Docker

Section 1 : Introduction

What is Docker ?

- **Docker is just a software which is used to perform operating system virtualization.**
- **This is also know as containerization.**
- **These containers are isolated from each other and bundled with their own application, tools, libraries and configuration files.**
- **Though these containers are isolated it can communicate with each other through well-defined channels.**
- **All these containers share the base operating system kernel thus its more lightweight then virtual machines.**

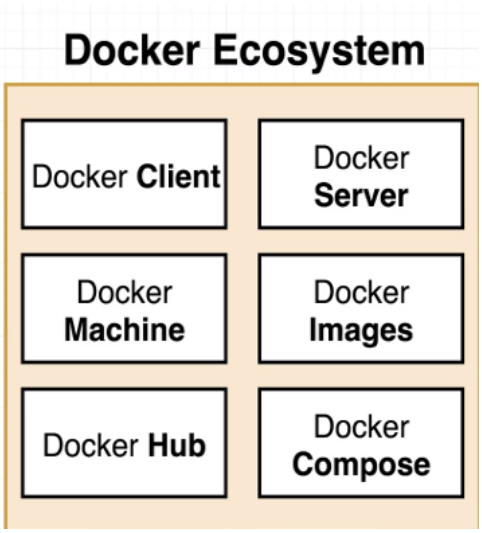
Why Docker

- **To understand docker, first lets try to understand simple scenario.**
- **Software installation: Which everyone would have tried to install software in any machine at least once in your life before.**
- **This is the flow how to install software in any machine.**
- **Eventually, every one would have faced multiple issues while installation.**
- **For instance, dependent software's not found [wget, curl, unzip, tar] to install.**
- **You will be trying hard to fix any issues which is coming upon every time when you install software.**
- **The is the core cause which Docker is trying to fix.**
- **Docker wants to make it easy & straight forward to install any software in computer. (On your laptop, VMs, Server or cloud instance)**

In Nutshell, the real use case of Docker is to install and run software without worrying about setup or dependencies.

Docker Ecosystem ?

- **Anytime you see someone who refer docker in blog post, article or forum they might referring to many terminologies like [Docker images, Hub, Compose, build, client, server etc..]**

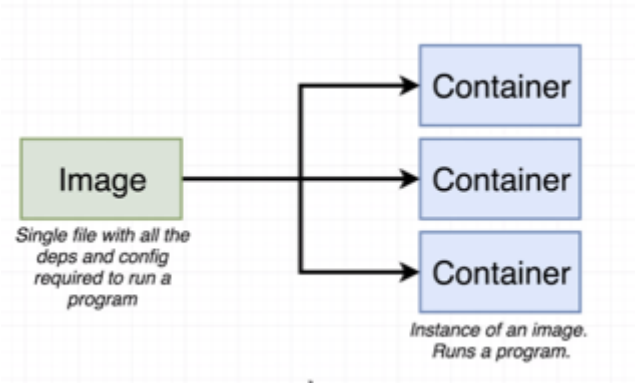


- **However these are tools which comes together to form a platform or ecosystem to create and run something called containers.**

Well your next question would be what is the container as we are speaking a lot about it ? Lets see about it in details.

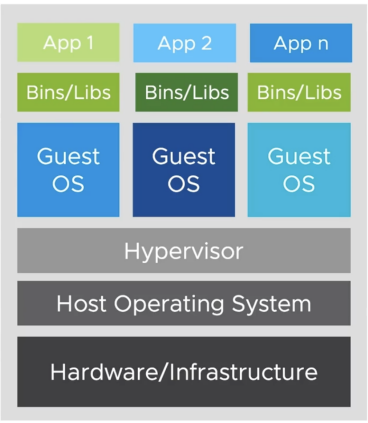
What is Container ?

- **As the image explains, you have an single file called image with all the dependencies and software installed and ready.**
- **This single file is going to store in your local machine.**
- **At some point of time you can use this image to create some thing called container.**
- **The container is an instance of an image, which is going to run your application.**

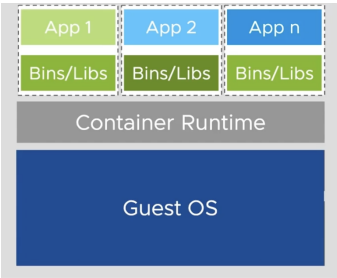


Difference between Virtual Machine and Container

It's an unfair comparison to compare VM's to Containers because they work together quiet commonly



Virtual Machines



Containers

Docker Installation

Create Dockerhub account

https://hub.docker.com/signup

Docker on MAC

https://docs.docker.com/docker-for-mac/install/

Docker on Windows

https://docs.docker.com/docker-for-windows/install/

Windows Home users will need to install Docker Toolbox which uses VirtualBox instead.

- **Docs and setup info available here:https://docs.docker.com/toolbox/toolbox_install_windows/**
- **Release downloads available here:https://github.com/docker/toolbox/releases**
- **Toolbox will install everything you need including VirtualBox.**

- **You may also need to enable virtualization in your computer's BIOS settings.**
- **This will be different for each manufacturer, please refer to their documentation on which keys to use to access these settings on reboot.**

> Docker on Linux

- **Installing on Centos**
- **Installing on Ubuntu**
- **Installing on Debian**

Unlike MAC and Windows Docker desktop version, we must install Docker compose

Starting Docker on boot : To start Automatically on linux boot

Section 2 : Linux Kernel Features

Namespace

- **Namespaces is the feature of Linux kernel.**
- **This feature helps to partition resources such that one set of processes sees one set of resource while another set of process sess another set of resources.**

How Namespace is used for Docker?

- **Docker creates an isolated environment for the container using this namespaces**.
- **Docker Engine uses namespaces such as the following on Linux.**
  - **The `pid` namespace:** Process isolation (PID: Process ID).
  - **The `net` namespace:** Managing network interfaces (NET: Networking).
  - **The `ipc` namespace:** Managing access to IPC resources (IPC: InterProcess Communication).
  - **The `mnt` namespace:** Managing filesystem mount points (MNT: Mount).

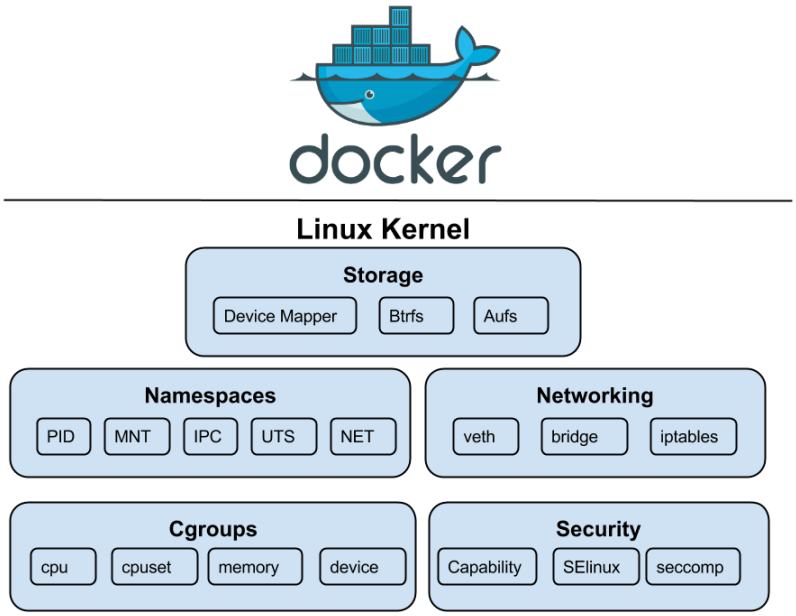
- **The `uts` namespace:** Isolating kernel and version identifiers. (UTS: Unix Timesharing System).

Cgroups (control group)

- **Cgroups is a Linux kernel feature that limits resource usage (CPU, memory, disk I/O, network, etc.) to each processes.**

Networking & Security

- **Docker also virtualize Network & Security components of base OS.**
- **From Network stand point base OS adapter is going to virtualized.**
- **From Security stand point SELinux of base OS is going to be virtualized.**



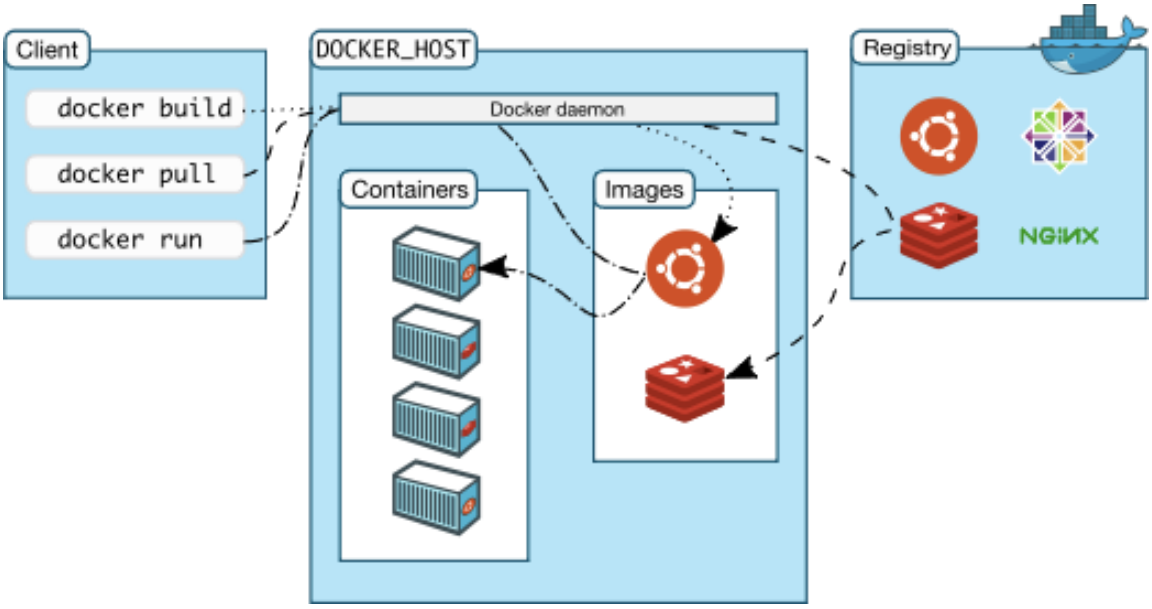
Section 3 - Architecture

Docker Architecture

- **Docker works in Server-Client model.**
- **Once docker is installed you can just run command**

```
docker version or docker info
```

- **Here, Docker demon (dockerd) is going to manage objects such as images, containers, networks and volumes.**
- **Docker client is the primary way to interact with docker demon (server)**
- **For example. when you used "docker run" command the client sends these command to dockerd which is going to execute it.**
- **Docker clinet uses an API to connect to the docker daemon.**



Container Runtime

What is Container Runtime ?

- **It's software that runs and manages the components required to run containers.**
- **Docker uses software called "containerd" as runtime.**
- **We have several other container runtime softwares in market such as CRI-O & RKT**

Session 4 : Images

Docker images

- **A Docker image is a read-only template that contains a set of instructions for creating a container that can run on the Docker platform.**
- **It provides a convenient way to package up applications and preconfigured server environments.**

Anatomy of Docker image

- **A Docker image is made up of a collection of files that bundle together all the essentials, such as installations, application code and dependencies, required to configure a fully operational container environment.**
- **You can create a Docker image in one of two ways:**
  - **Interactive Method: By running a container from an existing Docker image, manually changing that container environment through a series of live steps and saving the resulting state as a new image.**
  - **Dockerfile Method: By constructing a plain-text file, known as a Dockerfile, which provides the specifications for creating a Docker image.**

Container Registries

- **Container registries are catalogs of storage locations, known as repositories, where you can push and pull container images.**
- **Hosted: Docker Hub, Google Container Registry**
- **Self-Hosted: Artifactory, Harbor, Quay, DTR (Docker Trusted Registry)**

Deal with docker images

Pull image

```
docker pull centos
```

Return low-level information on Docker objects

```
docker image inspect centos
```

Get details about all images in local system

```
docker images
```

Shows history of image

```
docker history centos
```

Pull specific version images

```
docker pull ubuntu:14.04
```

To remove one image from local repository

```
docker rmi ubuntu:14.04
```

- **"rmi" refers remove image.**
- **Following to that image name to be deleted.**

To remove all image from local repository

```
docker rmi $(docker images -q)
```

Take the input from command "docker images -q" and delete all the images available in local machine.

Docker images - Interactive method

Quickest and simplest way to create Docker images.

Login into shell once machine is created

```
docker run -it --name myhost ubuntu bash
```

[i] Interactive, [t] tty

After login into container

```
apt-get update && apt-get install -y nginx
```

Command to list active container processes

```
docker ps
```

```
docker ps -a
```

Docker commit

```
docker commit container_id <docker_id>/centos-automation
```

Docker push

```
docker push <docker_id>/centos-automation:latest
```

Docker images - Docker file method

- **Clean, compact and repeatable recipe-based images.**
- **Easier lifecycle management.**
- **Through this mentod you can easily customize the image in an automated way.**

```
# Use the official Ubuntu 18.04 as base
FROM ubuntu:18.04
# Install nginx and curl
RUN apt-get update &&
apt-get upgrade -y &&
apt-get install -y nginx curl &&
rm -rf /var/lib/apt/lists/*
```

> ### How to run docker build file ?

```
docker image build -t imagename .
```

Here -t is to tag the image and . is the current dir where the Dockerfile and other app files are located.

> ### To check how images are build, you can use history command

```
docker history <imagename>
```

Session 5 : Container Management

> ### Create container and run it in foreground

```
docker run -t centos
```

> ### Create container and run it in background. To keep container running without exit using [t].

```
docker run -t -d centos
```

[t] tty, [d] detached

Assigning a name to my container

```
docker run -t -d --name myhost centos
```

Creating container by specifing tags

```
docker run -t -d --name myhost centos:7.5
```

Binding local port 80 to container port 80

```
docker run -t -d -p 80:80 --name myhost centos
```

[p] publish

Create HTTP container with port binding

```
docker pull httpd
docker run -t -d -p 80:80 --name myhost httpd
docker exec -it myhost bash
Create a sample html file under "htdocs"
Access your html file from laptop browser "localhost:80"
```

Run any shell commands in the container which is created

```
docker run --name myhost centos cat /etc/redhat-release
```

Login into the container shell after creating the machine

```
docker run -it --name myhost centos bash
```

[i] Interactive, [t] tty

Login to existing container which is already available [Only applicable for machines in exited state]

```
docker start -ai myhost
```

Login to existing machine which is up and running at background.

```
docker exec -it myhost bash
```

Run top on container without entering into it

```
docker top myhost
```

Attach the container back which is running in background. [But this may not give interactive shell]

```
docker attach myhost
```

View running containers

```
docker ps
```

View running and stopped containers

```
docker ps -a
```

To stop running container

```
docker stop myhost
```

To start container

```
docker start myhost
```

Remove docker container forcefully [It may remove machine even while running]

```
docker rm -f myhost
```

Remove docker container [Only when it is in stopped state]

```
docker rm myhost
```

To check the port flow of container

```
docker port myhost
```

Session 6 : Volumes

- **Docker allows to share storage among multiple containers.**
- **Does not delete contained data when deleting the container.**
- **Back up, restore, and migrate data easily.**

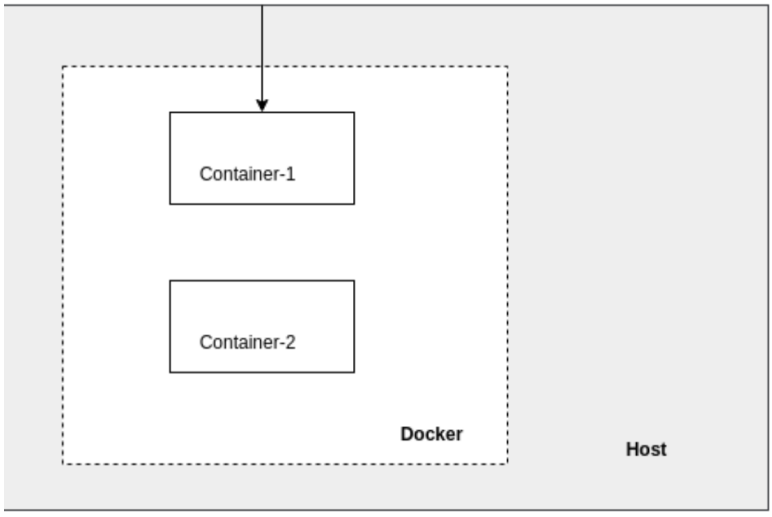
Attaching external volume to a docker container

```
docker run -t -d -v /Users/Docker:/data --name myhost centos
```

- **In linux machine hosted container, you can create logical volume on base machine and attach to docker container.**

Session 7 : Networking

- **When we talk about Docker, we say that containers are isolated.**
- **How do containers will communicate each other, or other applications like MySQL database? It is not useful if things are not able to communicate each other.**
- **Depending on how we want our container to behave, we can select our network.**
- **Docker has a network concept. It has several network drivers to work with. This helps us to connect container with a container or container with host.**



- **Also, Docker networking enables a user to link container to as many networks as requires.**

- **Docker networks are used to provide complete isolation for containers.**
- **It is also possible to add containers to more then one network.**

Advantages of Docker networking

- **Rapid deployment**
- **Portability**
- **Better Efficiency**
- **Faster configuration**
- **Scalability**
- **Security**

Network commands summary

List available networks

```
docker network ls
```

Create a network

```
docker network create
```

Remove a network

```
docker network rm
```

Inspect a network

```
docker network inspect
```

> Connect container to a network

```
docker network connect
```

> Disconnect container from a network

```
docker network disconnect
```

Network Drivers

The network drivers used in Docker are below

Bridge

- **Bridge is the default network.**
- **When the Docker daemon service starts, it configures a virtual bridge named docker0 .**
- **When we don't specify the network, this is the one Docker uses.**
- **Docker creates a private network inside the host which allows containers to communicate with each other. host tells Docker to use host computers network directly.**

Host

- **Remove network isolation between the container and the Docker host.**

None

- **Disable all networking for a specific container.**

Overlay

- **Overlay networks connect multiple Docker daemons together and enable swarm services to communicate with each other.**

Macvlan

- **Macvlan networks allow you to assign a MAC address to a container, making it appear as a physical device on your network.**

Session 8 : Container Orchestration

Container orchestration is going to help us to create containers in automated way.

To make this possible we are going to use docker-compose.

docker-compose.yml

```
version: "3.8"
services:
    Master:
        image: "centos-automation"
        container_name: master
        hostname: ansible_master
        ports:
            - "22:22"
        volumes:
            - /Users/prathang/Prathang/Learnings/Shared:/data

    Target-node-1:
        image: "centos-ssh"
        container_name: target1
        hostname: target-1
        volumes:
            - /Users/prathang/Prathang/Learnings/Shared:/data

    Target-node-2:
        image: "centos-ssh"
        container_name: target2
        hostname: target-2
        volumes:
            - /Users/prathang/Prathang/Learnings/Shared:/data
```

How to execute compose file

```
docker-compose up -d
```