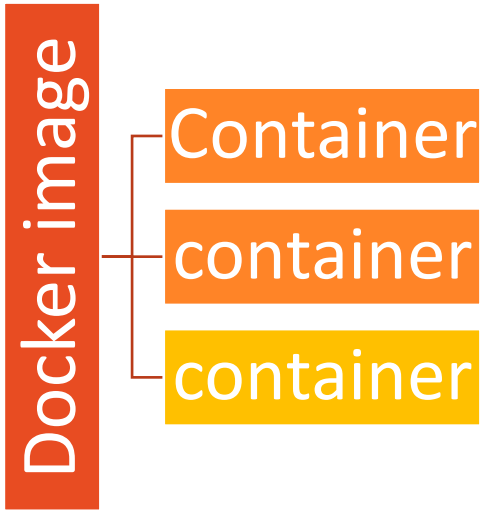
Docker

Docker is a ecosystem or platform for creating and running containers



Docker image is a single file consists of all code and dependencies to run a program- our java code files+jdk+tomcat

Image is a single file that stores on a hard drive

My project image:- generally we write code and to run the code we need jvm and tomcat server

Container -we will create container from the image

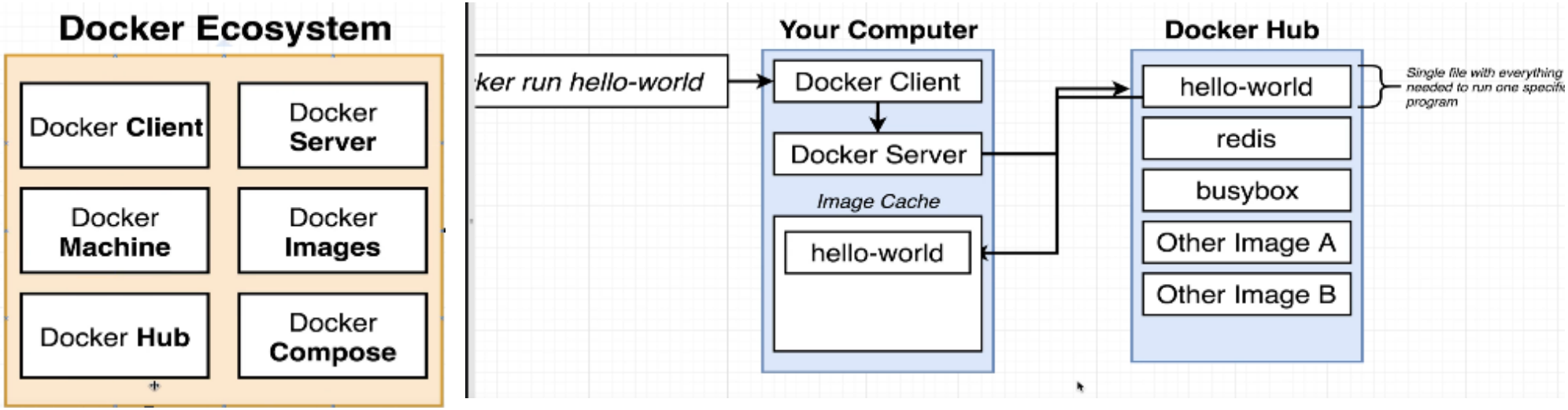
It's a isolated space in HDD, where all the mentioned s/w in the image will be installed

Container== a Running program

Container is a program running with an own isolated

set of resources

We will use the image to create a container, and container is isolated



Analogies:-

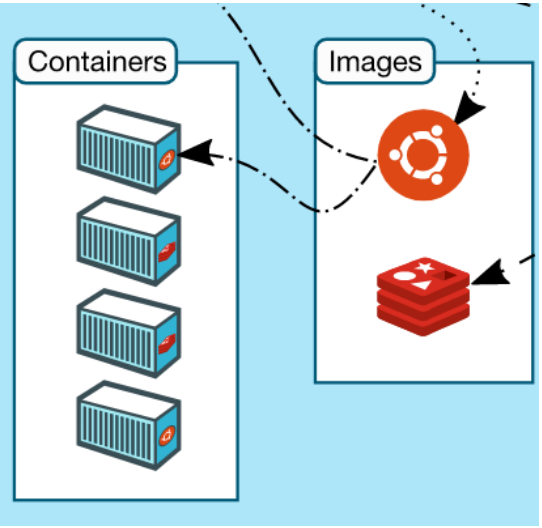
1) Container =Separate Drum, 1 drum with petrol,1 drum with diesel

In shipping industry, we had a problem while shipping the goods, like shipping in sea, by road,by flight

So they have chosen container as best way, so that goods in 1 container is completely isolated from others

Like transferring the entire fish +acquarium to another house while u are going on another vacation

1) Here fish acuqarium is a container- having fishes+ water+food+oxygen+stones..



Tech analogies

This docker is most similar to java

Java	Docker
We will write source code in .java	We will write source code in Docker file Here we will write what s/w to be installed
Once we build(compile and package) we will get a jar with group id and artefact id	Here once we build , we will get an image and We can tag that image with some name- Docker build –t <custom image name> .
That jar can be deployed in any jvm/any server And we can create object for those classes	Once we run that image,container will be created which is nothing but some separate space in HDD where all those softwares mentioned in the image will be installed

Points to remembers

```
C:\Users\Manideep>docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED         STATUS         PORTS
0e9698d27ca5   7ec45d18a115   "docker-entrypoint.s…"   2 hours ago     Up 2 hours     0.0.0.0:8
```

Once docker is installed, u can run any command even through command prompt and through Ubuntu both--------

When u are running in windows, you would have installed WSL, means windows subsystem for Linux, here a linux virtual machine will be created on your

Windows machine, so even though u are on windows machine, all the commands will be executed inside a linux machine

Every image has a startup command

1) What is there in the image??

   Software name to be installed, like image contains software names like-JVM,Tomcat...,node.nginx

   User created images will be pushed to https://hub.docker.com/ , It's a repository so that someone can pull the image from it.

2) Where those s/w needs to be installed?
   it should be installed in separate isolated space in the same o.s, Since we don't have multiple o.s ,as we have single O.S,

Container is nothing but some isolated space in a HDD.
Simple yaa, they have mentioned s/w names in the image, those software's should be installed in some separate space in Hdd na

3) Instance of an image== nothing but ,Object of a class
4) The files and folder u are creating in your local windows are present in the below folder
   \\wsl.localhost\Ubuntu\home\manideepvv
5) We can create a container from an image. Because image has all the softwares mentioned to install and
   When we create a container, an isolated space will be allocated to us, to install all mentioned s/w present in the image,
6) From a running container also we can create a snapshot and we can create a image from a running container
7) In java if u build, u will get a jar file, that jar file u can use in runtime to create objects of those classes, same way,
   in docker if u build u will get image, and u can create container which is some isolated space in system,where u can install s/ws
   mentioned in the image

-it ==means connect my terminal to the input stream

Sh== it will get a console inside that container

8) Docker run <image-name>
   This command will try to create a container from the given image, if its present locally it will use
   Else it will try to download from docker hub,no need to pull it first like "docker pull nginx"

About docker

1) If image is found locally it will use it,else it will download from internet

Problem that docker solves

Before docker, only code is packed to jar files and that code is being exported to another environment

But the problem here is ,in ur local jdk8 ,tomcat7,might be there, whereas in dev,SIT environment

Diff version might be there,

So the solution is –pack all the required softwares, + along with code

Container-package of code+required softwares

Now the problem will be solved, in dev same softwares,in QA also same software

Tips to remember

1) In local practice, u need internet
2) Running with starting digits of container id
   Lets say u ran the above command and u got a Big image id as 8f09..

docker commit -c 'CMD ["redis-server"]' db1e82040a3e

sha256:8f09e980c91e4e9c6297b6dc6b73770f878e5be95c6bd3e0fb7004e028f13b0c

And If u want to create a container from it , u just need to key only starting few digits of container id

docker run 8f09e980

Question and answers

1) What is a container

Container is nothing but a separate isolated space inside HDD where all the softwares mentioned in the Image will be installed,like in image if u mention s/w like, jdk,nginx, those 2 softwares will be installed In the container

Same image can be used to create multiple containers, like same java class to create multiple objects

2) Relationship between image and container

Ans:- whatever the software u mentioned in the image, those will be installed in a isolated space inside a container

Container is nothing but some isolated space inside HDD

```
1  FROM node:alpine as builder
2  WORKDIR '/app'
3  COPY package.json .
4  RUN npm install
5  COPY . .
6  RUN npm run build
7
8  FROM nginx
9  COPY --from=builder /app/build /usr/share/ngin
```

Here in the image, u mentioned s/w like node, nginx, then those 2 softwares will be installed in the HDD isolated space

3) Difference between image id and container id

Once u built using a docker file, u will get image id /image name

docker build -f MultiStepDockerfile .

```
manideepvv@DESKTOP-48ALSQ8:~/frontendapp$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
manideepvv@DESKTOP-48ALSQ8:~/frontendapp$ docker build -f MultiStepDockerfile .
Sending build context to Docker daemon  728.5kB
Step 1/8 : FROM node:16-alpine as builder
 ---> 448a08f1d2f9
Step 8/8 : COPY --from=builder /app/build /usr/share/nginx/html
 ---> Using cache
 ---> 23544802c1ee
Successfully built 23544802c1ee
manideepvv@DESKTOP-48ALSQ8:~/frontendapp$ docker run -p 8080:80 23544802c1ee
```

When we built, we got an image name/jar file ,each jar file having groupid, artifact id

Same way every image have image name, using that we can create container

docker run <image id>

once u created a container (separate isolated space in HDD and all software's will be installed)

Whereas once u started container, u will get the container id from "docker ps" command

```
manideepvv@DESKTOP-48ALSQ8:~$ docker stop 23544802c1ee
Error response from daemon: No such container: 23544802c1ee
manideepvv@DESKTOP-48ALSQ8:~$ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED
6ba65c575824   23544802c1ee   "/docker-entrypoint.…"   4 minutes a
manideepvv@DESKTOP-48ALSQ8:~$ docker stop 6ba65c575824
6ba65c575824
manideepvv@DESKTOP-48ALSQ8:~$
```

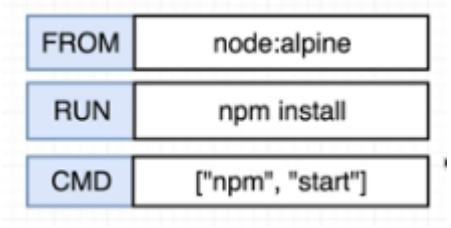
If u see here o tried "docker ps", we can see the running container id

4) Base image example

FROM node:14-alpine

//Generally every image will have mostly alpine versions, it's the most stripped and mini version of the image

It's the lightest version of it.it will have the basic programs



here node is the repository and alpine is the tagged version.

4) How to create a container?

1) First create a docker file –not .TXT file , just normal file type only
2) Build=== means ==Building an image==create an image for the Docker file with below 2 options
   1) docker build ."
   2) docker build -t manideep/redis:latest .

```
The command '/bin/sh -c npm install' returned a non-zero code: 1
manideepvv@DESKTOP-48ALSQ8:~/simplewebapp$ docker build -t manideep/npm:v1
Sending build context to Docker daemon  4.096kB
Step 1/4 : FROM node:14-alpine3.16
 ---> 79912fb6c330
Step 2/4 : COPY ./ ./
 ---> Using cache
 ---> 31782901be76
```

```
Removing intermediate container 55410350d966
 ---> b458f5751b09
Successfully built b458f5751b09
Successfully tagged manideep/npm:v1
manideepvv@DESKTOP-48ALSQ8:~/simplewebapp$
```

With this we successfully tagged the image with the name

3) Create a container from the existing image

docker run manideep/npm:v1

```
manideepvv@DESKTOP-48ALSQ8:~/simplewebapp$ docker run manideep/npm:v1

> @ start /
> node index.js

Listening on port 8080
^C
```

4) Why we need a image to create a container?
   Because container is nothing but a separate space in HDD, where as we need image, because in image
   Only we will mention which software names to install in that separate isolated space
5) How many processes can be inside a running container?
   Can be many, like in our sys we have word,chrome,edge.. lly in docker also many processes can run parallelly

6) What is diff b/n docker run and docker exec

docker exec –it <container-id > sh

docker exec will execute that command inside a running container whereas docker run will create a new container

docker run <image id>

for docker run we are giving image,simple from image we can create only container

7) why we need docker-compose.yml file

lets say we have 2 docker files, if we want to create 2 containers, we have to build 2 docker files to get 2 images and then we have to execute docker run <container-id> 2 times.. to get 2 containers from those 2 images so total 4 commands ,instead 1 command "docker run <image>" (if u give image name,based on that container will be created)

8) Tips while creating yml file

While writing yml file, always in next line, you have to give 2 spaces and start in next line

```
  api:
    build:
      dockerfile: Dockerfile.dev
      context: ./server
```

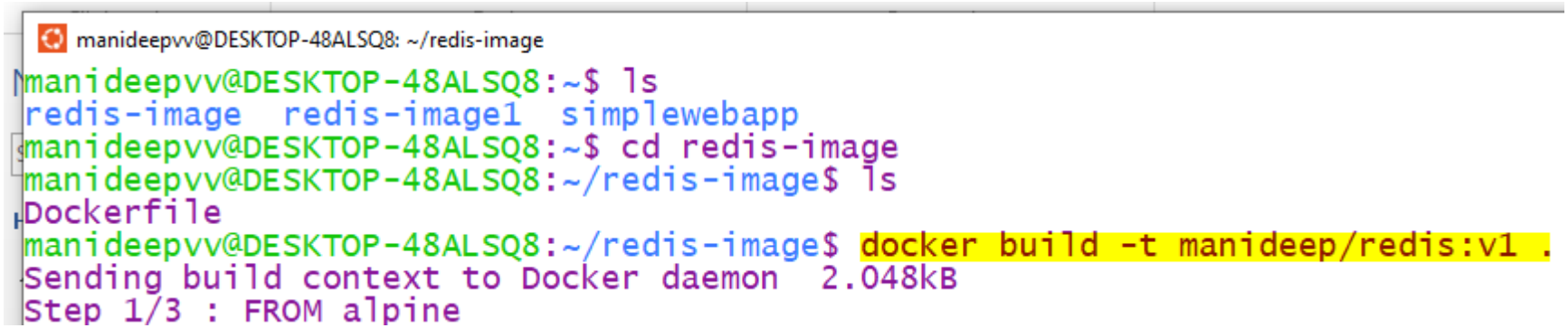
here if u see, in second line "build" was written after giving 2 spaces

word "dockerfile" is also written in 3rd line after giving 2 spaces under word "build"

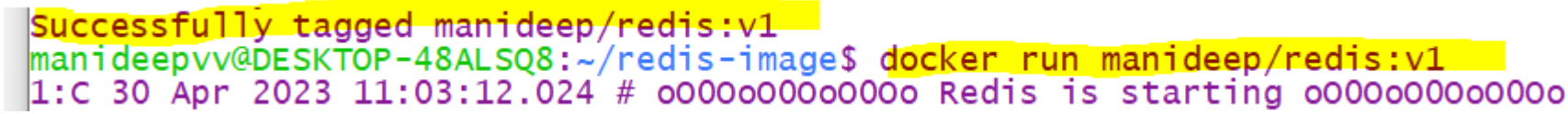
Creating image from docker file and running it

1) Same like java- first u will write .java file→ build → u will get .class file → can be pushed to mvn repo
2) Here we will create a Docker file→ build → image → can be pushed to docker repo
3) Create an image –while creating a image u can give a custom name
4) If u build u will get jar file-same thing, if u build u will get image file

   1st step is -1st u need to build the image using jar file



   Here I created a image and tagged a custom name to it.



   With that custom name, u can run that image, so that separate space will be created and required s/w mentioned in that Image will be installed

```
manideepvv@DESKTOP-48ALSQ8:~/simplewebapp$ docker build -t manideep/rnpm:v2 .
Sending build context to Docker daemon  4.096kB
Step 1/4 : FROM node:14-alpine3.16
 ---> 79912fb6c330
Step 2/4 : COPY ./ ./
 ---> Using cache
 ---> 31782901be76
Step 3/4 : RUN npm install
 ---> Using cache
 ---> c684695ed403
Step 4/4 : CMD ["npm","start"]
 ---> Using cache
 ---> b458f5751b09
Successfully built b458f5751b09
Successfully tagged manideep/rnpm:v2
manideepvv@DESKTOP-48ALSQ8:~/simplewebapp$ docker manideep/rnpm:v2
docker: 'manideep/rnpm:v2' is not a docker command.
See 'docker --help'
manideepvv@DESKTOP-48ALSQ8:~/simplewebapp$ docker run manideep/rnpm:v2

> @ start /
> node index.js

Listening on port 8080
```

1) To create a image from docker file – 1st we can create image with default name OR create with custom tagged name

   docker build-t <image name> .

   docker build –t projectName/RepoName:verison .

   docker build -t coreiq/npm:v1 .

-t represents tag the image with this name

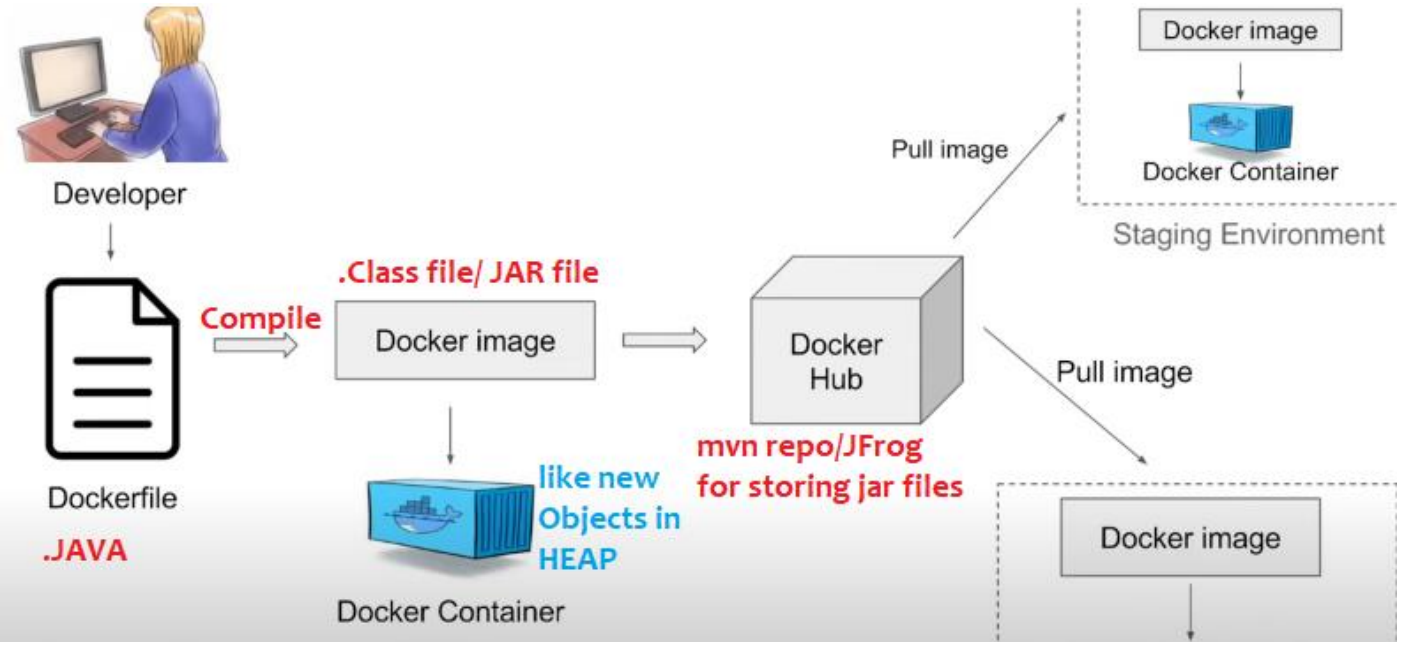
Dot . represents the current directory where the docker file is residing

2) To run the image use below command

docker run <tagged image name>

docker run coreiq/npm:v1

Docker flow

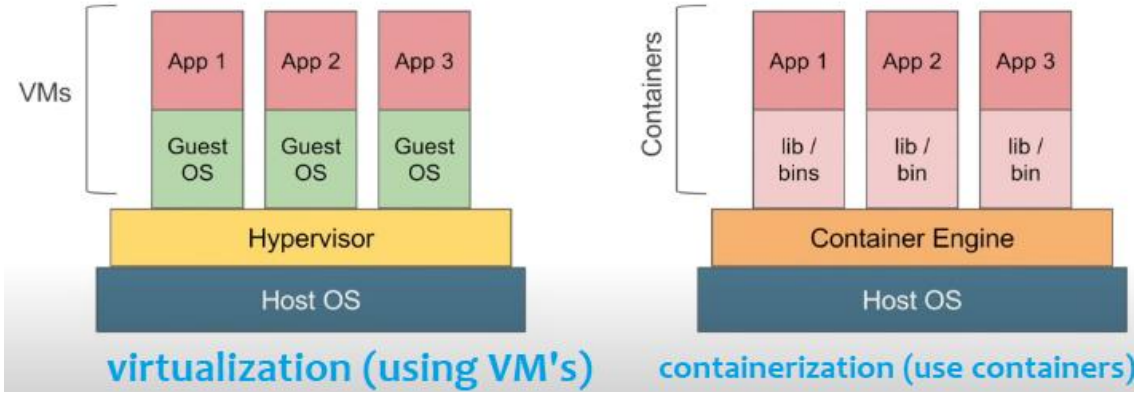


Here to run the application, if we need tomcat, JDK then we have to mention those as dependencies in docker file

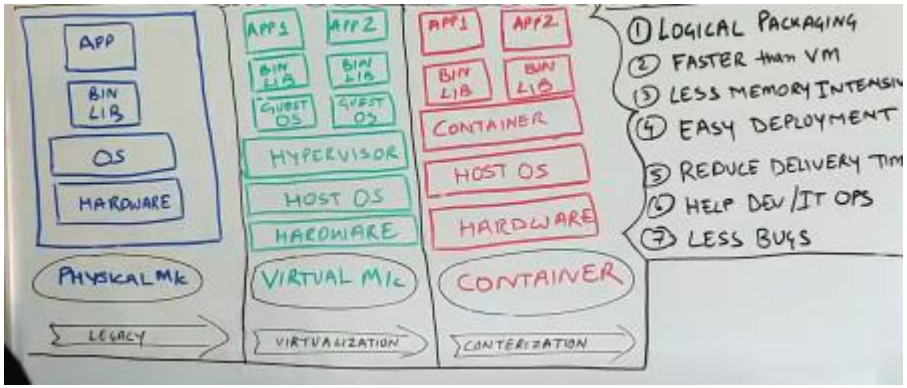
Once we compile/... then we will get docker image

We write java files	We write docker file- we have to mention all the dependencies what we want in this file like jdk,tomcat..
After compiling we will get .class file	After compiling we will get docker image file
All classes bundled to jar and we will push that to mvn repo	All docker images are stored in docker hub

Download that jar file, we can create instances of that class	We can download that image we can create instances of that image which is called as container
Same like local MVN repo where we store all the jars	In docker also we will store all the docker images locally, if we don't find it we will pull from docker hub

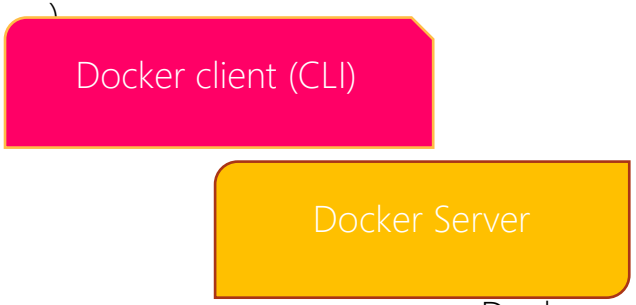


In containerization we don't have any guest os and all, containers are simple and light weight



Docker client (CLI) vs Docker server

Docker client is just a CLI, (its same like oracle command prompt,its not mail one, all the queries are again fired against oracle server)



Docker server is the one who creates the images and running containers..etc

Its same like oracle server, we can't interact with oracle server directly , we can interact via command prompt, this is also same thing, we can not interact with docker server directly we can go via Docker CLI only

NameSpace /segmenting the hardware for isolation

Namespacing = Isolating resources

Namespaces are used to provide isolation for running processes.its a os level feature, it provides segmenting the hardware ,make hard disk into segments,segment -1 have python-2, segment-2 install python -3, generally without segmentation we cant install 2 versions of python in same computer(but I still think we can install all versions at same time by changing path while installing)

With namespacing we can isolate resource per each process or for each application

When u run a container docker will create a set of namespaces

With name spacing we can restrict the area of hard drive available

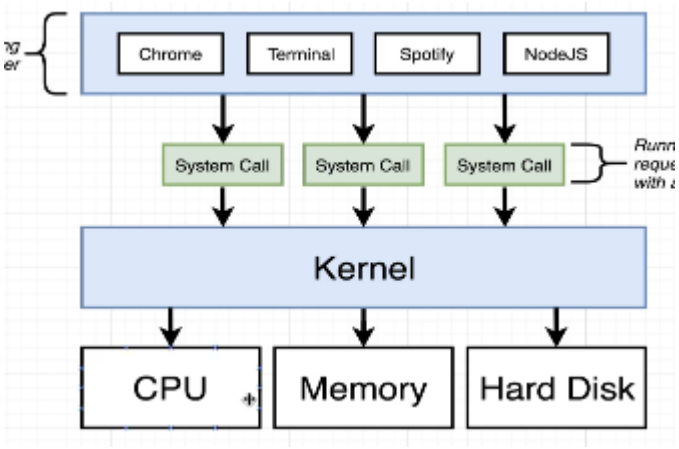
Namespacing says, this area of hard drive is for this particular process, like this area of hard drive is for chrome and that is for mongodb..

Ex:- in real companies, they will give vm's for our working in laptop where we connect to vm's

Means generally they will but 1000's of PETA Byte's as a single hard ware , and they will segregate that hardware to vm's ,like 1st user 100GB, 2nd user 100GB....

Namespaces will provide ,

Docker Engine uses namespaces such as the following on Linux:
• The pid namespace: Process isolation (PID: Process ID). // each container running on separate port
• 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).

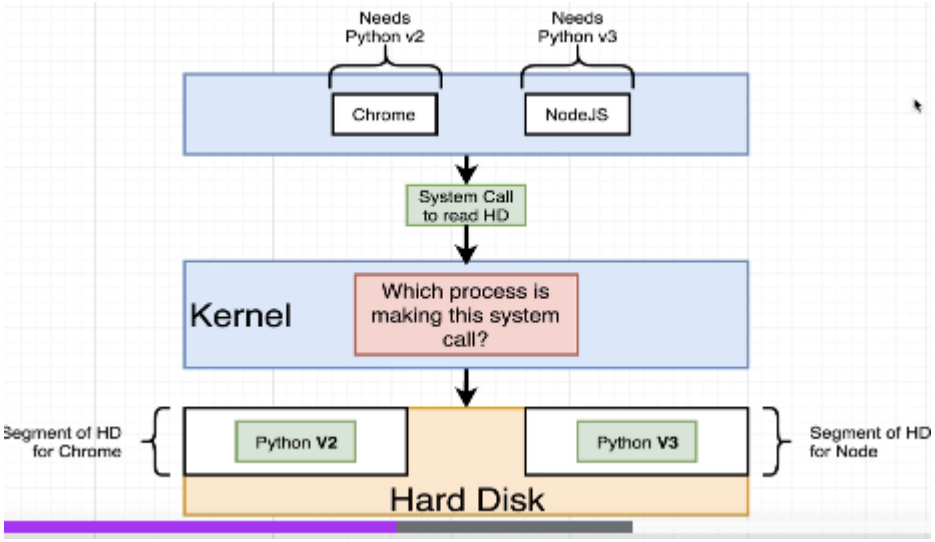


Generally, if any application wants to interact with hardware (HD,RAM..)they must go via kernel only

If spotify wants access to write data to a file, they must go via kernel only,if chrome wants access to RAM then also request should go via kernel only

Wordpad Appn→ go via kernel→ to interact with HardDisk

Edge Appn→ go via kernel→ to interact with RAM



Here in this example, there are 2 applications - chrome,Node.js where both needs python,

chrome needs python2 and nodejs needs python3, in real project world many appn's teams wants to deploy the code into jvm,

so here when u apply namespacing, Kernel will identify which chrome /nodejs is is requesting that will handle those and it will redirect those requests, if request comes from chrome kernel will redirect to python2, if request comes from nodeJS then kernel will redirect that request to python 3, this is called namespacing and isolation

here both python versions are isolated

in real world also , there will be many jvms isolated, and running continuously in same hard disk, all those are isolated and

if request is coming from 1DSTR it will redirect to 1JVM, and if request is coming to 1CASM appln it will redirect to another jvm

In docker all jvms are running inside same Hard disk

With namespacing we can redirect the requests that's coming from a particular process

100TB Hard disk would have been allocated to 10-15 teams

But in reality there might be one 1000Tb Hard disk would have been available, all teams wants jvm,now with name spacing in single hardware, they would have segregated the space 20GB to each application and everyone will have their jvm's ,so in

Like 1DSTR app-20GB namespace-they will have some JVM's

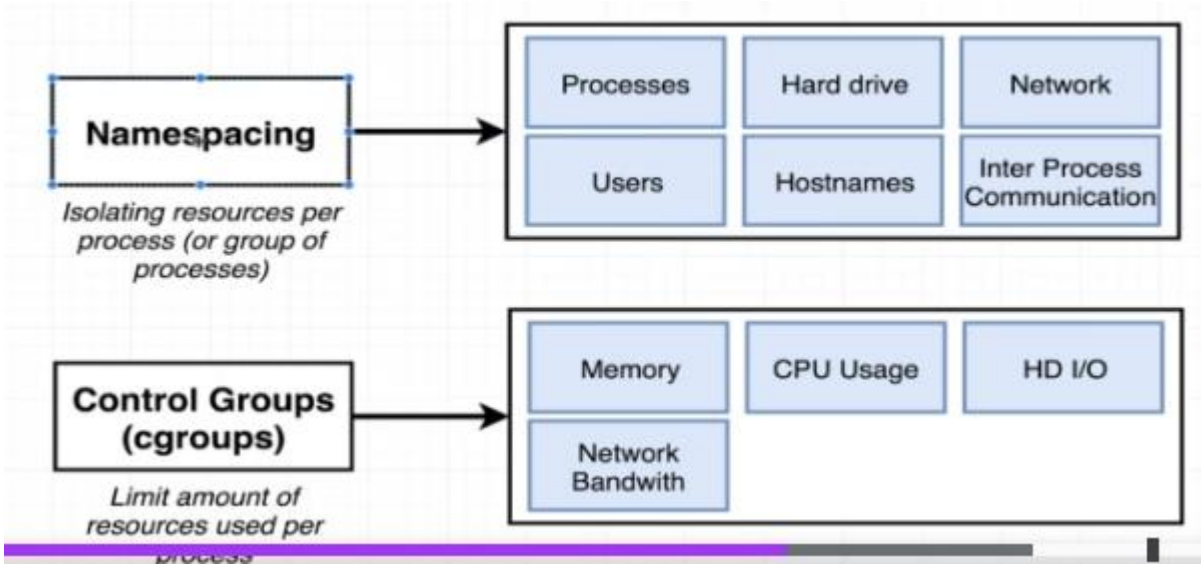
POPS application -20 GB namespace- they will have some JVM's

Here namespacing here is if any HDD read request comes from chrome, it will kernel will redirect it to the python2 Hard disk segment, if any request come from NodeJS kernel will redirect that request to python 3 segment

- ✓ With name spacing we can isolate resource per each process or for each application
- ✓ We can limit the resources for each and every process ,chrome only this segment of hard drive,
  Node Js -> only that 200GB hard drive
- ✓ We can redirect the request for particular process, like we redirect for chrome--?python2, nodeJs to speak with→python3
- ✓ Namespace says this area of hard drive for this process (application),this area is for coreIQ app,that 200Gb HDD is for POPS…
- ✓

Any application if it wants to interact with hardware first it must go via kernel only,

Different kinds of namespacing like pid, if 2 versions of python runs on different ports then both are isolated this is called pid namespace isolation,



Namespacing and control groups are linux specific properties

What is really a container

Its nothing but isolated space in Hard disk to install all required s/w mentioned in our image file & run your application alone

In isolated environments

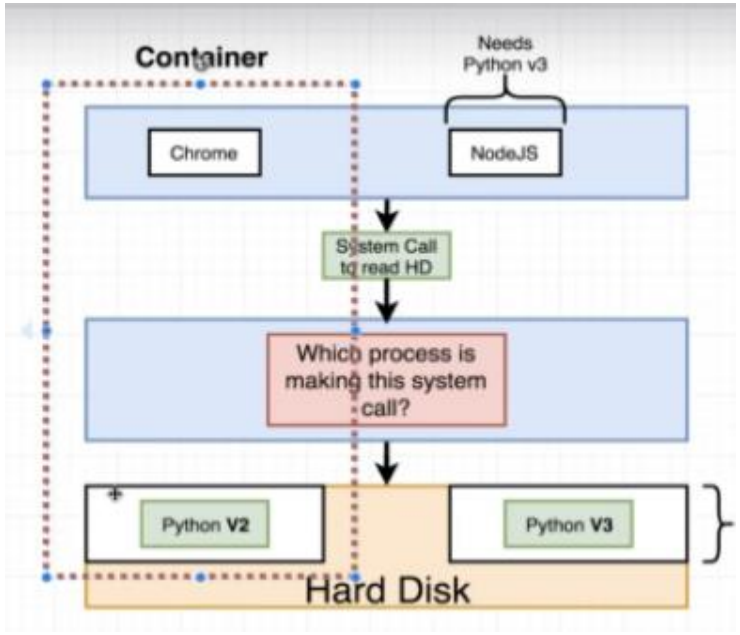
Containers are meant for isolated environments

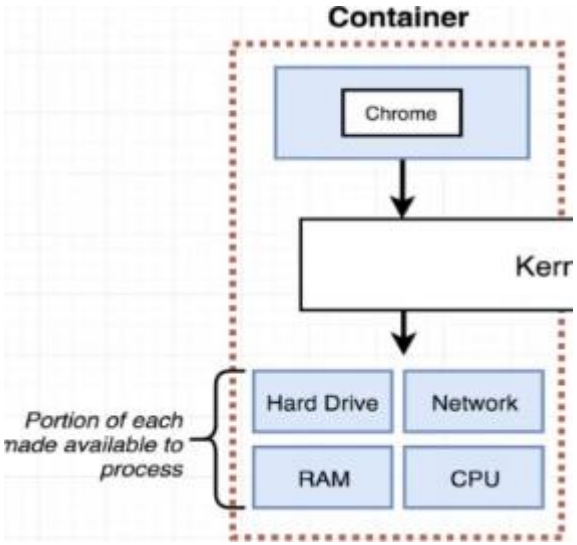
Its nothing but some isolated space in the os,

Isolated space in all system resource like --isolated HDD,Isolated RAM

Container is a running process(some program..may be our java program) along with the set of system resources

When u create a container a little portion of hard drive is available to your process



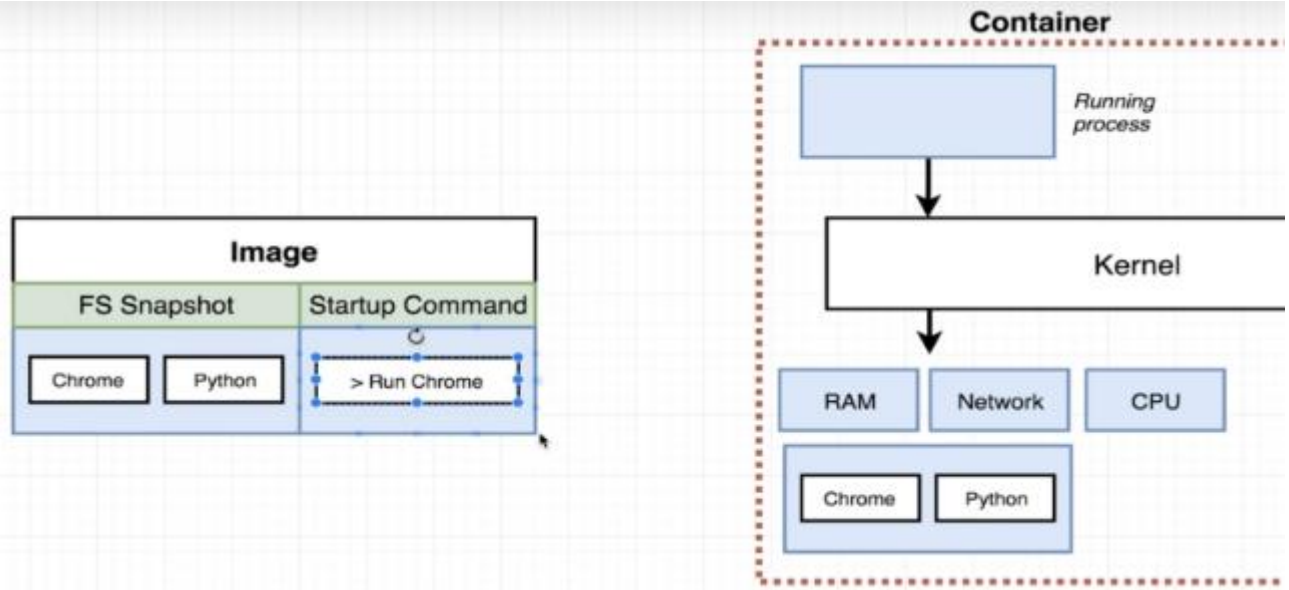


In left side,This entire set of running process and little portion of hardware is called a container

Container is a set of process , that have grouping

Here when request is coming from a process, Kernel will redirect that request to a specific portion of hard drive

Relationship between image and container

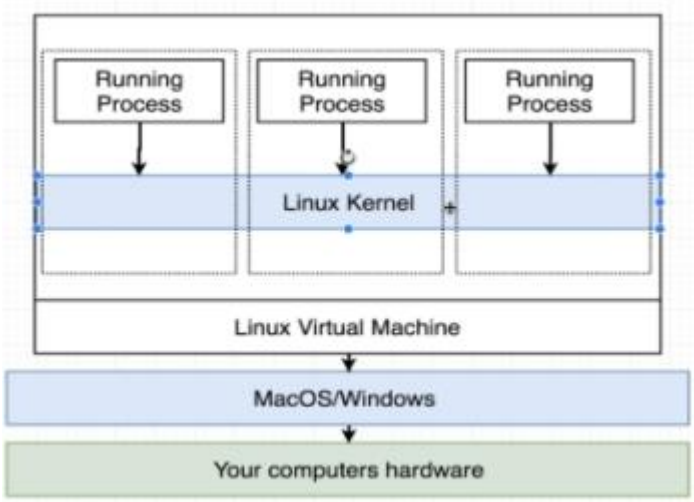


In the image , if u keep chrome, when u create a container from it, then you will be assigned with set of system resources like HDD,RAM,CPU..

in that little portion of hard disk chrome and python software's will be pre-installed and provided to you as you have mentioned python,Chrome softwares .

Lets say in that image if jdk is mentioned, then jdk is given and installed to you in that hard disk. And when you deploy and application , your application will only use those set of resources , I mean your application will use only those Hard disk and ,CPU,RAM,

Kernel also will identify the request is coming from which process and it will redirect all those requests to that set of hard drive



On windows,linux an linux virtual machine is created and on top of only docker containers will be created

Container life cycle

First it will check if image is present in local cache or not, if its not present then it will pull the image from the docker hub and store locally

Once container died after executing the base command

U can get the same container id and restart the same container, But the only problem is u can start that old container only with the old command, u can't start the old container with new command

Ex:- 1st check the status of all containers

manideepvv@DESKTOP-48ALSQ8:~$ docker ps --all

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
5be6c14d63ee	hello-world	"echo hi there"	10 minutes ago	Created		mystifying_liskov

2nd step

Start the same old container

Sample image names with tags

Jenkins:latest

Nginx:latest

nginx:stable-perl

Docker commands

Create and running a container from that image

DOCKER RUN=DOCKER CREATE+ DOCKER START

jvm,tomcat

gradlew bootrun

1) When u opened a shell inside a running container- to exit from command prompt type Ctrl+PQ
2)

1) Login to docker container

Docker login

First you have to login

2) Creating a container

Docker create image-name

1) First it will create an Isolated space in HDD & all system resources for the container to run manually
2) And installing mentioned software's mentioned in the image

Ex:- lets say in the image if u mentioned software's like jvm,tomcat, First it will create space in HDD and install those required software's

Now we created some space in HDD for the busy box softwares to store and install

manideepvv@DESKTOP-48ALSQ8:~$ **docker create busybox echo hi charan**

874db7b2fe327c2ab64c34f14905ec33ec7c3919661ca84d7623ea90b6fcbd05

//Now the above 874 is the container id

Now container is created means some space is created in HDD

3) Starting the container

Docker start –a container-id

Whereas –a is to print the logs to our console

Now to start the container with some command

Docker start –a <container id>

docker start -a 874db7b2fe327c2ab64c34f14905ec33ec7c3919661ca84d7623ea90b6fcbd05

hi charan

//hi charan is the output

Now we have started the container with that id, now the only problem is we can start that container with same old initial command

EX:- docker run hello-world echo hi there

Here once that container is created(if required software are installed)then the command

echo hi there will be executed and container will be started

Docker run <image-name> <command>

If u type docker run image name – it will create a container and install those softwares mentioned in that image in the assigned container Hard Disk

//whereas the command is the one that will be executed which will be executed once container is created

1) Download an image

docker pull nginx

docker pull nginx:latest

docker pull nginx:stable-perl

docker pull tomee //this is tomcat image official name

docker pull Ubuntu

More about images

==========

If u want tools like vi,.. only in Ubuntu these are present

2) See list of docker images

docker images

or

docker image ls

here ls means list

this will show the list of images downloaded, same like mvn local repo which will show list of jar downloaded

3) Create container ,run =create+start

Tip: always

1) D-run in detached mode
2) N-name a container
3) I- run in interactive /console mode if necessary

Final run command is

docker run –d –-name <container-name> <image-name>

here it will create a new container/space using the image

docker run <image-name>

Ex:- **docker run redis**

Here, if redis image is present locally, then it will use, else it will pull from docker hub

docker run hello-world echo hi there

//where hello-world is the image name

// whereas echo hi there is the command that will be executed after creating the container

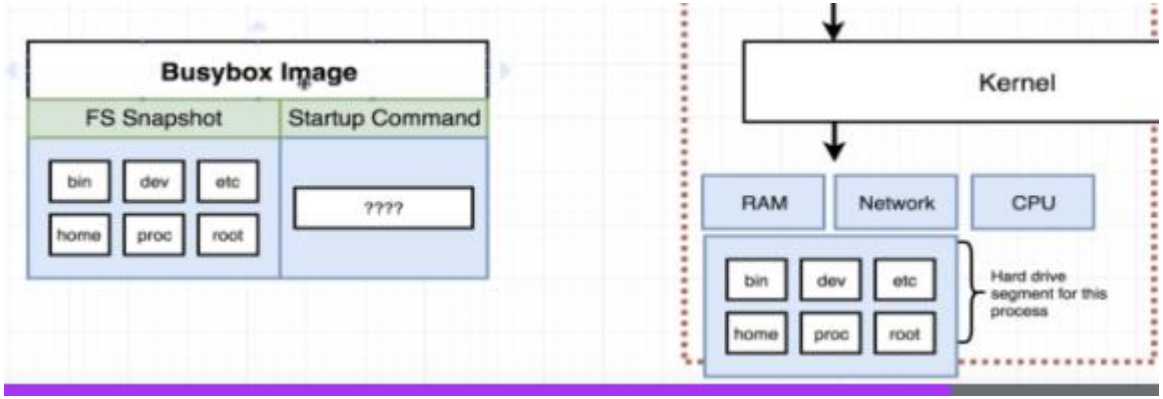
```
manideepvv@DESKTOP-48ALSQ8:~$ docker run  -d --name t1 tomee
Unable to find image 'tomee:latest' locally
latest: Pulling from library/tomee
99803d4b97f3: Downloading [==================================>                ]  17.68M
b493027a1bf3: Downloading [=========================================>         ]  17.19M
```

here since tomcat container is not there, it will download the image from hub

1) Docker run busybox
2) Docker run busybox ls

   //where ls is the command that will be run in the docker container

   manideepvv@DESKTOP-48ALSQ8:~$ **docker run busybox echo hi there**



Here in the busy box image we mentioned the folder.. whereas the same folders will be created
In our docker container too

Name and run a container

We can tag a name to an image, same like how we name a jar with group id ,artifact id

Similarly , we can name a running container, means we can give to a name running space

docker run --name <desired container name> image name:image version

from the image name/image id it will create a container

ex:- docker run --name mnginx nginx:latest

where "mnginx" is my custom name to this container

```
manideepvv@DESKTOP-48ALSQ8:~$ docker run --name manginx -d nginx:latest
aae78d952f9a82be7e5ba9763bc1fafb535b6b27f6492d2f315ea4aca1fcd4be
manideepvv@DESKTOP-48ALSQ8:~$ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED          STATUS          PORTS     NAMES
aae78d952f9a   nginx:latest   "/docker-entrypoint.…"   5 seconds ago    Up 4 seconds    80/tcp    manginx
manideepvv@DESKTOP-48ALSQ8:~$
```

Run a container in detached mode

Detached mode means u will not get the logs in same window

-d means detached mode, means our console will be detached from that containers console,

means we will not get the logs ,

if u want to see logs use "docker logs <container-name>

docker run –d <image-name>

docker run -d nginx:latest

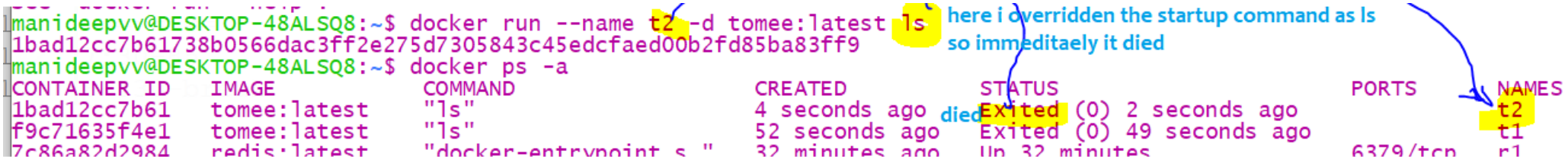
it will create container using that image

```
manideepvv@DESKTOP-48ALSQ8:~$ docker run -d nginx:latest
75a89a72b6b408a717c737f25ccbf16314230a436ba6f36c58107cc36cdfde58
manideepvv@DESKTOP-48ALSQ8:~$
```

Overriding the start-up command

docker run --name t2 -d tomee:latest ls

docker run --name <container-name> -d <imagename:version> <startup command>



2) List out all running, stopped containers

1) **docker ps**

This will show only running containers

2) docker ps -a

Here "–a" means all started +stopped

This will show both running and stopped containers

2) **docker-compose ps**

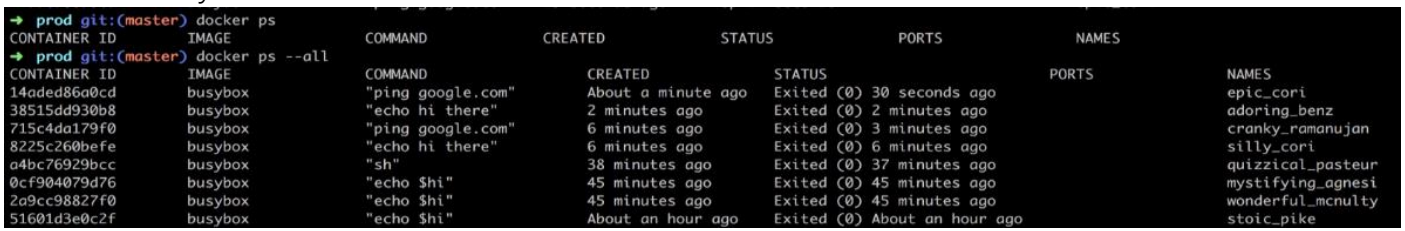
// this "docker-compose" will work only when we have a file called "docker-compose" in current directory

//where ps means process status

//list all running containers currently on our machine

docker ps --all

// it will list out all created and destroyed containers like 1 hour before what are all the containers created and naturally destroyed and all.....



3) Create a container

Creating a container means= creating some space in HDD+ Installing all the required software's mentioned in the image

docker create <image-name>

docker create hello-world

e5b1760fea69a3414794b2f878c64d71237adef87ed2758b8a98f275a12eeafd

if u execute the above command u will get container id—that e5b is the container id

4)

5)

6) see the logs of a container

docker logs <container id>

or

docker logs <container-name>

manideepvv@DESKTOP-48ALSQ8:~$ **docker logs fb58c19f5657**

here we cant give image name, because for 1 class many objects will be there,here also for 1 image-many containers

will be there

3) when container started for the first time it will show the logs, for second time /when we restarted then it wont show logs

6.1) Stop /kill a container

Kill means terminate the container abruptly

Stop means – we will send a stop signal to the running process, but it wont terminate immediately,

It will give 10 seconds it should shutdown before that gracefully, if it still running docker will automatically issues a kill command

docker stop <container-id> // stops slowly gracefully before 10 sec, else auto kill will be issued

docker kill <container-id>// kill the container abruptly

7) Start/restart the died container

Container Is nothing but some isolated space in linux os

here same container -space will not be wasted- in that same space, the same software will be started,

if u create new container un necessary space of hardware will be used

docker start <container-id>

docker start <container-name> //we should only give a name to a container while starting

container name is unique

docker start –a <container-id>

docker start -a e5b1760fea69a3414794b2f878c64d71237adef87ed2758b8a98f275a12eeafd

//-a is to show the logs into the console

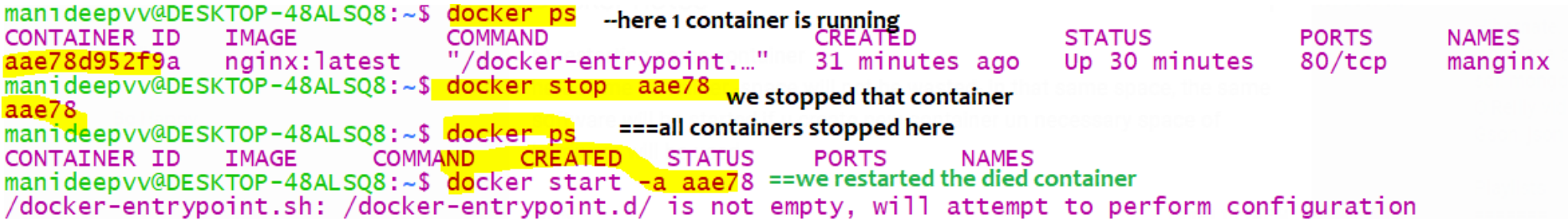
//-a will print the logs from vm – it will get the command output and print in our console

// all the numbers after –a is the container id

You can restart the old container, but the only problem is u can start only with old command

First check what and all container are born and died

The above is the ran and died container



8) Deleting an image

"docker system prune"→ this will delete all dangling images(which doesn't have name/untagged images), this wont delete normal images

docker rmi <image-name:tag>

docker rmi ngingx:latest

if u type docker image, then u can see the tags

if u delete a image, container wont be stopped /deleted.

in java also if we delete a .java file, then .jar file wont get deleted na same here also

Then when the image will be deleted instead of untagged?

First u have to stop the container, →delete the container→ then delete the image→ now image will be deleted instead of untagged

If u force delete a image when container is running→then image will be untagged, it wont be deleted

here rm-means remove

rmi-means remove image

it will remove that single image ,

Deleting image when a container exists even in stopped mode

If any container is created (whether it is in stopped/running mode) if container /space exists itself u cant use “rmi”

U must do a force delete, or u must delete the space/container and delete the image without force delete

```
manideepvv@DESKTOP-48ALSQ8:~$ docker ps
CONTAINER ID   IMAGE           COMMAND                  CREATED       STATUS         PORTS                    NAMES
707683aca701   complex-nginx   "/docker-entrypoint.…"   6 days ago    Up 2 seconds   0.0.0.0:3050->80/tcp     complex-nginx-1
manideepvv@DESKTOP-48ALSQ8:~$ docker run -d --name mn-perl nginx:stable-perl
8dbd43d89daa59b5dfa01a89ad227182b13dd92fecb5371b28414c897c4175c5
manideepvv@DESKTOP-48ALSQ8:~$ docker ps
CONTAINER ID   IMAGE                COMMAND                  CREATED          STATUS          PORTS                    NAMES
8dbd43d89daa   nginx:stable-perl    "/docker-entrypoint.…"   5 seconds ago    Up 3 seconds    80/tcp                   mn-perl
707683aca701   complex-nginx        "/docker-entrypoint.…"   6 days ago       Up 3 seconds    0.0.0.0:3050->80/tcp     complex-
nginx-1
manideepvv@DESKTOP-48ALSQ8:~$ docker ps
CONTAINER ID   IMAGE                COMMAND                  CREATED          STATUS          PORTS                    NAMES
8dbd43d89daa   nginx:stable-perl    "/docker-entrypoint.…"   12 seconds ago   Up 10 seconds   80/tcp                   mn-per
l
707683aca701   complex-nginx        "/docker-entrypoint.…"   6 days ago       Up 10 seconds   0.0.0.0:3050->80/tcp     comple
x-nginx-1
manideepvv@DESKTOP-48ALSQ8:~$ docker stop mn-perl
mn-perl
manideepvv@DESKTOP-48ALSQ8:~$ docker rmi nginx:stable-perl
Error response from daemon: conflict: unable to remove repository reference "nginx:stable-perl" (must force) - container 8
c0e0b8f8023 is using its referenced image b28f68aa8908
manideepvv@DESKTOP-48ALSQ8:~$ docker rmi -f nginx:stable-perl
Untagged: nginx:stable-perl
Untagged: nginx@sha256:e4dcfb41197293ede139a88a307802239f48f913242fd3130fa416b37fb53c52
Deleted: sha256:b28f68aa890803114f1713c048734b13cf2e6b93deeaf96a644a57ef0ad4b671
manideepvv@DESKTOP-48ALSQ8:~$
```

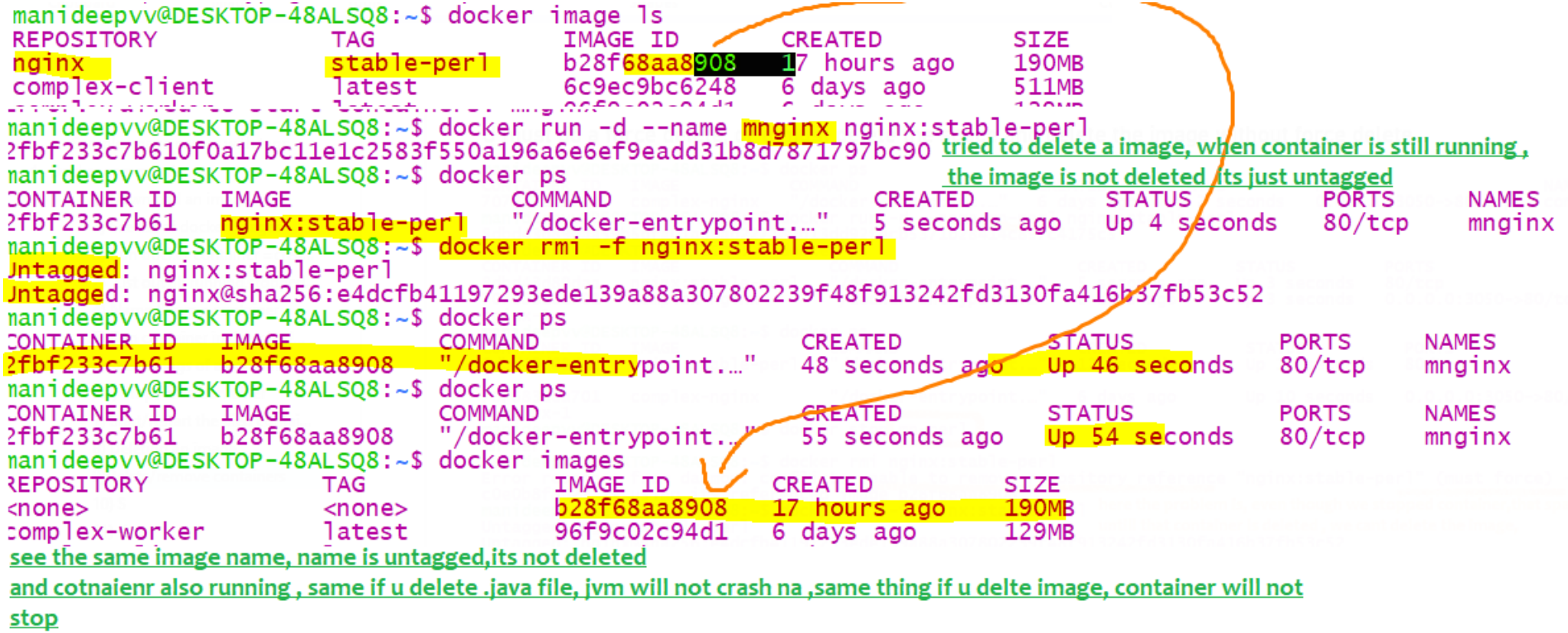
here the problem is, even though we stopped container,that space is there untill that container is deleted , we cant delete the image,

else we need -f flag only

Delete an image when container is running

If u delete a image when a container is running -using “docker rmi –f <image-name>”

, image will not be deleted, it will be untagged and container will not be stopped at all



9)Delete/remove containers

"docker system prune"→ this will delete all stopped containers

Deleting a particular container- if u delete space will be released-same like deleting a file

Once it got deleted, u cant see that in "docker ps –a" also

1) Either stop it and use docker rm <container-name> or
2) Don't stop and use docker rm –f <container-name>

docker rm <container-name> <container-name> <container-id>

if u want to delete a container/space, first it should be In stopped state,else if it is running u have to use "–f" force flag

u cant dele multiple containers by space, u can use either container name or container id

Note:- if u delete a container, image wont be deleted,

docker rm manginx

//here manginx is the container name-we can name a container using –name flag

docker run –name <any container name> <image-name>

This will delete all container and frees memory in hard disk

This will delete only space/container-image wont be deleted,

Deleting a running container

You cant delete a container when it is running it will throw error saying u cant delete a running container

But if u want to delete a running container use "–f" option (where f means **force**)

docker rm –f <container-name/container id>

now the container will be stopped and removed

```
manideepvv@DESKTOP-48ALSQ8:~$ docker run -d --name mnginx nginx:stable-perl
ef337c76a79a5fea2fff965c6cd9c4b60ff97e9207dbdd0f5fee0547b6425d98
manideepvv@DESKTOP-48ALSQ8:~$ docker ps
CONTAINER ID   IMAGE                COMMAND                  CREATED              STATUS              PORTS     NAMES
ef337c76a79a   nginx:stable-perl    "/docker-entrypoint.…"   About a minute ago   Up About a minute   80/tcp    mnginx
manideepvv@DESKTOP-48ALSQ8:~$ docker rm -f mnginx
mnginx
manideepvv@DESKTOP-48ALSQ8:~$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
manideepvv@DESKTOP-48ALSQ8:~$
```

here mnginx is the container name

-f means force

Deleting all containers

docker rm –f $(docker ps -aq)

Deleting all stopped containers+dangling images

docker system prune

If u create a container means, separate isolated space is allocated (along with separate system resources) the software's mentioned in the image will be download to our allocated space

1) When u delete the container that allocated space in HDD will be deleted
2) All the locally cached images will be deleted

10) S

11) K

12) Executing additional commands inside running containers

docker exec –it <container-id/name> <command>

u will get container id from docker ps, and container id is diff from image id
this exec command will not create any additional container like docker run
,it will get the shell of a running container
docker exec –it <container-name/id> sh

docker exec –it <container-name/id> /bin/bash

Note: to exit from command prompt type Ctrl+PQ

```
Step 8/8 : COPY --from=builder /app/build /usr/share/nginx/html
 ---> Using cache
 ---> 23544802c1ee
Successfully built 23544802c1ee
manideepvv@DESKTOP-48ALSQ8:~/frontendapp$ docker exec -it 23544802c1ee sh
Error: No such container: 23544802c1ee
manideepvv@DESKTOP-48ALSQ8:~/frontendapp$
```

Here I gave image id ,it wont consider,bec this wants to execute that command inside a container

exec- means execute this additional commands inside running containers

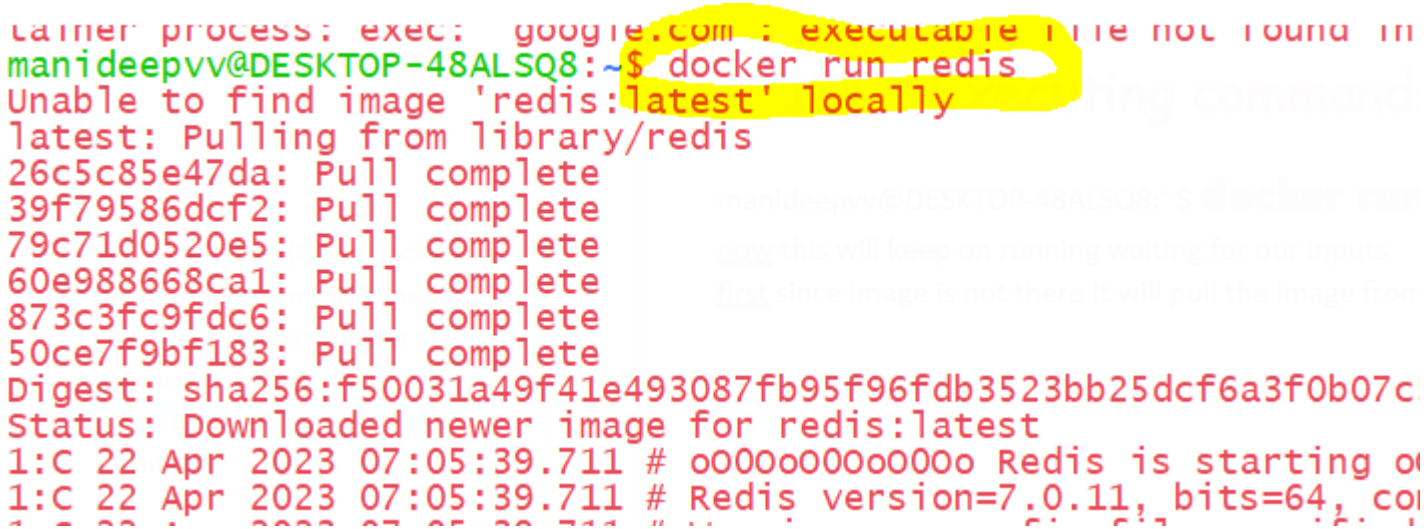
-it means connect my terminal to the input stream/interactive mode

if u miss –it flag that command wil be executed, but u wont get the input console CLI like below

manideepvv@DESKTOP-48ALSQ8:~$ **docker run redis**

now this will keep on running waiting for our inputs

first since image is not there it will pull the image from remote site hub.docker.com



See now this window is continoulsy running and redis is continuously running , now it wont take any inputs

Now redis is running , whereas redis is a cache, we need to give inputs so that cache can store all those the elements

1) So now we want to interact with redis client, but we cant type those commands here
2) If we open another terminal and connecting to CLI it wont , because we have to connect to CLI inside that container
3) So finally we have to go inside that running container and execute the connecting to CLI command

   Command is

```
manideepvv@DESKTOP-48ALSQ8:~$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS      NAMES
62668f117a27   redis     "docker-entrypoint.s…"   39 minutes ago   Up 39 minutes   6379/tcp   vigorous_kapitsa
manideepvv@DESKTOP-48ALSQ8:~$ docker exec -it 62668f117a27 redis-cli
127.0.0.1:6379> set myValue 5
OK
127.0.0.1:6379> set myValue 2
OK
127.0.0.1:6379> get myValue
"2"
127.0.0.1:6379>
manideepvv@DESKTOP-48ALSQ8:~$
```

redis-cli is the command name

since u kept -it a Cursor came to execute our input

only when u presedn ctrl+c , docker control came

Command

docker exec –it 5f33dc7dfef3 redis-cli

where redis-cli id the command name that will be executed inside the container id 5f…

-it means input when u keep this , then only the redis console came, it will accept the inputs from us

Adding -it flag will enable us a prompt window to enter our inputs

"-it", -i Says connect my standard terminal to input stream and –t means to format the stdout data

docker exec –it <containerid>sh <any additional command>

manideepvv@DESKTOP-48ALSQ8:~$ docker ps --all

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
5f33dc7dfef3	redis	"docker-entrypoint.s…"	50 minutes ago	Up 50 minutes	6379/tcp	ecstatic_hellman

Now u got the container id and type below

docker logs 5f33dc7dfef3

13) i mode- Get the cmnd prompt of a container

docker exec –it <container-id> sh

docker run –it <container-id> <image-name >

here interactive mode means – u will get the command prompt inside a container

if u want to exit from that mode/from that console type "ctrl+pq"simultaneously

Above can be sh/zsh/powershell/bash

The above command will open a terminal inside that docker container, I mean inside that HDD

Lets say a redis process is running inside a docker container and if u want to get the command prompt of that container

1) Earlier we had command as "docker exec –it <container-id> <command-name>
2) Now since we want entire command prompt new command is → "docker exec –it <container-id> sh"

```
manideepvv@DESKTOP-48ALSQ8:~$ docker ps --all
CONTAINER ID   IMAGE     COMMAND                  CREATED             STATUS                          PORTS      NAMES
5f33dc7dfef3   redis     "docker-entrypoint.s…"   About an hour ago   Up About an hour                6379/tcp   ecstatic_h
efiman
62668f117a27   redis     "docker-entrypoint.s…"   4 hours ago         Exited (0) About an hour ago               vigorous_k
apitsa

manideepvv@DESKTOP-48ALSQ8:~$ docker exec -it 5f33dc7dfef3 sh
# ls
dump.rdb
# cd /
# ls
bin   boot  data  dev  etc  home  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
# echo hi santhshi
hi santhshi
# export b=sex
# echo $b
sex
# redis-cli
127.0.0.1:6379> set lapname lenovo
OK
127.0.0.1:6379> get lapname
"lenovo"
127.0.0.1:6379>
```

15) Starting container with shell

docker run –it busybox sh

docker run –it <image-name> sh

from the image it will create the container and opens a shell in that command

here –it means, -i means connect my terminal to standard input stream, -t means format the data,

So when ever u want to have a console, u must place –it

Sh- open a shell console in linux

when u run the container like “docker run busybox echo hi there”,

First image will be pulled if it is not in the system

the container will be created (Isolated space will be allocated) it will execute the command "echo hi there" and then it will die

In the below image u an see it died immediately

Whereas, if u run the container with a shell like "docker run –it busybox sh " the container will keep on running, because we have started/opened a shell or command prompt ,hence u can see the status as keep on running

```
manideepvv@DESKTOP-48ALSQ8:~$ docker run busybox echo hi mom
hi mom
manideepvv@DESKTOP-48ALSQ8:~$ docker ps --all
CONTAINER ID   IMAGE     COMMAND                  CREATED              STATUS                          PORTS     NAMES
2c6627479de6   busybox   "echo hi mom"            4 seconds ago        Exited (0) 2 seconds ago                  hardcore_leavi
tt
20147dcd8a88   busybox   "sh"                     About a minute ago   Up About a minute                         vigilant_ride
```

16) Build/Create image from docker file

Commands	Explanation In java once we build , we will get a jar ,every jar will have Group id, artifactid ,same here also-once we build,we will get image like jar file
docker build .	A file called "Dockerfile"must be present in current directory "."
docker build –f <docker file location +file name>	docker build –f <location > <file name> docker build –f ./context/Dockerfile.dev
docker build –f <docker file name>	If we have any other file other than "Dockerfile" like if we wrote instructions in "Dockerfile.dev" ,then docker wont recognise this file, we have to use this –f flag

docker build –t <tag a name to that image>	In java, once we build for every jar we will have groupid, artifact id, here also similarly Once we build, We can tag a name to every image

way 4:-Tag the name to that image and build with an diff docker file name

1. `docker build -f <docker file name>-t <custom name to that image-user:projname>.`

docker build -f Dockerfile.dev -t manideep:node .

if ur docker file name is other than "Dockerfile", then u have to use –f command

"-f" stands for file name

"-t" means build the image with that name-like how we have groupid and artifactid for jar in java samething

"." Where dot means current directory

Way 1:- docker build .

//where dot refers the current directory,in current dir there must be a file named "Dockerfile"

This command needs a file named "dockerfile" as input and it will create image from it

Way 2:-Naming an image while building

docker build –t <yourDockerId> /projName:version .

docker build -t manideep/redis:latest .

way3:- building with custom file name

here , docker file name is “someDocker-dev”, the “docker build .” will work only when instructions are present in a file named “Dockerfile”, docker will recognise only this file, if u have any other name specify it

docker build –f <your custom docker file name> .

ex:- docker build –f Dockerfile.dev .

in above –f stands for custom file name and . refers current directory

17) Running the image by opening the console

Once u build the docker file , u will get the image

Once u got the image, u can run using command

Docker run <image id>

```
Step 8/8 : COPY --from=builder /app/build /usr/share/nginx/html
 ---> 23544802c1ee
Successfully built 23544802c1ee
manideepvv@DESKTOP-48ALSQ8:~/frontendapp$ docker run 23544802c1ee
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will att
```

Diff between docker exec vs docker run-

Run used to create a new container,exec is used to execute command on existing container

But qn is for both we gave 1 container id, how exec command knows on which container its running this command

Remember if u are giving a image name, with image we can create container only

Way 1:- docker run <image id>

Way 2:- docker run –it <image tagged name > sh

here –it,sh are optional

"-it" means connect my terminal to the input console

"sh" means open a console window inside the running container

```
manideepvv@DESKTOP-48ALSQ8:~/simplewebapp$ ls
Dockerfile  index.js  package.json
manideepvv@DESKTOP-48ALSQ8:~/simplewebapp$ docker run -it manideep/rnpm:v2 sh
/ # ls
Dockerfile          home                mnt                 package.json        sbin
bin                 index.js            node_modules        proc                srv
dev                 lib                 opt                 root                sys
etc                 media               package-lock.json   run                 tmp
/ #
```

docker run <container-id> <our new command>

docker run 858b47be28e9 npm run test

docker run -it 858b47be28e9 npm run test

way 3:- Exposing a port from docker container to our machine

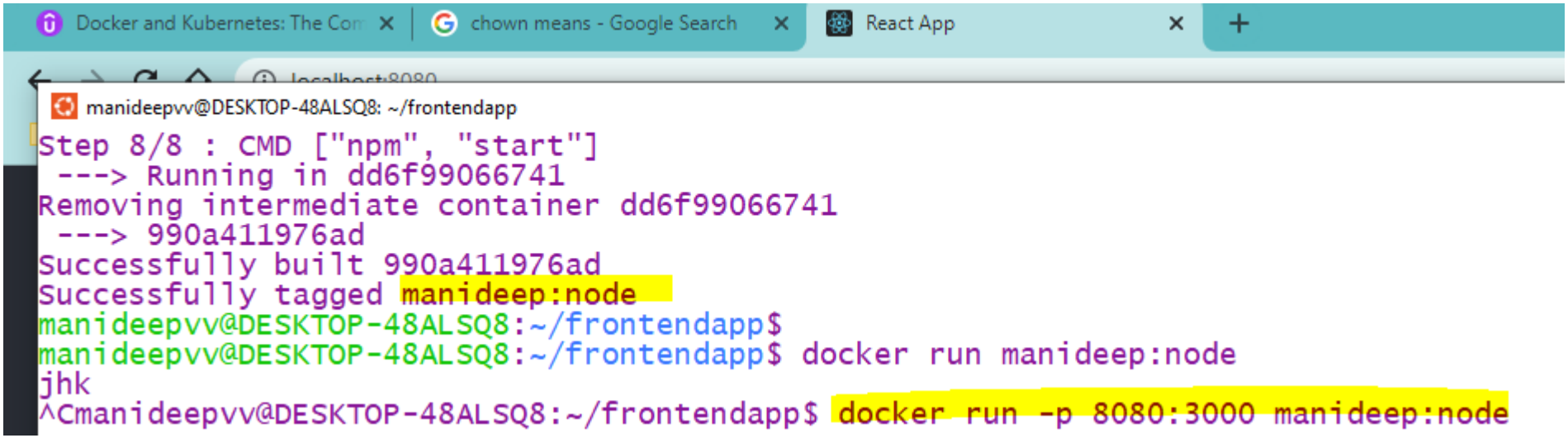
mapping and Running on different port: you have to map

mostly react apps runs on 3000 port

if u want to see ur app running on 8082 so that u can hit form browser, then use 8082 on left side,and map to where it is actually running inside container on right sde

docker run -p <input system port>: <output system port> <tagged image name>

docker run –p 8082:3000 coreiq/redis:v1



Here since we are mapping our windows port 8000 with docker container port 3000

Way4:- Alternate to docker run

Docker u can write a “docker-compose.yml” file and run using “docker-compose up”

Way 5:- Running additional commands with docker run

docker run <image name> <additional commands>

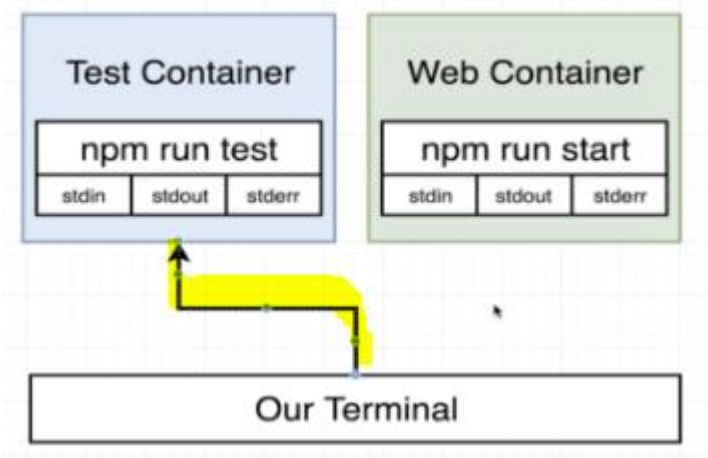
docker run ad99b9482ada npm run test

```
Successfully built ad99b9482ada
manideepvv@DESKTOP-48ALSQ8:~/frontendapp$ docker run ad99b9482ada npm run test

> frontend@0.1.0 test
> react-scripts test

PASS src/App.test.js
  □ renders learn react link (88 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        5.615 s
Ran all test suites.
```

18) Docker attach

"docker exec –it <image name> sh"

With above command we can get a terminal inside a running container due to "sh"



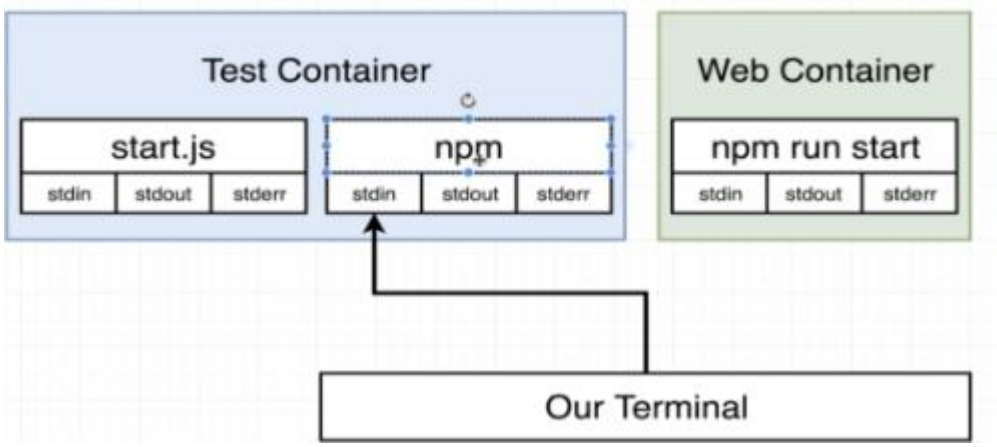
With docker attach we can forward a command from our terminal to the running container ,but why we need it,

We have command to execute the our command inside a running container

What docker attach does??

We can attach our your terminal's standard input, output, and error (or any combination of the three) to a running container

But the problem here is –if in that container if 2 processes are running , the docker attach can connect the input stream only to 1 process in that running container, with docker attach we can get handle only to primary process,not sec process



Alternate to docker attach <container id > is "docker exec –it <container id> sh",the only problem is long command

And we need to get container id

Containers Isolation

Isolation→ all container runs in isolation, if u create a file in 1st container it will not reflect that file in second container.

Lets start 2 containers of same image "busybox",Since these are containers they are always in isolation

Because container is nothing but an some isolated Space in Hdd, both containers are in isolated spaces in Hard disk

Doubts

1) When container stopped, will that earlier allocated space to that container in the HDD gets deleted?

No, the process will be died, that's it, the allocated space to the container is not yet deleted

And that allocated space will be deleted only when you execute this command "docker system prune"

Building custom images through docker server

1) Creating docker images

Specify base image

Run some commands to install additional programs

Specify a command to run on container startup

Sample code:-

1) Create a folder named "redis-image" mkdir redis-image
2) Move inside that folder "cd redis-image"
3) We have to create a file called "Dockerfile" of type file so type this command –"touch Dockerfile"
4) If u want to see the files-To open a linux folder type the command –" \\wsl$" like in windows we have "start ."

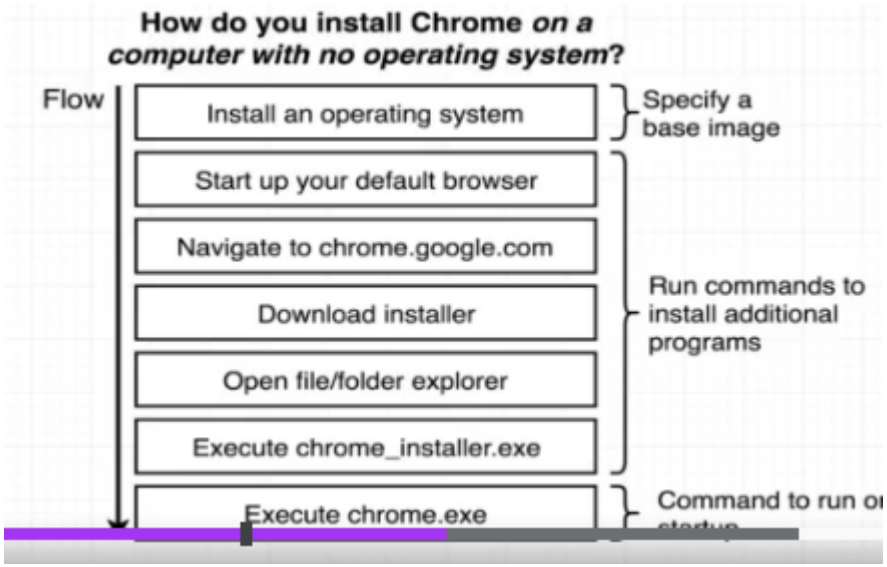
or go to location \\wsl.localhost\Ubuntu\home\manideepvv folder and open folder “redis-image” and see if file “Dockerfile” is created or not
With Extension File

5) Now open that file and type following commands

Sample docker file

```
FROM alpine
#here “node:alpine”node is the repository name ,alpine is the tag name
# Step 2: Download and install dependency

RUN apk add --update redis
RUN apk add --update gcc

# Step 3: Tell the image what to do when it starts as container

CMD ["redis-server"]
```

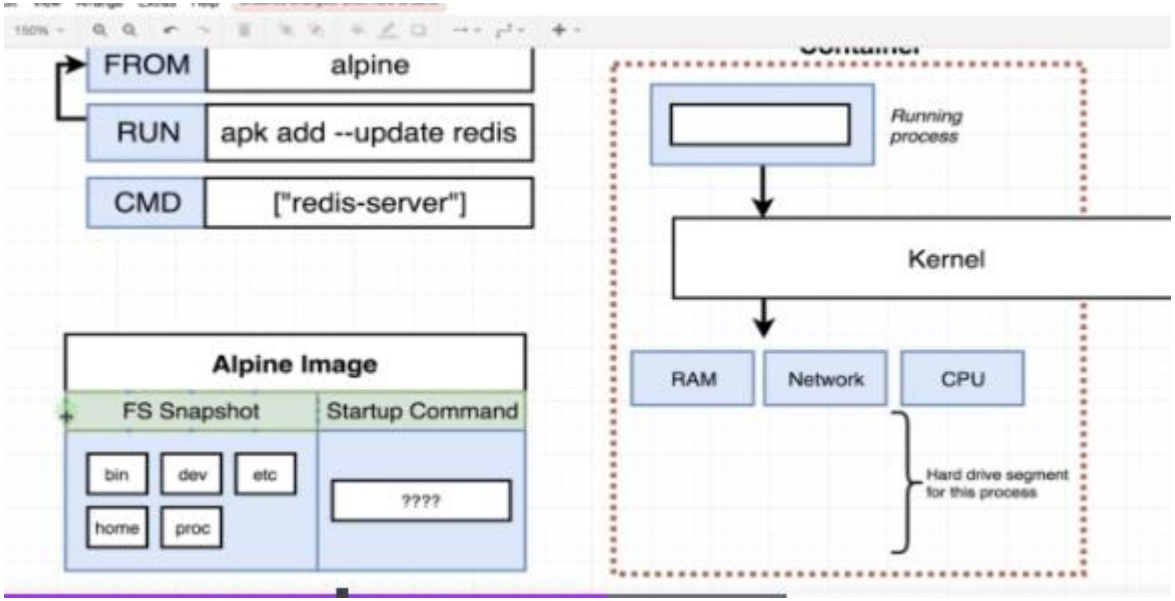
6) Type “docker build .”
Explanation of above commands
Alpine is also an image, first it will pull that image from docker hub
The above command “docker build .”is used to generate image from the docker file,
It will give the docker file to docker CLI, it will create a image from the given docker file



Among all o.s windows have lot of predefined functionalities which we need, same like windows We use alpine because it has lot of inbuilt features

That are very useful for installing and running redis

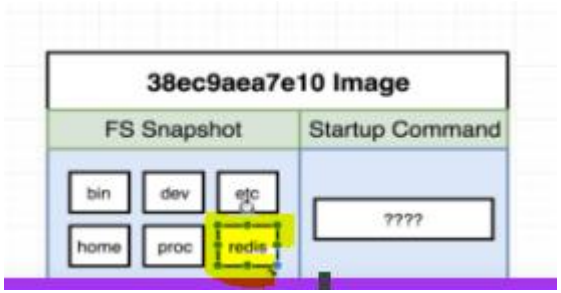
```
manideepvv@DESKTOP-48ALSQ8:~/redis-image$ docker build .
Sending build context to Docker daemon  2.048kB
Step 1/3 : FROM alpine
latest: Pulling from library/alpine
f56be85fc22e: Already exists
Digest: sha256:124c7d2707904eea7431fffe91522a01e5a861a624ee31d03372cc1d138a3126
Status: Downloaded newer image for alpine:latest
 ---> 9ed4aefc74f6
Step 2/3 : RUN apk add --update redis
 ---> Running in 1e6af360fbe3
fetch https://dl-cdn.alpinelinux.org/alpine/v3.17/main/x86_64/APKINDEX.tar.gz
fetch https://dl-cdn.alpinelinux.org/alpine/v3.17/community/x86_64/APKINDEX.tar.gz
(1/1) Installing redis (7.0.11-r0)
Executing redis-7.0.11-r0.pre-install
Executing redis-7.0.11-r0.post-install
Executing busybox-1.35.0-r29.trigger
OK: 10 MiB in 16 packages
Removing intermediate container 1e6af360fbe3
 ---> a120c6fca777
Step 3/3 : CMD ["redis-server"]
 ---> Running in d49e3097a489
Removing intermediate container d49e3097a489
 ---> 6df21e6e3b54
Successfully built 6df21e6e3b54
manideepvv@DESKTOP-48ALSQ8:~/redis-image$ docker ps --all
```



Alpine is also an image and it also have a start-up command
In above-FS snapshot means-File system snapshot

Internals of image creation

The main thing u should remember is – it Is also like streams, every intermediate steps create another stream
Here also in every intermediate step one container is created and based on that 1 image is created,
Step 1- based on alpine image 1 container is created(create some isolated space + install required s/w in that area)
Step 2- as per step 2 "RUN apk add --update redis" , a software called redis is installed into that container (HDD Isolated space)
Once s/w is installed, we took FS (File system snapshot) and this container has been stopped (not killed because we gave time)
As we took the file system snapshot of that container, in the new image we included redis



Therefore, in step 2 new image is generated,

Step 3- created a image from step 2 –started a container ,executed this CMD ["redis-server"] and installed this software and Created a File system snapshot, create a base image

Doubt:-

Why each step is creating and destroying a container and

why each step is creating a new image ,3 steps ==3 images,

without creating and destroying this many container, cant it create a final image,

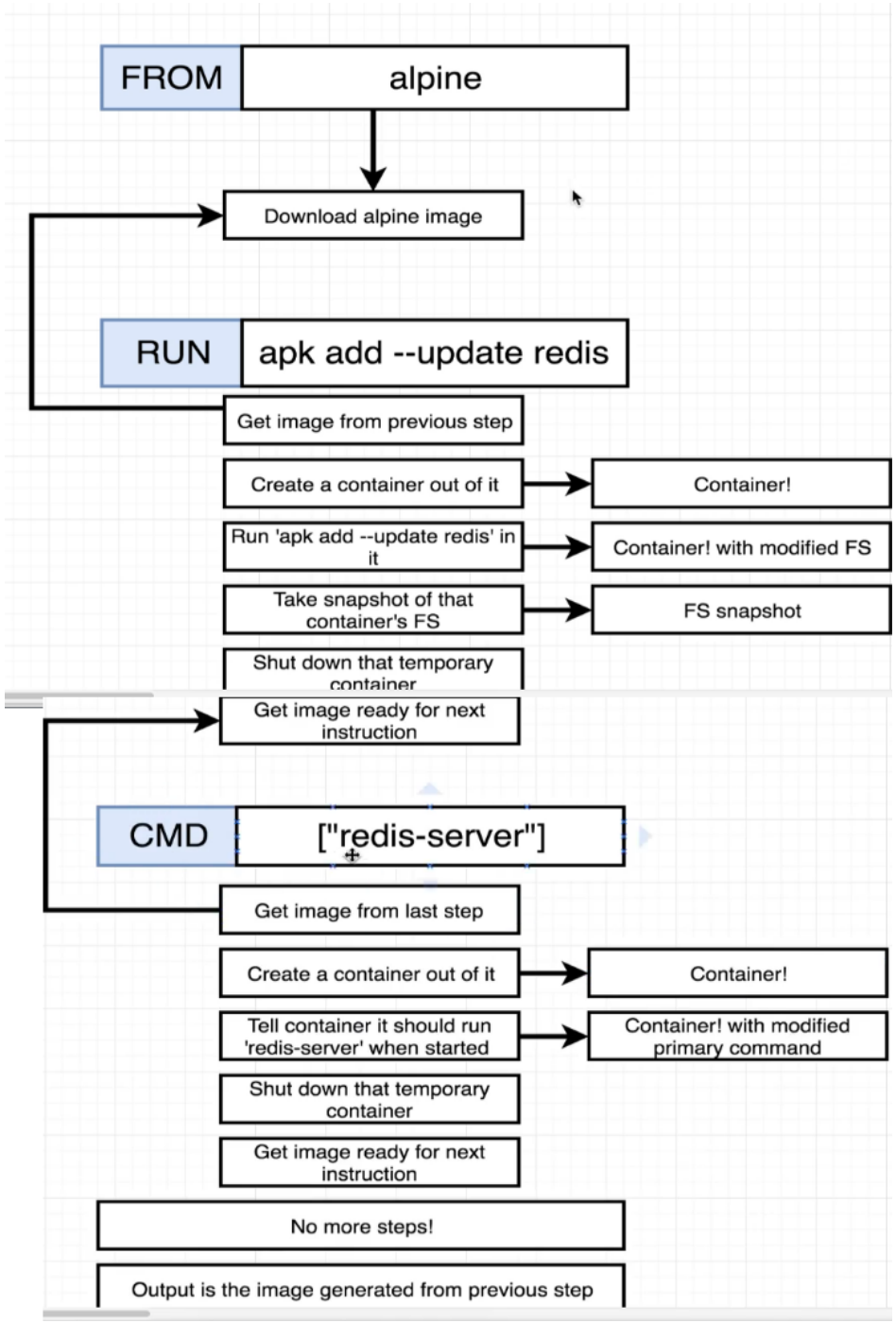
this is also behaving like java streams,creating a stream for each intermediate operation

What's happening in every step??

---

1) First it gathers the image from previous step(if no previous step-then its from base image)as it has mentioned s/w to be installed .
2) Here we will Create a isolates space {container} from it, and install required s/w and execute the current command like
   RUN apk add --update redis
   RUN apk add --update gcc
   Executing these commands will install these additional s/w in that container
3) see the container &create a File system snapshot from it (this is called image and this image will be cached..all intermediate images will be cached for future purposes
4) Destroy that current container, Every step creates a new image

Analogy:-
Every intermediate operation like a stream is created from another stream, here also every intermediate operation one image is created
From the running container


FROM
alpine
Download alpine image
RUN
apk add --update redis
Get image from previous step
Create a container out of it
Container!
Run 'apk add --update redis' in it
Container! with modified FS
Take snapshot of that container's FS
FS snapshot
Shut down that temporary container
Get image ready for next instruction
CMD
["redis-server"]
Get image from last step
Create a container out of it
Container!
Tell container it should run 'redis-server' when started
Container! with modified primary command
Shut down that temporary container
Get image ready for next instruction
No more steps!
Output is the image generated from previous step

Caching the intermediate images

We don't need to do it manually, automatically the intermediate steps images will get cached.

In the above scenario, in every intermediate step container is created, whereas in every step the container created is cached

In above we have a command called "apk add –update redis", when this step is executed a container will be created and generally

immediately it will be died, whereas this container is cached, next time this container won't be created if u execute the same command

it will be fetched from cache

```
manideepvv@DESKTOP-48ALSQ8:~/redis-image$ ls
Dockerfile
manideepvv@DESKTOP-48ALSQ8:~/redis-image$ docker build .
Sending build context to Docker daemon  2.048kB
Step 1/3 : FROM alpine
 ---> 9ed4aefc74f6
Step 2/3 : RUN apk add --update redis
 ---> Using cache
 ---> a120c6fca777
Step 3/3 : CMD ["redis-server"]
 ---> Using cache
 ---> 6df21e6e3b54
Successfully built 6df21e6e3b54
manideepvv@DESKTOP-48ALSQ8:~/redis-image$
```

since same kind of image is already built and already created container out of it,it using from cache

Earlier when I built the image then every time container are getting built and destroyed frequently, now when I built this image

instead of creating containers it used from the cache

Tagging name to an image

Many of times images we built will have same name, to avoid conflicts we tag image with following standards

Like we will create an image from docker file generally we use this command "**docker build .**" now instead of this command use below

docker build -t <yourDockerId> /projName:version .

1) in the above **dot** at the end specifies the direct of folder/files where the docker file is available to use for the build
2) we should specify where the docker file is present, whereas dot represents the current directory
3) **-t** stands for tag

docker build -t manideep/redis:latest .

Once image is created then u can run that image using this command "docker run Manideep/redis:latest"

Sample commands

manideepvv@DESKTOP-48ALSQ8:~/redis-image1$ **docker build -t manideep/redis:v1 .**


```
docker commit -c 'CMD ["redis-server"]' CONTAINERID
```

If you are a Windows user you may get an error like "**/bin/sh: [redis-server]: not found**" or **"No Such Container"**

Instead, try running the command like this:

```
docker commit -c "CMD 'redis-server'" CONTAINERID
```

Creating images from running containers

1) First run some basic alpine image, below are the commands

   docker run –it alpine sh

   -it says connect my terminal to the input stream

   Sh says open a shell terminal/ command prompt

2) Install any software like redis

   apk add --update redis

3) Open a new terminal and Get the container id of running process
   docker ps --all
4) Open a new terminal and we have to Create a file sys snapshot/image from the container

   docker commit –c 'CMD ["<startup-command>"]' <container-id>

Sample command

manideepvv@DESKTOP-48ALSQ8:~$ docker commit -c 'CMD ["redis-server"]' db1e82040a3e

sha256:8f09e980c91e4e9c6297b6dc6b73770f878e5be95c6bd3e0fb7004e028f13b0c

once u run docker commit, once image created successfully, u will get image id

with this image will be created and u can create and start a container from it as below

docker run 8f09e980

2) UTILISING EXISTING OFFICIAL IMAGES

Example of wrong base image

From alpine

RUN npm install

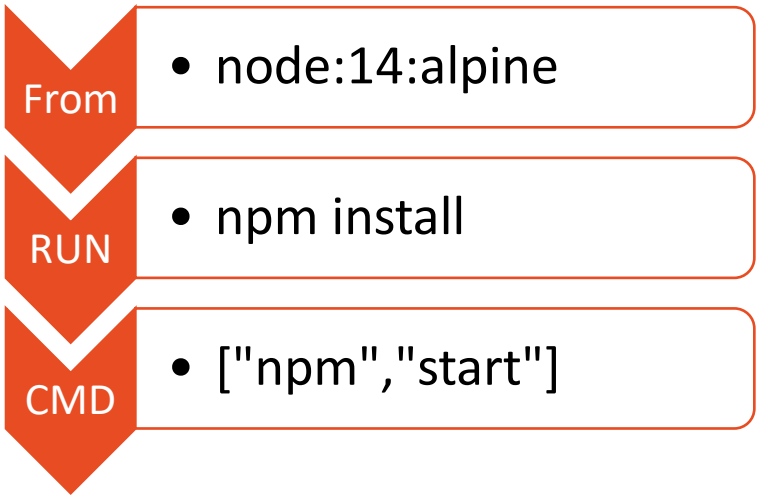
CMD ["npm","start"]

//This wont work because this base image doesn't contain npm

//NPM people installed this software and created an image out of it and pushed to docker hub

//Similarly java people also installed openjdk and created an image out of it and pushed that image to docker hub, so that we can pull.

Installing node /npm through docker images



or node:alpine

Here if we give plain alpine as the base image it wont work, bec it doesn't have all the required programs

You have to choose node as the base image

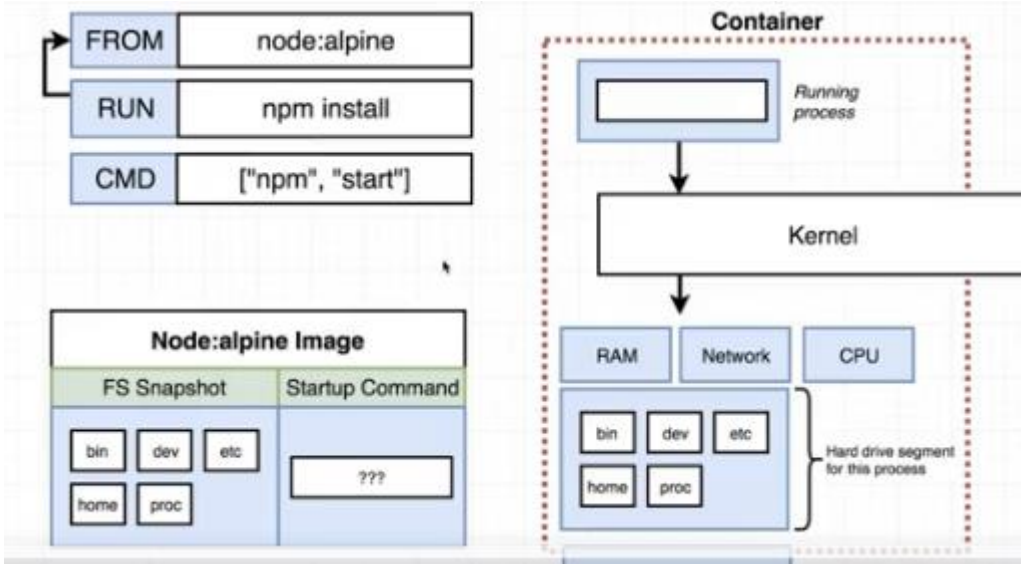
Steps of creating this container

1)Base image [node:14-alpine] will be pulled and as usual, since all s'w are mentioned in the image ,container will be created from that image in HDD

And s/ws will be installed,

3) While installing, this particular node needs 2 files named "package.json"..actually these 2 we kept in our current directory
   But the problem is this container is created in isolated space and our files are in our current directory, hence this container is not recognizing those files

```
manideepvv@DESKTOP-48ALSQ8:~/simplewebapp$ docker build .
Sending build context to Docker daemon  4.096kB
Step 1/3 : FROM node:14-alpine3.16
14-alpine3.16: Pulling from library/node
91d30c5bc195: Pull complete
69ac140ffefe: Pull complete
4f60c2188c3b: Pull complete
20f506de6304: Pull complete
Digest: sha256:d67eee535151b47137efe1d1c6c496eb7aae714588814d12eacd618558739d11
Status: Downloaded newer image for node:14-alpine3.16
 ---> 79912fb6c330
Step 2/3 : RUN npm install
 ---> Running in 76d26806867d
npm WARN saveError ENOENT: no such file or directory, open '/package.json'
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN enoent ENOENT: no such file or directory, open '/package.json'
npm WARN !invalid#2 No description
npm WARN !invalid#2 No repository field.
npm WARN !invalid#2 No README data
npm WARN !invalid#2 No license field.
```



Copying build files

If container needs those files while installing any required software

copy ./ ./

copy <source> <destination>

1st dot represents – path to the folder to copy from your machine relative to build context

2nd represents –path to copy stuff inside the container

Full program

1) Create a folder called simpleweb and paste those 2 files
   "mkdir simpleweb"
2) Touch dockerfile and edit it with below contents
   FROM node:14-alpine3.16
   COPY ./ ./
   RUN npm install
   CMD ["npm","start"]

```
manideepvv@DESKTOP-48ALSQ8:~/simplewebapp$ docker build -t manideep/npm:latest .
Sending build context to Docker daemon  4.096kB
Step 1/4 : FROM node:14-alpine3.16
 ---> 79912fb6c330
Step 2/4 : COPY ./ ./
 ---> c0be5d308d78
Step 3/4 : RUN npm install
 ---> Running in d7a8d5f2d9d8
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN !invalid#2 No description
npm WARN !invalid#2 No repository field.
npm WARN !invalid#2 No license field.

added 57 packages from 42 contributors and audited 57 packages in 13.278s

7 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities

Removing intermediate container d7a8d5f2d9d8
 ---> 2c61c3117ce0
Step 4/4 : CMD ["npm","start"]
 ---> Running in ef2cf9b31b16
Removing intermediate container ef2cf9b31b16
 ---> 493d58e95a18
Successfully built 493d58e95a18
Successfully tagged manideep/npm:latest
manideepvv@DESKTOP-48ALSQ8:~/simplewebapp$
```

Now after copying those 2 files, and I created image and tagged this name, it built successfully

And u can run the container with below tagged image name

docker run Manideep/npm

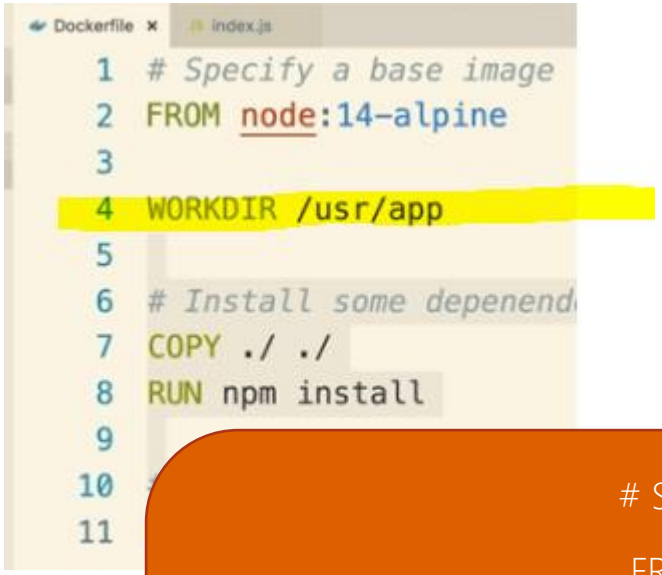
The problem with the above approach is it will copy those required files into the root directory

And it might replace the existing files ,

```
manideepvv@DESKTOP-48ALSQ8:~/simplewebapp$ docker run -it manideep/rnpm:v2 sh
/ # ls
Dockerfile          home                mnt                 package.json        sbin
bin                 index.js            node_modules        proc                srv
dev                 lib                 opt                 root                sys
etc                 media               package-lock.json   run                 tmp
/ #
```

See here docker copied our 2 files which we mentioned (packagege.json,index.js) those into existing root directory which should

Be avoided, and it should be copied to another specific folder as below



inside docker file u have to mention,where those 2 must be copied

Sample docker file

Specify a base image

FROM node:14-alpine

WORKDIR /usr/app #in this directory all those files will be placed any following

Command will be executed relative to this path in this container

Install some depenendencies

COPY ./ ./

```
manideepvv@DESKTOP-48ALSQ8:~/simplewebappWithBuildFolder$ docker build -t manideep/node:v2 .
Sending build context to Docker daemon  4.096kB

manideepvv@DESKTOP-48ALSQ8:~/simplewebappWithBuildFolder$ docker run -it manideep/node:v2 sh
/usr/app # ls
Dockerfile          index.js            node_modules        package-lock.json   package.json
/usr/app #
```

Now If u see all those files are placed somewhere

Container port mapping

Let's say we create a container (isolated space in the hard drive) and installed node js in that space that nodejs is running on port number 8080

And from browser u can't run hit the port 8080. Because that nodejs is running in an isolated space and this is a public port

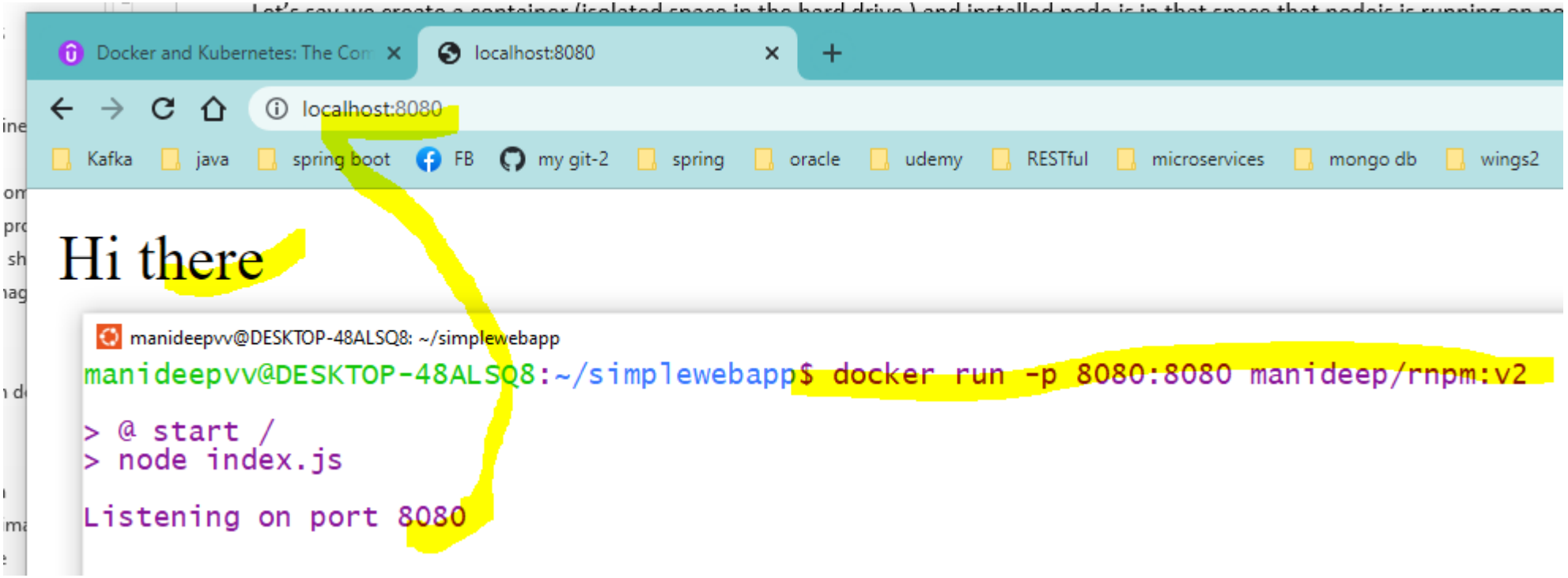
If u want it, this u have to map our system port with docker port and then only u can run

docker run -p <our general incoming port with> :

<docker container port where that s/w is actually running>

<tagged image name>

if u want to see ur app running on 8082 so that u can hit form browser, then use 8082 on left side,and map to where it is actually running inside container on right side-inside container if that app is already running on 5000, then give as 5000 in right side



Minimising the existing rebuilds

There are totally 2 files called index.json,package.json- the problem here is index.json contains only sysouts, even if we change this index.js

Where hello-world diagram contains, unnecessarily all the entire s/w will be downloaded again by deleting everything from cache, this can be avoided using RIGHT side approach, where

//here we are copying everything before we are downloading the s/w	Here we copy only 1 file,which we don't make any changes on it,hence s/w wont be downloaded again
# Install some depenendencies COPY ./ ./ RUN npm install	# Install some depenendencies COPY ./package.json ./ RUN npm install COPY ./ ./
For installing node we need both the files Here we copy everything , before we download and install node software	Every step identifies whether any changes made to previous step or not, If there are any changes to previous step everything will be downloaded again Whereas here, we copy only 1st one called package.json before installing node and after installing node we copy another file,called index.json Because, if we copy index.json first , then since there are changes to index.json now since this step onwards cache will be invalidated,and next step installing will not be used from the cache, it will delete all cache Hence we should install and then we should copy that index.json file after installing the software

Docker compose

Refer 53-54 Zips number folder for code

Separate CLI that gets installed along with Docker

Used to start up multiple Docker containers at the same time

Automates some of the long-winded arguments we were passing to 'docker run

And the main advantage is u don't need to build+run , both it will do as part of "docker-compose up"

In docker-compose.yml file u will write all the below commands

docker-compose.yml

Here are the containers I want created:

redis-server

Make it using the 'redis' image

node-app

Make it using the Dockerfile in the current directory

Man port 8081 to 8081

//Below are the contents of docker-compose.yml file

```
version: '3'
services:
    redis-server: //this says create a container called redis-server
```

If u want to run the image and if u want to create a container from it below is the command

This docker-compose up will look for docker-compose.yml file

Up- means "up" all the containers

docker run <image name>

docker-compose up

docker compose= docker build+docker run

Rebuilding all containers in yml

If u have source code changes-then do rebuild, It builds the source code responsible to create 2 containers

docker-compose up --build

```
docker compose up –d

// if u want to start all containers at once  use this option with –d flag
```

Rebuild everything and start all those containers again

Rebuild- so that u can get all those latest changes

```
//build the image using docker file present in current directory
docker build .
```

```
docker run <image name>
```

docker-compose up --build

STOPPING ALL CONTAINERS AT ONCE WITH DOCKER COMPOSE

This **docker-compose up** –d will start 2-3 containers at once, if u want to stop all containers at once

Use docker-compose down

```
manideepvv@DESKTOP-48ALSQ8:~/visits$ docker-compose up -d
manideepvv@DESKTOP-48ALSQ8:~/visits$ visits-node-app-1]) for
manideepvv@DESKTOP-48ALSQ8:~/visits$ d with the --remove-orpl
manideepvv@DESKTOP-48ALSQ8:~/visits$
manideepvv@DESKTOP-48ALSQ8:~/visits$ docker-compose down
[+] Running 2/2
 ⠿ Container visits-redis-server-1  Removed
 ⠿ Network visits_default           Removed
```

Restart policies

Restart Policies

"no"	Never attempt to restart this. container if it stops or crashes Mentioning ‘no’ in single quotes is
always	If this container stops "for any reason" always attempt to restart it
on-failure	Only restart if the container stops with an error code
unless-stopped	Always restart unless we (the developers) forcibly stop it

Restarting containers automatically inside ym file

```
version: '3'
services:
  redis-server:
    image: 'redis'
  node-app:
    restart: on-failure
    build: .
    ports:
```

In yml file no is interpreted as false,

So , if u want to provide restart policy as no, then provide in single or double quotes as

restart :'no'

to see all the running containers list type

docker-compose ps

we should have different docker files for each env

"dockerfile.dev"

Custom docker file name

So far , we worked without code, I mean we worked only with docker files, now we will start working with

code+docker files

for that refer project num "65-creating"," 67-starting" these proj will be there as part of my repo

1) Download nodejs software and test installation status using command “node –v” in command prompt
2) Create a sample node js samle project with command
   “npx create-react-app frontend”
3) The above command wont work in Ubuntu console, so create a project in windows folder and move
   That project to Ubuntu \\wsl.localhost\Ubuntu\home\manideepvv, type “\\wsl$”
4) In that create Dockerfile and add node related files like “index.js”,”package.json”
5) Since we are creating prod ready proj, instead of creating file named“dockerfile” created “Dockerfile.Dev”
   For dev environment

docker build –f <your custom docker file name> .

ex:- docker build –f Dockerfile.dev .

in above –f stands for custom file name and “.” refers to current directory

```
manideepvv@DESKTOP-48ALSQ8:~/frontendapp$ docker build -f Dockerfile.dev .
Sending build context to Docker daemon  724.5kB
Step 1/6 : FROM node:16-alpine
 ---> 8cf71856f96e
```

Run that container and map with system port

Rebuild when code changes are made

manideepvv@DESKTOP-48ALSQ8:~/frontendapp$ *docker build -f Dockerfile.dev .*

Step 6/6 : CMD ["npm", "run", "start"]

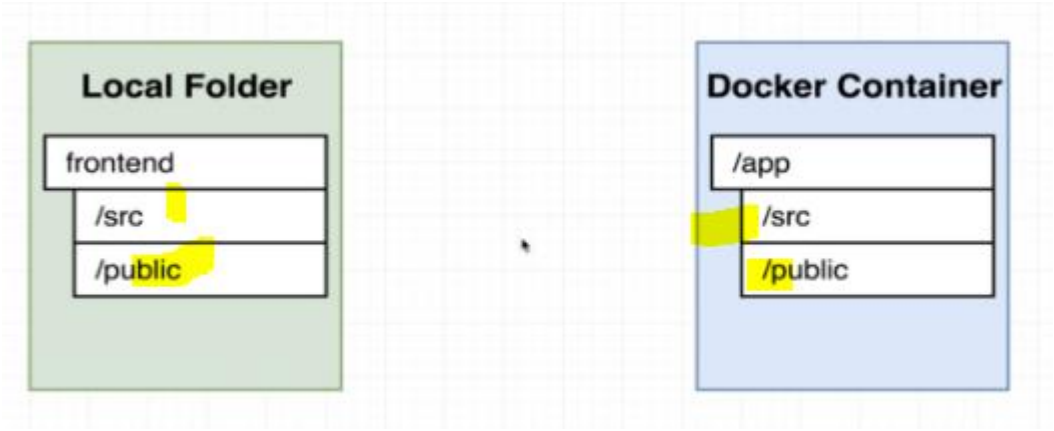
---> Running in 7a325cea22fc

Removing intermediate container 7a325cea22fc

---> 7ec45d18a115

Successfully built **7ec45d18a115**

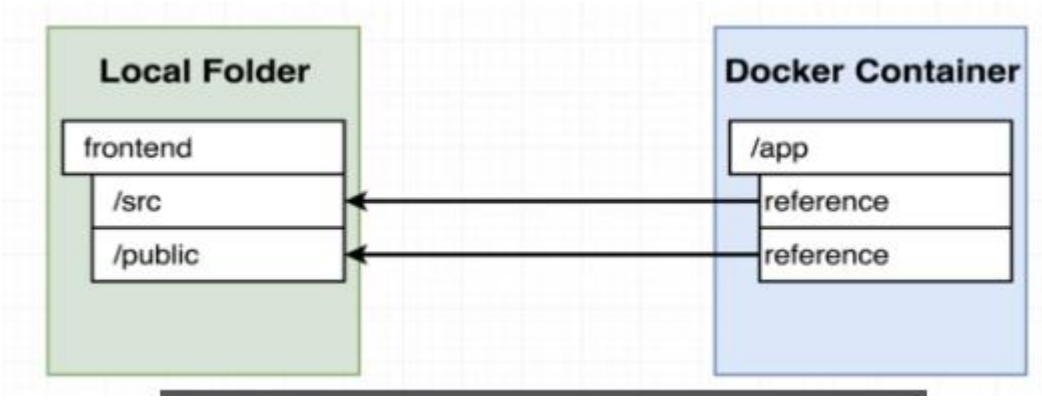
manideepvv@DESKTOP-48ALSQ8:~/frontendapp$ **docker run -p 8081:3000 7ec45d18a115**



when we rebuild again separate space is created

to that space all those files should get copied

Docker volumes



With docker volumes , u can map a folder inside a container to a folder outside a container

When code changes are made , we need to rebuild , same like java and deploy that code into jvm

Here also we should rebuild and instead of rebuilding ,we have a option of point to the source code from docker

Container , same like how we map the ports

Dynamic code refresh

Refer proj 70,71,72(docker compose yml file proj-best)

In production we will never create a docker volume,bec we don't want dynamic code refresh

In java we will build and we will get a jar file and we will deploy that jar file in jvm ,

Every time when we make a code change, we have to rebuild and deploy that latest jar in jvm/any server

Way 1:-

Here in docker world also, generally we have to do build every time and run that new continaer,

Way 2:-

Instead of building every time and getting the new image file we can go for hot code replacement

While running that container we cant point those requests to source code

So without rebuilding the image we can get the latest changes every time.

```
docker run -p 8080:3000 -v /home/node/app/node_modules -v $(pwd):/home/node/app
<image id/jar name>
```

here –p means port, first port is map with system port and 2nd one is docker port

-v means =volume

/home/node/app is the default folder created inside a container

-v /home/node/app/node_modules says don't map this folder called node_modules, this folder will be there in the container, if any request comes to this let it be, don't map to a folder outside of a container

And when request comes to this folder in the container, don't map this, because we have deleted node_modules

$(pwd) :/app means

```
Successfully built 007e76d4b40f
manideepvv@DESKTOP-48ALSQ8:~/frontendapp$ docker run -p 8080:3000 -v /home/node/app/node_modules -v $(pwd):/home/node/ap
p 007e76d4b40f

> frontend@0.1.0 start
```

Map a folder (/app) present inside a container to a folder outside a container

If anytime container reaches /app directory, its going to redirect to the pwd

what ever is there in pwd-present working directory map to app folder present inside a running container

```
docker run -it -p 3000:3000 -v /home/node/app/node_modules -v ~/frontend:/home/node/app USERNAME:frontend
```

Short hand docker compose

In docker file below are the commands

==========

RUN mkdir -p /home/node/app

WORKDIR /home/node/app

WAY 3:- USING DOCKER COMPOSE COMMAND

Alternate to big run command

The above run command is big, so even for 1 container also we can go with “docker-compose.yml” file

Refer “1.1.comments-dynamic code refresh .YML” in this directory git hub

```
version: '3'
services:
        web: ## this says create a container called  web
          build: . ## says build using the docker file present in this current   directory
          ports:
                -"8080:3000 ## here the application is running inside docker port 3000
```

The above will face issues, because build specifies "." Means current directory

Refer proj 72

Here context specifies

Where should we pull the information like files and folders

```
version:'3'
services :
        web:
         build:
                context:
```

Running tests without docker compose

Here also we will run those tests in a new container

docker run <image-id> <our new command>

remember with image we can create container only, so in above step new container will be created

docker run 858b47be28e9 npm run test

```
Step 8/8 : CMD ["npm","start"]
 ---> Using cache
 ---> 858b47be28e9
Successfully built 858b47be28e9
manideepvv@DESKTOP-48ALSQ8:~/frontendapp$ docker run 858b47be28e9 npm run test

> frontend@0.1.0 test
> react-scripts test
```

Create a new space /container & run this additional command [npm run test] in that space

But ,u can create space only when we have image

docker run -it <image name> <command name>

docker run -it 858b47be28e9 npm run test

run command will create a new container and execute this command

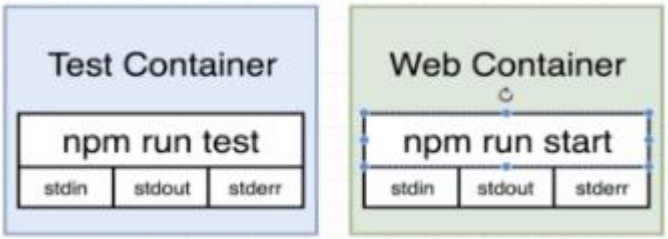
docker run -t manideep:node npm run test

Running tests with docker compose

Refer project 76(it has both for linux, for wsl) ," 2.0.seperate container to run the tests.yml"

It says create a separate container and change the start up command

Like below in docker-compose.yml file ,create a separate container called tests



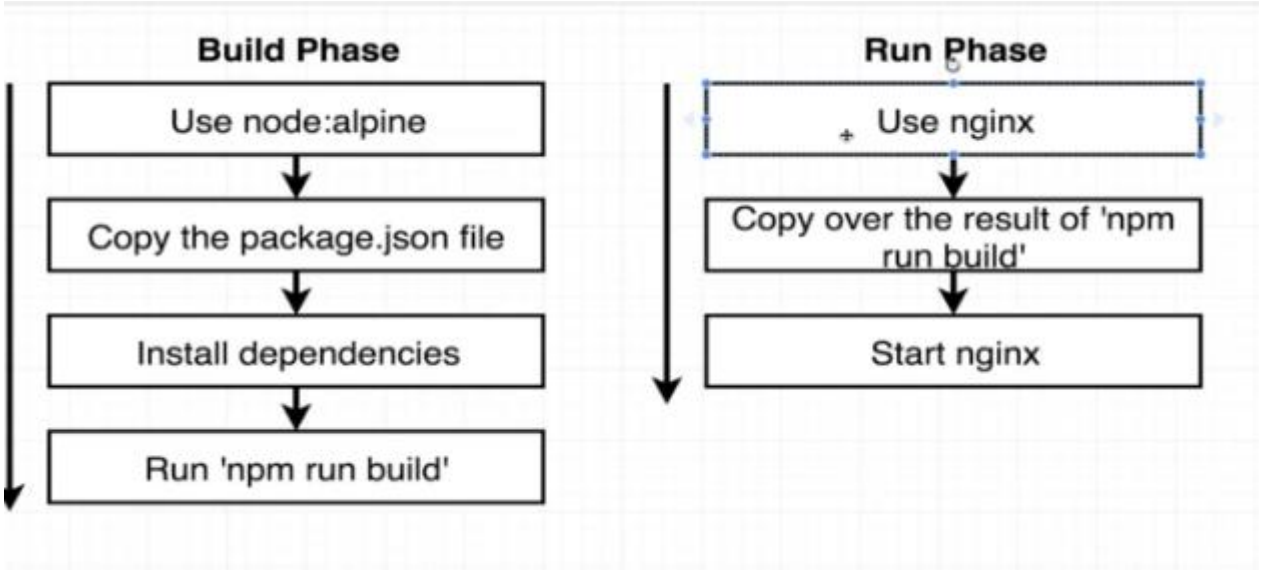
here 2 containers-both test container,web containers are seperate

```
tests:// says create a new container called tests
  stdin_open: true
  build:
    context: .
    dockerfile: Dockerfile.dev
  volumes:
    - /home/node/app/node_modules
    - .:/home/node/app  ##when any request comes to this folder ,map to pwd
  command: ["npm", "run", "test"]
```

Implementing Nginix –Multi step builds

Nginix is a server same like tomcat, it's a web server whereas tomcat is an application server

Refer prj number 80 wsl version for windows



So far our base image is only-1 , that is node:alpine version

But now we need 2 different softwares- we need nginix also

So we will follow multi step process, now the plan Is lets go with phase-1 ,execute "npm run build" command

And sample docker file and we will use the output of step-1 /phase-1 to the new phase

Refer “3.multi docker steps,,node,nginx.txt”

```
FROM node:16-alpine as builder #here builder is the phase name

WORKDIR ‘/app’  ## set the current dir to /app,so that here after all commands will be executed in this dir

                        # once proj is built ,build folder also will be created here

Copy package.json  .

RUN npm install

COPY  .  .

RUN npm run build


FROM nginx

COPY –from =builder  /app/build  / usr/share/nginx/html
```

```
2023/05/13 14:44:34 [notice] 1#1: exit
manideepvv@DESKTOP-48ALSQ8:~/frontendapp$ docker run -p8080:80 23544802c1ee
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
```

Nginx always runs on port 80

Once the image is created, ran that using image name

docker run <image name>