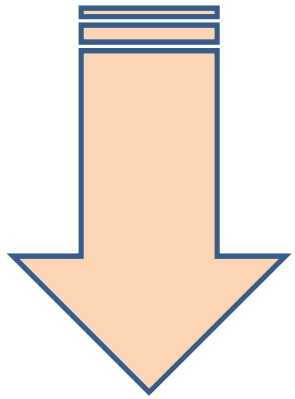
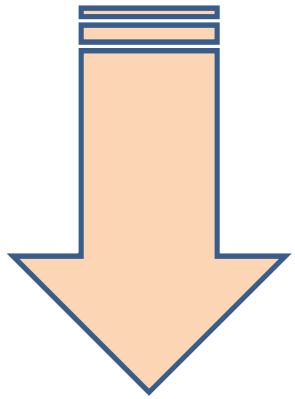
DEVOPS
UNIT-III

DevOps



UNIT-3 : Containerization using Docker, Containerization using Kubernetes



Containerization using Docker

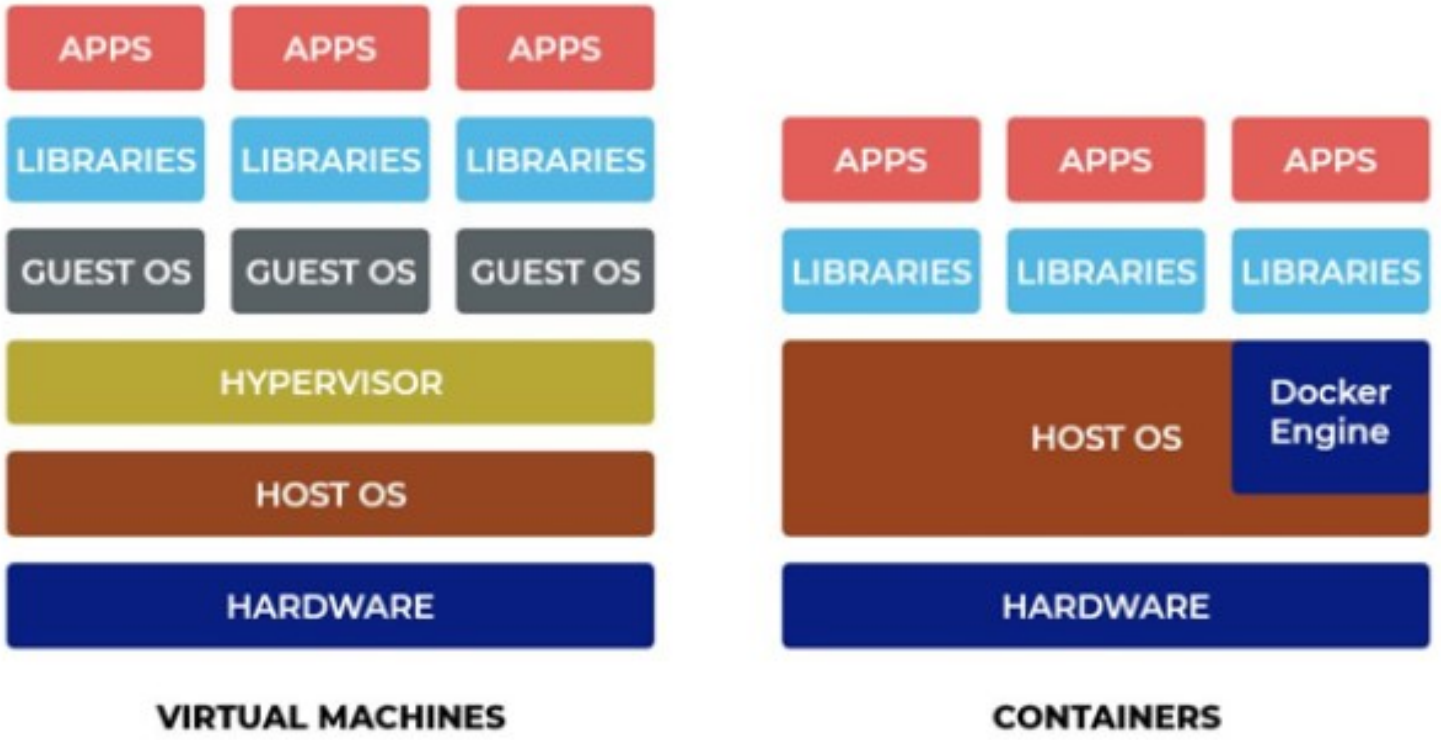
- **Containerization** is OS-based virtualization that creates multiple virtual units in the userspace, known as Containers. Containers share the same host kernel but are isolated from each other through private namespaces and resource control mechanisms at the OS level.
- **Container-based Virtualization** provides a different level of abstraction in terms of virtualization and isolation when compared with hypervisors. Hypervisors use a lot of hardware which results in overhead in terms of virtualizing hardware and virtual device drivers.
- Containers can run virtually anywhere, greatly easy development and deployment: on Linux, Windows, and Mac operating systems; on virtual machines or bare metal, on a developer's machine or in data centers on-premises; and of course, in the public cloud.
- Docker is a tool to perform operating system-level virtualization, which is also known as containerization.
- **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 developed or tested or in production. Docker is a tool designed to make it easier to create, deploy, and run applications by using containers.
- **Docker** comes into play at the deployment stage of the software development cycle
- Docker is the world's leading software container platform. It was launched in 2013 by a company called **Dotcloud**, Inc which was later renamed **Docker, Inc.**
- 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.

Containerization using Docker

- 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.
- A full operating system (e.g -Linux, Windows) runs on top of this virtualized hardware in each virtual machine instance.
- But in contrast, containers implement isolation of processes at the operating system level, thus avoiding such overhead. These containers run on top of the same shared operating system kernel of the underlying host machine and one or more processes can be run within each container. In containers you don't have to pre-allocate any RAM, it is allocated dynamically during the creation of containers while i**n VMs you need to first pre-allocate the memory and then create the virtual machine**.
- **Containerization has better resource utilization compared to VMs** and a short boot-up process. It is the next evolution in virtualization.

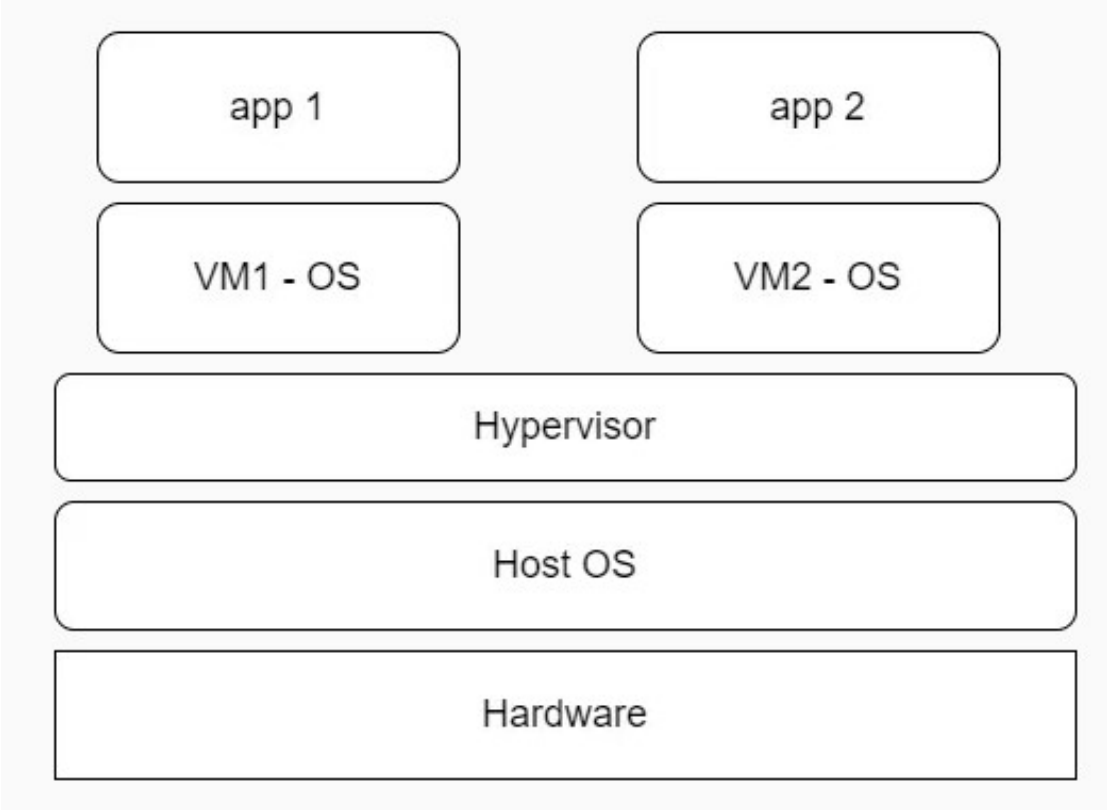
Containerization using Docker

Virtual machines vs containers



VMs

- A virtual machine (VM) is another way of creating an isolated environment.
- A VM is effectively an individual computer that lives inside a host machine; multiple VMs can live inside a single host machine.
- VMs are created by virtualising the host machine's underlying hardware (processing, memory and disk). The hardware is virtualised and split up, with a piece representing a portion of the underlying physical hardware, which a VM can be run on.

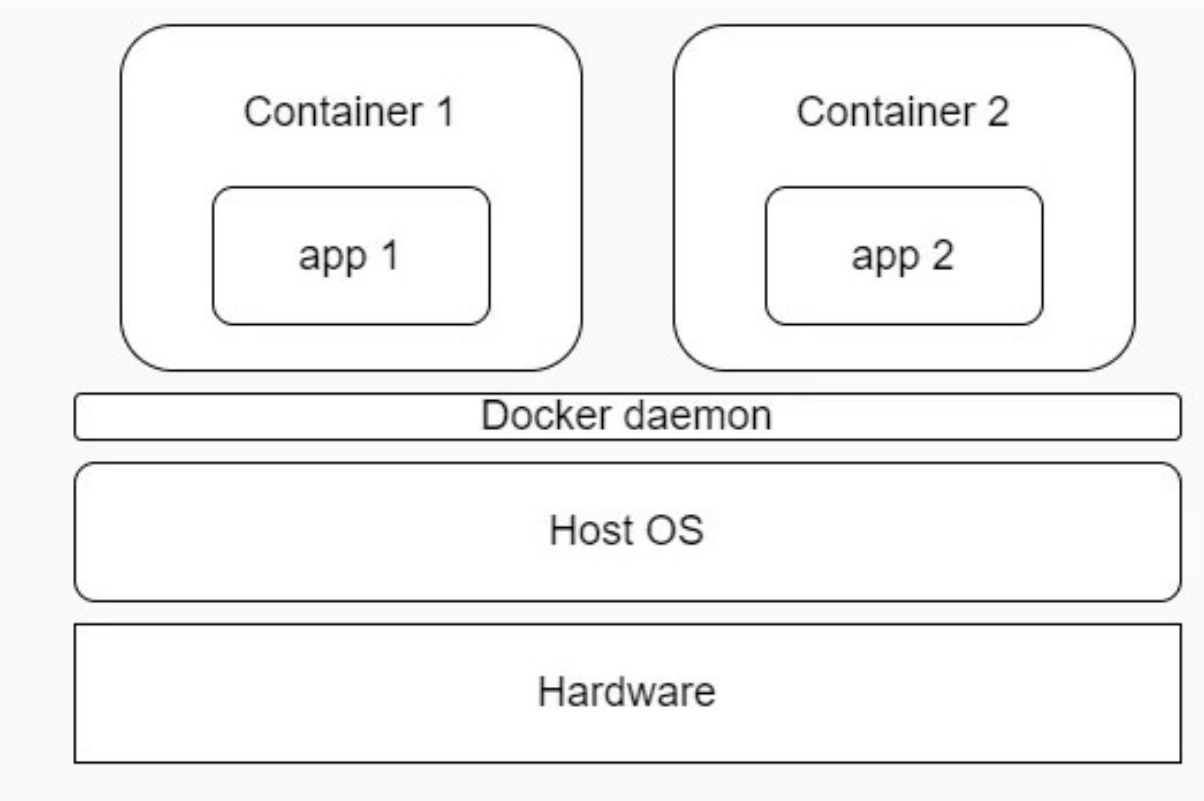


VMs

- As you can see in the figure above, the thing that sits between the VMs and the host is the hypervisor layer. The hypervisor is a piece of software that virtualises the host's hardware and acts as the broker: managing the virtualised hardware and feeding resources to the VMs.
- This virtualisation process brings with it substantial computational overhead. Furthermore, since each VM is basically its own machine, they have their own OS installed, which typically require tens of gigabytes of storage, and which therefore takes time to install, which has to be done every time you want to spin up a new VM.
-

Containers

- Containers take a different approach to producing isolation: like VMs, containers live on top of a host machine and use its resources, however, instead of virtualising the underlying hardware, they virtualise the host OS. Meaning containers don't need to have their own OS, making them much more lightweight than VMs, and consequently quicker to spin up.



Containers

- The parallel to the hypervisor layer with containers is the Docker daemon (assuming you're using Docker), it acts as the broker between the host OS and containers. It comes with less computational overhead than hypervisor software (as depicted by the thinner box in the figure above), again making containers more lightweight compared to VMs.
- VMs suffer from duplication: many of the capabilities and features of the guest OS(s) are found in the host OS, so why not just use the host OS? This is what containers aim to do, whilst still providing isolation and decoupling from software in the host machine. With containers, only the things that the app absolutely needs are copied into the container, as opposed to VMs were the whole OS is installed – even the things from the OS that aren't used by the app.
- What containerisation is actually doing under the covers is some clever misdirection whereby a container only gets to see a virtual view of the host OS; a view that only contains the things that have been prescribed for the container – certain things in the file system, for example.

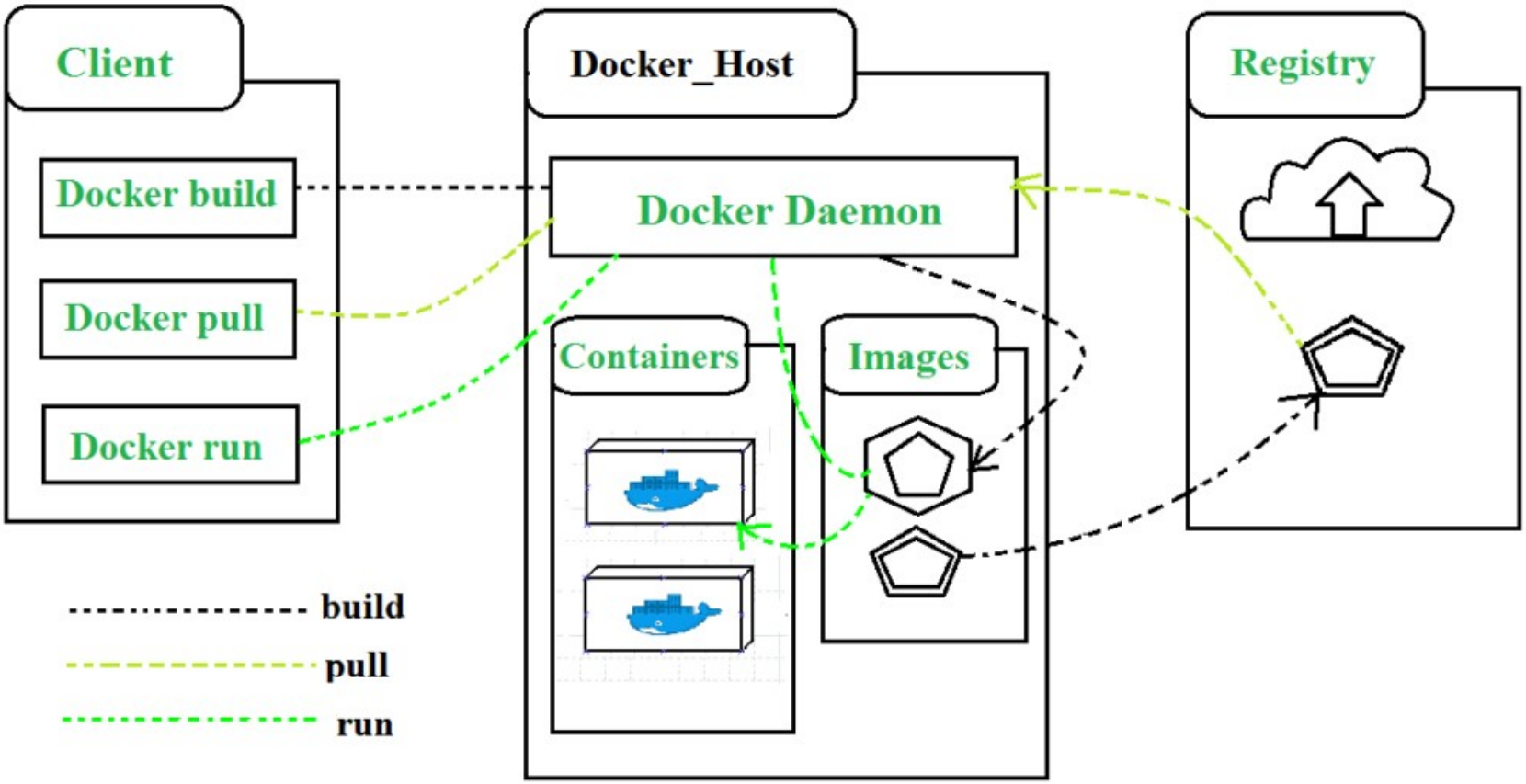
Docker

- Docker is a containerisation platform – it is a toolkit that allows you to build, deploy and manage containerised applications.
- There are alternative containerisation platforms, such as podman, however, Docker is the leading player in this space. Docker is an open source platform, free to download.
- There is also Docker Inc, the company that sells the commercial version of Docker. Docker comes with a command line interface (CLI), using which you can do all of the operations that the platform provides.
- **Docker terminology**
- **Images**: The blueprints of our application which form the basis of containers. These contain all of the configuration settings that define the isolated environment.
- **Containers**: Are instances of a Docker image and are what run the actual application.
- **Docker Daemon**: That background service running on the host that listens to API calls (via the Docker client), manages images and building, running and distributing containers. The Deamon is the process that runs in the operating system which the client talks to – playing the role of the broker.
- **Docker Client**: The command line tool that allows the user to interact with the daemon. There are other forms of clients too.
- **Docker Hub**: A registry of Docker images containing all available Docker images. A user can have their own registry, from which they can pull images.

Docker Architecture

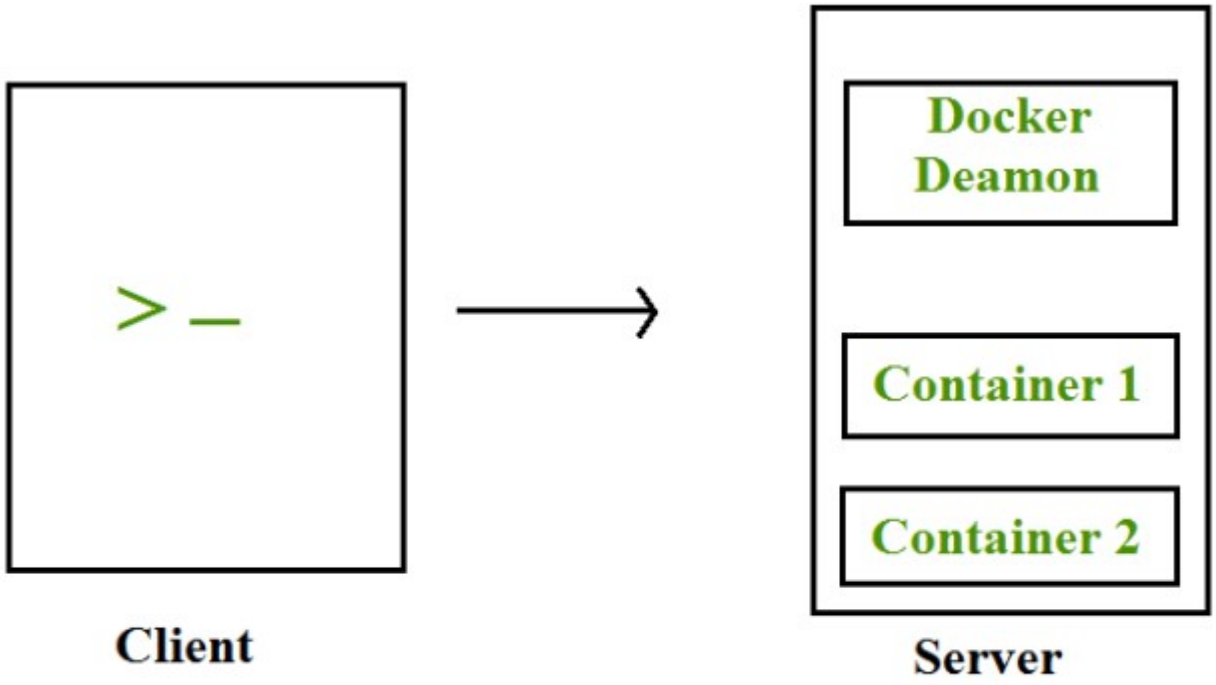
- Docker architecture consists of Docker client, Docker Daemon running on Docker Host, and Docker Hub repository.
- Docker has client-server architecture in which the client communicates with the Docker Daemon running on the **Docker Host** using a combination of REST APIs, Socket IO, and TCP.
- If we have to build the Docker image, then we use the client to execute the build command to Docker Daemon then Docker Daemon builds an image based on given inputs and saves it into the Docker registry.
- If you don't want to create an image then just execute the pull command from the client and then Docker Daemon will pull the image from the Docker Hub finally if we want to run the image then execute the run command from the client which will create the container.

Containerization using Docker



Containerization using Docker

1. **Docker Clients and Servers**– Docker has a client-server architecture. The Docker Daemon/Server consists of all containers. The Docker Daemon/Server receives the request from the Docker client through CLI or REST APIs and thus processes the request accordingly. Docker client and Daemon can be present on the same host or different host.



Containerization using Docker

Docker Images– *Docker images are used to build docker containers by using a read-only templat*e. The foundation of every image is a base image eg. base images such as – ubuntu14.04 LTS, and Fedora 20. Base images can also be created from scratch and then required applications can be added to the base image by modifying it thus this process of creating a new image is called "committing the change".

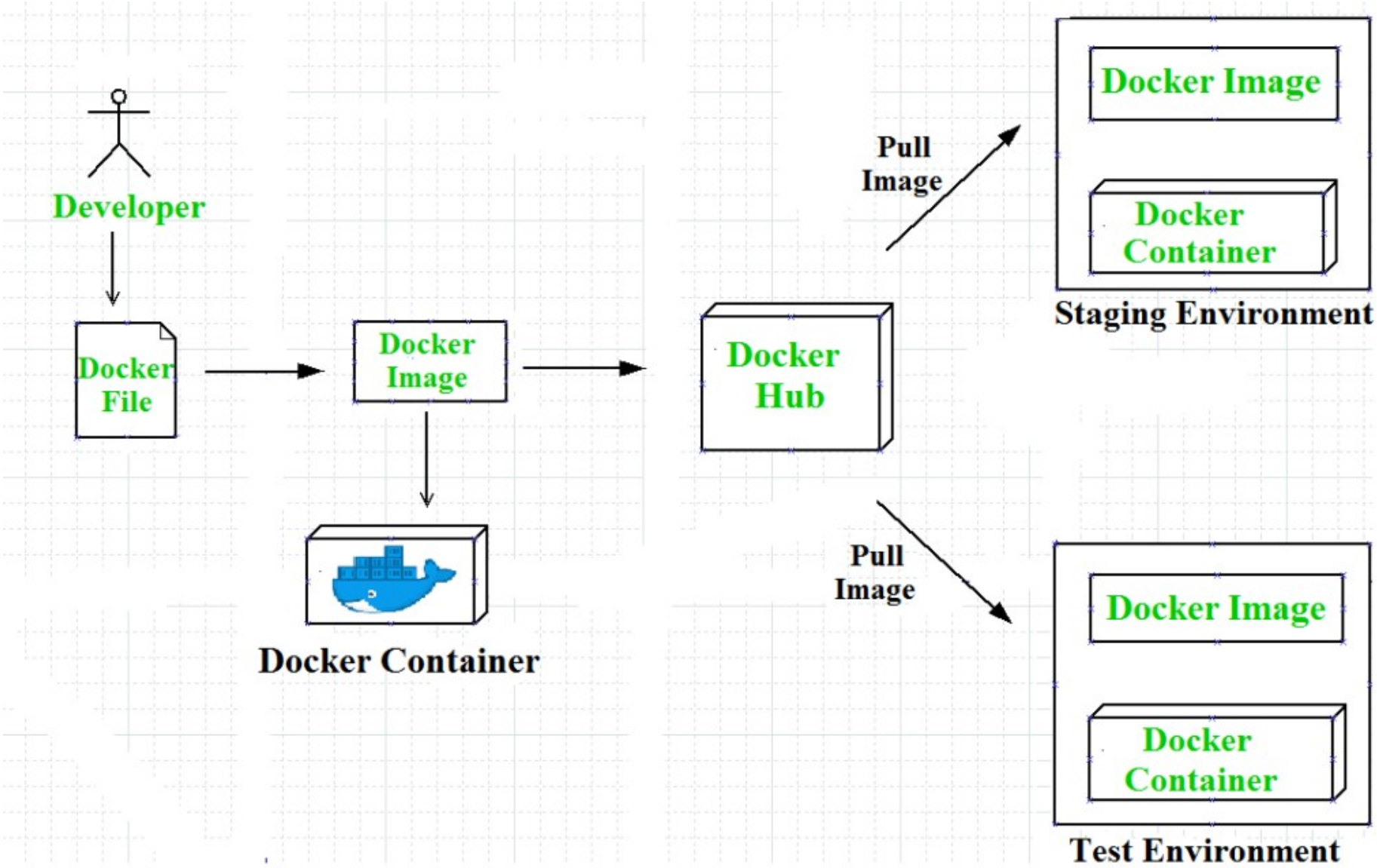
Docker File–image. The final image can be uploaded to Docker Hub and shared among various *Dockerfile is a text file that contains a series of instructions on how to build your Docker image.* This image contains all the project code and its dependencies. The same Docker image can be used to spin 'n' number of containers each with modification to the underlying collaborators for testing and deployment. The set of commands that you need to use in your Docker File is FROM, CMD, ENTRYPOINT, VOLUME, ENV, and many more.

Docker Registries– *Docker Registry is a storage component for Docker images.* We can store the images in either public/private repositories so that multiple users can collaborate in building the application.

***Docker Hub** is Docker's cloud repository*. Docker Hub is called a public registry where everyone can pull available images and push their images without creating an image from scratch.

Docker Containers– *Docker Containers are runtime instances of Docker images*. Containers contain the whole kit required for an application, so the application can be run in an isolated way. For eg.- Suppose there is an image of Ubuntu OS with NGINX SERVER when this image is run with the docker run command, then a container will be created and NGINX SERVER will be running on Ubuntu OS.

Containerization using Docker



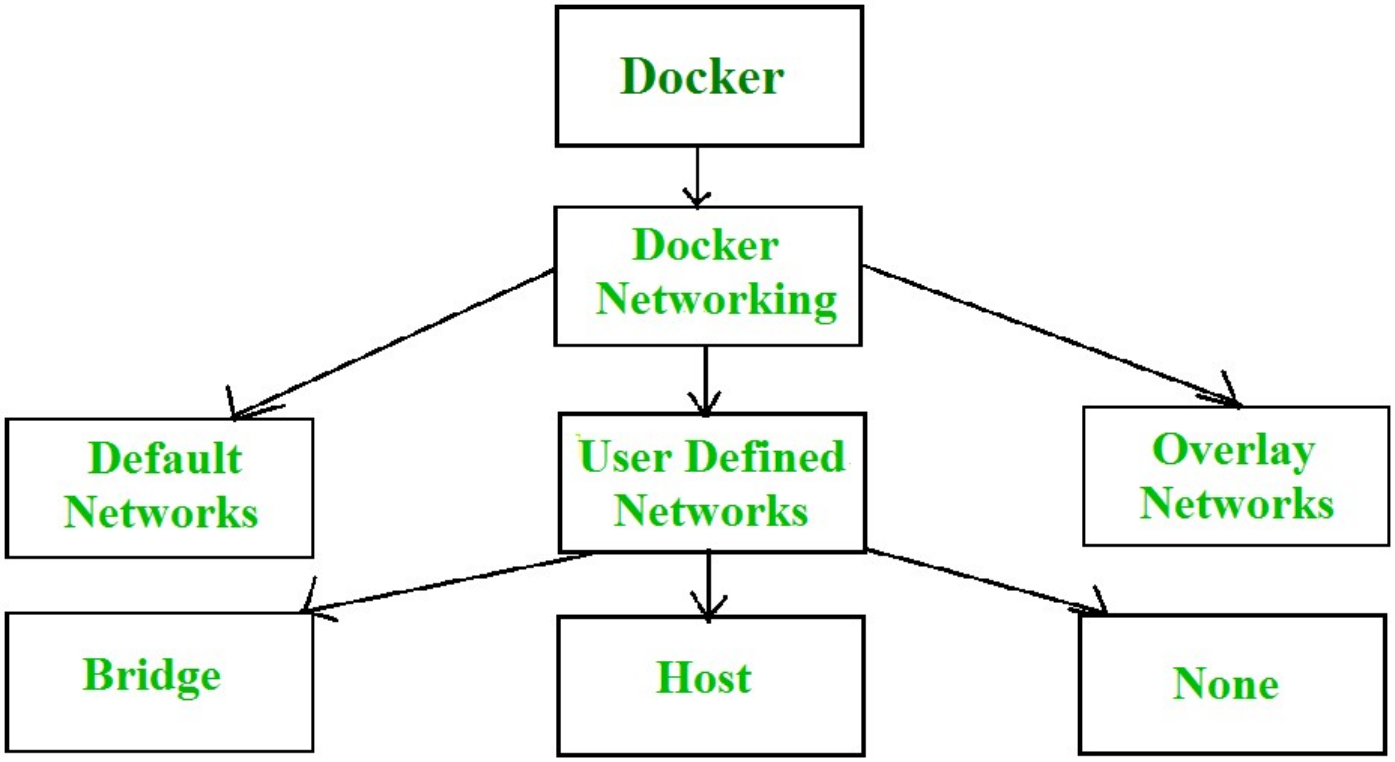
Containerization using Docker

Docker Compose

- Docker Compose is a tool with which we can create a multi-container application. It makes it easier to configure and
  run applications made up of multiple containers. For example, suppose you had an application that required WordPress and MySQL, you could create one file which would start both the containers as a service without the need to start each one separately. We define a multi-container application in a YAML file. With the docker-compose-up command, we can start the application in the foreground. Docker-compose will look for the docker-compose. YAML file in the current folder to start the application. By adding the -d option to the docker-compose-up command, we can start the application in the background. Creating a docker-compose. YAML file for WordPress application

Containerization using Docker

- **Docker Networks**
- When we create and run a container, Docker by itself assigns an IP address to it, by default. Most of the time, it is required to create and deploy Docker networks as per our needs. So, Docker let us design the network as per our requirements. There are three types of Docker networks- default networks, user-defined networks, and overlay networks.



Docker Advantages

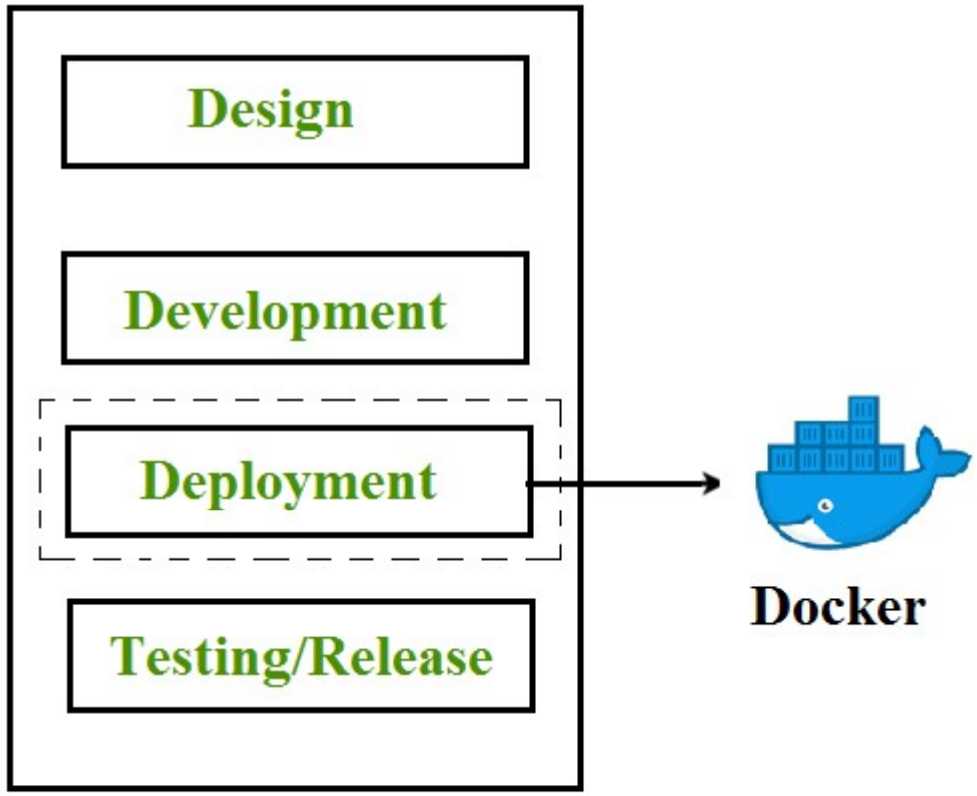
Speed – The speed of Docker containers compared to a virtual machine is very fast. The time required to build a container is very fast because they are tiny and lightweight. Development, testing, and deployment can be done faster as containers are small. Containers can be pushed for testing once they have been built and then from there on to the production environment.

Portability – The applications that are built inside docker containers are extremely portable. These portable applications can easily be moved anywhere as a single element and their performance also remains the same.

Scalability – Docker has the ability that it can be deployed on several physical servers, data servers, and cloud platforms. It can also be run on every Linux machine. Containers can easily be moved from a cloud environment to a local host and from there back to the cloud again at a fast pace.

Density – Docker uses the resources that are available more efficiently because it does not use a hypervisor. This is the reason that more containers can be run on a single host as compared to virtual machines. Docker Containers have higher performance because of their high density and no overhead wastage of resources.

Containerization using Docker



Installing a FSD App on AWS EC2 Docker Container

-

Installing a FSD App on AWS EC2 Docker Container

-