DOCKER

Date

- Container
  - A way to package application with all necessary dependencies and configuration
  - Portable artifact, easily stored & moved around
  - Makes development & deployment more efficient
  - Lightweight in nature

→ Common Problems Occuring When replicating an env.

(1) Need to manually install all dependencies
(2) There might be some version conflicts.
(3) Our system depends on a specific version of a dependency and when replicating we installed the latest one which would cause bugs.
(4) Installation problems.
(5) Some CLI Cmds that run on Linux might not run on MACOS / Windows.
(6) Installation process on each OS is different

→ A container helps us to bundle our application along with it dependencies & configurations in a single unit
↳ It is basically a standardized way of replicating enviornments

→ container is a running env for the image

Date

→ Each container has it's own isolated enviornment
→ Single cmd to install the app

class → image
object → container

- Image
→ Docker image is a blueprint that helps build/creat container
→ Docker image & container has similar relationship like that of Class & Object.
↳ Class is a blueprint that helps us create multiple objects
→ A single docker image can create multiple containers
→ Basically we share images with developer than from images containers are created
→ System resources are used by container not images same as the case is with class & object
→ Docker image is a static snapshot of what the local dev env should look like

actually these are also images

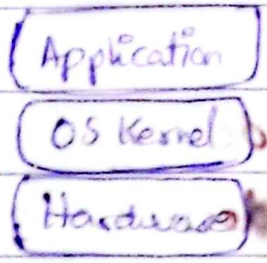
→ A dockes ~~container~~ image consist of multiple layers. like when we pull an image different layers of images are being downloaded. Like if I pull

→ Each image has a specific tag (version)

Docker desktop runs a small linux based VM that helps run our containers
↳ Docker was initially built for linux

Date

psql version 6 so it will download multiple layers.
And now if we install v10 so the layers which already exist will not be downloaded.



▷ Docker Vs Virtual Machine

↳ Docker only virtualizes the application layer whereas VM virtualize App + Kernal
↳ The VM has it's own kernel, docker uses kernel of host machine
↳ Docker is lightweight
↳ Docker is much faster

▷ Docker Commands

- docker pull IMAGE_NAME → fetches image from dockerhub
- docker images → List all images
- docker run IMAGE_NAME → fetches image if not available & run the container
  -i → to accept std input
- docker run -it IMAGE_NAME
  ↳ runs image in interactive mode
  -t ↳ terminal
- docker ps → To view running containers
- docker ps -a → View all containers

Page No.

docker inspect CONT_NAME → to get details about container

Date

- docker start CONTAINER_NAME or ID
  ↳ to start an already created container
- docker stop " or ID

- docker rmi IMAGE_NAME → used to remove an image
- docker rm CONT_NAME → " " " container
  ↳ to remove an image we would 1st need to remove all the containers build from it
- docker run -d redis → runs container in detached mode (run in background)
- docker run redis:4.0 → specifying version

- docker run -d -e MYSQL_ROOT_PASSWORD=hello mysql
  ↳ environment variable (must of mysql container)

- Giving custom names to container

docker run --name my-container mysql:8.0

↳ terminal

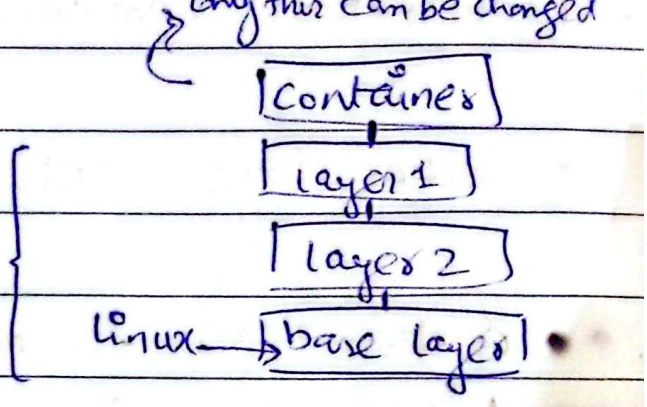
→ To attach to a container after running it in detached mode
↳ docker attach CONT_NAME/ID

Page No.

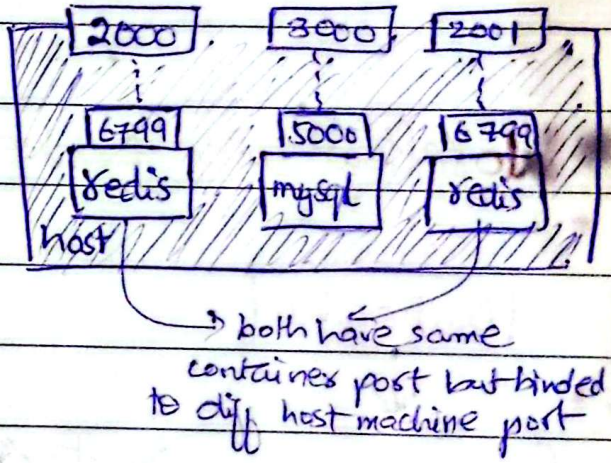
Unlike VM's containers are not meant to run Operating System. Therefore when we run ubuntu image it exits immideatly. Containers only runs when process inside it is alive

Date

- Docker Image Layers



- Port Binding
  - ↳ Each container has a port associated with it
  - ↳ Therefore if we want to access redis we need to bind it to a host machine port



- docker run -p2000:6799 redis
  - ↳ Now the 2000 port of Host is binded with one container
    If we try to bind any other containes with 2000 it will give error

Page No.

We can also limit the amount of host's CPU utilization by a container

docker run --cpus=.5 ubuntu → container will not utilize more than 50% of host CPUs

docker run --memory=100m ubuntu

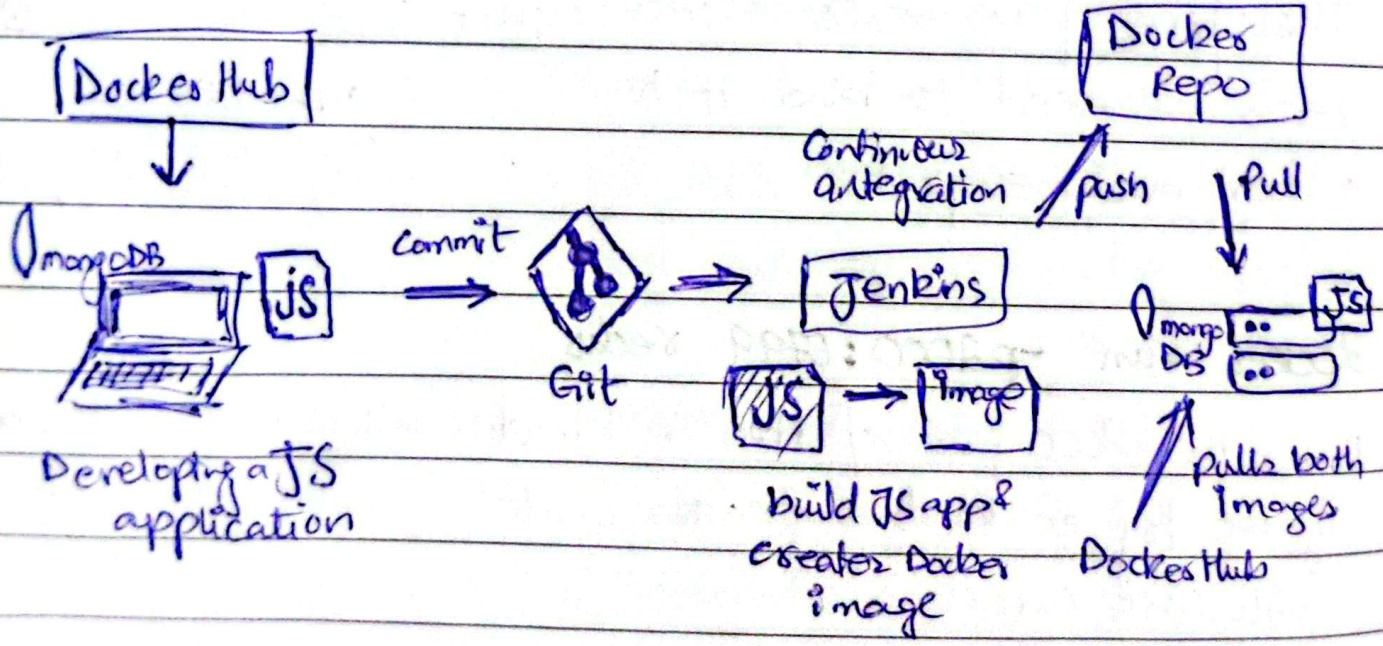
Date

- Troubleshoot Commands

  - docker logs CONT_NAME

  - docker exec -it mysql-latest /bin/bash
  - docker exec -it mysql-latest /bin/sh

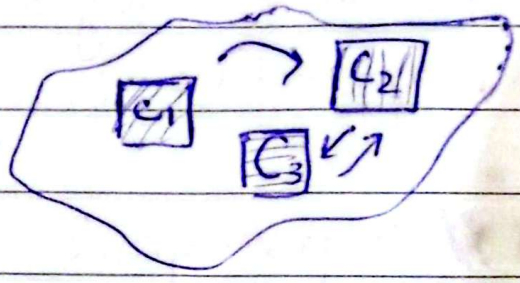
- Workflow with Dockers



Date

- Docker Network

→ If we want two containers to directly interact with each other without using any port or local host we can make network.



- docker network ls → view all docker networks
- docker network create NETWORK_NAME
- docker network rm "
- Running a MongoDb container

```
docker run -d \
> -p 27017:27017 \
> --name mongo-db \
> --network mongo-network \
> -e MONGO_INITDB_ROOT_USERNAME=admin \
> -e MONGO_INITDB_ROOT_PASSWORD=query \
> mongo
```

→ Docker has a built in DNS that maps the containes name to the pvt IP of the contain. So we can connect the apps using container name.

→ IP of DNS → 127.0.0.11

Page No.

Date

- Docker Compose

  - Instead of using cmds on terminal we can run the cmds thru .yaml (yet Another Markup Language)
  - It is basically a tool for defining and running multi-container applications

// compose.yaml (for mongo containes)

```
services:
  mongo:
    image: mongo
    ports:
      - 27017:27017
    environment:
      MONGO_INITDB_ROOT_USERNAME: admin
      MONGO_INITDB_ROOT_PASSWORD: qwerty

  mongo-express:
    image: mongo-express
    ports:
      - 2021:8081
    environment
      ME_CONFIG
```

Page No.

Date

→ Compose by default creates a seperate network for all the service in the file

- docker-compose (-f) compose.yaml up -d (-f → file)
- docker-compose (-f) 4 down → delets all container
  → will also delete data

- Dockerizing App
  ↳ Every image is build using a Dockerfile
  ↳ This file contains all the basic commands that are used to build an image.
  ↳ This processes of making our app into images is done by Jenkins
  ↳ "Dockerfile is basically a blue print to build a Docker image"
    ↳ It contains all instructions to dockerize our app

testapp
node
alpine
(the node itself has a base image)
(a node application needs node image)

FROM node → every image is build on a base image (layered structure)

WORKDIR → here we specify our working directory
  ↳ where files will be copied & cmds will be executed

COPY → to copy files from HOST to Image

ENTRYPOINT → FROM Ubuntu → docker run ubuntu-sleeper 10
ENTRYPOINT ["sleep"] ← the 10 appends here
or
CMD ["sleep", 10] → it is now hardcoded if we need to change sleep value we should change dockerfile each time

RUN → helps to run multiple cmds
>> we can have multiple RUN cmd but single CMD cmd

CMD → The last cmd which the container of this image will use to run.
like "python manage.py runserver"

EXPOSE → To expose the ports of an image
ENV → help defin env variables

// Dockerfile → for node application

```
FROM node                          → recommended to use env in compose
ENV  MONGO_DB_USERNAME=admin \
     M  4    PASSWORD=query
RUN  mkdir -p testapp
COPY . /testapp
```

. → current folder of host machine where we have dockerfile
/testapp → destination folder of container

testapp/
- Dockerfile
- src_code
- compose.yaml

```
CMD ["node", "/testapp/server.js"]
```

Page No.

any name of image

specify any tag (version)

Date

docker build -t testapp:1.0 . ← location of file

-> If we do any changes in Dockerfile we need to rebuild the image.

- Pushing Docker image

(1) Create any repo in DockerHub (samozeali)

(2) Now run docker build -t samozeali . ← should be same

(3) We now need to login

docker login

(4) After login then do docker push

- Docker Volumes

↳ Volumes are pessistent data stores for containers

- docker run -v /users/samoze/desktop/data/myfolder:/test/data ubu

absolute path of host machine — container path

↳ In Compose-yaml

services:

mongo:

...

volumes:

- /user/samoze/desktop/data:/data/db

Page No.

Date

- docker volume ls → to view all volumes
- docker volume create myVolume
  - but it is isolated right now, not attached to any containers
  - ↳ by default the volume will be created at C:\ProgramData\Docker\Volumes
- docker volume rm myVolume

- Types of Docker Volume
  1. Named Volumes — docker run -v /opt/datadir:/var/lib/mysql
     - docker run -v VOL_Name : CONTAINER_DIR
       - ↳ most prefered way
  2. Anonymous Volumes
     - docker run -v MOUNT-PATH (container directory)

  } Volume managed by docker

  3. Bind Mount
     - docker run -v HOST_DIR : CONTAINER_DIR

  } volume managed by host machine

- docker volume prune → To delete unused volumes
  - ↳ by default targets anonymous volumes

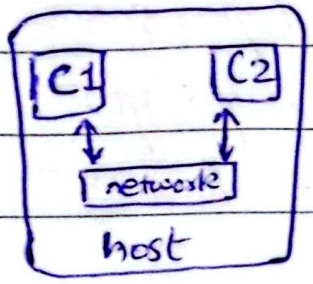
Page No.

Date

- Docker Network Drivers :

(1) Bridge

↳ It is the default network drives



(2) Host

↳ Removes network isolation b/w containers and the Docker host. Removes the IP address from containers Uses the host's network directly.

(3) Null

↳ When we need to make an isolated container. That doesn't interact either with any other container or host