What is Docker?

Docker is an **open-source centralized platform designed** to create, deploy, and run applications. Docker uses **container** on the host's operating system to run applications. It allows applications to use the same **Linux kernel** as a system on the host computer, rather than creating a whole virtual operating system. Containers ensure that our application works in any environment like development, test, or production.Unmute

Docker includes components such as **Docker client, Docker server, Docker machine, Docker hub, Docker composes,** etc.

Let's understand the Docker containers and virtual machine.

Docker Containers

Docker containers are the **lightweight** alternatives of the virtual machine. It allows developers to package up the application with all its libraries and dependencies, and ship it as a single package. The advantage of using a docker container is that you don't need to allocate any RAM and disk space for the applications. It automatically generates storage and space according to the application requirement.

Virtual Machine

A virtual machine is a software that allows us to install and use other operating systems (Windows, Linux, and Debian) simultaneously on our machine. The operating system in which virtual machine runs are called virtualized operating systems. These virtualized operating systems can run programs and preforms tasks that we perform in a real operating system.

Containers Vs. Virtual Machine

|  |  |
| --- | --- |
| **Containers** | **Virtual Machine** |
| Integration in a container is faster and cheap. | Integration in virtual is slow and costly. |
| No wastage of memory. | Wastage of memory. |
| It uses the same kernel, but different distribution. | It uses multiple independent operating systems. |

Why Docker?

Docker is designed to benefit both the Developer and System Administrator. There are the following reasons to use Docker -

* Docker allows us to easily install and run software without worrying about setup or dependencies.
* Developers use Docker to eliminate machine problems, i.e. "**but code is worked on my laptop**." when working on code together with co-workers.
* Operators use Docker to run and manage apps in isolated containers for better compute density.
* Enterprises use Docker to securely built agile software delivery pipelines to ship new application features faster and more securely.
* Since docker is not only used for the deployment, but it is also a great platform for development, that's why we can efficiently increase our customer's satisfaction.

Advantages of Docker

There are the following advantages of Docker -

* It runs the container in seconds instead of minutes.
* It uses less memory.
* It provides lightweight virtualization.
* It does not a require full operating system to run applications.
* It uses application dependencies to reduce the risk.
* Docker allows you to use a remote repository to share your container with others.
* It provides continuous deployment and testing environment.

Disadvantages of Docker

There are the following disadvantages of Docker -

* It increases complexity due to an additional layer.
* In Docker, it is difficult to manage large amount of containers.
* Some features such as container self -registration, containers self-inspects, copying files form host to the container, and more are missing in the Docker.
* Docker is not a good solution for applications that require rich graphical interface.
* Docker provides cross-platform compatibility means if an application is designed to run in a Docker container on Windows, then it can't run on Linux or vice versa.

### Important Terminologies in Docker

#### 1. Docker Image

* It is a file, comprised of multiple layers, used to execute code in a Docker container.
* They are a set of instructions used to create docker containers.

#### 2. Docker Container

* It is a runtime instance of an image.
* Allows developers to package applications with all parts needed such as libraries and other dependencies.

#### 3. Docker file

* It is a text document that contains necessary commands which on execution helps assemble a Docker Image.
* Docker image is created using a Docker file.

#### 4. Docker Engine

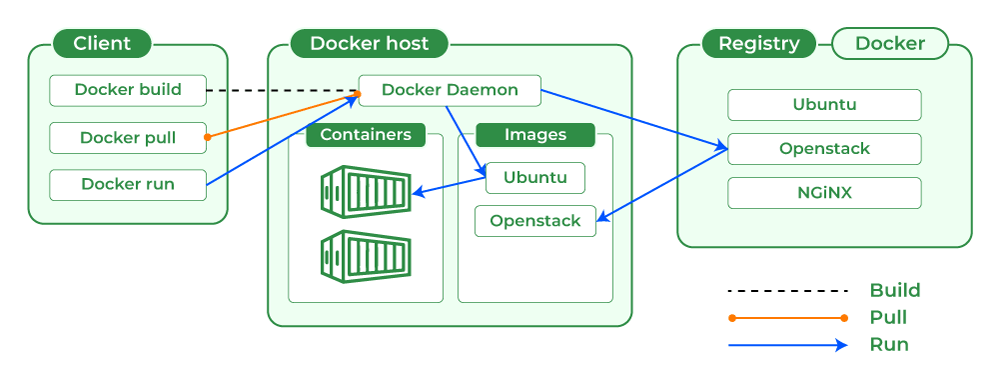
* The software that hosts the containers is named Docker Engine.
* Docker Engine is a client-server based application
* The docker engine has **3 main** components:
  + **Server**: It is responsible for creating and managing Docker images, containers, networks, and volumes on the Docker. It is referred to as a daemon process.
  + **REST API**: It specifies how the applications can interact with the Server and instructs it what to do.
  + **Client**: The Client is a docker command-line interface (CLI), that allows us to interact with Docker using the docker commands.

#### 5. Docker Hub

* Docker Hub is the official online repository where you can find other Docker Images that are available for use.
* It makes it easy to find, manage, and share container images with others.

# **Architecture of Docker**

Docker makes use of a client-server architecture. The Docker client talk with the docker daemon which helps in building, running, and distributing the docker containers. The Docker client run 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.



### 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 which is responsible for running more than one container. It comprises of 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.

**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:**

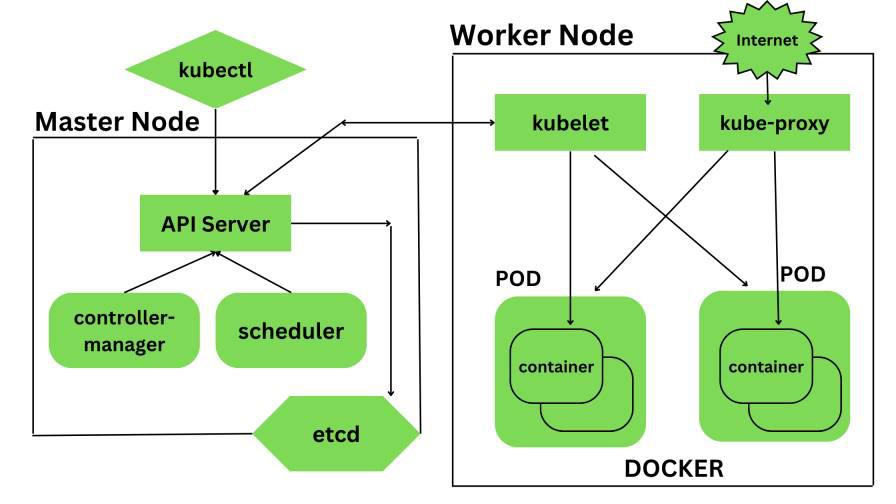
1. **Automated Scheduling**– Kubernetes provides an advanced scheduler to launch containers on cluster nodes. It performs resource optimization.
2. **Self-Healing Capabilities**– It provides rescheduling, replacing, and restarting the containers which are dead.
3. **Automated Rollouts and Rollbacks**– 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 |

**Architecture of Kubernetes**

Kubernetes follows the client-server architecture where we have the master installed on one machine and the node on separate Linux machines. It follows the master-slave model, which uses a master to manage Docker containers across multiple Kubernetes nodes. A master and its controlled nodes(worker nodes) constitute a **“Kubernetes cluster”**. A developer can deploy an application in the docker containers with the assistance of the Kubernetes master.



*Architecture of Kubernetes*

**1. Kubernetes- Master Node Components –**

Kubernetes master is responsible for managing the entire cluster, coordinates all activities inside the cluster, and communicates with the worker nodes to keep the Kubernetes and your application running. This is the entry point of all administrative tasks. When we install Kubernetes on our system we have four primary components of Kubernetes Master that will get installed. The components of the Kubernetes Master node are:

**a.) API Server**– The API server is the entry point for all the REST commands used to control the cluster. All the administrative tasks are done by the API server within the master node. If we want to create, delete, update or display in Kubernetes object it has to go through this API server.API server validates and configures the API objects such as ports, services, replication, controllers, and deployments and it is responsible for exposing APIs for every operation. We can interact with these APIs using a tool called **kubectl**. *‘kubectl’ is a very tiny go language binary that basically talks to the API server to perform any operations that we issue from the command line. It is a command-line interface for running commands against Kubernetes clusters*

**b.) Scheduler**– It is a service in the master responsible for distributing the workload. It is responsible for tracking the utilization of the working load of each worker node and then placing the workload on which resources are available and can accept the workload. The scheduler is responsible for scheduling pods across available nodes depending on the constraints you mention in the configuration file it schedules these pods accordingly. The scheduler is responsible for workload utilization and allocating the pod to the new node.

**c.) Controller Manager**– Also known as controllers. It is a daemon that runs in a non terminating loop and is responsible for collecting and sending information to the API server. It regulates the Kubernetes cluster by performing lifestyle functions such as namespace creation and lifecycle event garbage collections, terminated pod garbage collection, cascading deleted garbage collection, node garbage collection, and many more. Basically, the controller watches the desired state of the cluster if the current state of the cluster does not meet the desired state then the control loop takes the corrective steps to make sure that the current state is the same as that of the desired state. The key controllers are the replication controller, endpoint controller, namespace controller, and service account, controller. So in this way controllers are responsible for the overall health of the entire cluster by ensuring that nodes are up and running all the time and correct pods are running as mentioned in the specs file.

**d.) etc**– It is a distributed key-value lightweight database. In Kubernetes, it is a central database for storing the current cluster state at any point in time and is also used to store the configuration details such as subnets, config maps, etc. It is written in the Go programming language.

**2. Kubernetes- Worker Node Components –**

Kubernetes Worker node contains all the necessary services to manage the networking between the containers, communicate with the master node, and assign resources to the containers scheduled. The components of the Kubernetes Worker node are:

**a.) Kubelet**– It is a primary node agent which communicates with the master node and executes on each worker node inside the cluster. It gets the pod specifications through the API server and executes the container associated with the pods and ensures that the containers described in the pods are running and healthy. If kubelet notices any issues with the pods running on the worker nodes then it tries to restart the pod on the same node. If the issue is with the worker node itself then the Kubernetes master node detects the node failure and decides to recreate the pods on the other healthy node.

**b.) Kube-Proxy**– It is the core networking component inside the Kubernetes cluster. It is responsible for maintaining the entire network configuration. Kube-Proxy maintains the distributed network across all the nodes, pods, and containers and exposes the services across the outside world. It acts as a network proxy and load balancer for a service on a single worker node and manages the network routing for TCP and UDP packets. It listens to the API server for each service endpoint creation and deletion so for each service endpoint it sets up the route so that you can reach it.

**c.) Pods**– A pod is a group of containers that are deployed together on the same host. With the help of pods, we can deploy multiple dependent containers together so it acts as a wrapper around these containers so we can interact and manage these containers primarily through pods.

**d.) Docker**– Docker is the containerization platform that is used to package your application and all its dependencies together in the form of containers to make sure that your application works seamlessly in any environment which can be development or test or production. Docker is a tool designed to make it easier to create, deploy, and run applications by using containers. Docker is the world’s leading software container platform. It was launched in 2013 by a company called Dot cloud. It is written in the Go language. It has been just six years since Docker was launched yet communities have already shifted to it from VMs. Docker is designed to benefit both developers and system administrators making it a part of many DevOps toolchains. Developers can write code without worrying about the testing and production environment. Sysadmins need not worry about infrastructure as Docker can easily scale up and scale down the number of systems. Docker comes into play at the deployment stage of the software development cycle.

