr–grqJ–uuid ––flavor "3tandard J" vmJ*
- *# nova boot ––image imageJ ––hint grouq=svr–grqJ–uuid ––flavor "3tandard J" vm2*
- This will make sure that the virtual machines, vm1 and vm2, are placed in the same compute node.

Changing the color of the hypervisor

- While we have decided to use KVM for nova-compute, it would be great to learn how OpenStack could support a wide range of hypervisors by means of nova-compute drivers.
- You might be suggested to run your OpenStack environment with two or more hypervisors.
- It can be a user requirement to provide a choice of more than one hypervisor.
- This will help the end user resolve the challenge of native platform compatibility for their application, and then we can calibrate the performance of the virtual machine between different hypervisor environments.
- This could be a common topic in a hybrid cloud environment.

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:

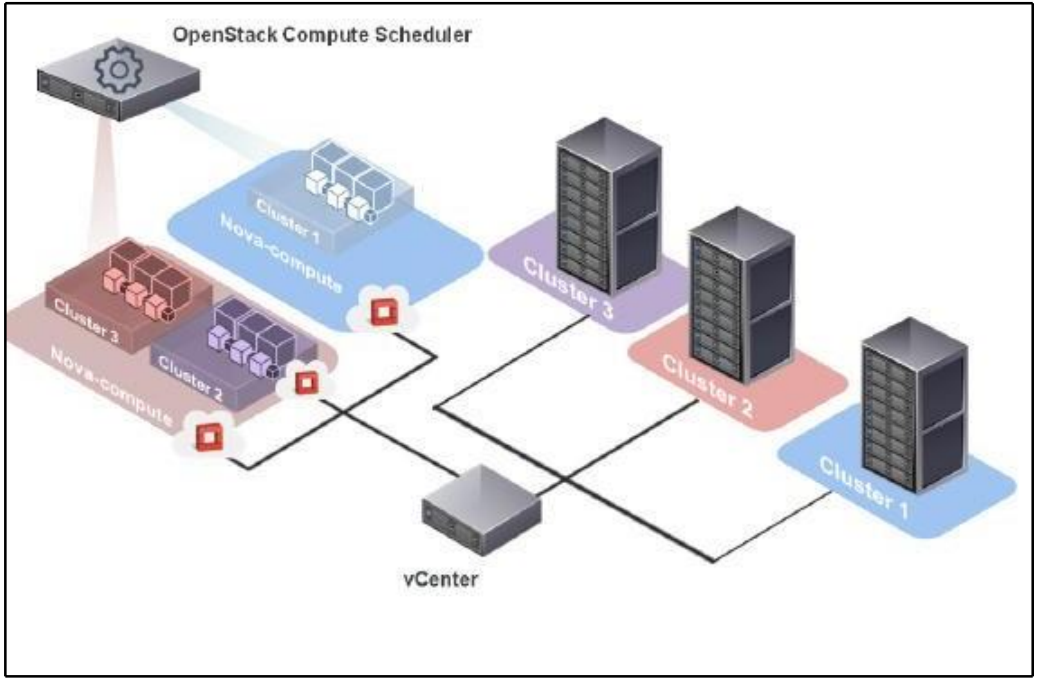


Changing the color of the hypervisor

- 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.
- Practically, the term integration on the hypervisor level refers to the OpenStack driver that will be provided to manage vSphere by nova-compute.
- Eventually, OpenStack exposes two compute drivers that have been coded:
- *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

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

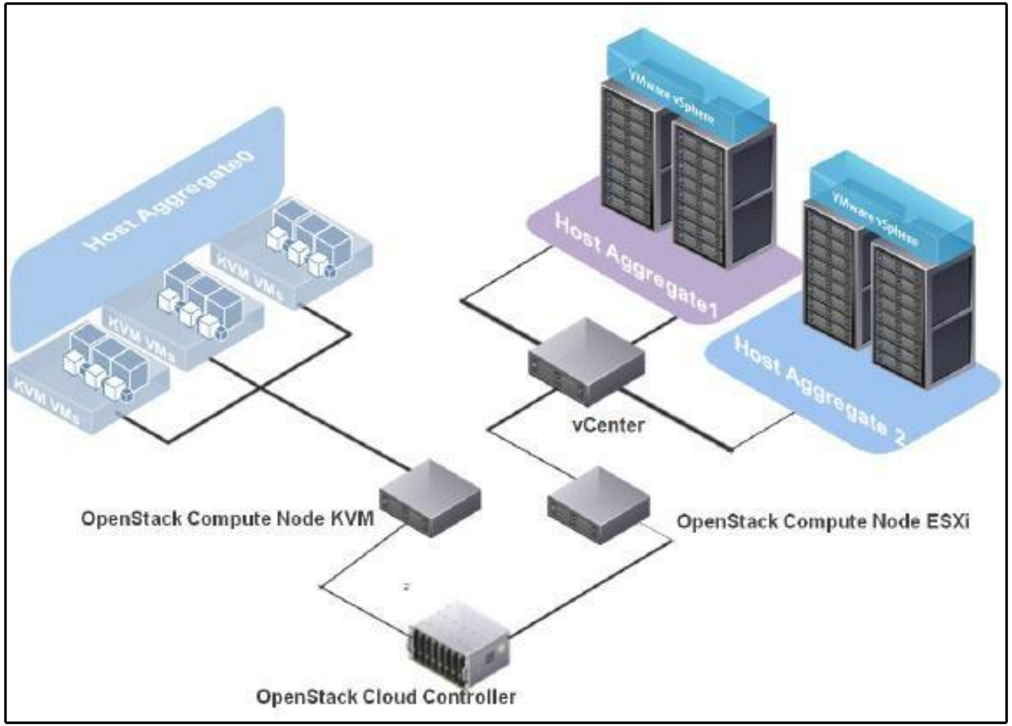
- A brief example can be conducted with the following steps:
  - Create a new host aggregate; this can be done through Horizon.
  - Select Admin project. Point to the Admin tab and open System Panel. Click on the Host Aggregates category and create new host named vSphere- Cluster_01.
  - Assign the compute 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

- vMotion is a component of VMware vSphere that allows the live migration of a running virtual machine from one host to another with no downtime.
- VMware's vSphere virtualization suite also provides a load- balancing utility called DRS, which moves computing workloads to available hardware resources.
- A good practice retrieved from this layout implementation is to place the compute node in a separate management vSphere cluster so that nodes that run nova-compute can take advantage of vSphere HA and DRS.
- vCenter can be managed by the OpenStack compute nodes only if a management vSphere cluster is created outside the OpenStack cluster.

Changing the color of the hypervisor



Changing the color of the hypervisor

- At this point, we consider that running multiple hypervisors in a single OpenStack installation is possible using Host Aggregates or using Nova cells.
- If you factor in hypervisors' varieties, do not get confused by the fact that an individual compute node always runs a single hypervisor.
- Finally, in the previous figure, the VM instance is running on KVM that is hosted directly on a nova-compute node, whereas the vSphere with vCenter on OpenStack requires a separate vCenter server host where the VM instances will be hosted on ESXi.

05

Overcommitment considerations

Overcommitment considerations

- Meaning: **to commit excessively**
- **Three main steps**
  - **Estimate a sample calculation for the CPU and RAM size.**
  - **Use OpenStack resources' overcommitment without overlooking.**
  - **As much as possible, gather resources' usage statistics periodically.**
- The art of memory or CPU overcommitment is a hypervisor feature, allowing the usage of more resource power by the virtual machine than the compute host has.
- For example, it allows a host server with 4 GB of physical memory to run eight virtual machines, each with 1 GB of memory space allocated.
- Well, there is no secrecy in this case! You should think about the hypervisor; just calculate the portion of physical memory not used per virtual machine and assign it to one that may need more RAM at certain moments. This is a technique based on the dynamic relocation of unused resources that are being held in an idle state. On the other hand, it might be a nice feature but must be used without exaggeration!
- It might be dangerous if resources are exhausted and can lead to a server crash. Therefore, we need to dive into overcommitment use cases.

Overcommitment considerations

- OpenStack allows you to overcommit CPU and RAM on compute nodes.
- This allows you to increase the number of instances running on your cloud at the cost of reducing the performance of the instances.
- The Compute service uses the following ratios by default:
- **CPU allocation ratio: 16:1**
- **RAM allocation ratio: 1.5:1**
- Using a RAM allocation ratio above 1:1 can impact running VMs if all available memory on the hypervisor is used.
- The default CPU allocation ratio of 16:1 means that the scheduler allocates up to 16 virtual cores per physical core.
- For example, if a physical node has 12 cores, the scheduler sees 192 available virtual cores.
- With typical flavor definitions of 4 virtual cores per instance, this ratio would provide 48 instances on a physical node.

CPU allocation ratio

The formula for the number of virtual instances on a compute node is

(OR*PC)/VC

where:

OR
CPU overcommit ratio (virtual cores per physical core)

PC
Number of physical cores

VC
Number of virtual cores per instance

RAM allocation ratio: 1.5:1

- The default RAM allocation ratio of 1.5:1 means that the scheduler allocates instances to a physical node as long as the total amount of RAM associated with the instances is less than 1.5 times the amount of RAM available on the physical node.

- For example, if a physical node has **48 GB of RAM**, the scheduler allocates instances to that node until the sum of the RAM associated with the instances reaches 72 GB (such as nine instances, in the case where each instance has 8 GB of RAM).

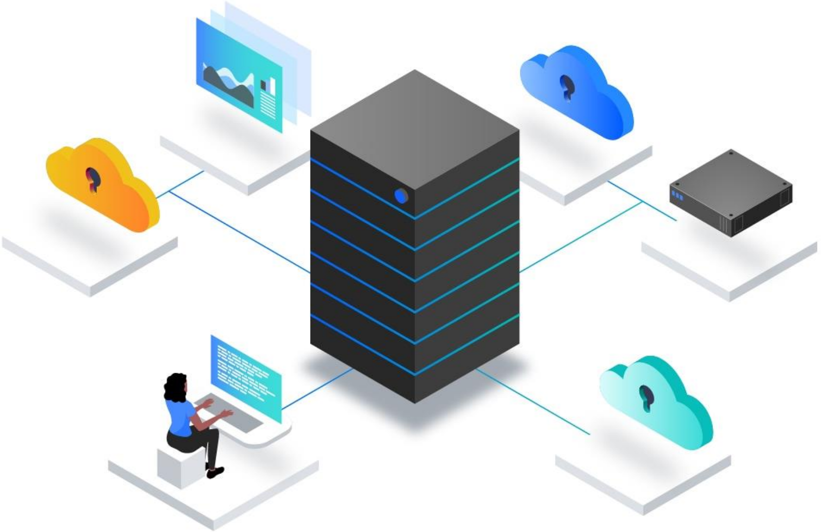
- Use the cpu_allocation_ratio and ram_allocation_ratio directives in /etc/nova/nova.conf to change the default settings.

Overcommitment considerations

- Remember that we only use overcommitment when it is needed.
- To make it more valuable, you should keep an eye on your servers. Bear in mind that collecting resource utilization statistics is essential and will eventually conduct a better ratio update when needed.
- Overcommitting is the starting point for performance improvement of your compute nodes; when you think about adjusting such a value, you will need to know exactly what you need!
- To answer this question, you will need to actively monitor the hardware usage at certain periods.
- For example, you might miss a sudden huge increase in resources' utilization requirements during the first or the last days of the month for certain user machines, whereas you were satisfied by their performance in the middle part of the month.

06

Storing instances' alternatives



Storing instances' alternatives

- Compute nodes have been sized with the total CPU and RAM capacity, but we did not cover the disk space capacity.
- 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

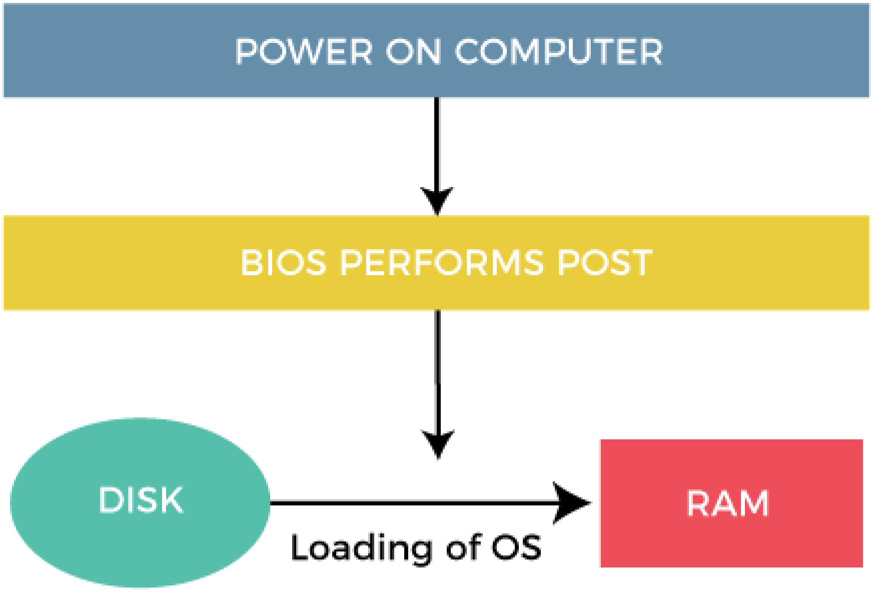
- However, some further disadvantages can be seen, such as the following:
- Inability to scale when additional storage is needed
- Difficulties in migrating instances from one compute node to another Failure of compute nodes automatically leading to instance loss

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.

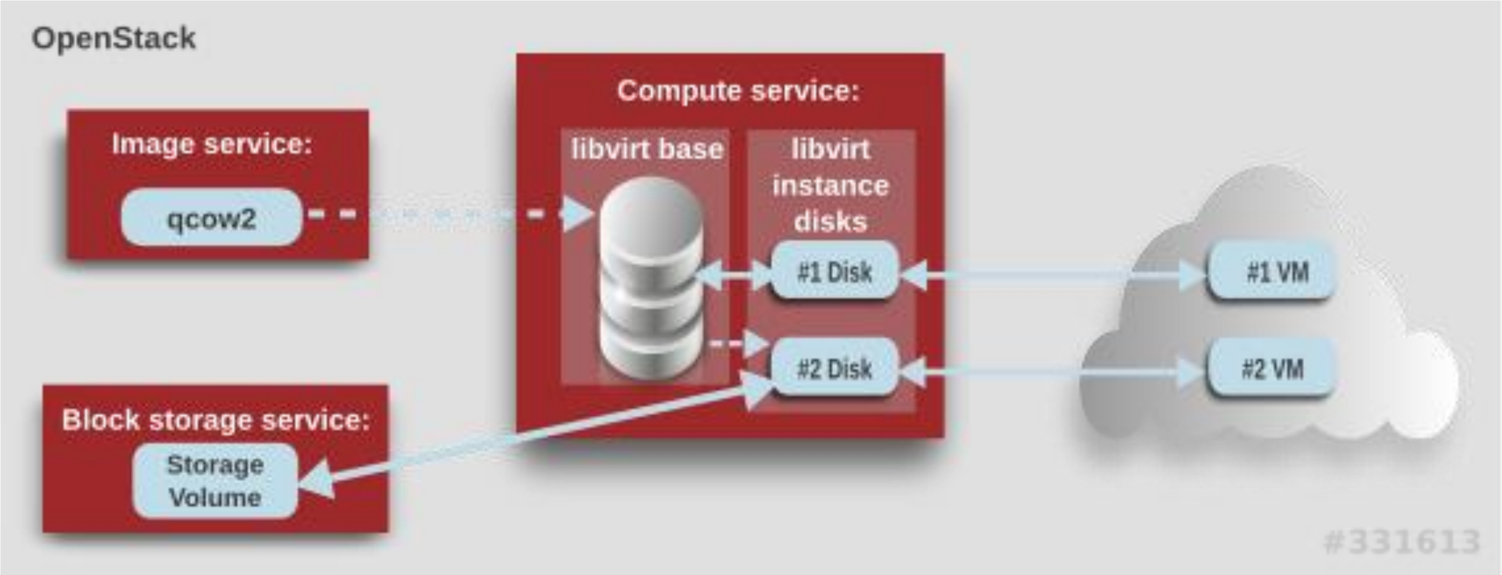
07

Understanding instance booting

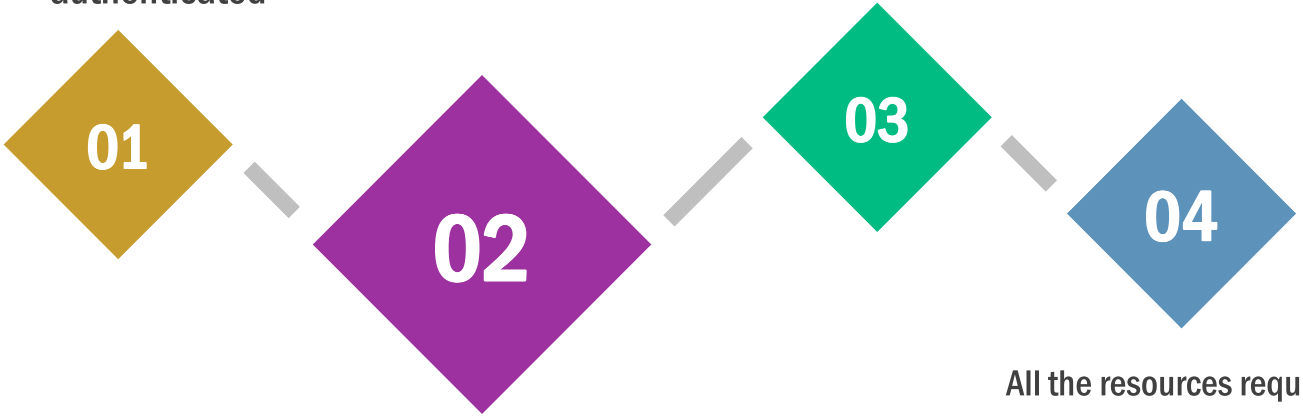


Understanding instance booting

Launching an instance on your OpenStack cloud requires interaction with multiple services.



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



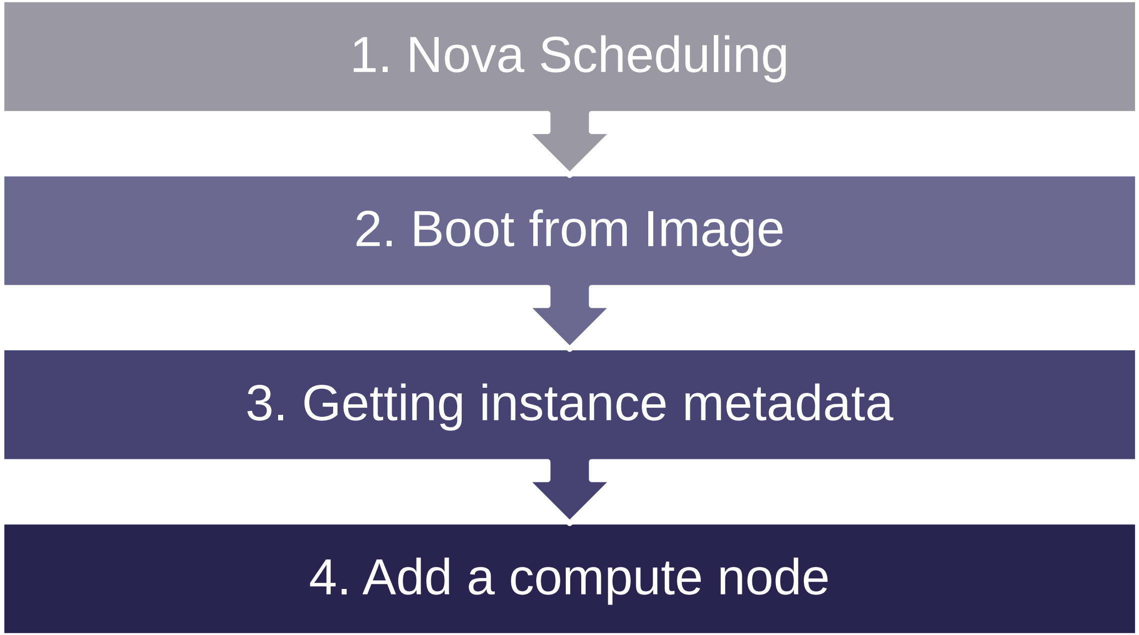
01: The user request must be authenticated

02: A compute node with adequate resources to host the virtual machine must be selected

03: Requests must be made to the image store to get the correct image for the virtual machine

04: All the resources required to launch the virtual machine must be allocated

Steps of instance booting



1. Nova Scheduling

2. Boot from Image

3. Getting instance metadata

4. Add a compute node

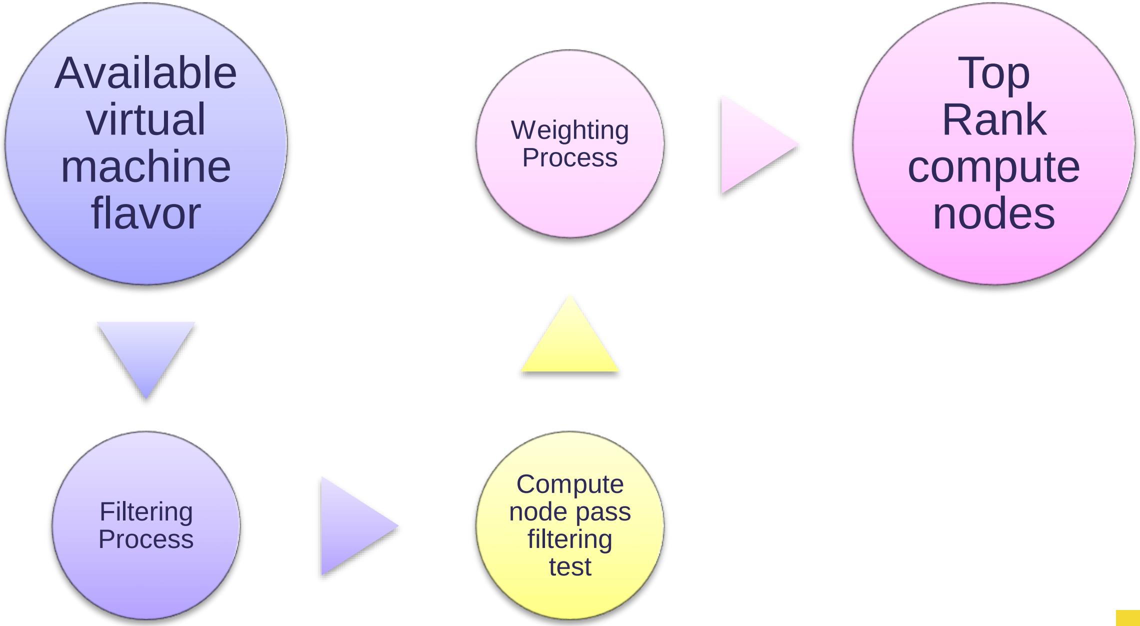
Step 1: Understanding the Nova scheduling process

- One of the critical steps in the process of launching the virtual machine
- It involves the process of selecting the best candidate compute node to host a virtual machine.
- The default scheduler used for placing the virtual machine is the filter scheduler that uses a scheme of filtering and weighting to find the right compute node for the virtual machine.
- **The scheduling process consists of going through the following steps:**

1. The virtual machine flavor itself describes the kind of resources that must be provided by the hosting compute node.
2. All the candidates must pass through a filtering process to make sure they provide adequate physical resources to host the new virtual machine. Any compute node not meeting the resource requirements is filtered out.
3. Once the compute nodes pass the filtering process, they go through a process of **weighting** that ranks the compute nodes according to the resource availability.

Step 1: Understanding the Nova scheduling process

- The filter scheduler uses a pluggable list of filters and weights to calculate the best compute node to host an instance.
- Changing the list of filters or weights can change the scheduler behavior. Setting the value of *scheduler_default_filters* can do this.



Available virtual machine flavor

Filtering Process

Compute node pass filtering test

Weighting Process

Top Rank compute nodes

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

- It provides metadata service over an HTTP server running at a special IP address of 169.256.169.254.
- To retrieve the metadata, the instances must already have networking configured and be able to reach the metadata web server.
- The metadata and user data can be retrieved on the virtual machine by sending a GET request to the metadata IP address using the **curl or wget** command line as follows:
- *# curl httq://J69.254.J69.254/latest/meta–data/*

Step 3: Getting the instance metadata

- Possible instance information hierarchically organized and can be requested 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:

1. Adjust the /etc/openstack_deploy/openstack_user_config.yml file by adding a new compute_hosts stanza pointing to the new compute node:

```
compute_hosts: cn-01:
ip: 172.47.0.20
```

Step 4: Add a compute node

2. Additional settings can be added to our compute node, including the type of hypervisor, CPU, RAM allocation ratio, and the maximum number of instances that can be spawned per host. This can be defined in the

   /etc/openstack_deploy/user_variables.yml file:

```
## Nova options
# Hypervisor type for Nova
nova_virt_type: kvm
# CPU overcommitment ratio
nova_cpu_allocation_ratio: 2.0
# RAM overcommitment ratio
nova_ram_allocation_ratio: 1.5
# Maximum number of virtual machines per compute node
nova_max_instances_per_host: 100
```

Step 4: Add a compute node

3. Install the containers in the target compute node by running the setup- hosts.yml Playbook under /etc/openstack_deploy/. If an OpenStack environment is fully running, we can instruct Ansible to limit the deployment only for the new host using the --limit option followed by the new hostname in the Ansible wrapper command line as follows:

```
# oqenstack–ansible setuq–hosts.yml ––limit cn–OJ
```

4. Optionally, it is possible to monitor the new compute node using the telemetry service by including a new metering-compute_hosts stanza in the /etc/openstack_deploy/conf.d/ceilometer.yml file:

```
...
metering-compute_hosts:
cn-01:
ip: 172.47.0.20
...
```

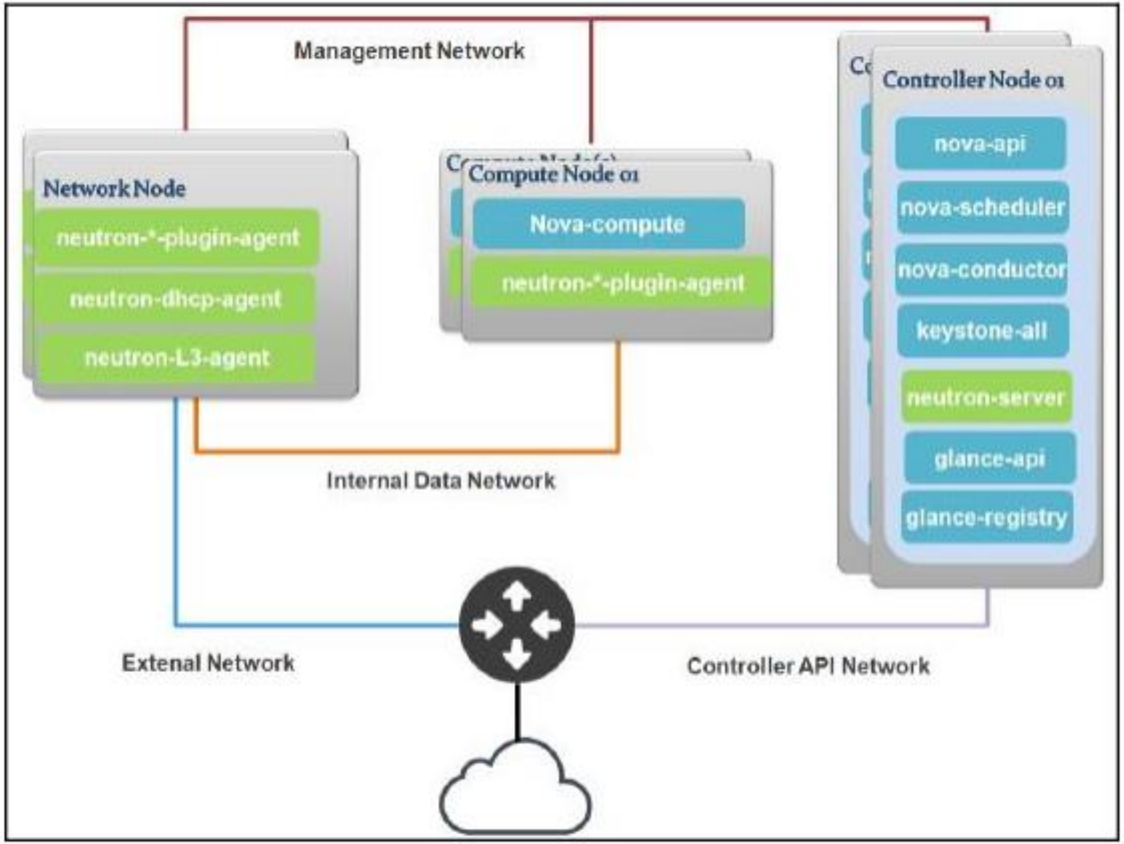
Step 4: Add a compute node

5. For a more refined update of the OpenStack infrastructure, we can instruct Ansible to deploy the new service only in the compute_hosts group added previously in the openstack_user_config.yml file :

```
# oqenstack–ansible setuq–oqenstack.yml ––limit comqute_hosts
––skiq–tags nova–key–distribute
# oqenstack–ansible setuq–oqenstack.yml ––limit comqute_hosts
––tags nova–key
```

- The new compute node should join the OpenStack environment and be ready to host instances. This can be verified in different ways by accessing the compute container.
- To identify the newly deployed host, use the ssh command line to access the compute node by filtering the utility container in the deployment machine.
- All deployed hosts should be listed in the/etc/hosts file.

Step 4: Add a compute node



08

Planning for service recovery

Planning for service recovery

- One of the most critical tasks for a system administrator or cloud operator is to plan a backup.
- Building an infrastructure and starting in production without a disaster recovery background is considered highly risky and you will need to start taking immediate action.
- We may find a bunch of property software in the cloud computing area that does the job, such as the VMware backup solution.
- However, backing up open source clouds will not be that easy. OpenStack does not, for instance, support any special tool for backup.
- As it is merely a collection of components combined to deliver services, an OpenStack operator should think how to map the components used in its infrastructure and prepare a backup strategy for each; the strategy should be easy, efficient, and auto-recovery enabled.

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.

Thanks!

Do you have any questions?

navyamolkt@amaljyothi.ac.in
me@AJCE