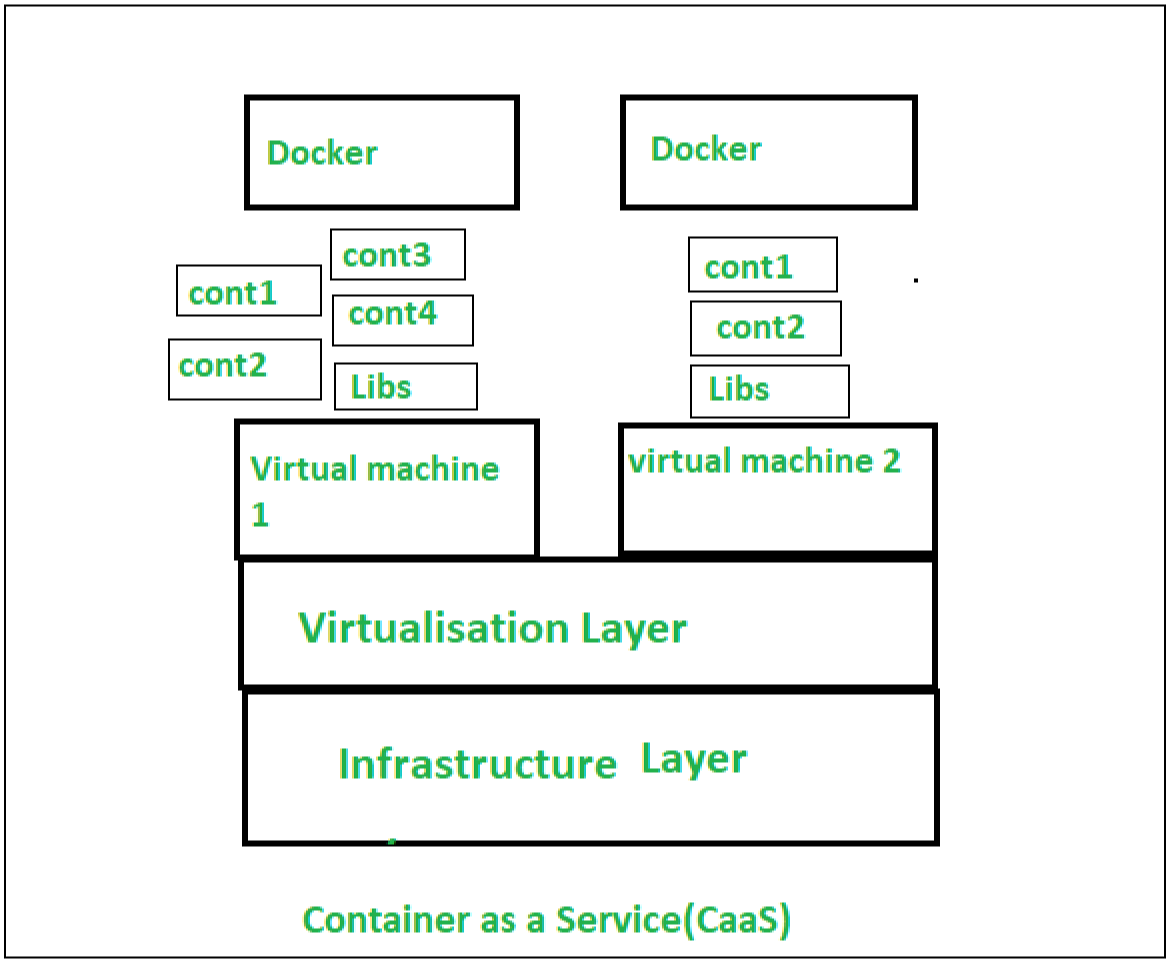
Container as a Service (CaaS)

What is a Container :
Containers are a usable unit of software in which application code is inserted, as well as libraries and their dependencies, in the same way that they can be run anywhere, be it on desktop, traditional IT, or in the cloud.
To do this, the containers take advantage of the virtual operating system (OS) in which OS features (in the case of Linux kernel, which are the first names and groups of domains) are used in both CPU partition, memory, and disk access processes.

Container as a Service (CaaS) :
Containers as a service (CaaS) is a cloud service model that allows users to upload, edit, start, stop, rate and otherwise manage containers, applications and collections. It enables these processes through tool-based virtualization, a programming interface (API) or a web portal interface. CaaS helps users create rich, secure and fragmented applications through local or cloud data centres. Containers and collections are used as a service with this model and are installed in the cloud or data centres on the site.

CaaS assists development teams to deploy and manage systems efficiently while providing more control of container orchestration than is permitted by PaaS.
Containers-as-a-service (CaaS) is part of cloud services where the service provider empowers customers to manage and distribute applications containing containers and collections. CaaS is sometimes regarded as a special kind of Infrastructure-as-a-service (IaaS) model for cloud service delivery, but where larger assets are containers there are virtual machines and physical hardware.



Advantages of Container as a Service (CaaS) :

- Containers and CaaS make it much easier to deploy and design distributed applications or build smaller service.
- During development, a collection of containers may handle different responsibilities or different coding environments.
- Network protocol relationships between containers can be defined and binding to forwarding.
- CaaS promise that these defined and dedicated container structures can be deployed quickly in cloud capture.
- For considering an example, Think of a fake software program designed with a microservice design, in which the service plan is organized with a business domain ID. Service domains can be: payments, authentication, and a shopping cart.
- Using CaaS, these application containers can be sent instantly to a live system.
- Posting applications installed on the CaaS platform enables program performance using tools such as log integration and monitoring.
- CaaS also includes built-in automated measurement performance and orchestration management.
- It enables teams to quickly build high visibility and distributed systems for high availability.
- In addition, CaaS enhances team development with power by enabling faster deployment.

- Containers are used to prevent targeted deployment while CaaS can reduce engineering operating costs by reducing the DevOps resources needed to manage deployments.

Disadvantages of Container as a Service (CaaS) :

- Depending on the provider, there are limits to the technology available.
- Extracting business data from the cloud is dangerous.

Security Issues :

- Containers are considered safer than their counterparts like Microsoft Machines, but have some risks.
- Although they are agnostic platforms, containers share the same kernel as the operating system.
- This puts the containers at risk of being targeted if they are targeted.
- The risks increase exponentially as containers are deployed in the Cloud via CaaS.

Performance Limitations:

- Containers are visible areas and do not run directly on the bare metal.
- There is something missing with the extra layer between the bare metal and the application containers and its characters.
- Combine this with the network loss of the container associated with the hosting plan; the result is a significant loss of performance.
- Therefore, businesses must face some losses in the functionality of the containers even after the high quality hardware available.
- Therefore, it is sometimes preferred to use programs with a bare metal to test the full potential of the application.

How CaaS Works?

A container as a service is a computing and accessible computer cloud. Used by users to upload, create, manage and deploy container-based applications on the cloud platform. Cloud-based environment connections can be made using a graphical interface (GUI) or through API calls. The essence of the entire CaaS platform is an orchestration tool that enables the management of complex container structures. Orchestration tools combine between active containers and enable automated operations. The existing orchestrator in the CaaS framework has a direct impact on the services provided by service users.

Why containers are important:

- With the help of containers application code can be packaged so that we can run it anywhere.
- Helps in promoting portability between multiple platforms.
- Helps in faster release of products.
- Provides increased efficiency to develop and deploy innovative solutions and compose distributed systems.

Why Caas is important :

- Helps developers to develop fully scaled container as well as application deployment.
- Helps in simplifying container management.
- Helps in automating key IT functions like Google Kubernetes and docker.
- Helps in increasing team development velocity which results rapid development and deployment

Containers (container-based virtualization or containerization)

What are containers (container-based virtualization or containerization)?

Containers are a type of software that can virtually package and isolate applications for deployment. Containers package an application's code and dependencies together, letting the application reliably run in all computing environments.

Containers share access to an operating system (OS) kernel without the traditional need for virtual machines (VMs). Containers can be used to run small microservices, larger applications or even lightweight container OSes.

Container technology has its roots in partitioning, dating back to the 1960s, and chroot process isolation developed as part of Unix in the 1970s. Its modern form is expressed in application containerization, such as Docker, and system containerization, such as LXC, part of the Linux Containers Project. Both container styles let an IT team abstract application code from the underlying infrastructure, which simplifies version management and enables portability across various deployment environments.

Developers use containers for development and test environments. IT operations teams might deploy live production IT environments on containers, which can run on bare-metal servers, VMs and the cloud.

What is container management and why is it important?

How containers work

Containers hold the components necessary to run desired software. These components include files, environment variables, dependencies and libraries. The host OS limits the container's access to physical resources, such as CPU, storage and memory, so a single container can't consume all of a host's physical resources.

Container image files are complete, static and executable versions of an application or service that differ from one technology to another. Docker images, for example, are made up of multiple layers. The first layer, the base image, includes all the dependencies needed to execute code in a container. Each image has a readable/writable layer on top of static unchanging layers. Because each container has its own specific container layer that customizes that specific container, underlying image layers can be saved and reused in multiple containers. Likewise, multiple instances of an image can run in a container simultaneously, and new instances can replace failed ones without disrupting the application's operation.

An Open Container Initiative (OCI) image is made up of a manifest, file system layers and configurations. An OCI image has two specifications to operate: a runtime and an image specification. Runtime specifications outline the functioning of a file system bundle, which are files containing all necessary data for performance and runtimes. The image specification contains the information needed to launch an application or service in the OCI container.

The container engine executes images, and many organizations use a container scheduler and orchestration technology, such as Kubernetes, to manage deployments. Containers have high portability because each image includes the dependencies needed to execute the code. For example, container users can execute the same image on an Amazon Web Services (AWS) cloud instance during a test, then an on-premises Dell server for production without changing the application code in the container.

Benefits of containers

Containers offer the following benefits:

- **Efficiency.** Because containers share the same OS kernel as the host, they're more efficient than VMs, which require separate OS instances.
- **Portability.** Containers have better portability than other application hosting technologies. They can move among any systems that share the host OS type without requiring code changes. This encapsulation of the application operating code in the container means there are no guest OS environment variables or library dependencies to manage.
- **Memory, CPU and storage gains.** Proponents of containerization point to gains in efficiency for memory, CPU and storage as key benefits of this approach compared with traditional virtualization. Because containers don't have the overhead required by VMs, such as separate OS instances, it's possible to support many more containers on the same infrastructure. An average physical host

could support dozens of VMs or hundreds of containers. However, in actual operations, the host, container and VM sizes are highly variable and subject to the demands of a specific application or applications.

- **Consistency.** Containers are consistent throughout the application lifecycle. This fosters an agile environment and facilitates new approaches, such as continuous integration and continuous delivery (CI/CD). They also spin up faster than VMs, which is important for distributed applications.

Disadvantages of containers

Containers have the following drawbacks:

- **Lack of isolation.** A potential drawback of containerization is the lack of isolation from the host OS. Because containers share a host OS, security threats have easier access to the entire system when compared with hypervisor-based virtualization. One approach to addressing this security concern has been to create containers from within an OS running on a VM. This approach ensures that, if a security breach occurs at the container level, the attacker can only gain access to that VM's OS, not other VMs or the physical host.
- **Lack of OS flexibility.** In typical deployments, each container must use the same OS as the base OS, whereas hypervisor instances have more flexibility. For example, a container created on a Linux-based host could not run an instance of the Windows Server OS or applications designed to run on Windows Server.
- **Difficulty monitoring visibility.** With up to hundreds or more containers running on a server, it might be difficult to see what's happening in each container.

Various technologies from container and other vendors as well as open source projects are available and under development to address the operational challenges of containers. They include security tracking systems, monitoring systems based on log data as well as orchestrators and schedulers that oversee operations.

Common container uses

Containers are frequently paired with microservices and the cloud but offer benefits to monolithic applications and on-premises data centers as well.

Containers are well-adapted to work with microservices. Each service that makes up the application is packaged in an independently scalable container. For example, a microservices application can be composed of containerized services that generate alerts, log data, handle user identification and provide many other services.

Each service operates on the same OS while staying individually isolated. Each service can scale up and down to respond to demand. Cloud infrastructure is designed for elastic, unlimited scaling.

Traditional monolithic application architectures are designed so that all the code in a program is written in a single executable file. Monolithic applications don't readily scale in the way that distributed applications do, but they can be containerized. For example, the Docker Modernize Traditional Applications program helps users transition monolithic applications to Docker containers as is, with adaptations for better scaling, or via a full rebuild and rearchitecting.

Containers are also used to run applications in different environments. Because all of an application's code and dependencies are included in the container, developers can lift and shift the application without needing to redesign it to work in a new environment. If changes do need to be made, then containerized applications might only need to go through code refactoring, where only small segments of the code need to be restructured.

Container tool and platform providers

Numerous vendors offer container platforms and container management tools, such as cloud services and orchestrators. Docker and Kubernetes are well-known product names in the container technology space, and the technologies underpin many other products.

- Docker is an open source application container platform designed for Linux and, more recently, Windows, Apple and mainframe OSes. Docker uses resource isolation features, such as cgroups and Linux kernels, to create isolated containers. Docker is an eponymous company formed to sell enterprise-supported container hosting and management products. In November 2019, the company sold the Docker Enterprise business to Mirantis.
- The open source container orchestrator Kubernetes, created by Google, has become the de facto standard for container orchestration. It organizes containers into pods on nodes, which are the hosting resources. Kubernetes can automate, deploy, scale, maintain and otherwise operate application containers. A plethora of products are based on Kubernetes with added features and or support, such as Rancher, acquired by SUSE in December 2020; Red Hat OpenShift; and Platform9.

- Microsoft offers containerization technologies, including Hyper-V and Windows Server containers. Both types are created, maintained and operated similarly, as they use the same container images. However, the services differ in terms of the level of isolation. Isolation in Windows Server containers is achieved through namespaces, resource control and other techniques. Hyper-V containers provide isolation through the container instances running inside a lightweight VM, which makes the product more of a system container.
- Amazon Elastic Container Service (ECS) is a cloud computing service in AWS that manages containers and lets developers run applications in the cloud without having to configure an environment for the code to run in. Amazon ECS launches containers through AWS Fargate or Amazon Elastic Compute Cloud. Its features include scheduling, Docker integration, container deployments, container auto-recovery and container security.

Besides these, other container orchestration tools are also available, such as DC/OS from D2iQ -- formerly Mesosphere -- and LXC.

The major cloud vendors all offer diverse containers as a service products as well, including Amazon ECS and Amazon Elastic Kubernetes Service, Google Kubernetes Engine, Microsoft Azure Container Instances, Azure Kubernetes Service and IBM Cloud Kubernetes Service. Containers can also be deployed on public or private cloud infrastructure without the use of dedicated container products from the cloud vendor.

Future of containers

Enterprises have gradually increased their production deployment of container software beyond application development and testing. For most organizations, their focus has shifted to container orchestration and especially Kubernetes, which most vendors now support. As organizations consolidate their processes and toolsets for IT operations, they want more granular control to monitor and secure what's inside containers.

The adoption of container software has expanded across various areas of IT -- from security to networking to storage. Some organizations have deployed stateful applications, such as databases and machine learning (ML) apps on containers and Kubernetes, to ensure consistent management. For example, containerized ML doesn't slow down a machine as much compared to containerless ML, and it doesn't take up as many resources over time.

Some organizations use containers as a part of a broader Agile or DevOps transformations. One example includes containerizing microservices in a CI/CD environment. Another potential use is to proliferate the use of containers and Kubernetes out to the network edge, to remotely deploy and manage software in various locations and on a variety of devices.

It's unlikely that containers will replace server virtualization, as both technologies are complementary to one another. As containers are run in lightweight environments and VMs take more resources, hardware virtualization makes it easier to manage the infrastructure needed for containers.

In addition to the acquisitions noted above, other major vendors have acquired smaller startups to bolster their toolchain offerings. For example, Cisco acquired Portshift -- now Panoptica -- in October 2020, and Red Hat acquired StackRox in January 2021.

Gartner predicts that by 2024, 15% of all enterprise applications will run in a container environment. This is a more than 10% increase since 2020.

Difference between Virtual Machines and Containers

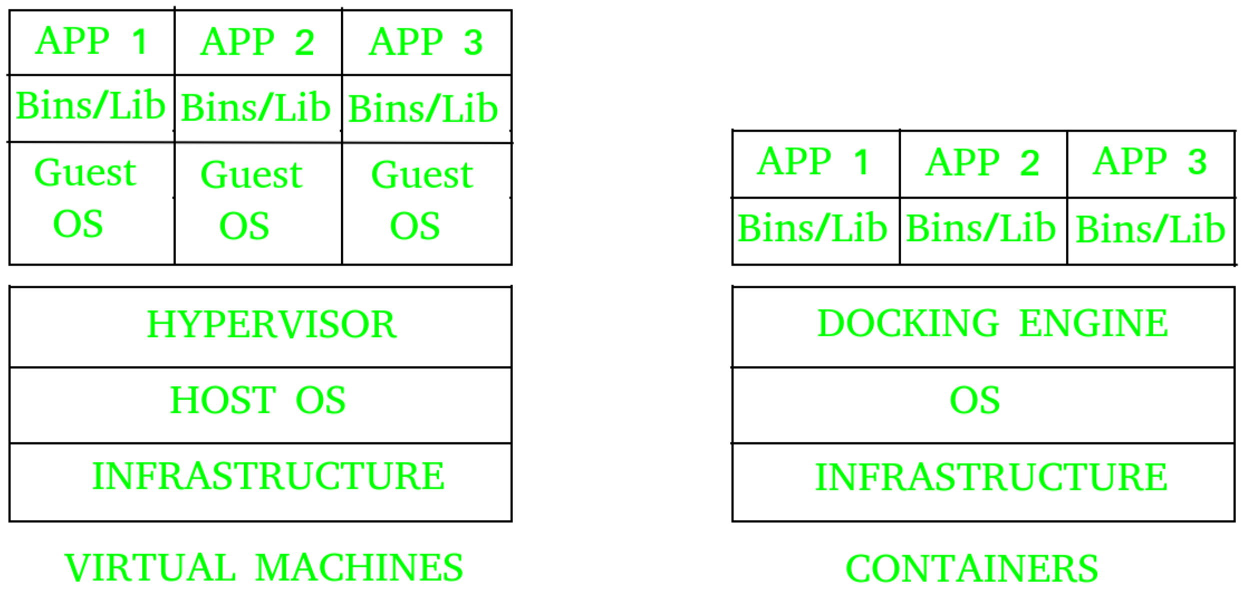
Virtual machines and Containers are two ways of deploying multiple, isolated services on a single platform.

Virtual Machine ***:***

It runs on top of an emulating software called the hypervisor which sits between the hardware and the virtual machine. The hypervisor is the key to enabling virtualization. It manages the sharing of physical resources into virtual machines. Each virtual machine runs its guest operating system. They are less agile and have lower portability than containers.

Container:

It sits on the top of a physical server and its host operating system. They share a common operating system that requires care and feeding for bug fixes and patches. They are more agile and have higher portability than virtual machines.



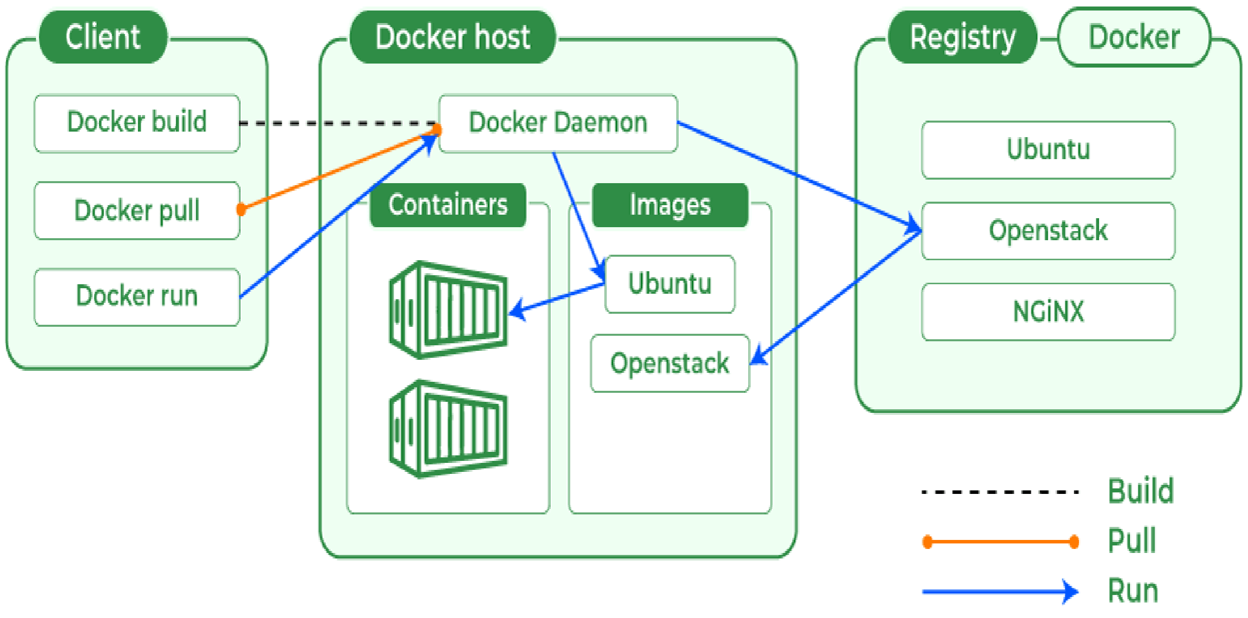
Let’s see the difference between Virtual machines and Containers.

Virtual Machines(VM)	**Containers**
VM is a piece of software that allows you to install other software inside of it so you control it virtually as opposed to installing the software directly on the computer.	While a container is software that allows different functionalities of an application independently.
Applications running on a VM system, or hypervisor, can run different OS.	While applications running in a container environment share a single OS.
VM virtualizes the computer system, meaning its hardware.	While containers virtualize the operating system, or the software only.
VM size is very large, generally in gigabytes.	While the size of the container is very light, generally a few hundred megabytes, though it may vary as per use.
VM takes longer to run than containers, the exact time depending on the underlying hardware.	While containers take far less time to run.

VM uses a lot of system memory.	While containers require very less memory.
VM is more secure, as the underlying hardware isn't shared between processes.	While containers are less secure, as the virtualization is software-based, and memory is shared.
VMs are useful when we require all of the OS resources to run various applications.	While containers are useful when we are required to maximize the running applications using minimal servers.
Examples of Type 1 hypervisors are KVM, Xen, and VMware. Virtualbox is a Type 2 hypervisor	Examples of containers are RancherOS, PhotonOS, and Containers by Docker.

Architecture of Docker

Docker makes use of a client-server architecture. The Docker client talks with the docker daemon which helps in building, running, and distributing the docker containers. The Docker client runs with the daemon on the same system or we can connect the Docker client with the Docker daemon remotely. With the help of REST API over a UNIX socket or a network, the docker client and daemon interact with each other.



What is Docker Daemon?
Docker daemon manages all the services by communicating with other daemons. It manages docker objects such as images, containers, networks, and volumes with the help of the API requests of Docker.

Docker Client
With the help of the docker client, the docker users can interact with the docker. The docker command uses the Docker API. The Docker client can communicate with multiple daemons. When a docker client runs any docker command on the docker terminal then the terminal sends instructions to the daemon. The Docker daemon gets those instructions from the docker client withinside the shape of the command and REST API's request.

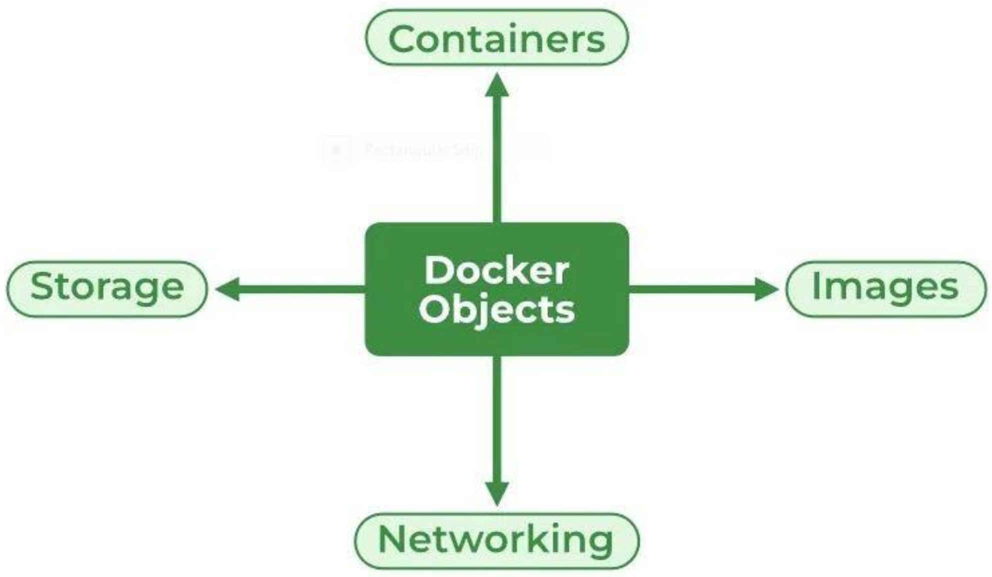
The main objective of the docker client is to provide a way to direct the pull of images from the docker registry and run them on the docker host. The common commands which are used by clients are **docker build, docker pull,** and **docker run.**

Docker Host
A Docker host is a type of machine that is responsible for running more than one container. It comprises the Docker daemon, Images, Containers, Networks, and Storage.

Docker Registry
All the docker images are stored in the docker registry. There is a public registry which is known as a **docker hub** that can be used by anyone. We can run our private registry also. With the help of **docker run** or **docker pull** commands, we can pull the required images from our configured registry. Images are pushed into configured registry with the help of the **docker push** command.

Docker Objects

Whenever we are using a docker, we are creating and use images, containers, volumes, networks, and other objects. Now, we are going to discuss docker objects:



Docker Images

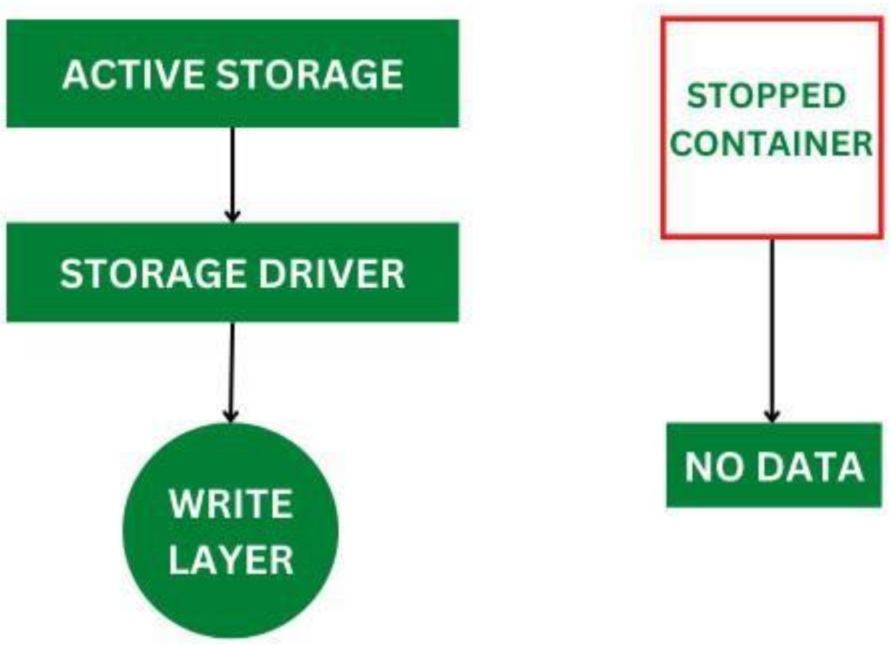
An image contains instructions for creating a docker container. It is just a **read-only template**. It is used to store and ship applications. Images are an important part of the docker experience as they enable collaboration between developers in any way which is not possible earlier.

Docker Containers

Containers are created from docker images as they are ready applications. With the help of Docker API or CLI, we can start, stop, delete, or move a container. A container can access only those resources which are defined in the image unless additional access is defined during the building of an image in the container.

Docker Storage

We can store data within the writable layer of the container but it requires a storage driver. Storage driver controls and manages the images and containers on our docker host.



Types of Docker Storage

1. **Data Volumes:** Data Volumes can be mounted directly into the filesystem of the container and are essentially directories or files on the Docker Host filesystem.
2. **Volume Container**: In order to maintain the state of the containers (data) produced by the running container, Docker volumes file systems are mounted on Docker containers. independent container life cycle, the volumes are stored on the host. This makes it simple for users to exchange file systems among containers and backup data.
3. **Directory Mounts:** A host directory that is mounted as a volume in your container might be specified.
4. **Storage Plugins:** Docker volume plugins enable us to integrate the Docker containers with external volumes like Amazon EBS by this we can maintain the state of the container.

Docker Networking

Docker networking provides complete isolation for docker containers. It means a user can link a docker container to many networks. It requires very less OS instances to run the workload.

Types of Docker Network

1. **Bridge:** It is the default network driver. We can use this when different containers communicate with the same docker host.

2. **Host:** When you don’t need any isolation between the container and host then it is used.
3. **Overlay:** For communication with each other, it will enable the swarm services.
4. **None:** It disables all networking.
5. **macvlan:** This network assigns MAC(Media Access control) address to the containers which look like a physical address.

Understanding Docker Containers

Container platform technology is packaged items of software applications in virtual containers. Each software container is a standardized executable component. It combines the core OS needed to run it with every dependency and library as an “image”. They are portable and reusable in any operating system. They have separated execution environments from each other.

Launched in 2013, Docker has recently become the de facto industry standard open-source software for containers. It is increasingly getting more popular alongside the integration of companies using cloud-native and hybrid solutions. It is also characterized as making containerizing easier and simpler. With a few commands, developers can create a container environment all in one place.

Usually, the Docker container images become containers when run by Docker Engine. It is available for every Linux and Windows-based OS and application, enabling containerization to run natively regardless of the environment or infrastructure. This seeks to put an end to the “dependency” tragedy and eliminate the curse of “it doesn’t work on my laptop!.”

Benefits of Docker Containers

The main goals and benefits of Docker containers are to allow developers to improve the following processes:

- Shortening the delay of writing code until running it.
- Flexibility in deploying, cloning, and moving the workload.
- Unifying a one-way streamline for shipping, running, and testing.
- Reducing the need for server resources and costs with better performance.
- Simplifying infrastructure maintenance, update, and support.

- Increasing security strength by containing isolation capabilities.

Common Docker Usage

We cannot imagine that Docker containers as a giant project have been developed, for instance, just to enable a handful of Linux users around the world to run Windows applications in Linux OS! That is not really the case or function. There are some other solutions for that use case. Docker mainly exists to serve software developing projects and increase the consistent delivery of applications.

We cannot define every detail of Docker’s common usage. Instead, we will look at the most major use cases which are mainly related to better delivery of (CI/CD) and enhancement of workflows. These uses include but are not limited to:

- Simplifying and standardizing the software development lifecycle (SDLC) by creating a worldwide unified environment to integrate the processes of providing your applications and services.
- Enabling developers to easily share local written code with colleagues and testers for peer review and executing automated and manual tests smoothly.
- Facilitating fixing bugs and composability thanks to easy movements between development and test environments, in addition to the final customer.
- Creating dynamic and high portability workloads, from local laptops, on-prime servers, virtual machines, on the cloud, on local data centers, and hybrid platforms.
- Scaling up and down workloads and applications in real-time as per the needs of the business and to run more than one workload in high-density environments.
- Facilitating the orchestration of the deployment of many items. The Kubernetes orchestration software can organize them in clusters.

How Does Docker Architecture Work?

Docker is a manifestation of the new generation of virtualization. It uses a client-server architecture to call with the Docker daemon which can run on the same OS or be connected remotely. The heavy workload of building and running containers falls on the shoulders of the Docker engine. And the user interacting is with the Docker client. They communicate through a REST API either on UNIX sockets or a network interface.

In order to list all the Docker Objects, let’s make the following list:

- **Docker Client CLI**: An interface for users to interact with Docker.
- **Docker Daemon and Engine**: A process for building and running the containers.
- **Docker Registry**: To store the Docker images in public or private Docker Hub.

- **Docker Volumes**: The persisting data of Docker containers.
- **Docker Image**: A read-only object has the instructions to create a Docker container.
- **Docker File**: A simple text file contains the command-line of building Docker images.
- **Docker Container**: The final runnable Docker image.

Difference between Docker Images and Containers

Docker Image	**Docker Container**
It is a blueprint of the Container.	It is an instance of the Image.
Image is a logical entity.	The container is a real-world entity.
Images are created only once.	Containers are created any number of times using an image.
Images are immutable. One cannot attach volumes and networks.	Containers change only if the old image is deleted and a new one is used to build the container. One can attach volumes, networks, etc.
Images do not require computing resources to work.	Containers require computing resources to run as they run with a Docker Virtual Machine.
To make a docker image, you have to write a script in a Dockerfile.	To make a container from an image, you have to run the “docker run <image>” command
Docker Images are used to package up applications and pre-configured server environments.	Containers use server information and a file system provided by an image in order to operate.
Images can be shared on Docker Hub.	It makes no sense in sharing a running entity, always docker images are shared.

Docker Image	Docker Container
There is no such thing as a running state of a Docker Image.	Containers use RAM when created and in a running state.
An image must not refer to any state to remove the image.	A container must be in a running state to remove it.
One cannot connect to the images as these images are like snapshots.	In this, one cannot connect them and execute the commands.
Sharing of Docker Images is possible.	Sharing of containers is not possible directly.
It has multiple read-only layers.	It has a single writable layer.
These image templates can exist in isolation.	These containers cannot exist without images.

Introduction to Kubernetes (K8S)

Kubernetes is an open-source Container Management tool that automates container deployment, container scaling, descaling, and container load balancing (also called a container orchestration tool). It is written in Golang and has a vast community because it was first developed by Google and later donated to CNCF (Cloud Native Computing Foundation). Kubernetes can group ‘n’ number of containers into one logical unit for managing and deploying them easily. It works brilliantly with all cloud vendors i.e. public, hybrid, and on-premises.

Kubernetes is an open-source platform that manages Docker containers in the form of a cluster. Along with the automated deployment and scaling of containers, it provides healing by automatically restarting failed containers and rescheduling them when their hosts die. This capability improves the application’s availability.

Features of Kubernetes:

1. **Automated Scheduling**– Kubernetes provides an advanced scheduler to launch containers on cluster nodes. It performs resource optimization.
2. **Self-Healing Capabilities**– It provides rescheduling, replacing, and restarting the containers which are dead.
3. **Automated Rollouts and Rollbacks**– It supports rollouts and rollbacks for the desired state of the containerized application.
4. **Horizontal Scaling and Load Balancing**– Kubernetes can scale up and scale down the application as per the requirements.
5. **Resource Utilization**– Kubernetes provides resource utilization monitoring and optimization, ensuring containers are using their resources efficiently.
6. **Support for multiple clouds and hybrid clouds**– Kubernetes can be deployed on different cloud platforms and run containerized applications across multiple clouds.
7. **Extensibility**– Kubernetes is very extensible and can be extended with custom plugins and controllers.
8. **Community Support-** Kubernetes has a large and active community with frequent updates, bug fixes, and new features being added.

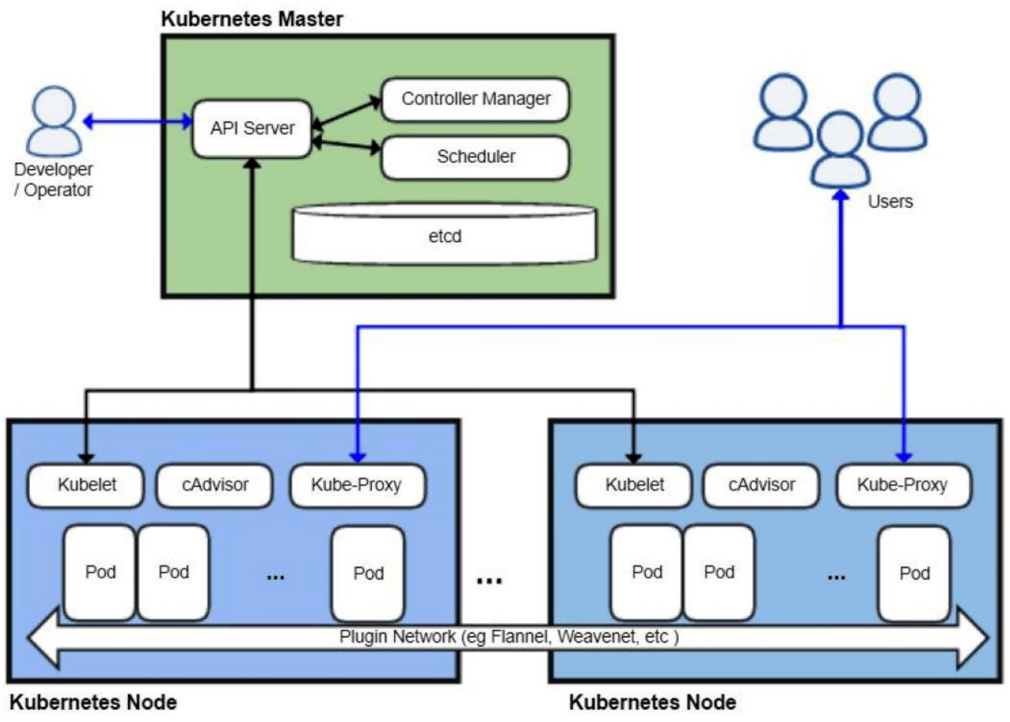
Kubernetes Vs Docker:

Docker Swarm	**Kubernetes**
Developed by Docker Inc.	Developed by Google, now managed by CNCF
No Auto-Scaling	Auto-Scaling
Does Auto Load-Balancing	Manually configure your Load-Balancing settings
It performs rolling updates to containers straightaway	K8S performs rolling updates to Pods as a whole
Share storage volumes with any other containers	Share storage volumes between multiple containers inside the same pods
It uses 3rd party tools like ELK	K8S provides in-built tools for logging and monitoring

Kubernetes – Architecture

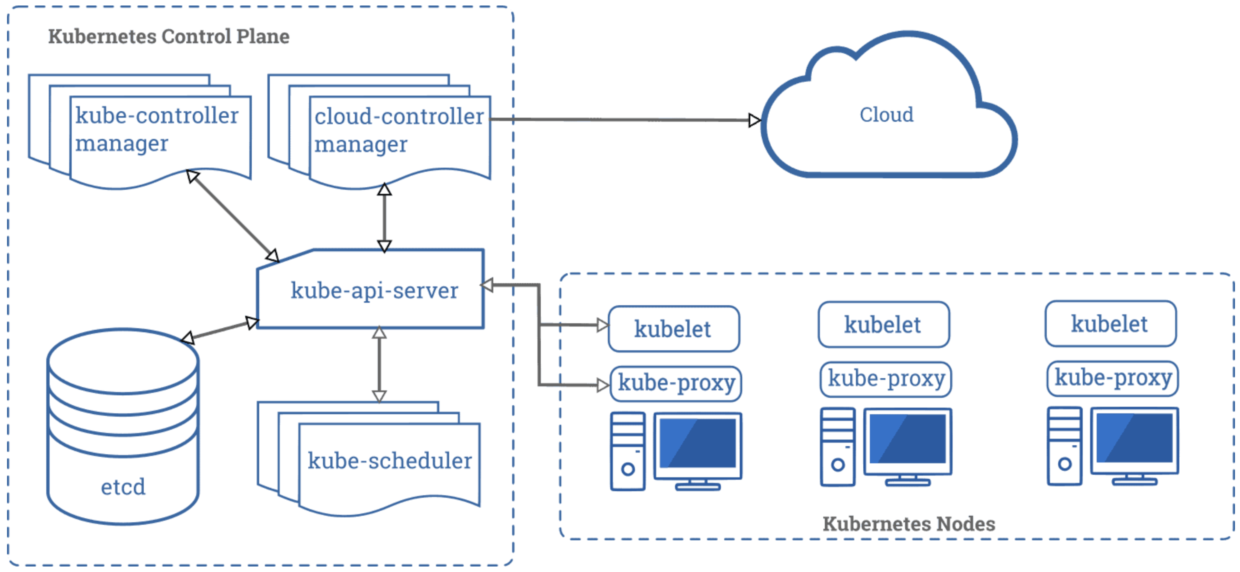
Kubernetes is an open source container deployment and management platform. It offers container orchestration, a container runtime, container-centric infrastructure orchestration, load balancing, self-healing mechanisms, and service discovery. Kubernetes architecture, also sometimes called Kubernetes application deployment architecture or Kubernetes client server architecture, is used to compose, scale, deploy, and manage application containers across host clusters.

An environment running Kubernetes consists of the following basic components: a control plane (Kubernetes control plane), a distributed key-value storage system for keeping the cluster state consistent (etcd), and cluster nodes (Kubelets, also called worker nodes or minions).



1) In the Kubernetes architecture diagram above you can see, there is one master and multiple nodes.
2) The Master node communicates with Worker nodes using Kube API-server to kubelet communication.
3) In the Worker node, there can be one or more pods and pods can contain one or more containers.
4) Containers can be deployed using the image also can be deployed externally by the user.

Kubernetes Architecture Components



1. Kubernetes Master Node

In Kubernetes (k8s), a master node is the **control plane component** responsible for **managing the cluster**. It coordinates and schedules tasks, maintains cluster state, and monitors node health. It includes components like API server, scheduler, and controller manager, ensuring overall cluster functionality and orchestration of containerized applications.

Master Node Components:

1) Kube API server handles administrative tasks on the master node. Users send REST commands in YAML/JSON to the API server, which processes and executes them. The Kube API server acts as the front end of the Kubernetes control plane.

2) etcd, a distributed key-value store, maintains the cluster state and configuration details like subnets and config maps in Kubernetes' database. It's where Kubernetes stores its information.

3) Kube-scheduler assigns tasks to worker nodes and manages new requests from the API Server, ensuring they are directed to healthy nodes.

4) Kube Controller Manager task is to retrieve the desired state from the API Server. If the desired state does not match the current state of the object, corrective steps are taken by the control loop to align the current state with the desired state.

There are different types of control manager in Kubernetes architecture:

- **Node Manager:** It oversees nodes, creating new ones in case of unavailability or destruction.
- **Replication Controller:** It ensures the desired container count is maintained within the replication group.

- **Endpoints Controller:** This controller populates the endpoints object, connecting Services & Pods.

2. Kubernetes Worker Node

Worker nodes in a cluster are machines or servers running applications, controlled by the Kubernetes master. Multiple nodes connect to the master. On each node, multiple pods and containers operate.

Components of Worker Nodes:
1) Kubelet, an agent on each node, communicates with the master. It ensures pod containers' health, executing tasks like deploying or destroying containers, reporting back to the Master.
2) Kube-proxy enables worker node communication, managing network rules. It ensures rules are set for containers to communicate across nodes.
3) A **Kubernetes pod** is a set of containers on a single host, sharing storage and network. It includes specifications for container execution, enabling easy inter-container communication.
4) Container Runtime, responsible for container execution, supports multiple runtimes: Docker, containers.

Application of Kubernetes

- Microservices architecture: Kubernetes is well-suited for managing microservices architectures, which involve breaking down complex applications into smaller, modular components that can be independently deployed and managed.
- Cloud-native development: Kubernetes is a key component of cloud-native development, which involves building applications that are designed to run on cloud infrastructure and take advantage of the scalability, flexibility, and resilience of the cloud.
- Continuous integration and delivery: Kubernetes integrates well with CI/CD pipelines, making it easier to automate the deployment process and roll out new versions of your application with minimal downtime.
- Hybrid and multi-cloud deployments: Kubernetes provides a consistent deployment and management experience across different cloud providers, on-premise data centers, and even developer laptops, making it easier to build and manage hybrid and multi-cloud deployments.
- High-performance computing: Kubernetes can be used to manage high-performance computing workloads, such as scientific simulations, machine learning, and big data processing.
- Edge computing: Kubernetes is also being used in edge computing applications, where it can be used to manage containerized applications running on edge devices such as IoT devices or network appliances.

Kubernetes vs Docker

Kubernetes	**Docker**
Kubernetes is an open-source platform used for maintaining and deploying a group of containers	Docker is a tool that is used to automate the deployment of applications in lightweight containers so that applications can work efficiently in different environments.
In practice, Kubernetes is most commonly used alongside Docker for better control and implantation of containerized applications.	With Docker, multiple containers run on the same hardware much more efficiently than the VM environment & productivity of Docker is extremely high.
Applications are deployed as a combination of pods, Deployment, and services.	Apps are deployed in the form of services.
It supports auto-scaling of the container in a cluster.	Docker does not support auto-scaling.
The health check is of two kinds: liveness and readiness.	Health checks are limited to service.
Hard to set up and configure.	Docker's setup and installation are easy.
It does not have extensive documentation but is quite less than Docker. But it does include everything from installation to deployment.	Docker documentation is more effective, more extensive, and has even more capabilities & it includes everything from installation to deployment & quick-start instructions as well as a more detailed tutorial.
Kubernetes installation is provided to be quite difficult than Docker and even the command for Kubernetes is quite more complex than Docker.	Docker installation is quite easier, by using fewer commands you can install Docker in your virtual machine or even on the cloud.
Azure, buffer, intel, Evernote, and Shopify Using Kubernetes.	Google, Amazon, ADP, VISA, citizens bank, and MetLife companies using Docker.