**Docker**

**Virtualization software**

Packaging an app into a container that has everything that it needs to run the application like the code, libraries, dependencies, runtime, and app configuration.

**How does development work without a docker?**

Developers used to install everything in their dev environment. All developer needs to install and configure the services on their local devices, which are different based on the developers' OS

With containers, u don’t have to directly install services in ur local machine

we have service packaged with its whole configuration in a container

so, dev can just run the docker cmd, and that fetches the container package from the internet and strts it on your device

So , docker Standardize the process of running any service on any local dev environment

**Application Deployment:**

Without docker

Dev produce artifact/package(jar) +installation instructions+ database+how to set it up in server

Dev give these to DevOps and they will configure it and set it up in server. This is very error-prone for 1. Miscommunication (dev -devops), 2.

Now with containers devs not only create package but also dependencies and configuration inside app artication so devops don’t need to configure that in server. They just need to run docker cmd to get the container package that devs created and run it in server [they just need to install docker and set up docker runtime just once]

**Docker Vs Virtual Machine**

How Docker works:

How OS are made:

1. OS Kernal : Core of every OS. iIt interacts between software and hardware. it communicates with hardware -cpu, memory storage etc.
2. OS Application layer

Docker:

Docker virtualize the application layers only. Services and apps are installed on top of that layer. It uses the host kernal

VM:

VM virtualizes application layer and Kernal. So when download a vm image on ur host, it doesn’t use host kernal , it boots its own kernel

SIZE-- Docker images , MB | VM images GB.

SPEED-- Containers take seconds to run | VM take min

Compatibility-- Docker compatable with linux | VM is compatable with all OS

For Docker compatability issue: if u wanna run a linux based docker image, it cannont run in windows since it cannot use windows kernal

Solution: Docker Desktop

This uses a hypervisor layer with a lightweight linux distribution that let u run linux based images

Image Vs Containers::

Docker Image:

Excetubale Application artifact that includes source code, environment config-OS layer, any services needed, application, environment var, directories, files

All these are packaged together in the image

Containers:

A running instance of an Image. One image can run multiple containers

When u have an image, u download it to server or local computer and run it in our OS, application starts and gives us a container

Docker Registry:

Staorage for all the docker Images. Companies create the official images and put in the biggest docker registry that’s called docker hub

**Docker cmds:**

To see docker image in ur machine:

docker images

to see running docker images:

docker ps

list all running and stopped containers:

docker ps -a

pull an image from registry

docker pull {name}:{tag}

Ex: docker pull nginx:1.29

Creates a container from given image and starts it

docker run {name}:{tag}

Ex: docker run nginx:1.29

Runs container in background and prints the containerId without blocking the terminal with logs

docker run -d {name}:{tag}

Specify the container name while creating the container[else docker will assign a random name]

Docker run --name {container name} nginx:1.29

Check logs from service running inside the container

docker logs {container Id}

stop running container

docker stop {container Id}

start a stopped container

docker start {container Id}/{container name}

How to access the container

Application inside container runs in an isolated Docker network. This allows to run the same app running on the same port multiple times

We need to expose the container port to the host (our local network)- to do that we would do port binding

Port Binding: Bind the containers port to the hosts port to make the service available to the outside world

U can do this while creating the container with a -p [--publish]tag:

docker run -p {Host\_port}:{container\_port} {name}:{tag}

Ex: docker run -p 9000:80 nginx:1.29

Now checkout <http://localhost:9000>, then check logs , we should see the request logs

Note: Docker run cmd creates a new container everytime , it doesn’t reuse the old container

Docker Registry Vs Repository

Registry

A service providing storage

Collection of repositories

Repositories

Collections of related images with same name but different versions

**Custom Docker Images for our app:**

Create a definition of how to build an image from our application

That definition is in DockerFile which is a text doc that contains cmds to assemble an image

Docker can then build an image reading those instructions

Dockerfile starts from a parent base Image. U choose the base image depending on tools u need to have

Base image ex- node, tomcat, python

Strucure of Docker file:

FROM [build an image from the specified image]

\*\*FROM node:{version} [for base image]

COPY [copies files from local to container path]

\*\* COPY package.json /app/

\*\* COPY src /app/

WORKDIR [set the working directory for nxt cmds, same as cd]

\*\* WORKDIR /app/

RUN [will execute any cmd in a shell inside the container env]

\*\* RUN npm install

CMD [for the last cmd, when docker container strts. There can be only one CMD]

\*\* CMD[“node”,”server.js”] [syntax CMD[“executable”, “param”] [basically its saying ‘node server.js ‘]

Finally run the Dockerfile:

docker build {path}

Ex: docker build -t {name}:{tag} .

Note: Docker image consists of layers . Each instruction creates one layer . Layers are stacked and each one is a delta of the changes from previous layer