# Dockers Step by Step

## Introduction to Docker :-

What is Docker ?

A platform for building, running, and shipping applications.

Dockers help consistently build, run and ship applications.

Reasons:

One or more files missing

Software version mismatch

Different configurations settings

This is when docker comes to rescue.

Package

Node 14

Mongo 4

App

Virtual Machines vs Containers

## Container :-

An isolated environment for running an application.

* Allow running multiple apps in isolation

Are lightweight

* Use OS of the host
* Start quickly
* Need less hardware resources

## Virtual Machine :-

An abstraction of a machine (Physical machine). 🡪 using Hypervisor (VirtualBix, VMware, Hyper-V (windows machine)

Run application in isolation.

Problems

* Each VM needs a full-blown OS
* Slow to start
* Resource intensive

## Architecture of Docker :-

Diagram

Description automatically generated

Containers is process like other processes running on CP. Container Share kernel of the host .

A Kerner manages applications and hardware resources. Every OS has it’s own kernel. Kernel has Different API’s.

On Linux we can run only linux container. On windows we can run both windows + Linux containers.

Mac OS own kernel don’t have. Native support (Docker on MAC uses Linux VM).

## Installing Docker :-

docs.docker.com/get-docker/

Windows Machine

Enabling Hyper-V and Containers windows features

Upgrade Linux Kernel on windows and restart machine

## Development Workflow :-

We take application and dockerised it which means it can run thru’ docker using dockerfile using image.

Image

* A cut-down OS
* A runtime environment (eg Node)
* Application files
* Third party libraries
* Environment variables

Once we have a image we tell docker using image start the container. Application gets loaded inside a container or a process and this is how we run application locally on machine.

Instead of directly running we tell docker to run our application on container in an isolated environment.

Once we have an image we can push it to registry (Docker hub) storage can any one use

Then we can pull on any machine.

A picture containing text, jack, first-aid kit, vector graphics

Description automatically generated

Walkthrough of development workflow

Mkdir hello-docker

Cd hello-docker

Code .

Create app.js

Instructions

Start with an OS

Install Node

Copy app files

Run node app.js

Hub.docker.com

Create Dockerfile

And give commands

FROM node:alpine

COPY . /app

WORKDIR /app

CMD node app.js

Now on terminal window run

Docker build -t hello-docker .

Docker image ls

Docker run hello-docker

Now we can publish to docker hub and pull on another machine

Dockerfile contains instructions for packaging application into an image. Once we have image we can run on any machine using docker.



The Linux Command Line

Introduction

Docker is built on basic Linux commands

Own versions of Linux Distributions

Open Source Software

Ubuntu

Debian

Alphine

Fedora

CentOS

Ubuntu Linux

Running Linux

Ubuntu Linux

hub.docker.com/search?q=ubuntu&type=image

docker pull ubuntu 🡪 use docker run ubuntu

docker ps 🡪 running containers

docker ps -a -> stop containers

docker run -it ubuntu 🡪 in interactive mode start container

echo hello

whoami

echo $0

bash -> born against shell

/ forward slash for directory for linux

Case sensitive

History

Managing Packages

Package Managers

Npm

Yarn

Pip

Nuget

In ubuntu apt for package managers

Apt( advanced package tool)

Apt-get

Apt update

Apt install nano

Before installing package use apt update 🡪 to update package

Then install the package

Apt remove nano

Clear terminal window my using control and L

Linux File System

Linux just like windows files and directories and like hierarchy

Graphical user interface

Description automatically generated with medium confidence

Everything is a file! in Linux

/ root directory

Bin – programs/binary files

Boot – botting

Dev – devices

Etc – configurations

Home – users (each users will have home directory)

Root – only root user can access

Lib – Libraries (Software lib dependencies

Var – variables like app data

Proc –Running processes

## Navigating File System

Pwd – current working directory

Ls

Ls -1

Ls -l

Cd relative or absolute path

Cd root 🡪 relative path

Cd ..

Cd ../..

Cd ~ 🡪 home directory

## Manipulating Files and Directories

Cd ~ 🡪 to go to home directory

Blue represents directory

Mv test docker

Touch hello.txt

Touch file1.txt file2.txt

Mv hello.txt hello-docker.txt

Control and W to remove word in one go.

Rm file1.txt file2.txt

Rm file\*

Rm -r docker 🡪 remove directory and all it’s content recursively

## Editing and viewing files

Apt install nano

Nano file1.txt

Cat file1.txt

Cat user for short file

For long file use more command

More we can only scroll down

To scroll up use less

Apt install less

Head to display first few lines

Head -n 5 /etc/adduser.conf

Tail -n 5 /etc/adduser.conf

## Redirection

Cat file1.txt > file2.txt

Cat command using redirection operator we can write in another file

echo hello > hello.txt

If you want to write a single line on file don’t use nano use echo command

Ls -l -> long listing

Ls -l /etc > file.txt 🡪 > redirect standard output

## Searching for text in file

grep hello file1.txt

grep -I hello file.txt -> case insensitive use i

grep -I root /etc/passwd

grep -I hello file1.txt file2.txt -> search text in more files

grep -I hello file\* 🡪 search In more files using wildcards

grep -I -r hello . -> search text in directory

grep -ir hello . -> we can combine commands to search text in directory

Files and Directories

Find -> finding files in directories

Ls doesn’t show hidden files and directories

Ls -a 🡪 to show hidden files

find -type d 🡪 only directories

find -type f -> only files

find -type f -name “f\*” -🡪 case sensitive

find -type f -iname “f\*” -> case insensitive use I command

find / -type f -iname “\*.py” > python.txt

Chaining Commands

Mkdir test; cd test; echo done

Mkdir test && cd test && echo done 🡪 to stop the command

Mkdir test || echo “directory exists”

Ls /bin | less 🡪 now less don’t need file name it get input from ls /bin

How to split commands in multiple lines

Mkdir hello;\

* Cd hello;\
* Echo done

Environment Variables

Storing application settings in configuration

Printenv

Printenv PATH

Echo $PATH

Export DB\_USER=rakesh 🡪 available only in current terminal window

Echo $DB\_USER

Printenv DB\_USER

Docker start -I 2f7(container id)

Echo $DB\_USER 🡪 doesn’t exist

To make it persistent we have to write in special file

Cd ~

Ls -a

.bashrc -> user personal startup file

echo DB\_USER=rakesh >> .bashrc 🡪 user append sign to append in bashrc

Don’t store sensitive information in bashrc file and it will be store in text file and anyone has access to the machine and view the password.

.bashrc is loaded only once when we start the terminal session

Source .bashrc -> reload the bashrc file

Source ~/ .bashrc

## Managing Processes

Process is an instance of a running programs

Ps command

TTY -> Telle Time

Pts/0 -> first sudo terminal window

Time -> cpu time

Create a processs and put it in background

Sleep 3

Sleep 100 &

Kill processed

Ps

## Managing Users

useradd

usermod

userdel

useradd -m john

user stored in config file in /etc/passwd

john:x:1000:1000::/home/john:/bin/sh

user:password:suerid:grpid::/homedirectory:/shell program use when this user logs in

usermod -s /bin/bash john

cat /etc/shadow 🡪 password stored in encrypted format and only accessible for root user

open new terminal window

docker exec -it containerid bash

docker exec -it -u john containerid bash

adduser -> postscript and more interactive and uses useradd underthe wood.

Adduser bob

We have chance to set the password

And specify additional Information

## Managing Groups

To provide same permissions

groupadd developers

cat /etc/group

Now add john to grp

-g 🡪 one primary grp

-G -> 0 or more secondary grps

Primary grp is created when we create user with same name as user

usermod –G developers john

cat /etc/passwd | grep john or grep john /etc/passwd

groups john

## File Permissions

Login as root d

Deploy.sh

Echo echo hello >deploy.sh

Cat deploy.sh

Ls -l 🡪 to check permisions of files

Cat deploy.sh

ls -l

drwxr-xr-x 2

-rw-r--r-- 1

D means directory

* Means files

9 letters divided in 3 grps

Rw means read and write permissions

First grp permission for user who owns this file

Second grp permission for group users

3rd grp permission for everyone else

Chmod u+x deploy.sh

./deploy.sh

Chmod o+x deploy.sh

Chmod og+x+w-r deploy.sh

## Building Images

Image includes everything which application needs to run (all files and conf settings)

Containers is kind of like a VM provides an isolated env.

Can be stopped and restarted

Is just a process !

Steps

Install Node

npm install

npm start

We will dockerised this application and package into an image.

## Docker file Instructions

A dockerfile contains instructions for building an image.

DockerFile

From

WorkDir

Copy

Add

Run

Env

Expose 🡪 container starting on given port

User

Cmd

Entrypoint

### Choosing the right base Image

FROM 🡪 Os+runtime env 🡪 docs.docker.com/engine/examples/dotnetcore/

Docker build -t react-app

Docker image ls

Docker run -it react-app

## Copying Files and Directories

FROM node:14.16.0-alpine3.13

WORKDIR /app

COPY . .

## Excluding Files and Directories

.dockerignore

node\_modules/

npm install to install project dependencies

alphine don’t have apt it has apk

## Setting Environment Variables

ENV API\_URL=http://api.myapp.com/

Printenv API\_URL

Echo $API\_URL

## Exposing Ports

EXPOSE 3000

Container will listen on port 3000

## Setting the USER

Addgroup add

Adduser -S -G app app

Groups app

RUN addgroup app && adduser -S -G app app

## Defining Entrypoints

Docker run react-app npm start

If you have multiple cmd instructions in docker file then only last one will take effect

`RUN instructions are building instructions

Cmd instructions are runtime instructions

#shell form

CMD npm start

#Exec form

CM [“npm”, “start”]

## Speeding Up Builds

Image is collection of layers

Docker history react-app

To optimize we should build dockerfile like that the instructions which don’t change frequently should be on top and instructions which changes frequently should be at the bottom

## Removing Images

docker images

dangling images (loose images) 🡪 When we were building image docker was creating image and some point these layers lost relation.

Docker image prune

Container in stop state

Docker container prune

## Tagging Images

Always use explicit tag for development we can use latest tag but for production always use explicit tag.

docker build -t react-app .

docker build -t react-app:1 .

docker image remove react-app:1 .

docker image tag react-app:latest react-app:1

latest doesn’t apply to be latest image we have to explicitly tag to make it latest

docker image tag b06 react-app:latest

docker images

## Sharing Images

Docker image tag b06 rmk123/react-app:2

First we need to login

Use

docker login

docker push rmk123/react-app:2

## Saving and Loading Images

Save and load in another machine without going thru’ docker hub

docker image save -o react-app.tar react-app:3

docker image load -i react-app.tar

docker images

## Working with Containers

### Starting containers

docker run react-app

docker run -d react-app (Run container in detach mode i.e in background)

docker run -d –name blue-sky react-app (To give container a name)

### Viewing the logs

docker logs 1fd5

### Publishing ports

docker run -d -p 80:3000 –name c1 react-app

### Executing Commands in Running Containers

docker exec c1 ls 🡪 command in running container

docker exec -it c1 sh

### Stopping and Starting Containers

docker stop c1

docker ps

docker start c1 🡪

docker run we start new container

docker start we start the stopped container

### Removing Containers

docker rm c1

to remove running container we can use force option

docker rm -f c1

docker ps -a | grep c1 🡪 to check stop container using grep command

docker container prune -🡪 to remove stopped containers in 1 go

### Container File System

docker ps

never store data in container file system

docker exec -it 655 sh

/app $ echo data > data.txt

/app $ exit

docker exec -it 6eb sh

/app $ ls | grep data

/app $

### Persisting data and volumes

Volume is a storage outside of container it can be directory on the host or somewhere on the cloud.

docker volume

docker volume create app-data

docker volume inspect app-data

docker run -d -p 4000:3000 -v app-data:/app/data react-app

docker exec -it 716 sh

/app $ cd data

/app/data $ echo data > data.txt 🡪 this will not get write permission so to achieve this go to dockerfile and modify the details

Also, we can share volumes between multiple containers

### Copying Files between the Host and Containers

docker ps

docker exec -it e1c sh

/app $ echo hello > log.txt

/app $ exit

docker cp e1c:/app/log.txt

ls

### Sharing Source code with a Container

Publishing Changes

Production Machine -> Build a new Image tag it properly and deploy

Development 🡪

don’t rebuild for every change

Copy files don’t manually copy from dev machine to container

We can create mapping or binding between a directory on host and directory inside a container

This way any change we make immediately visible in container

Docker run -d –p 5001:3000 -v $(pwd):/app react-app

Docker logs -f 696

## Running Multiple Container Applications

* Installing Docker compose

Docs.docker.com/compose/install/

Docker compose built on top of docker engine

Easy to start multiple docker containers

For Mac and windows docker compose is already present on machine

docker-compose --version

* Cleaning up Our Workspace

docker image ls -q 🡪 you will only get Image ID’s

First remove containers and then it’s images

docker container rm -f $(docker container ls -a -q) 🡪 -a will bring stopped container as well

docker image rm -f $(docker image ls -q)

* Sample Web Application
* Json And YAML Formats

Parsing YAML is slower than parsing json files

YAML file for configuration files and json file for exchanging data between multiple computers like client and server.

* Creating Compose File

Each service in docker compose should have their own dockerfiles

* Building Images

docker-compose build –no-cache

* Starting and stopping the application

We can combine docker-compose build and up using docker-compose up --build

docker-compose up

other useful option is docker-compose up -d 🡪 detach mode

docker-compose ps 🡪 all containers related to our application in contrast if we type docker ps it will show all running containers.

docker-compose down

* Docker Networking

When we start our application with docker compose, docker compose will automatically start container and add to a network.So these container talk to each other

docker network ls

containers can talk to each other with there names

docker exec -it 8c6 sh

ping api

we need to login as a root user to ping container

docker exec -it -u root f7c9. Sh

Docker comes with an embedded DNS server that contains name and ip of container

Inside a container we have a component called as a DNS resolver. DNS resolver talks to DNS server to find the IP address of the target container. So when we ping api container,

So, each container has the ip address and it’s part of the network

* Viewing Logs

docker-compose logs 🡪 It will give all containers logs in 1 place

docker logs 8cd -f 🡪 specific container logs

* Publishing Changes

Every time code is changed we don’t want to rebuild our application images.

volumes

- ./backend:/app 🡪 host directory mapping with container one

* Migrating the database

Docker volume ls

Docker voume rm vidly-vidly

* Running Tests

Deploying Applications

* Deployment Options

1. Single-host deployment
2. Cluster deployment (Group of servers)

Cluster Solutions (Orchestration tools)

1. Docker Swarn
2. Kubernetes 🡪 google products

* Getting a Virtual Private Server

VPS Options

Digital Ocean

Google Cloud Platform (GCP)

Microsoft Azure

Amazon Web Services (AWS)

* Installing Docker Machine

A picture containing chart

Description automatically generated

github.com/docker/machine/releases

docker-machine –version

* Provisioning a Host

Works on only powershell not on command prompt on windows -- docker-machine create \

* --driver digitalocean \
* --digitalocean-access-token
* Connecting to the Host

docker-machine ls

connect to server we will use ssh

docker-machine ssh vidly

* Defining the Production Configurations

Compose file formats

* Reducing Image Size

Optimize asset for production