# 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

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

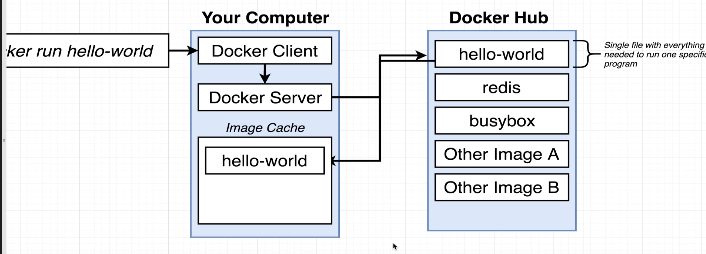
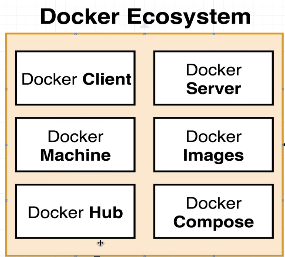
It’s a isolated space in HDD, where all the mentioned s/w in the image will be installed

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:-

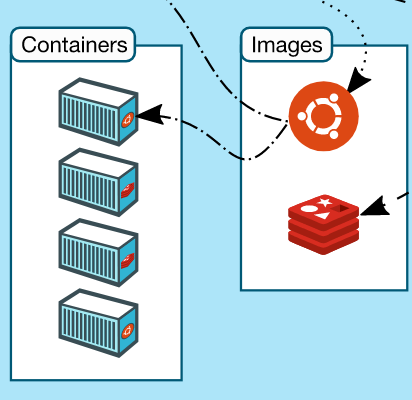
1. Container =Separate Drum, 1 drum with petrol,1 drum with diesel

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



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…

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

1. 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

1. Instance of an image== nothing but ,Object of a class
2. The files and folder u are creating in your local windows are present in the below folder

[\\wsl.localhost\Ubuntu\home\manideepvv](file:///\\wsl.localhost\Ubuntu\home\manideepvv)

1. 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,

1. From a running container also we can create a snapshot and we can create a image from a running container
2. 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

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

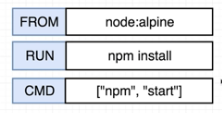
### Question and answers

1. Base image example

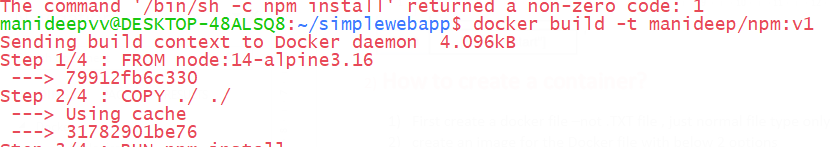
FROM node:14-alpine

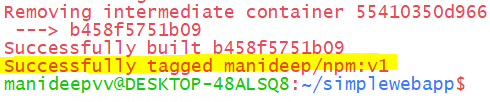
//Generally every image will have mostly alpine versions, it’s the most stripped and mini version of the image

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

here node is the repository and alpine is the tagged version.  
2) 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
3. **docker build .**”
4. **docker build -t manideep/redis:latest .**

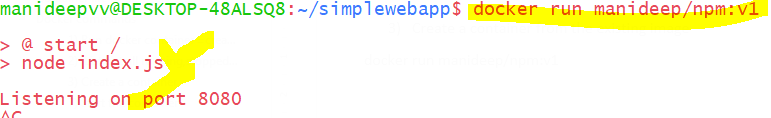




With this we successfully tagged the image with the name

1. Create a container from the existing image

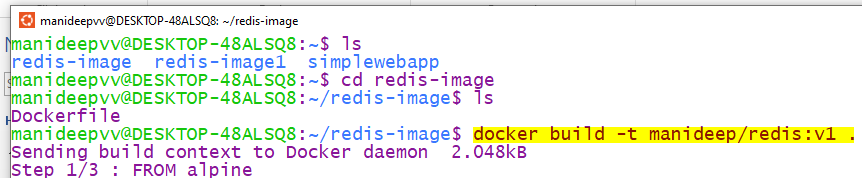
docker run manideep/npm:v1



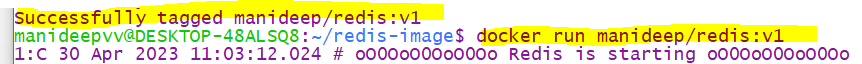
Creating image from docker file and running it

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



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

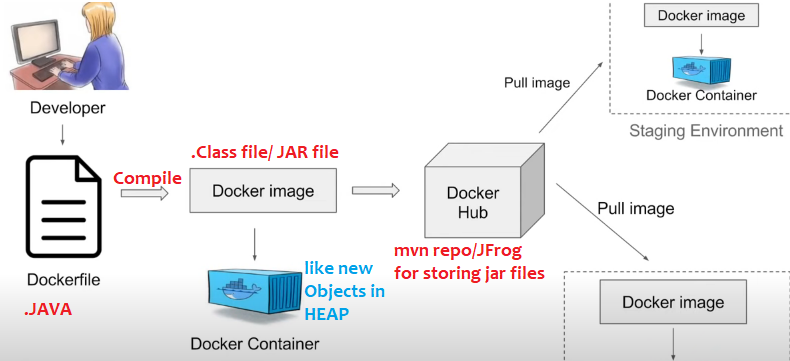
Dot . represents the current directory where the docker file is residing

1. *To run the image use below command*

docker run <tagged image name>coreiq/npm:v1

docker run coreiq/npm:v1

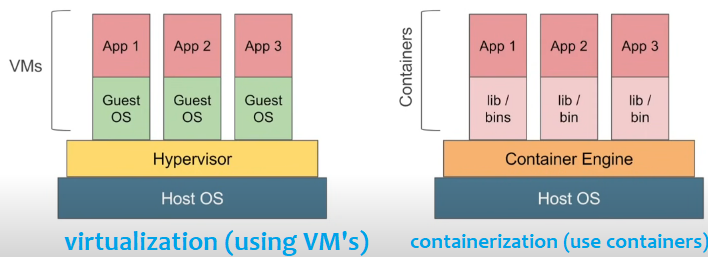
## **Docker flow**



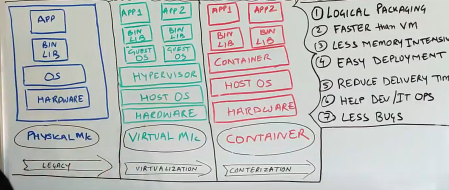
Here to run the application, if we need tomcat, JDK then we have to mention those as dependencies in docker file

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 client (CLI)

Docker 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:

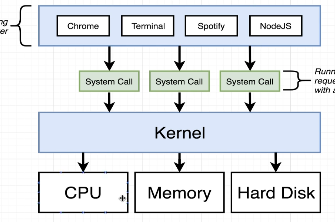
• The pid namespace: Process isolation (PID: Process ID). // each container running on separate port

• 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).

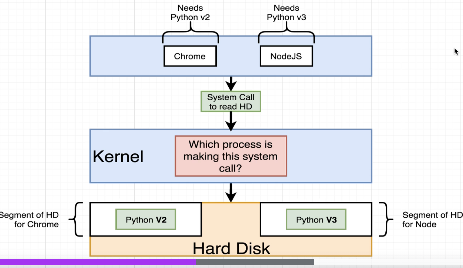


**Generally, if any application wants to interact with hardware (HD,RAM..)they must go via kernel only**

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

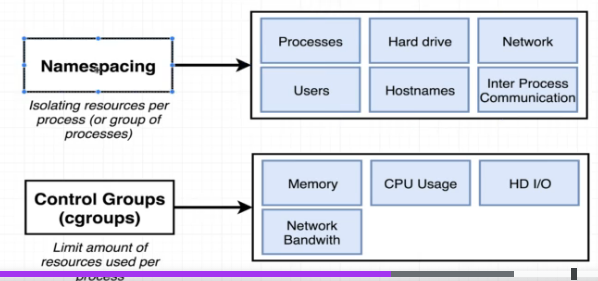
* With name spacing we can isolate resource per each process or for each application
* We can limit the resources for each and every process ,chrome only this segment of hard drive,

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

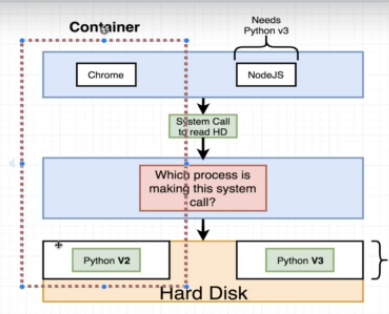
In isolated environments

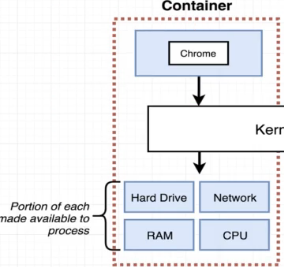
Containers are meant for isolated environments

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

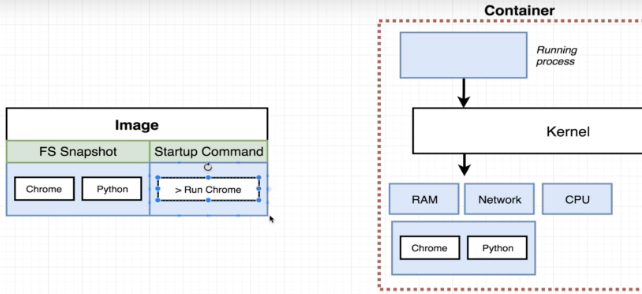


In left side,This entire set of running process and little portion of hardware is called a container

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

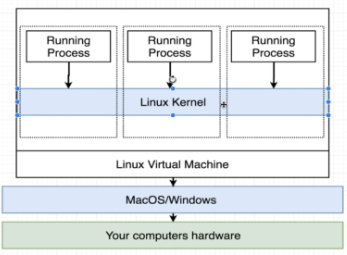


In the image , if u keep chrome, when u create a container from it, then you will be assigned with set of system resources like HDD,RAM,CPU..

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

# Docker commands

Create and running a container from that image

Docker run=Docker create+ Docker Start

1. Login to docker container

Docker login

First you have to login

1. 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

1. 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

### Run docker container =create+start

Docker run <image-name>

Ex:- docker run redis

Docker run **hello-world** echo hi there

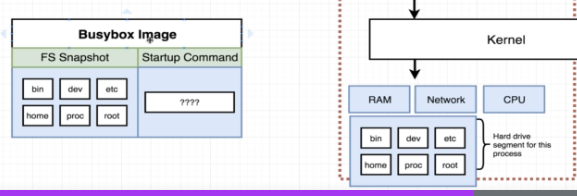
//where hello-world is the image name

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

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

### 2) List out all running,stopped containers

1) docker ps

2) docker-compose ps

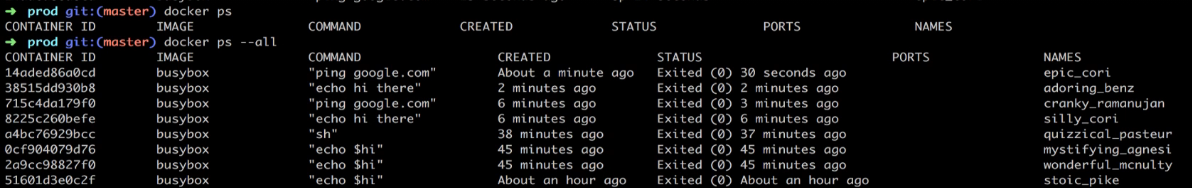
// this “docker-compose” will work only when we have a file called “docker-compose” in current directory

//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

### 7) Start the container

//TO START THE container we need the container id

We can start a container with some command

docker start –a <container-id>

**docker start -a e5b1760fea69a3414794b2f878c64d71237adef87ed2758b8a98f275a12eeafd**

//-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

### 8) Restart the died container

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

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

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

85558c5960fa busybox "echo hi my dear wife" 9 seconds ago Exited (0) 8 seconds ago elegant\_chatelet

The above is the ran and died container

No if u want to restart use below

manideepvv@DESKTOP-48ALSQ8:~$ **docker start -a 85558c5960fa**

hi my dear wife

where 85558c5960fa is the container id

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

### 9) Delete containers

This will delete all container and frees memory in hard disk

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) See the logs of a container

docker logs <container id>

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

### 11) Kill vs stop a conatiner

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

### 12) Executing additional commands inside running containers

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

exec- means execute this additional commands inside running containers

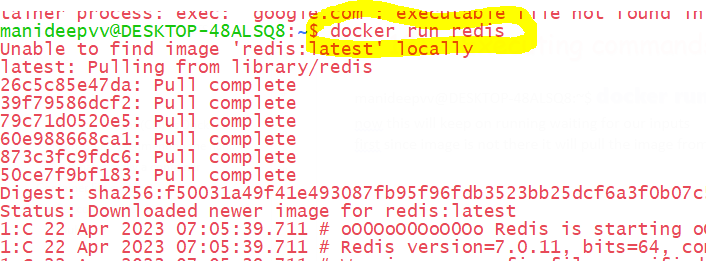
-it means connect my terminal to the input stream

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

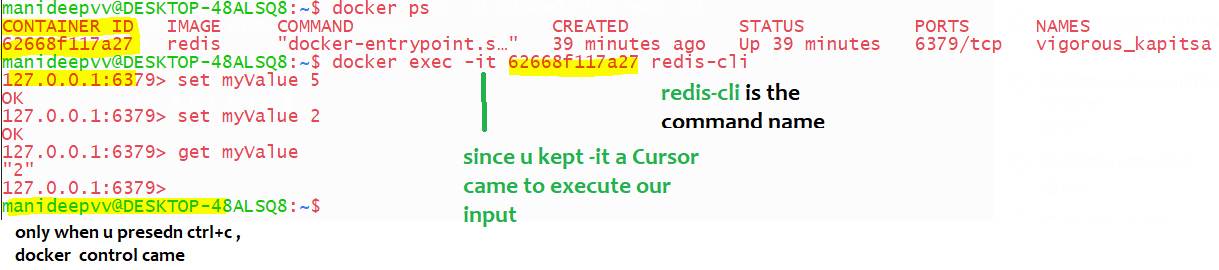


See now this window is continoulsy running and redis is continuously running , now it wont take any inputs

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



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

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) Getting the command prompt of a container

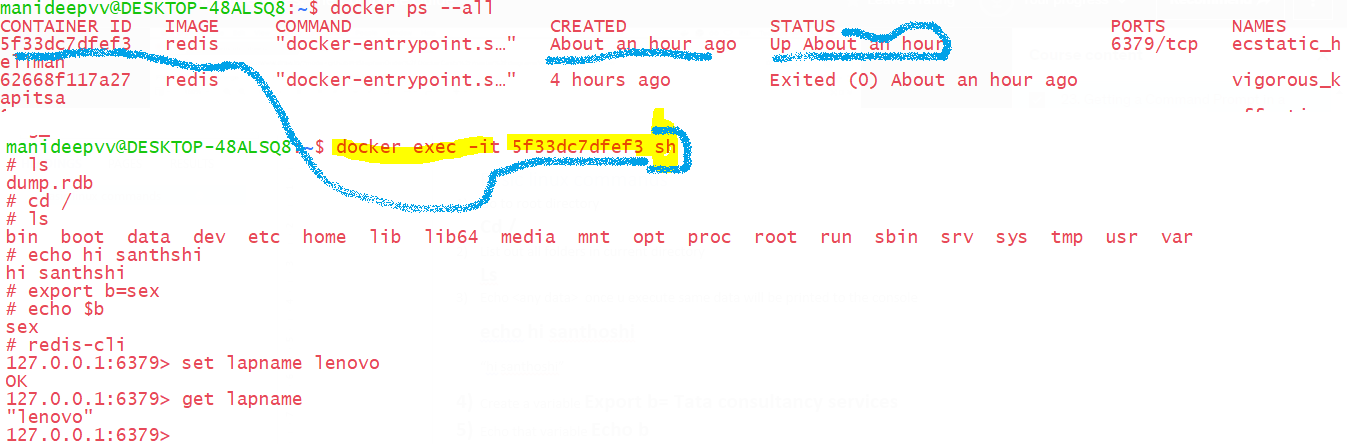
docker exec –it <container-id> sh

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”



15) Starting container with shell

docker run –it busybox sh

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

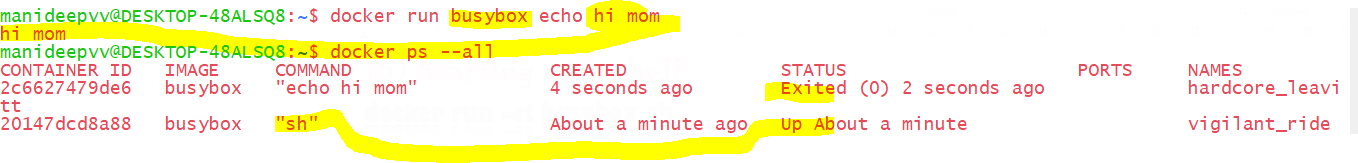
when u run the container like “docker run busybox echo hi there”,

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**



### 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 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**

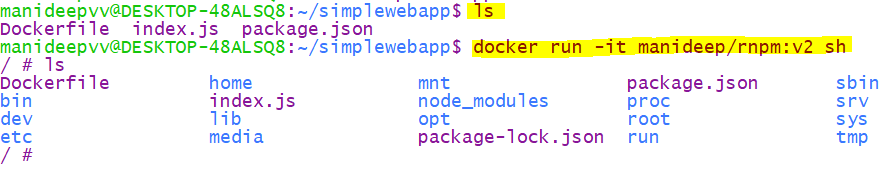
Way 1:- docker run <containerid>

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



1. Exposing a port from docker container to our machine

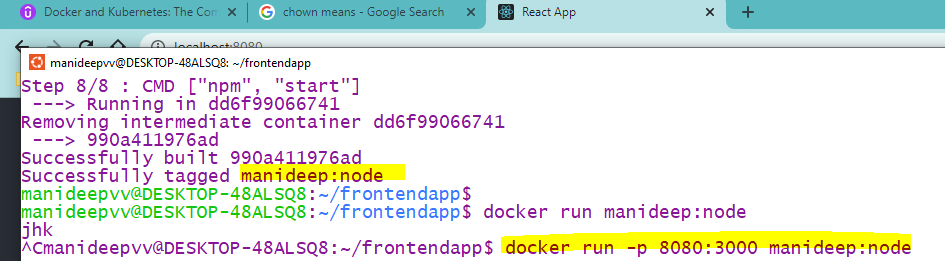
mapping and Running on different port: you have to map

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



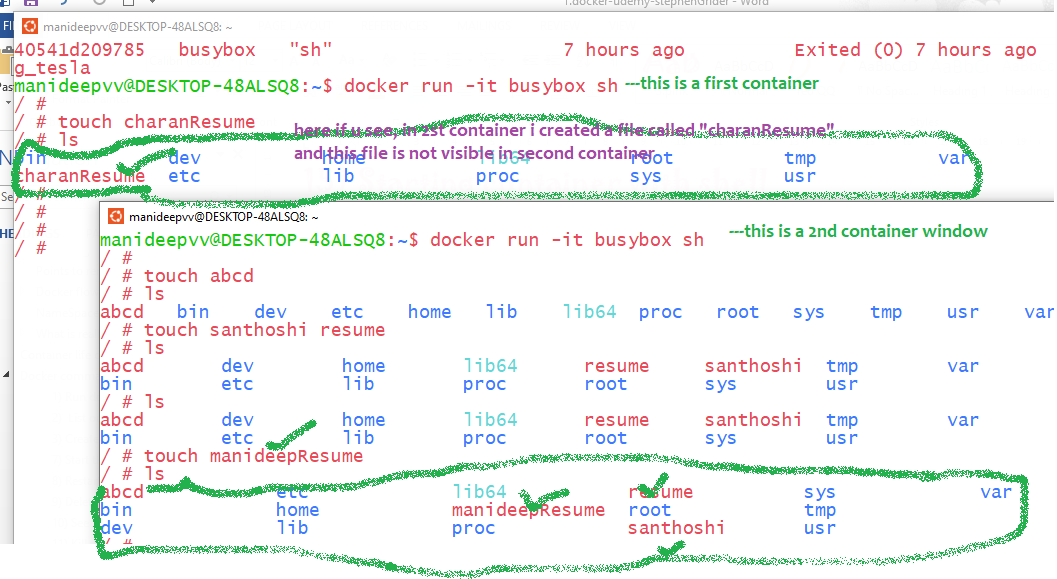
Here since we are mapping our windows port 8000 with docker container port 3000

## 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

## **Creating docker images**

Specify a command to run on container startup

Run some commands to install additional programs

Specify base image

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$](file:///\\wsl$)” like in windows we have “start .”

or go to location [\\wsl.localhost\Ubuntu\home\manideepvv](file:///\\wsl.localhost\Ubuntu\home\manideepvv) folder and open folder “redis-image” and see if file “Dockerfile” is created or not

With Extension File

1. 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"]

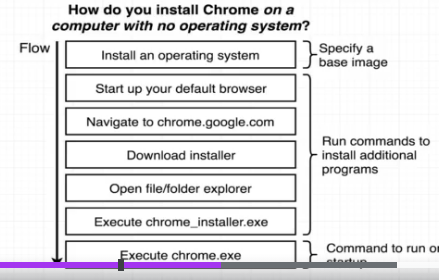
1. Type “**docker build .**”

Explanation of above commands

Alpine is also an image, first it will pull that image from docker hub

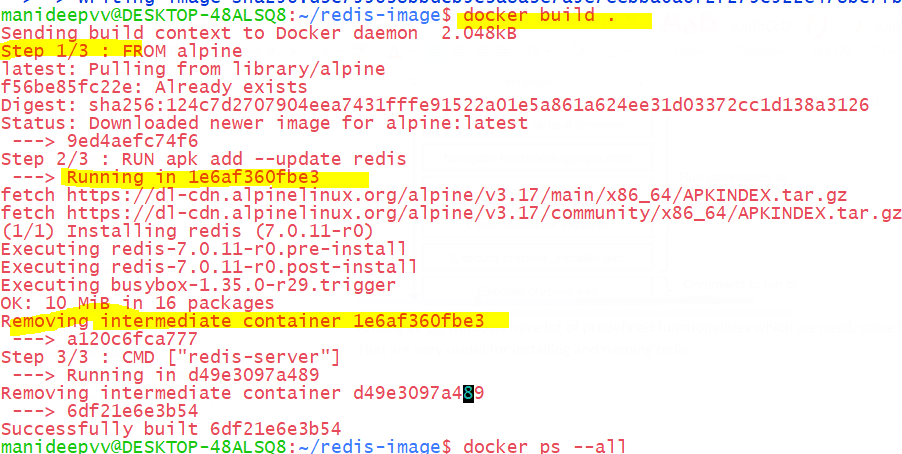
The above command “docker build .”is used to generate image from the docker file,

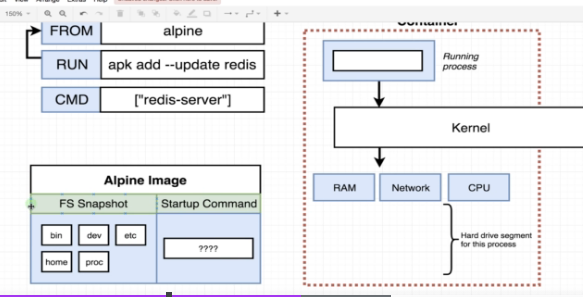
It will give the docker file to docker CLI, it will create a image from the given docker file



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





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

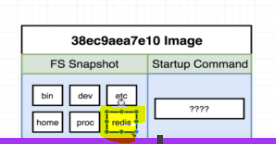
Here also in every intermediate step one container is created and based on that 1 image is created,

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

1. see the container &create a File system snapshot from it (this is called image and this image will be cached..all

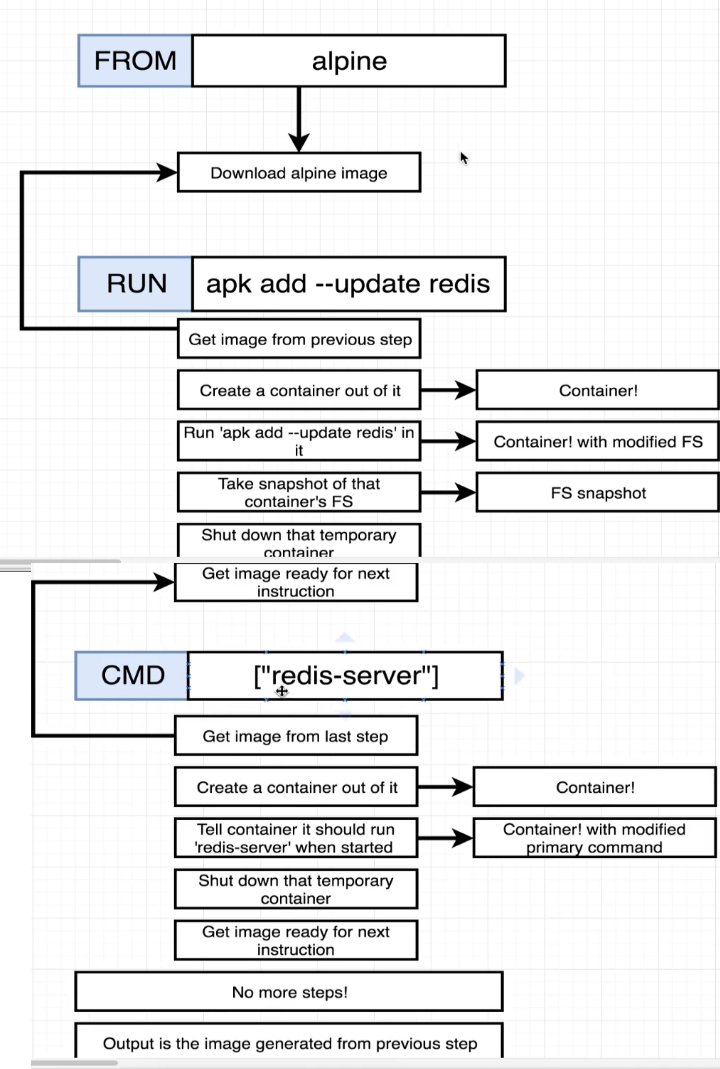
intermediate images will be cached for future purposes

1. 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



Caching the intermediate images

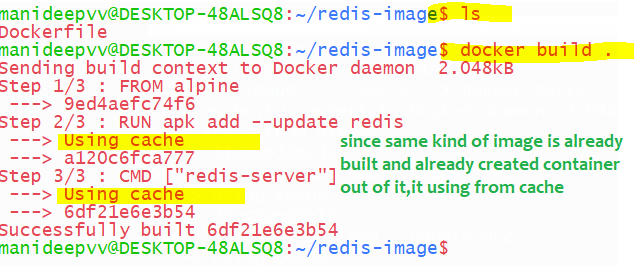
We don’t need to do it manually, automatically the intermediate steps images will get cached.

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



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 .

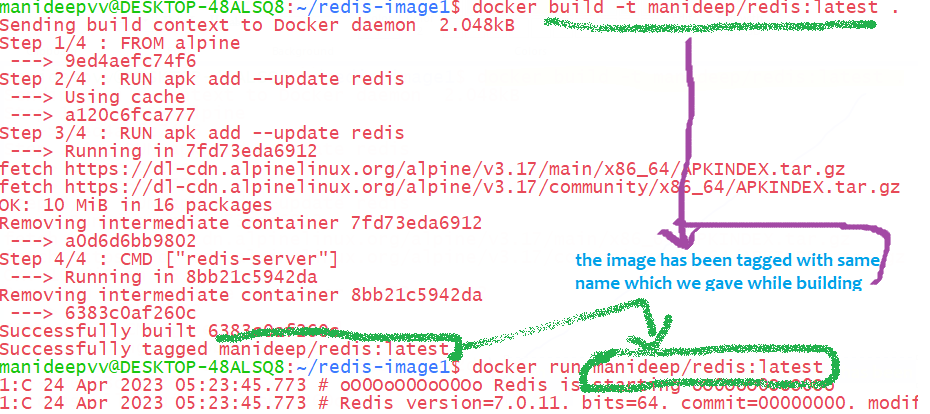
1. in the above dot at the end specifies the direct of folder/files where the docker file is available to use for the build
2. we should specify where the docker file is present, whereas dot represents the current directory
3. –t stands for tag

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

1. Install any software like redis

apk add --update redis

1. Open a new terminal and Get the container id of running process

docker ps --all

1. 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

## 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

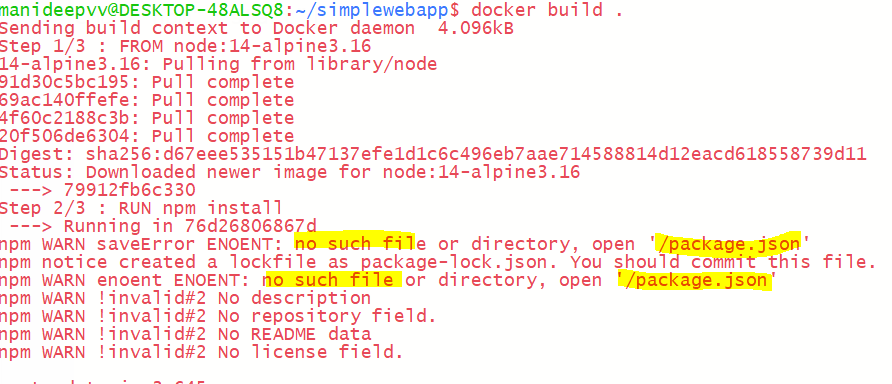
Steps of creating this container

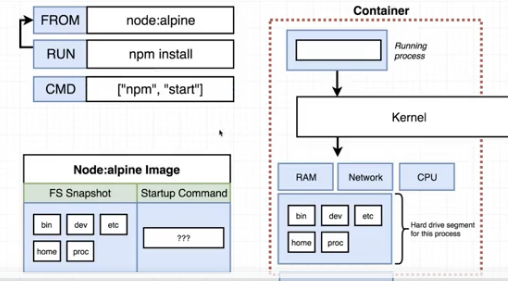
1)Base image [node:14-alpine] will be pulled and as usual, since all s’w are mentioned in the image ,container will be created from that image in HDD

And s/ws will be installed,

1. 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





### **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”

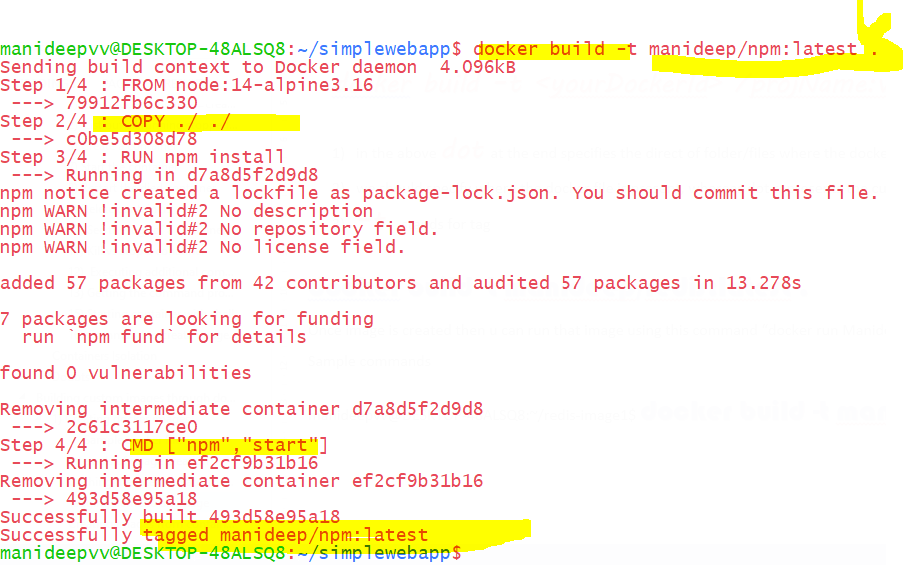
1. Touch dockerfile and edit it with below contents

FROM node:14-alpine3.16

COPY ./ ./

RUN npm install

CMD ["npm","start"]



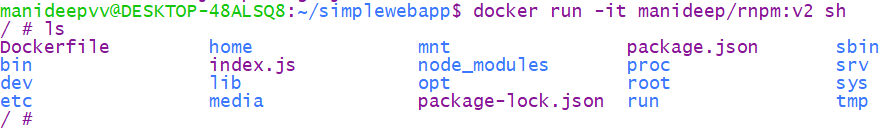
Now after copying those 2 files, and I created image and tagged this name, it built successfully

And u can run the container with below tagged image name

docker run Manideep/npm

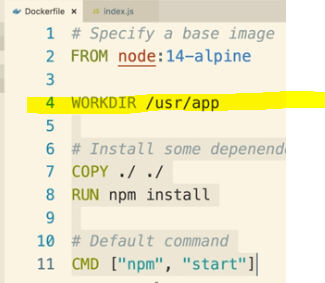
The problem with the above approach is it will copy those required files into the root directory

And it might replace the existing files ,



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

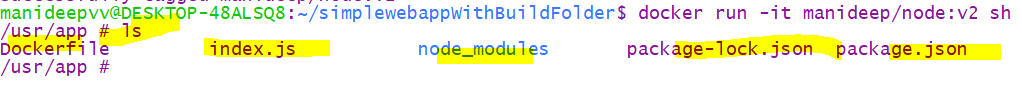
COPY ./ ./

RUN npm install

# Default command

CMD ["npm", "start"]





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

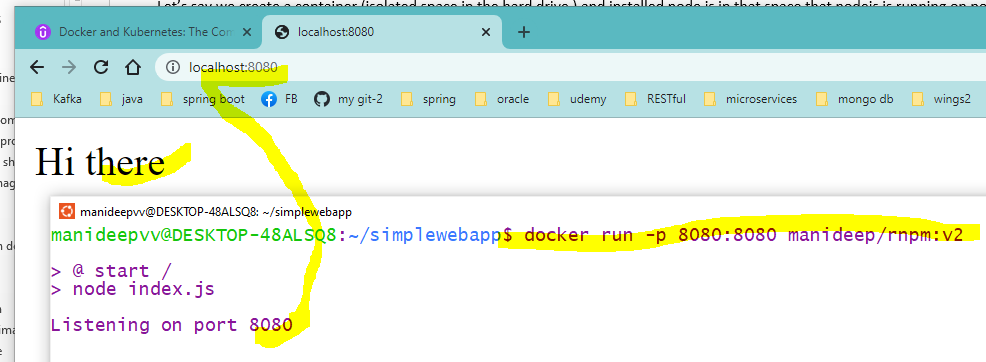
And from browser u can’t run hit the port 8080. Because that nodejs is running in an isolated space and this is a public port

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

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

image: 'redis' //using the image named redis

node-app: //this says create a container called node-app

build: . // this says create the above container using this docker file present in the

//current directory, where node needs 2 addnal files index.json..

ports:

- "4001:8081"

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 all the containers “up”

docker run <image name> 🡪

docker-compose up

Rebuilding

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

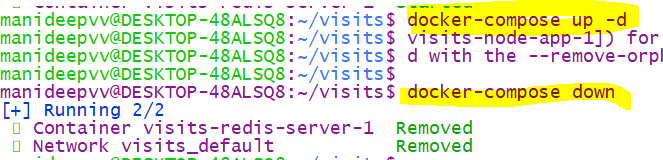
|  |
| --- |
| //build the image using docker file present in current directory  docker build . |
| docker run <image name> |

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



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:

- '4001:8081'

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”

1. 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](file:///\\wsl.localhost\Ubuntu\home\manideepvv), type “\\wsl$”

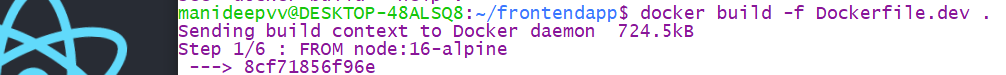
1. In that create Dockerfile and add node related files like “index.js”,”package.json”
2. Since we are creating prod ready proj, instead of creating file named“dockerfile” created “Dockerfile.Dev”

For dev environment

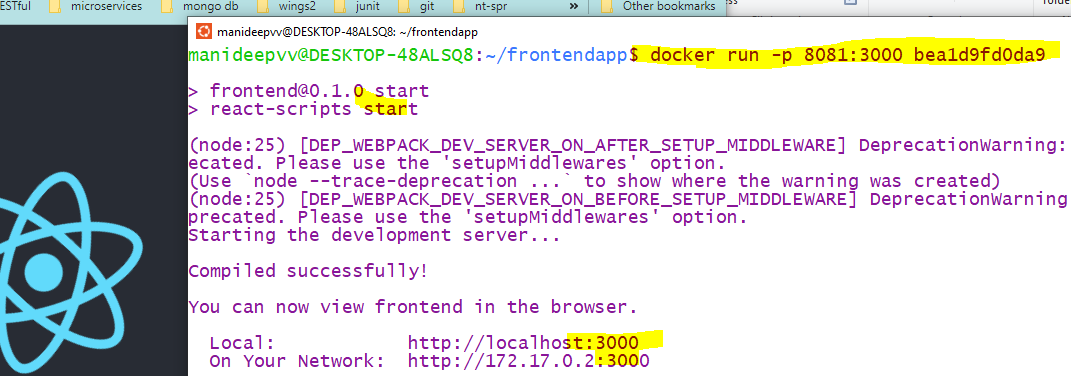
docker build –f <your custom docker file name> .

ex:- docker build –f Dockerfile.dev .

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



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"]

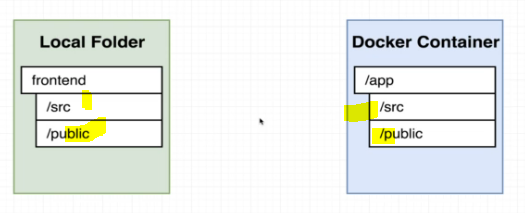
---> Running in 7a325cea22fc

Removing intermediate container 7a325cea22fc

---> 7ec45d18a115

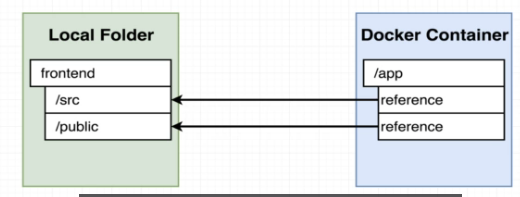
Successfully built **7ec45d18a115**

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

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

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

Instead of that,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 8000:3000 –v /app/node\_modules –v **$(pwd) :/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

–v /app/node\_modules says don’t map this folder called node\_modules, this folder will be there in the container

And when request comes to this folder in the container, don’t map this

**$(pwd) :/app** means

Map a folder 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

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

version: '3'

services:

web: ## this says create a container called web

build: . ## means create a container called web using the docker file present in this current directory

ports:

-"8080:3000 ## here the application is running inside docker port 3000

map the system port 8080 to docker container port 3000

so when we hit 8080 in our browser this is mapped to container port 3000

volumes:

- /app/node\_modules

- .:/app # this says when request comes to app folder inside a container that will be

#mapped to current directory