**Docker**

Docker is the one implementation of container-based virtualization technologies. Docker is a software platform that allows you to build, test, and deploy applications quickly. Docker packages software into standardized units called containers that have everything the software needs to run including libraries, system tools, code, and runtime. Using Docker, you can quickly deploy and scale applications into any environment and know your code will run.

Types of Virtualizations-

Pre-virtualization –

**Problems-**

* Huge Cost
* Slow Deployment
* Hard to migrate

Physical Server

Host Os

Bins/Libs

App

Pre-Virtualization

Hypervisor-based Virtualization-

App3

App2

App1

**Hypervisor providers:**

VMware, Virtual Box

Host Operating System

Guest OS

Guest OS

Guest OS

Hypervisor

Physical Layer

Bins/Libs

Bins/Libs

Bins/Libs

**Benefits:**

Cost-Efficient

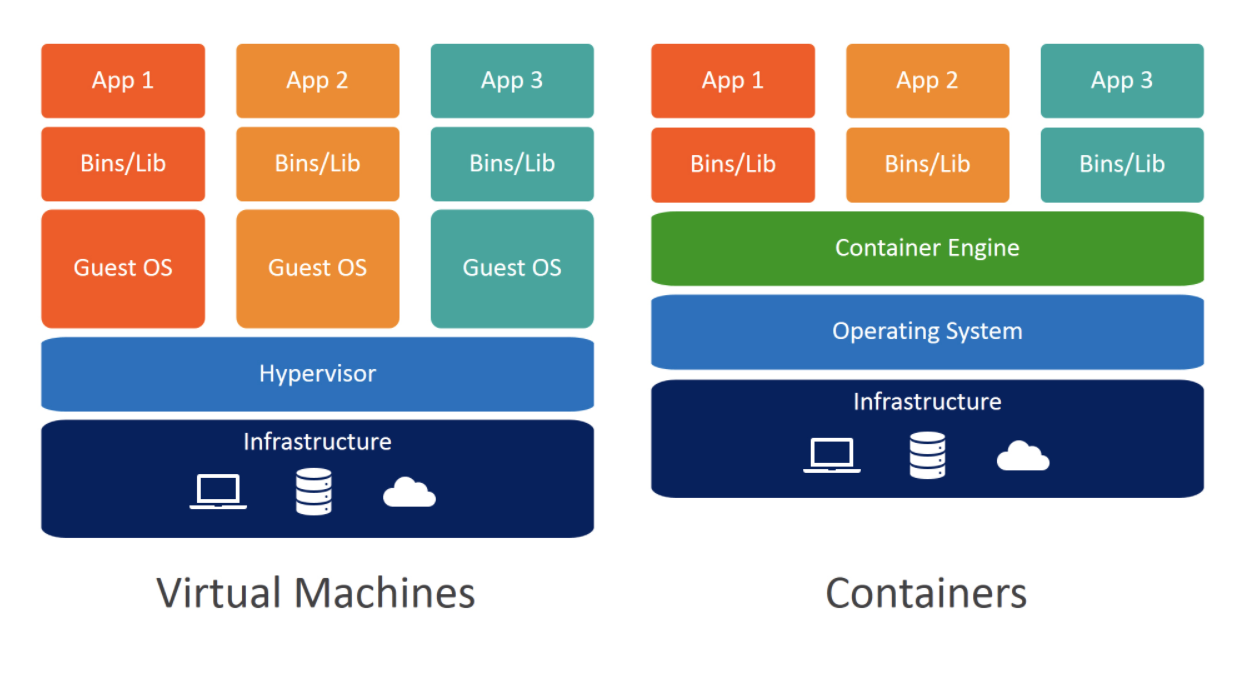
Easy to scale

**Limitation:**

Kernel Resource Duplication

Application Portability Issue

Container Virtualization-



**Benefits:**

Cost Efficient

Fast Deployment

Guaranteed Portability

Container Virtualization

Container virtualization (often referred as operating system virtualization) is more than just a different kind of hypervisor. Containers use the **host operating system**as their base, and **not** the hypervisor.

Client-server Architecture-

Docker uses a client-server architecture with daemon being the server. The user does not directly interact with the daemon but instead through the docker client.

The Docker client is the primary user interface to Docker. It accepts commands from the user and communicates back and forth with docker daemon.

There are two types of Docker clients-

1. Command Line
2. Kitematic

The daemon is the persistent process which does the heavy lifting of building, running and distributing docker containers. Docker daemon is often referred as Docker engine or Docker server.

Docker daemon can’t run on non linux platforms natively because it uses Linux specific kernel features.

Docker machine is a lightweight Linux VM made specially run the docker daemon on OS X or Windows.

**Images-**

Images are read only templates used to create containers.

Images are created with docker build command, either by us or by other docker users.

Images are composed of layers of other images.

Images are stored in docker registry.

**Containers-**

If an image is a class, then a container is an instance of class-a runtime object

Containers are lightweight and portable encapsulations of an environment in which to run applications.

Containers are created from images. Inside the container, it has all the binaries and dependencies needed to run the application.

**Registry & Repositories-**

A registry is where we store our images.

Inside registry, images are stored in repositories.

**Docker Hub-**

Docker hub is a public registry which contains a large number of images.

**Steps to create & Run Docker Container (Example - Hello World)**

1. Search image(busybox) at dockerhub website
2. Select image(busybox) and scroll down to documents
3. Then select tiny image(busybox) which have less size
4. Select tag(1.24) in image(busybox)
5. Open quickstart terminal (this is only for docker toolbox) or normal terminal
6. To check which image have the run command

Command - docker images

1. To create a container

Command – docker run repository(busybox):tag(1.24) echo

[arguments]”Hello World”

**Commands =**

docker image – manage image

docker images – list images

docker inspects – return low level information on docker object

docker log – fetch the logs of container

docker ps – list the containers

docker bulid – to bulid the image using the instructions

**Build Docker Images-**

1. **By using docker commit in a docker container**

Steps-(image - debian)

1. Spin up a container from a base image
2. Install git package in the container
3. Commit changes made in the container

Docker commit – command would save the changes we made to docker conatiner’s file system to a new image.

docker commit container\_ID repository\_name:tag

1. **By writing Dockerfile**

A Dockerfile is a text document that contains all the instructions users provide to assemble an image. Each instruction will create a new image layer to the image. Instructions specify what to do when building the image. Instructions are not case sensitive.

Instructions-

From – used for base image

Run- it will specify a command to execute

Docker build command takes the path to the build context as an argument.

**Docker Cache-**

Docker creates container images using layers. Each command that is found in a Dockerfile creates a new layer. Each layer contains the filesystem changes to the image for the state before the execution of the command and the state after the execution of the command.

Docker uses a layer cache to optimize and speed up the process of building Docker images.

Docker Layer Caching mainly works on the RUN, COPY and ADD commands, which will be explained in more detail next.

RUN Command-

The RUN command allows you to execute a command in the Docker image. If the layer that is generated by the RUN command already exists in cache, the RUN command will be executed only once.

COPY Command-

The COPY command in a Dockerfile allows you to import one or more external files into a Docker image. When executed, the COPY commands ensure you have the latest version of all relevant external files.

ADD Command-

The ADD command in a Dockerfile allows you to import external files into a Docker image.

**Docker Container links-**

The main use for docker container links is when we build an application with microservices architecture, we are able to run many independent components in different containers.

Docker creates a secure tunnel between the containers that doesn’t need to expose any ports externally on the container.

**Docker Compose-**

Docker compose is a tool for defining and running multi-container Docker applications. With Compose, you use a YAML file to configure your application’s services.

**Docker Networking-**

Docker takes care of the networking aspects so that the containers can communicate with other containers and also with the Docker Host.

Types-

1. None network-

This network does not have any access to the outside world. The none network adds a container to container-specific network stack. That container stacks a network interface so it is totally isolated.

1. Bridge Network-

The default network driver. If you don’t specify a driver, this is the type of network you are creating. Bridge networks are usually used when your applications run in standalone containers that need to communicate.

1. Host Network-

For standalone containers, remove network isolation between the container and the Docker host, and use the host’s networking directly. host is only available for swarm services on Docker 17.06 and higher. Minimum. network security level

1. Overlay Network-

Overlay networks connect multiple Docker daemons together and enable swarm services to communicate with each other. You can also use overlay networks to facilitate communication between a swarm service and a standalone container, or between two standalone containers on different Docker daemons. This strategy removes the need to do OS-level routing between these containers.

**Unit Test in container-**

Unit tests should test some basic functionality of our docker app code, with no reliance on external services.

Unit tests should run as quickly as possible so that developers can iterate much faster without being blocked by waiting for the tests results.

Docker containers can spin up in second and create a clean and isolated environment which is great tool to run unit tests with.

**Continuous Integration-**

CI is a software engineering practice in which isolated changes are immediately tested and reported when they are added to a larger code base.

The goal of CI is to provide rapid feedback so that is a defect is introduced into the code base, it can be identified and corrected as soon as possible.

**Set up SSH keys for GitHub account-**

-SSH keys are way to identify trusted computers without involving password.

-Generate a SSH key pair and save the private SSH key in your local box and add the public key to your GitHub account.

-Then you can directly push your changes to GitHub repository without typing password.

Check the SSH public key available or not-

The SSH public key file usually sits under ~/.ssh/ directory and ends with .pub extension

**Docker Swarm-**

Docker swarm is a tool that clusters many Docker Engines and schedules containers.

Docker swarm decides which host to run the container based on your scheduling methods.

A swarm consists of multiple Docker hosts which run in **swarm mode** and act as managers (to manage membership and delegation) and workers (which run [swarm services](https://docs.docker.com/engine/swarm/key-concepts/#services-and-tasks)). A given Docker host can be a manager, a worker, or perform both roles. When you create a service, you define its optimal state (number of replicas, network and storage resources available to it, ports the service exposes to the outside world, and more). Docker works to maintain that desired state. For instance, if a worker node becomes unavailable, Docker schedules that node’s tasks on other nodes. A task is a running container which is part of a swarm service and managed by a swarm manager, as opposed to a standalone container.

When Docker is running in swarm mode, you can still run standalone containers on any of the Docker hosts participating in the swarm, as well as swarm services. A key difference between standalone containers and swarm services is that only swarm managers can manage a swarm, while standalone containers can be started on any daemon. Docker daemons can participate in a swarm as managers, workers, or both.

Provision a Swarm Cluster-

1. Deploy two VMs, one will be used for the Swarm manager node, and the other one will be used as a worker node.
2. Appoint the first VM as Swarm manger node and initialize a Swarm cluster

Command = docker swarm init

1. Let the second VM join the swarm cluster as a worker node.
2. Command = docker swarm join

Docker Services-

The services can be defined in our Docker compose file.

The service definition includes which Docker images to run, the port mapping and dependency between services.

**Docker Stack-**

A docker stack is a group of interrelated services that share dependencies, and can be orchestrated and scaled together.

Create a stack from your docker compose file:

* docker stack deploy

In the Swarm Mode,

* Docker compose files can be used for service definitions.
* Docker compose commands can’t be reused. Docker compose commands can only schedule the containers to a single node.