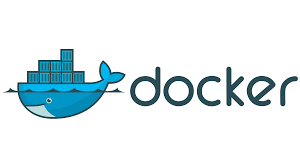
DOCKER

**What is Docker?**

****

Docker is an open-source platform that allows you to create, deploy, and run applications inside containers.

Containers are lightweight, portable, and self-contained environments that enable you to package and distribute your applications with all their dependencies.

Docker is a set of platform as a service (PaaS) products that uses OS-level virtualization to deliver software in packages called containers.

Containers are isolated from one another and bundle their own software, libraries and configuration files, they can communicate with each other through well-defined channels. All containers are run by a single operating system Kernel and therefore use fewer resources than virtual machines.

**History:**

The first edition of Docker was released in **2013**.

In 2013, Docker was first introduced by Solomon Hykes at PyCon.

Docker was originally a tool developed to simplify the deployment process of applications, by using containers to package software and its dependencies.

In March 2013, the first version of Docker was released as an open-source project.

Docker is developed using the **GO** programming language.

Looking at the rich set of functionality Docker has got to offer, it’s been widely accepted by some of the world’s leading organizations and universities, such as **Visa, PayPal, Cornell University and Indiana University** (just to name a few) to run and manage their applications using Docker.

Today, Docker is one of the most popular containerization technologies in use, with a large and active community of developers and users. It has become an essential tool for software development, deployment, and management.

**Docker Editions:**

Docker is available in 2 different editions, as listed below:

* **Community Edition (CE)**
* **Enterprise Edition (EE)**

The **Community Edition** is suitable for individual developers and small teams. It offers limited functionality, in comparison to the Enterprise Edition.

The **Enterprise Edition,**on the other hand, is suitable for large teams and for using Docker in production environments.

The Enterprise Edition is further categorized into three different editions, as listed below:

* **Basic Edition**
* **Standard Edition**
* **Advanced Edition**

**Why Docker?**

****

Docker is a popular platform for containerizing applications. It provides a way to package an application and all its dependencies into a single container, which can then be run on any system that has Docker installed. Here are some reasons why Docker is a popular choice for containerization:

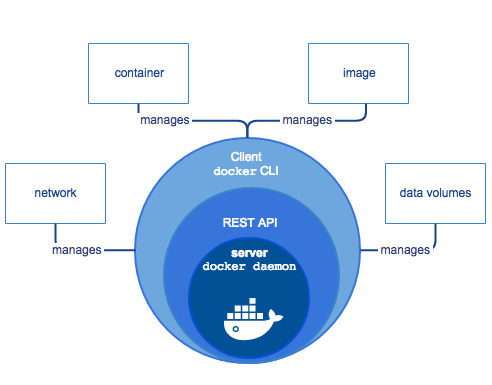
* **Portability:** Docker containers can run on any system that supports Docker, regardless of the underlying infrastructure or operating system. This makes it easy to move applications between different environments, such as development, testing, and production.
* **Consistency:** Docker containers provide a consistent environment for running applications, regardless of the host system. This helps to ensure that applications behave the same way in different environments.
* **Isolation:** Docker containers provide a level of isolation between applications and the host system, which helps to prevent conflicts and ensure that applications run reliably.
* **Efficiency:** Docker containers are lightweight and efficient, which means that they can be created and destroyed quickly, and multiple containers can run on the same system without consuming excessive resources.
* **Scalability:** Docker containers can be easily scaled up or down to meet changing demands, making it easy to deploy and manage applications in a dynamic environment.

**Core Components of Docker:**

Docker Engine is one of the core components of Docker. It is responsible for the overall functioning of the Docker platform.

Docker Engine is a client-server based application and consists of 3 main components.

1. Server
2. REST API
3. Client

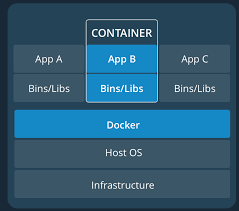


The **Server** runs a daemon known as **dockerd** **(Docker Daemon)**, which is nothing but a process. It is responsible for creating and managing Docker Images, Containers, Networks and Volumes on the Docker platform.

The **REST API** specifies how the applications can interact with the Server, and instruct it to get their job done.

The **Client** is nothing but a command line interface, that allows users to interact with Docker using the commands.

**What are Containers?**

****

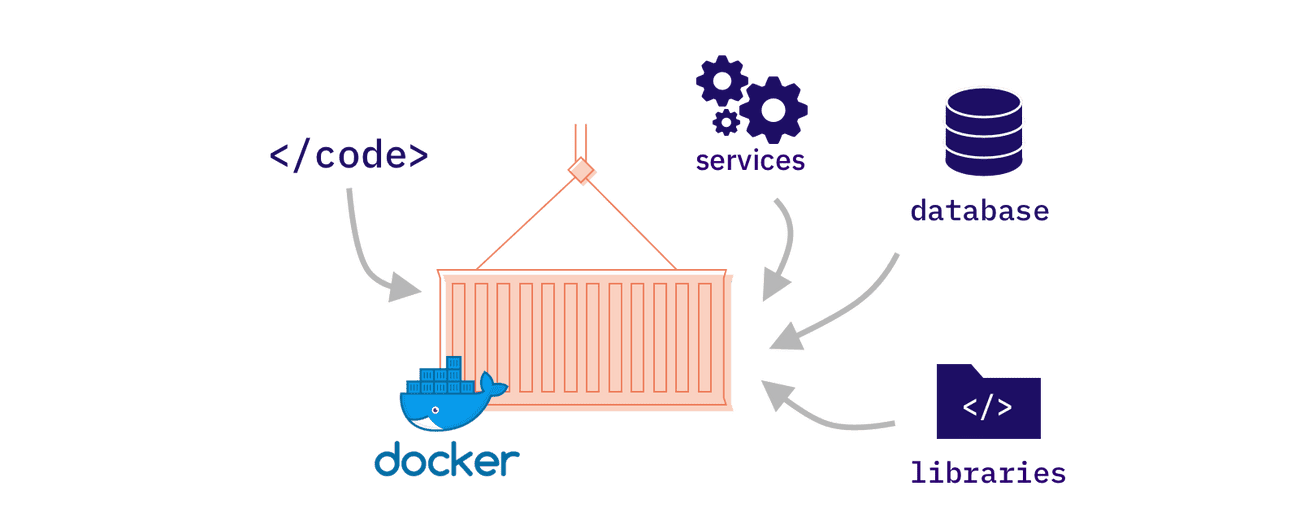
A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another.

Containers are isolated from each other and from the host system, using the concept of namespaces. Namespaces provide a way to isolate system resources, such as network interfaces, processes, and file systems.

Containers are a key component of the Docker platform, providing a flexible, efficient, and scalable way to build, run, and manage software applications.

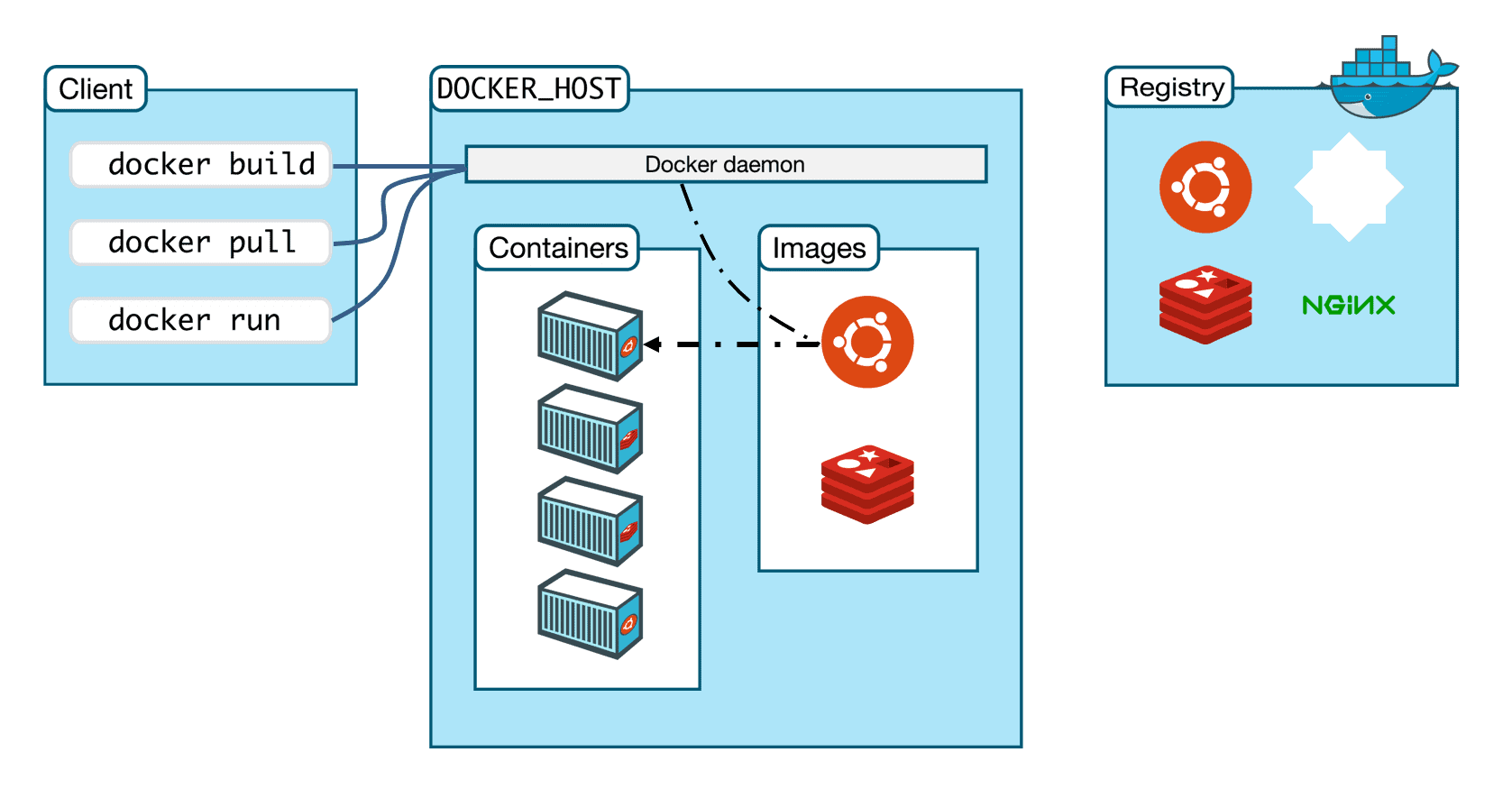
Docker containers are created from Docker images, which are essentially snapshots of a container at a specific point in time. Each container has its own file system, networking, and processes, and can be started, stopped, or deleted independently of other containers on the same system.

A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.



Containers isolate software from its environment and ensure that it works uniformly despite differences for instance between development and staging.

**Docker Architecture:**

****

Docker uses a client-server architecture, where the Docker client communicates with the Docker daemon (also known as the Docker engine) to manage containers, images, networks, and other resources. The Docker client can run on the same host as the Docker daemon, or on a remote host that communicates with the Docker daemon over a network.

Here's a detailed breakdown of the Docker architecture:

**Docker Client and Server:**

Docker client is accessed from the terminal and a Docker Host runs the Docker Daemon and registry.

A user can build Docker Images and run Docker Containers by passing commands from the Docker Client to Docker server.

**Docker client:** The Docker client is a command-line tool that allows users to interact with the Docker daemon using a series of commands.

The client sends commands to the daemon, which then executes them and returns the results to the client.

The Docker Client can run on the same host as the Docker daemon or on a remote host that communicates with the daemon over a network.

**Docker daemon:** The Docker daemon is a background process that runs on the host machine and manages the lifecycle of containers, images, networks, and other resources.

It listens for commands from the Docker Client and executes them using the container runtime, which is responsible for starting, stopping, and managing containers.

**Container runtime:**

The container runtime is a low-level component that is responsible for starting, stopping, and managing containers.

Docker supports multiple container runtimes, including runc, containerd, and CRI-O.

The container runtime works with the Docker daemon to manage container lifecycles and resources, such as namespaces and file systems.

**Docker Registry:**

It is an open source server-side service used for hosting and disturbing images.

Docker also has its own default registry called Docker Hub.

Here, images can be stored in either public or private repositories.

Pull and Push are the commands used by users in order to interact with a Docker Registry.

**Docker image:**

A Docker image is a pre-built package that contains everything needed to run a software application, including the code, runtime, system tools, libraries, and settings.

Images are built from a Dockerfile, which specifies how to configure and package the application.

Images can be stored in a Docker registry, such as Docker Hub, and can be shared and distributed to other users.

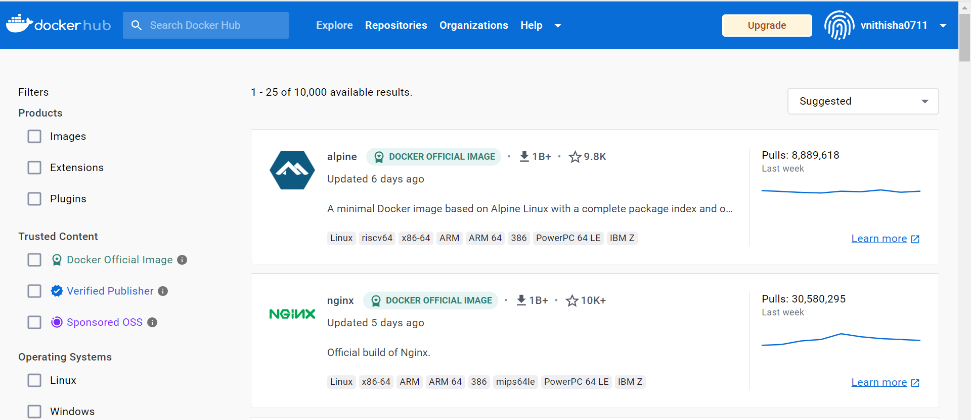
**Docker container:**

A Docker container is a lightweight and standalone executable package that includes the code, runtime, system tools, libraries, and settings needed to run a software application.

Containers are isolated from each other and from the host system, using the concept of namespaces.

Containers can be started, stopped, or deleted independently of other containers on the same system.

**What is Docker Hub?**

****

Docker Hub is the official online repository where you could find all the Docker Images that are available for us to use.

Docker Hub also allows us to store and distribute our custom images as well if we wish to do so. We could also make them either public or private, based on our requirements.

Docker Hub is a service provided by Docker for finding and sharing container images with your team.

It provides the following major features:

**Repositories:** Push and pull container images.

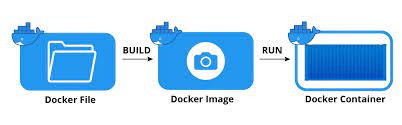
**Teams & Organisations:** Manage access to private repositories of container images.

Docker Hub is a cloud-based registry where you can store and share Docker images. It provides a central location for storing and distributing Docker images, making it easy for developers to access and use them. Docker Hub is also a community where developers can share their images with others and collaborate on open-source projects.

**Note:** Free users are only allowed to keep one Docker Image as private. If we wish to keep more than one Docker Image as private, we need to subscribe to a paid subscription plan.

**What is Dockerfile?**

A Dockerfile is a text file that contains a set of instructions for building a Docker image. It is used to automate the process of building a Docker image by specifying the software dependencies, configuration settings, and other parameters that are required to create the image.



A Dockerfile consists of a series of commands, each of which corresponds to a specific action that is performed during the image building process. The commands in a Dockerfile are executed in order, and each command generates a new intermediate image that is used as the basis for the next command.

Some common commands that can be used in a Dockerfile include:

**FROM:** specifies the base image to use as the starting point for the image being built.

**RUN:** executes a command in the container during the build process.

**COPY:** copies files or directories from the host machine to the container.

**ADD:** copies files or directories from the host machine to the container, with additional support for URLs and compressed files.

**EXPOSE:** specifies the network ports that the container should listen on.

**CMD:** specifies the default command to be executed when the container is run.

**What is Docker Image?**

A Docker image is a packaged and distributable file that contains everything needed to run a software application, including the application's code, runtime, libraries, environment variables, and any other dependencies. Docker images are built from a set of instructions contained in a Dockerfile, which specify how to configure and package the application.

A Docker image is essentially a snapshot of a container at a specific point in time. Once an image is created, it can be stored in a registry, such as Docker Hub, where it can be easily shared and distributed to other users.

Docker images are designed to be lightweight, portable, and scalable. They can be easily moved between different environments, such as development, testing, and production, without any changes to the underlying application or infrastructure.

In simple terms, a Docker Image is a template that contains the application, and all the dependencies required to run that application on Docker.

On the other hand, as stated earlier, a Docker Container is a logical entity. In more precise terms, it is a running instance of the Docker Image.

**Steps:**

The process of installing and using Docker on Linux in Amazon Web Services (AWS).

Before we begin, you will need:

* An AWS account: Sign up for an AWS account if you haven't already.
* An instance running Linux: Launch an instance running Linux in AWS.

Installation:

To install Docker on Linux in AWS, follow these steps:

Connect to your instance: Use SSH to connect to your instance.

Update the package manager: Run the following command to update the package manager:

***$ sudo yum update***

Install Docker: Run the following command to install Docker:

***$ sudo yum install docker***

Start Docker: Run the following command to start Docker:

***$ sudo service docker start***

To install Docker, follow these steps:

Check the system requirements: Docker requires a 64-bit version of Windows, macOS, or Linux with a compatible kernel version. Check the Docker documentation for the specific requirements for your system.

**Download Docker:** Visit the Docker website and download the appropriate version of Docker for your operating system.

**Install Docker:** Follow the installation instructions provided by the Docker installer to complete the installation process.

Once Docker is installed, you can use it to build and run containerized applications. Here are some basic commands to get you started:

**docker run:** Use this command to run a containerized application. For example, to run a containerized version of the nginx web server, run the following command:

$ sudo docker run -d -p 80:80 nginx

This command will start a container running nginx and expose it on port 80 of your local machine.

**docker build:** Use this command to build a Docker image from a Dockerfile. A Dockerfile is a text file that contains instructions for building a Docker image. For example, to build a Docker image for a simple Node.js application, create a file named Dockerfile with the following contents:

FROM node:14

WORKDIR /app

COPY package\*.json./

RUN npm install

COPY..

EXPOSE 3000

CMD [“npm”,”start”]

Then, run the following command to build the Docker image:

$ sudo docker build -t my-node-app

This command will build a Docker image with the tag "my-node-app" from the Dockerfile in the current directory.

**docker push:** Use this command to push a Docker image to a registry, such as Docker Hub. For example, to push the "my-node-app" image to Docker Hub, run the following commands:

$ sudo docker tag my-node-app username/my-node-app

$ sudo docker push username/my-node-app

These commands will tag the "my-node-app" image with your Docker Hub username and push it to Docker Hub.

**docker-compose:** Use this command to manage multi-container Docker applications. docker-compose allows you to define a YAML file that describes the containers, networks, and volumes for your application. For example, to define a simple Docker Compose file for a Node.js application and a Redis database, create a file named docker-compose.yml with the following contents:

version:’3’

services:

app:

build:.

ports:

-“3000:3000”

depends\_on:

-redis

redis:

image:”redis:alpine”

Then, run the following command to start the application:

$ sudo docker-compose up

This command will start the two containers (one for the Node.js application and one for Redis) and create a network between them.

**Docker Commands:**

#### docker create

The first command which we will be looking at is the **docker create**command.

This command allows us to create a new container.

The syntax for this command is as shown below:

docker create [options] IMAGE [commands] [arguments]

#### docker ps

The next command we will look at is the **docker ps** command.

The **docker ps** command allows us to view all the containers that are running on the Docker Host.

$ docker ps

If you want to view all the containers that were created on this Docker Host, irrespective of their current status, such as whether they are running or exited, then you would need to include the option -a, which in turn would display all the containers that were created on this Docker Host.

$ docker ps -a

#### docker start

The next command we will look at, is the **docker start** command.

This command starts any stopped container(s).

The syntax for this command is as shown below:

docker start [options] CONTAINER ID/NAME [CONTAINER ID/NAME…]

#### docker stop

The next command on the list is the **docker stop**command.

This command stops any running container(s).

The syntax for this command is as shown below:

docker stop [options] CONTAINER ID/NAME [CONTAINER ID/NAME…]

#### docker run

The next command we will be looking at is the **docker run** command.

This command first creates the container, and then it starts the container. In short, this command is a combination of the docker create and the docker start command.

The syntax for this command is as shown below:

docker run [options] IMAGE [commands] [arguments]

#### docker images

This command lists out all the Docker Images that are present on your Docker Host.

$ docker images

#### docker rm

Moving on to the next command — if we want to delete a container, we use the **docker rm** command.

#### docker rmi

The next command on the list is the **docker rmi** command.

The **docker rmi** command allows us to remove an image(s) from the Docker Host.

In simple terms, Docker is a software platform that simplifies the process of building, running, managing and distributing applications.