Module 1

Overview of OpenStack

Introduction to Cloud

Cloud computing is the delivery of computing services such as servers, storage, databases, networking, software analytics, intelligence and more over the cloud. Cloud computing provides an alternative to on – premises datacentre.

With an on – premises datacentre we have to manage everything such as purchasing and installing hardware, virtualization, installing the OS and required applications, setting up the network and configuring the firewall. After doing all this steps we become responsible for maintaining it through its entire life cycle.

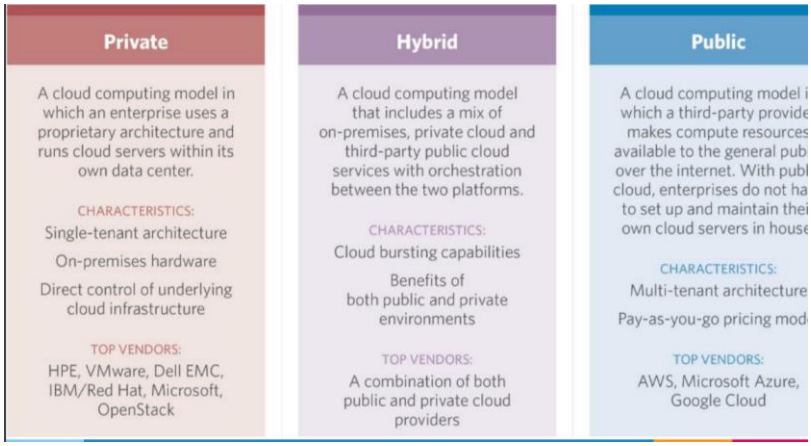
With cloud computing, a cloud vender is responsible for the hardware purchase and maintenance. They also provide a wide variety of software and platform as a service. We can take any required services on rent. The cloud computing services will be changed based on usage. The cloud environment provides an easily accessible online portal that makes handy for the user to manage the compute, storage, network and application resources.

Advantages of cloud computing

1. **Cost**: it reduces the huge capital cost of buying hardware and software.
2. **Speed:** Resources and be accessed in a minute, typically within a few clicks.
3. **Scalability:** We can increase or decrease the requirements of resources according to the business requirements.
4. **Productivity:** While using cloud computing we use less operational effort. We do nit need to apply patching as well as no need to maintain hardware and software.
5. **Reliability:** Backup and recovery of data are less expensive and very fast for business continuity.
6. **Security:** Many cloud venders offer a broad set of policies, technologies and controls that strengthen our data security.

Types of cloud computing or Cloud computing deployment models

1. **Public Cloud:** The public cloud infrastructure is available for the public use and cloud resources that are owned and operated by a third-party cloud service provider are termed as public clouds. It delivers computing resources such as servers, software and storage over the internet.
2. **Private Cloud:** The cloud computing resources that are exclusively used inside a single business or organization are termed as private cloud. A private cloud may physically be located on the company's own datacentre or hosted by a third-party service provider.
3. **Hybrid Cloud:** It is the combination of public and private clouds which is bounded together by technologies that allows data applications to be shared between them. Hybrid cloud provides flexibility and more deployment options to the business.



Types of Cloud services

1. **Infrastructure as a Service (IaaS)**

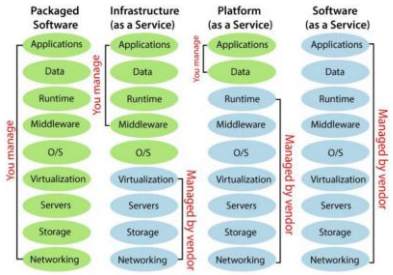
In IaaS, we can rent IT infrastructure like servers and virtual machines, storage and networks from a cloud service vender. We can create VM running Windows or Linux and installing anything we want on it. Using IaaS, we don't need to care about the hardware or virtualization software, but other than that we do have to manage everything else. Using IaaS, we get maximum flexibility but we need to put more effort into the maintenance.

2. **Platform as a Service (PaaS)**

This service provides an on demanding environment for developing, testing, delivering and managing software applications. The developer is responsible for the applications and the PaaS vender provides the ability to deploy and run it. Using PaaS, the flexibility gets reduce but the management of the environment is taken care of by the cloud venders.

3. **Software as a Service (SaaS)**

It provides a centrally hosted and managed software services to the end users. It delivers software over the internet, on demand, and typically on a subscription basis. SaaS is used to minimize the operational cost to the maximum extent. Example: Microsoft OneDrive, Dropbox, WordPress, and Amazon Kindle.



OpenStack is an open-source private cloud platform designed to manage distributed compute, network and storage resources in the data centre. It is mostly deployed as infrastructure-as-a-service (IaaS) in both public and private clouds where virtual servers and other resources are made available to users.

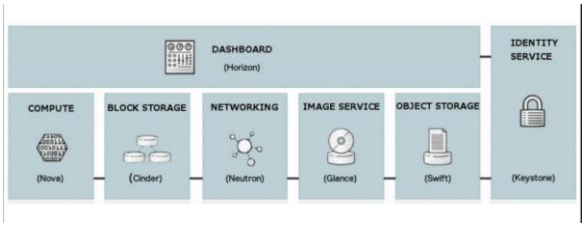
OpenStack - The new data center paradigm

OpenStack promises the features of a next-generation data center operating system. big global cloud enterprises that are influenced by OpenStack: VMware, Cisco, Juniper, IBM, RedHat, Rackspace, PayPal, and eBay. (Today, many of them are running a very large scalable private cloud based on OpenStack in their production environment.)

Introducing the OpenStack logical architecture

Services of OpenStack

Different services catering to different needs, they follow a common theme in their design that can be summarized as follows: Most OpenStack services are developed in Python, which aids rapid development. All OpenStack services provide REST APIs. These APIs are the main external communication interfaces for services and are used by the other services or end users. The OpenStack service itself may be implemented as different components. The components of a service communicate with each other over the message queue. The message queue provides various advantages such as queuing of requests, loose coupling, and load distribution among the worker daemons.



1. **Keystone - identity management**

It is the core component and provides an identity service comprising authentication and authorization of tenants in OpenStack. Communications between different OpenStack services are authorized by Keystone to ensure that the right user or service is able to utilize the requested OpenStack service. Keystone integrates with numerous authentication mechanisms such as username/password and token/authentication-based systems. It is possible to integrate it with an existing backend such as the Lightweight Directory Access Protocol (LDAP) and the Pluggable Authentication Module (PAM). Keystone also provides a service catalog as a registry of all the OpenStack services.

2. **Swift - object storage**

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

- **Benefits of Swift - object storage**

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

3. **Cinder - block storage**

Its main capability is to provide persistent block-level storage to the virtual machine. Cinder provides raw volumes that can be used as hard disks in virtual machines.

The features that Cinder:

1. Volume management: This allows the creation or deletion of a volume.
2. Snapshot management: This allows the creation or deletion of a snapshot of volumes.
3. Attaching or detaching volumes from instances
4. Cloning volumes

5. Creating volumes from snapshots
6. Copy of images to volumes and vice versa.

4. **Manila - File share**

   Manila is a file-share-based OpenStack storage service. It provides storage as a remote file system. In operation, it resembles the Network File System (NFS) or SAMBA storage service that we are used on Linux. In contrast to Cinder, it resembles the Storage Area Network (SAN) service. In fact, NFS and SAMBA or the Common Internet File System (CIFS) are supported as backend drivers to the Manila service. The Manila service provides the orchestration of shares on the share servers.

5. **Glance - Image registry**

   The Glance service provides a registry of images and metadata that the OpenStack user can launch as a virtual machine. Various image formats are supported and can be used based on the choice of hypervisor. Glance supports images for KVM/Qemu, XEN, VMware, Docker, and so on.

   Swift is a storage system, whereas Glance is an image registry. The difference between the two is that Glance is a service that keeps track of virtual machine images and metadata associated with the images. Metadata can be information such as a kernel, disk images, disk format, and so on. Glance makes this information available to OpenStack users over REST APIs. Glance can use a variety of backends for storing images. The default is to use directories, but in a massive production environment it can use other approaches such as NFS and even Swift. Swift, on the other hand, is a storage system. It is designed for object storage where you can keep data such as virtual disks, images, backup archiving, and so on. The mission of Glance is to be an image registry. From an architectural point of view, the goal of Glance is to focus on advanced ways to store and query image information via the Image Service API.

   A typical use case for Glance is to allow a client (which can be a user or an external service) to register a new virtual disk image, while a storage system focuses on providing a highly scalable and redundant data store. At this level, as a technical operator, your challenge is to provide the right storage solution to meet cost and performance requirements.

6. **Nova-Compute service**

   Nova is the original core component of OpenStack. From an architectural level, it is considered one of the most complicated components of OpenStack. Nova provides the compute service in OpenStack and manages virtual machines in response to service requests made by OpenStack users. What makes Nova complex is its interaction with a large number of other OpenStack services and internal components, which it must collaborate with to respond to user requests for running a VM.

   nova-api

   The nova-api component accepts and responds to the end user and computes API calls. The end users or other components communicate with the OpenStack nova-api interface to create instances via the OpenStack API or EC2 API.

   nova-compute

   The nova-compute component is primarily a worker daemon that creates and terminates VM instances via the hypervisor's APIs (XenAPI for XenServer, Libvirt KVM, and the VMware API for VMware).

   nova-network

   The nova-network component accepts networking tasks from the queue and then performs these tasks to manipulate the network (such as setting up bridging interfaces or changing IP table rules). Neutron is a replacement for the nova-network service.

nova-scheduler
The nova-scheduler component takes a VM instance's request from the queue and determines where it should run (specifically which compute host it should run on). At an application architecture level, the term scheduling or scheduler invokes a systematic search for the best outfit for a given infrastructure to improve its performance.
nova-conductor
The nova-conductor service provides database access to compute nodes. The idea behind this service is to prevent direct database access from the compute nodes, thus enhancing database security in case one of the compute nodes gets compromised.
Nova interacts with several services such as Keystone for authentication, Glance for images, and Horizon for the web interface. For example, the Glance interaction is central; the API process can upload any query to Glance, while nova-compute will download images to launch instances. Nova also provides console services that allow end users to access the console of the virtual instance through a proxy such as nova- console, nova-novncproxy, and nova-consoleauth.
Neutron - Networking services
Neutron provides a real Network as a Service (NaaS) capability between interface devices that are managed by OpenStack services such as Nova. There are various characteristics that should be considered for Neutron:

It allows users to create their own networks and then attaches server interfaces to them.
Its pluggable backend architecture lets users take advantage of commodity gear or vendor supported equipment.
It provides extensions to allow additional network services to be integrated.

Neutron has many core network features that are constantly growing and maturing. Some of these features are useful for routers, virtual switches, and SDN networking controllers.

Neutron introduces the following core resources:

Ports: Ports in Neutron refer to the virtual switch connections. These connections are where instances and network services are attached to networks. When attached to subnets, the defined MAC and IP addresses of the interfaces are plugged into them. Networks: Neutron defines networks as isolated Layer 2 network segments. Operators will see networks as logical switches that are implemented by the Linux bridging tools, Open vSwitch, or some other virtual switch software. Unlike physical networks, either the operators or users in OpenStack can define this.

Subnet: Subnets in Neutron represent a block of IP addresses associated with a network. IP addresses from this block are allocated to the ports.

Neutron provides additional resources as extensions. The following are some of the commonly used extensions:

Routers: Routers provide gateways between various networks.

Private IPs: Neutron defines two types of networks. They are as follows:

- **Tenant networks**: Tenant networks use private IP addresses. Private IP addresses are visible within the instance and this allows the tenant's instances to communicate while maintaining isolation from the other tenant's traffic. Private IP addresses are not visible to the Internet.
- **External networks**: External networks are visible and routable from the Internet. They must use routable subnet blocks. (iii)Floating IPs: A floating IP is an IP address allocated on an external network that Neutron maps to the private IP of an instance. Floating IP addresses are assigned to an instance so that they can connect to external networks and access the Internet. Neutron achieves the mapping of floating IPs to the private IP of the instance by using Network Address Translation (NAT).

Neutron also provides advanced services to rule additional network OpenStack capabilities as follows:

Load Balancing as a Service (LBaaS) to distribute the traffic among multiple compute node instances.
Firewall as a Service (FWaaS) to secure layer 3 and 4 network perimeter access.
Virtual Private Network as a Service (VPNaaS) to build secured tunnels between instances or hosts.

The Neutron architecture

The three main components of the Neutron architecture are:

Neutron server: It accepts API requests and routes them to the appropriate Neutron plugin for action.
Neutron plugins: They perform the actual work for the orchestration of backend devices such as the plugging in or unplugging ports, creating networks and subnets, or IP addressing.

Neutron agents: Neutron agents run on the compute and network nodes. The agents receive commands from the plugins on the Neutron server and bring the changes into effect on the individual compute or network nodes. Different types of Neutron agents implement different functionality. Neutron is a service that manages network connectivity between the OpenStack instances. It ensures that the network will not be turned into a bottleneck or limiting factor in a cloud deployment and gives users real self-service, even over their network configurations. Neutron has the ability to provide a way to integrate vendor networking solutions and a flexible way to extend network services. It is designed to provide a plugin and extension mechanism that presents an option for network operators to enable different technologies via the Neutron API.

Ceilometer, Aodh, and Gnocchi – Telemetry

Ceilometer provides a metering service in OpenStack. In a shared, multitenant environment such as OpenStack, metering resource utilization is of prime importance. Ceilometer collects data associated with resources. Resources can be any entity in the OpenStack cloud such as VMs, disks, networks, routers, and so on. Resources are associated with meters. The utilization data is stored in the form of samples in units defined by the associated meter. Ceilometer has an inbuilt summarization capability. Ceilometer allows data collection from various sources, such as the message bus, polling resources, centralized agents, and so on. The Alarming service has been decoupled from the Ceilometer project to make use of a new incubated project code-named Aodh. The Telemetry Alarming service will be dedicated to managing alarms and triggering them based on collected metering and scheduled events. More Telemetry service enhancements have been proposed to adopt a Time Series Database as a Service project code-named Gnoochi.

Heat – Orchestration

Heat is an OpenStack Orchestration project. Initial development for Heat was limited to a few OpenStack resources including compute, image, block storage, and network services. Heat has boosted the emergence of resource management in OpenStack by orchestrating different cloud resources resulting in the creation of stacks to run applications with a few pushes of a button.

From simple template engine text files referred to as HOT templates (Heat Orchestration Template), users are able to provision the desired resources and run applications in no time. Heat uses templates files in YAML or JSON format; indentation is important!

Horizon – Dashboard

Horizon is the web dashboard that pulls all the different pieces together from the OpenStack ecosystem. Horizon provides a web frontend for OpenStack services. Currently, it includes all the

OpenStack services as well as some incubated projects. It was designed as a stateless and data-less web application. It does nothing more than initiate actions in the OpenStack services via API callsand display information that OpenStack returns to Horizon. It does not keep any data except the session information in its own data store. Horizon is based on a series of modules called panels that define the interaction of each service. Its modules can be enabled or disabled, depending on the service availability of the particular cloud. In addition to this functional flexibility, Horizon is easy to style with Cascading Style Sheets (CSS).

Message Queue

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

The database

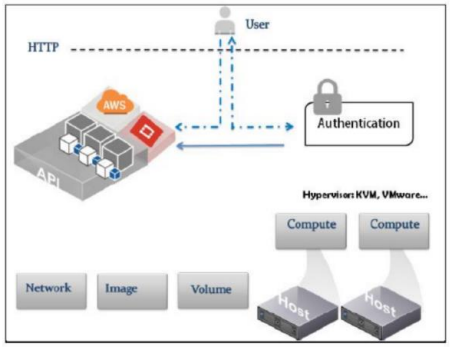
Its database stores most of the build-time and run-time states for the cloud infrastructure, including instance types that are available for use, instances in use, available networks, and projects. It provides a persistent storage for preserving the state of the cloud infrastructure. It is the second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

How OpenStack works by chaining all the service cores

The series of steps:

1. Authentication is the first action performed. This is where Keystone comes into the picture. Keystone authenticates the user based on credentials such as the username and password.
2. The service catalog is then provided by Keystone. This contains information about the OpenStack services and the API endpoints.
3. You can use the Openstack CLI to get the catalog: $ openstack catalog list. The service catalog is a JSON structure that exposes the resources available on a token request.
4. Typically, once authenticated, you can talk to an API node. There are different APIs in the OpenStack ecosystem (the OpenStack API and EC2 API):

high-level view of how OpenStack works:



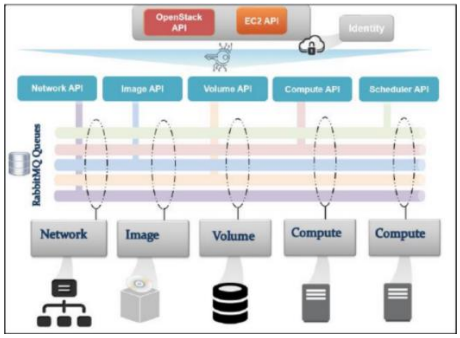
5. Another element in the architecture is the instance scheduler. Schedulers are implemented by OpenStack services that are architected around worker daemons. The worker daemons manage the launching of instances on individual nodes and keep track of resources available to the physical nodes on which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

decides the best candidate node to launch a virtual instance on. An example of this architecture is nova-scheduler.

Provisioning a VM under the hood

How Different services in OpenStack work together, leading to a running virtual machine. How things work

– simple architecture diagram:



The process of launching a virtual machine involves the interaction of the main OpenStack services that form the building blocks of an instance including compute, network, storage, and the base image. OpenStack services interact with each other via a message bus to submit and retrieve RPC calls. The information of each step of the provisioning process is verified and passed by different OpenStack services via the message bus.

The provisioning workflow based on API calls in OpenStack:

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

A sample architecture setup

There are three phases of design:

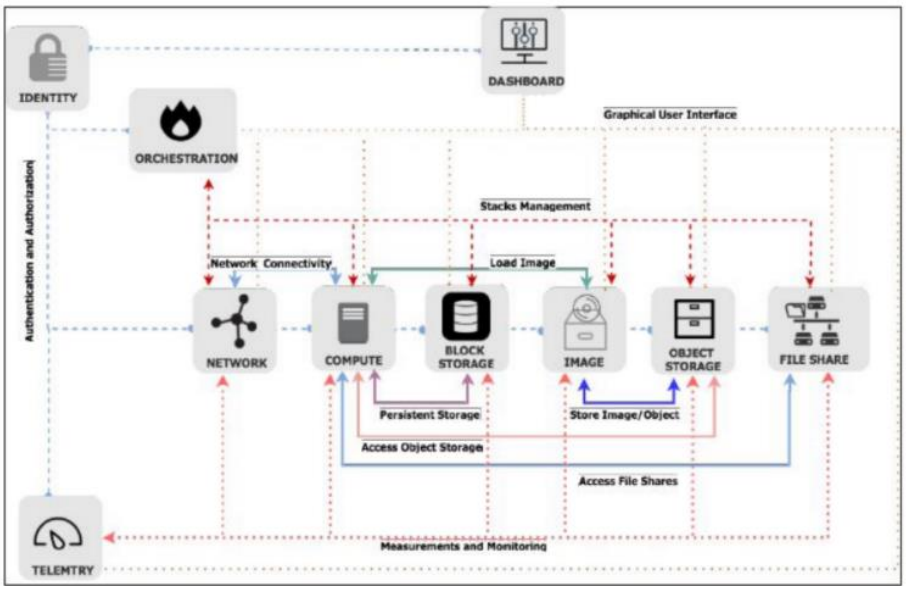
1. Designing a conceptual model
2. Designing a logical model,
3. Realizing the physical design.

The conceptual model design

Class	Role
Compute	Stores virtual machine images Provides a user interface
Image	Stores disk files Provides a user interface
Object storage	Stores objects Provides a user interface
Block storage	Provides volumes Provides a user interface
Network	Provides network connectivity Provides a user interface

Class	Role
Telemetry	Provides measurements, metrics, and alerts Provides a user interface
File Share	Provides a scale-out file share system for OpenStack Provides a user interface
Identity	Provides authentication
Dashboard	Provides a graphical user interface
Orchestration	Provides orchestration engine for stack creation Provides a user interface

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



Storage

Ephemeral storage is the volatile temporary storage attached to your instances which is only present during the running lifetime of the instance. ... Ephemeral storage is ideally used for any temporary data such as cache, buffers, session data, swap volume etc. ephemeral storage as the place where the end user will not be able to access the virtual disk associated with its VM when it is terminated. Ephemeral storage should mainly be used in production when the VM state is non- critical, where users or application don't store data on the VM. Ephemeral storage can also be a choice for certain users; for example, when they consider building a test environment. If you need your data to be persistent, you must plan for a storage service such as Cinder or Manila. Swift can be used when you have a sufficient volume of critical data in your cloud environment and start to feel the need for replication and redundancy.

The tenant data network

The main feature of a data network that it provides the physical path for the virtual networks created by the OpenStack tenants. It separates the tenant data traffic from the infrastructure communication path required for the communication between the OpenStack component itself.

Management and the API network

In a smaller deployment, the traffic for management and communication between the OpenStack components can be on the same physical link. This physical network provides a path for communication between the various OpenStack components such as REST API access and DB traffic, as well as for managing the OpenStack nodes.

The Storage network

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

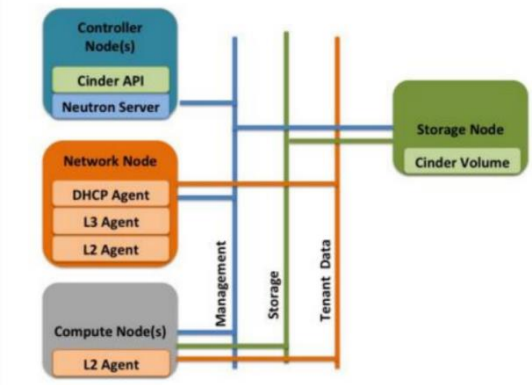
Virtual Network types

The external network The features of an external or a public network : It provides global connectivity and uses routable IP addressing. It is used by the virtual router to perform SNAT from the VM instances and provide external access to traffic originating from the VM and going to the Internet. SNAT refers to Source Network Address Translation. It allows traffic from a private network to go out to the Internet. OpenStack supports SNAT through its Neutron APIs for routers. It is used to provide a DNAT service for traffic from the Internet to reach a service running on the VM instance. While using VLANs, by tagging networks and combining multiple networks into one Network Interface Card (NIC), you can optionally leave the public network untagged for that NIC, to make the access to the OpenStack dashboard and the public OpenStack API endpoints simple.

The tenant networks

The features of the tenant network are as follows:

It provides a private network between virtual machines
It uses private IP space It provides isolation of tenant traffic and allows multi-tenancy requirements for networking services
The next step is to validate our network design in a simple diagram.



The physical model design

Estimating the hardware capabilities
Since the architecture is being designed to scale horizontally, we can add more servers to the setup. *We will start by using cost effective hardware. *make some basic hardware calculations for the first estimation of our exact requirements.
There is a possibility of experiencing contentions for resources such as CPU, RAM, network, and disk. Capacity management is considered a day-to-day responsibility where you have to stay updated with regard to software or hardware upgrades.
A real-life example of the impact of underestimating capacity planning

A cloud-hosting company set up two medium servers, one for an e-mail server and the other to host the official website. The company, which is one of our several clients, grew in a few months and eventually ran out of disk space. The expected time to resolve such an issue is a few hours, but it took days. The problem was that all the parties did not make proper use of the cloud, due to the on demand nature of the service. This led to Mean Time To Repair (MTTR) increasing exponentially. The cloud provider did not expect this!

Capacity management processes

1st step: Looping through tuning, monitoring, and analysis.
2nd step: take into account your tuned parameters and introduce the right change, within your hardware/software, which involves a synergy of the change management process.

CPU calculations

The following are the calculation-related assumptions:

200 virtual machines
GHz per physical core = 2.6 GHz
Physical core hyper-threading support = use factor 2
GHz per VM (AVG compute units) = 2 GHz
GHz per VM (MAX compute units) = 16 GHz
Intel Xeon E5-2648L v2 core CPU(total cores) = 10
CPU sockets per server = 2

The formula for calculating the total number of CPU cores:
(number of VMs x number of GHz per VM) / number of GHz per core
(200 * 2) / 2.6 = 153.846

We have 153 CPU cores for 200 VMs. The formula for calculating the number of core CPU sockets :
Total number of sockets / total cores
153 / 10 = 15.3
We will need 15 sockets

The formula for calculating the number of socket servers:
Total number of sockets / Number of sockets per server
15 / 2 = 7.5
You will need around seven to eight dual socket servers. The number of virtual machines per server with eight dual socket servers :
Number of virtual machines / number of servers
200 / 8 = 25
We can deploy 25 virtual machines per server

Memory calculations

Memory sizing is also important to avoid making unreasonable resource allocations. Let's make an assumption list (keep in mind that it always depends on your budget and needs):
2 GB RAM per VM
8 GB RAM maximum dynamic allocations per VM
RAM available per compute node:
8 * 25 = 200 GB

Network calculations assumptions:

200 Mbits/second is needed per VM

it might be possible to serve our VMs by using a 10 GB link foreach server, which will give:
10,000 Mbits/second / 25VMs = 400 Mbits/second

Storage calculations

we need to plan for an initial storage capacity per server that will serve 25 VMs each. assuming 100 GB ephemeral storage per VM, will require a space of 25*100 = 2.5 TB of local storage on each compute node. You can assign 250 GB of persistent storage per VM to have 25*250 = 5 TB of persistent storage per compute node.