**What is Virtualization?**

Virtualization is a creation of logical object version in place of an actual version.

Few examples are virtual computer hardware platform, virtual storage, and virtual LANs.

Hardware virtualization means creating virtual machine that acts like a real physical computer with an OS.

For example, a virtual machine (VM) hosted on a computer with Microsoft Windows may behave like Ubuntu and Ubuntu supported software may run on the VM.

**Need for a Virtual Machine**

Virtual machine setup has the following benefits.

* Multiple operating systems can be hosted on the same machine simultaneously with complete isolation.
* Multiple VMs can be deployed on the same physical box. This reduces the total number of physical machines.
* Easy maintenance, app provisioning, and quick recovery.

**Problems with Virtual Machine**

* A lot of wastage of resources like ram, processor, disk space due to fixed space slicing for every application deployed. Hence this is not ideal for a large scale application developed using micro services.
* Inconsistent computing environment across the software delivery life cycle (Prod/Dev/QA).
* Hardware failures like malfunctioning and power supply loss will stop all working servers since many servers run on a single physical server.

**What are Containers?**

Containers are multiple isolated services that are run on a single control host (underlying infrastructure) and they access a single kernel.

Container based virtualization is an OS-level virtualization method for deploying and running distributed applications without launching an entire VM for each application.

They isolate applications from one another.

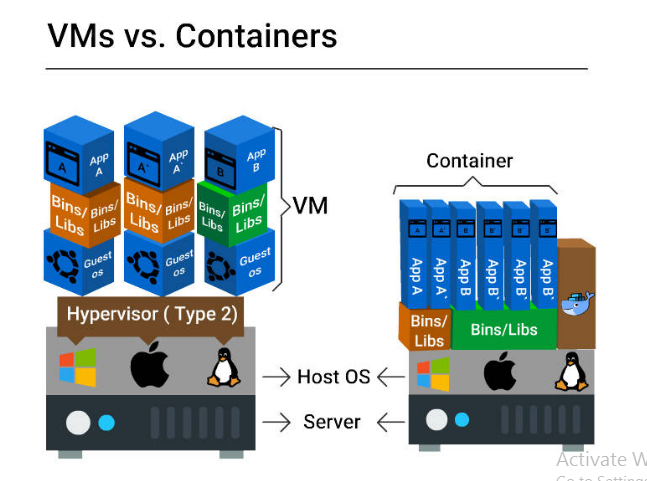
**Difference between VMs Vs Containers**

**Size:**

* VMs - Resource wastage is high
* Containers - No wastage

**Start up:**

* VMs - Starts slow
* Containers - Starts really quick



**Integration with DevOps tools:**

* VMs - Complex (infrastructure wastage, repetitive configuration and minimal scalability)
* Containers - Very simple

**Container - Benefits**

* Improved portability
* Better performance
* Optimum RAM/disk space/cloud utilization
* Suited for agile environment
* Facilitates approaches such as micro services, continuous integration, and delivery.
* Eliminates environment inconsistencies

**Benefits from Container Orchestration Tools**

* Container Orchestration tools like Docker Swarm, Amazon ECS, and Azure Container Service:
* Facilitate auto deployment
* Scale application easily
* Quickly push application from one environment to another
* Enable automated rollbacks and backups
* Support load balancing and service healing

**Docker - Introduction**

Docker is a tool intended to make the process of creating, deploying and running applications easier by using container based virtualization technology.

Docker is an open source container technology that provisions far more apps running on the same old servers compared to traditional VMs.

**Docker Engine**

Docker engine is the Docker core component that is responsible for creating Docker Images and running them as services.

**Docker Engine Core Components:**

* **Docker Daemon**

Continuous running program (daemon process) that manages the service and other docker objects tied to it.

* **REST API**

Specifies interfaces that programs can utilize to speak to the daemon and direct it what to do.

* **Docker Client**

CLI is utilized to interact with the daemon (docker command).

**Docker on Linux and non-Linux Kernel**

* **Docker hosted on Linux:**

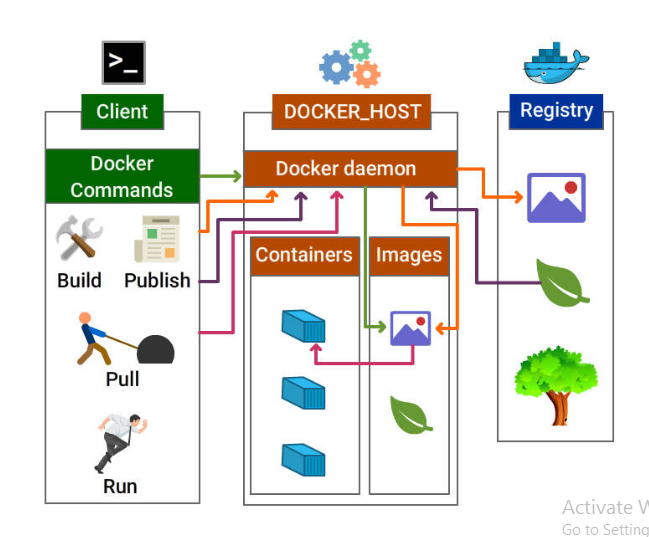
This requires just a Docker client and Docker daemon.

* **Docker hosted on non-Linux:**

Docker desktop for mac uses HyperKit VM, which handles virtualization.

Docker desktop for windows uses Microsoft Hyper-V to manage virtualization.

**Docker Architecture**



**Docker Components**

**Docker Daemon**

Docker daemon is the docker process that receives requests from docker client and is responsible for managing docker objects such as containers, images, networks, and volumes.

**Docker Client**

Docker client communicates with the docker daemon through Rest API calls. A docker client can send a request to many docker daemons.

**Docker Image**

Docker image is the collection of all files, libraries, binaries and other dependencies forming an executable software application, which can run everywhere without glitches.

* An image is an inert, immutable file.
* Docker image is read-only, i.e., the image and its content cannot be altered.
* Although the alteration is not allowed in Docker, we are allowed to add the new layer with the changes.
* After having many alterations, a docker image may be visualized as several layers one above another.

Parent and Child Images

* The layering concept in docker images leads to the addition of required capabilities efficiently by adding a new layer to the existing one resulting in a new image.
* Hence, the image has the parent-child relationship where the original image is termed as the base image upon which several child images are added.

**Docker Service / Container**

Containers are run-time instances of Docker images that can be run using the Docker run command.

The fundamental purpose of Docker is to run containers.

You can run a docker image to create as many docker containers as you want.

Docker Container Lifecycle

* create docker image
* docker service -running
* docker service -pause
* docker service -stop
* docker service -kill

**Docker Network**

The concept of networking in Docker comes into account when working with Docker in a real time scenario at a large scale.

Docker networking helps us to share data across various containers.

Host and containers in Docker are tied with 1: N relationship, which means one host can command multiple containers.

Various **modes for networking** is all about how we manage connections between containers.

* Bridge mode Networking
* Host Mode Networking
* Container Mode Networking
* No Networking

**Docker File**

Dockerfile is a script, formed of different arguments and commands (instructions) listed successively to automatically execute actions on a base image to form or create a new one.

A Docker File is a simple text file with instructions on how to build your images.

**Docker Registry**

* Docker Registry (Docker Repository) is a storage house for the Docker Images. It can be accessed publicly or privately by developers across the world.
* Docker images can be sent to registry by using docker push subcommand.
* Docker images can be downloaded from the registry using docker pull subcommand.

Following are the places where Docker registry can be hosted:

Docker Hub

AWS Container Registry

Google Container Registry and lot more

Docker Hub Define

Docker hub is one of the repositories of images which can be accessed at index.docker.io.

Docker Hub is the official repository by Docker development community. Any third party images can be pulled from the repository.

e.g.: docker pull thedockerbook/helloworld

**Docker Storage**

Storage drivers

* Container layer contains a very thin writable layer, unlike images that have read only layers.
* Each container has its storage layer whereas they share the read only image layer across containers in the same host.
* Docker uses storage drivers that will manage the data using copy-on-write mechanism.

Docker Storage.

* Copy-on-write mechanism
* Docker engine does not copy the whole image when we try to launch it. Instead, it uses Copy-on-write mechanism by which it uses a single copy of shared data until the data within the image is modified.
* This saves a lot of disk volume, and the startup process is quick.

Volumes

* Volume is a directory mounted in the container that is created using docker command.
* They are used to share data between containers by using the same volume across various containers.

**Docker workflow includes the following components:**

* **Docker Image** - Read only template that stores the application and environment.
* **Docker Container** - Runtime instance of a docker image
* **Docker registry** - Public and private repositories to store images
* **Docker File** - Automates Image construction
* **Docker file Compose** - Compose is a tool that can be used to manage multiple containers containing different applications.

**Docker Workflow - In Detail**

**Part 1**

* Create a docker file that includes details on the base and child images to be built.
* Build the docker file using docker build command and tag a name to the image.
* Verify if the image is built successfully using docker images command and Run docker inspect command to view the complete details of the image.
* Now the image is ready, push the same to the image repository using docker push command.

**Part 2**

* Pull the newly created image from the repository using docker pull command.
* Run the image using docker run command or using Dockerfile compose.
* Now you can verify the running container using docker ps command.

**Docker Basic Commands**

* docker version : Check the version of Docker:
* docker info : Check the detailed information on the running/stopped containers:
* docker pull <<image name>>: Docker images can be downloaded from Docker hub using docker commands.
* docker images ↵: Verify the downloaded docker images:
* docker history <<Image Name>> ↵: View all the commands that were run with an image via a container.
* docker rmi <<Image Name>> ↵ :Remove Docker Images
* docker run <<Image Name>> ↵ : Download and run an image in docker container using run command
* docker search Ubuntu | head -20
* docker pull busybox:1.24
* docker ps :List running containers
* docker inspect <Container Name> :Know the IP address of the running container

E.g. docker inspect nginxservice

* docker stats <<Container Name>> ↵ :Print the stats for a running Container

E.g. docker stats nginxservice

* docker pause <<Container Name >> ↵ :Pause the processes in a running container

E.g. docker pause nginxservice

* docker unpause <<Container Name >> ↵ :Unpause the processes in a running container

E.g. docker unpause nginxservice

* Export a container

docker export <<Container Name>> <<filename>>.tar ↵

e.g.

Let’s run a service using docker run command.

docker run --name newnginxservice -d nginx

docker export newnginxservice > test.tar

* Import a container

docker import <<Remote URL/Image Name.tar>> ↵

E.g. docker import test.tar

Docker daemon Commands

* Stop Docker daemon process

service docker stop

* Start Docker daemon process

service docker start

Diagnose Run Issues

* In case you are having a problem with downloading the images and running them, please follow these steps to check whether the docker service is running on your system or not:
* Check the running status of docker:

service docker status

* Restart Docker service in your system:

service docker restart

**EXAMPLE OF DOCKER**

**Step 1 - Create a Dockerfile**

Create a new directory DockerExample

* mkdir DockerExample.

Move the newly created directory

* cd DockerExample.

Add a file named Dockerfile

* vi Dockerfile.

####Dockerfile content###

FROM tomcat: jre8-alpine

# For wget to work

RUN apk update \

&& apk add ca-certificates wget \

&& update-ca-certificates

# Copy tomcat server.xml

WORKDIR /usr/local/tomcat

# Start tomcat

CMD ["catalina.sh", "run"]

**Step 2 - Build a Docker Image**

Build a docker image using below command.

* docker build -t tomcatimage .

-t --> Tags the name 'tomcatimage' to the newly created image.

Note: Do not forget to add a '.' dot at the end of the build command.

You can run the below command to view the complete details of the image.

* docker inspect tomcatimage

**Step 3 - Verify Image**

Verify the image retrieved by running command. This should list the newly created image.

* docker images

**Step 4 - Push to Registry**

Let us push the created image to Docker hub.

Set up a Docker Hub account. Fill your profile details on the main Docker Hub website and click Sign Up. Activate your account from the email sent.

Login to docker hub using the below command

* docker login --username <username>

key in the password once prompted.

Command to tag the image with the repository image name

* docker tag tomcat01 <username>/tomcatimage

Now let us push the image to the hub

* docker push <username>/tomcatimage

Now the image 'tomcat01' is available in the docker hub.

**Step 5 - Pull from Registry**

Now let’s learn to pull an image from the docker hub.

Login to docker hub using the same docker login command

Type the below command to pull the added image

* docker pull <username>/tomcatimage

Verify if the image is available by running docker command

* docker images

**Step 6 - Run Image**

To run the tomcat image, type the below command

* docker run --name tomcatRunner -p 8080:80 -d tomcatimage

tomcatRunner - container name

8080 - port of host machine

80 - port of container

-d - Run the daemon process in the background

**Step 7 - Verify Container**

Now let us verify if the new container created is running. Type in command

* docker ps

This will list the newly created container.