# 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

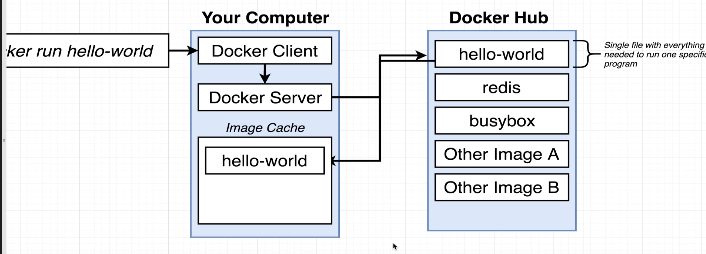
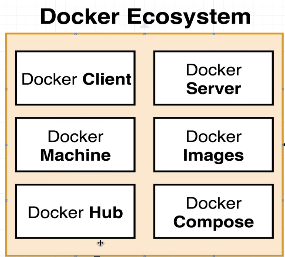
It’s a isolated space in HDD, where all the mentioned s/w in the image will be installed

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:-

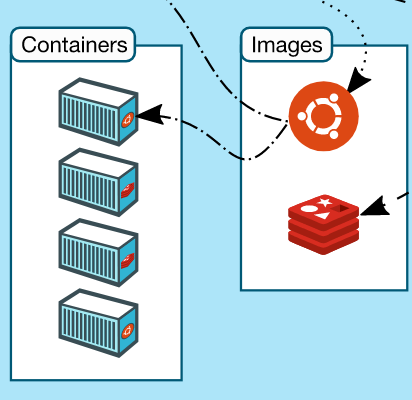
1. Container =Separate Drum, 1 drum with petrol,1 drum with diesel

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  Docker host is nothing but a machine where docker is installed |
| 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 software’s 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

Terminologies

Host machine-machine where docker is installed

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.

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

1. 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 Engine

Docker engine is the Docker core component that is responsible for creating Docker Images and running them as services. Like we use “javac” tool to compile and get the .class file , here docker engine is used to create the image

1. If image is found locally it will use it,else it will download from internet
2. The machine where docker is installed is called-docker host

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

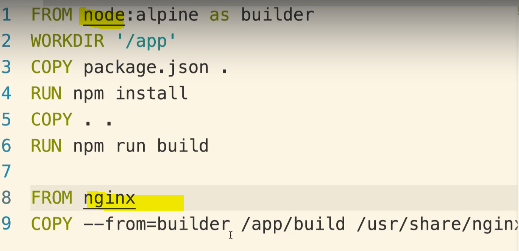
In the container

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

1. 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

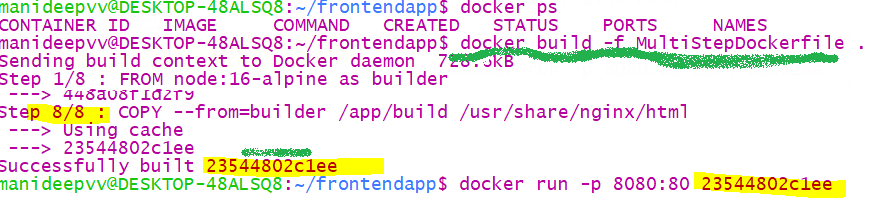


Here in the image, u mentioned s/w like node, nginx, then those 2 softwares will be installed in the HDD isolated space

1. 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 .



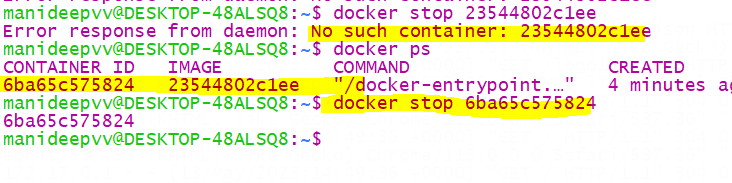
When we built, we got an image name/jar file ,each jar file having groupid, artifact id

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



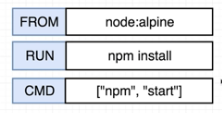
If u see here o tried “docker ps”, we can see the running container id

1. Base image example

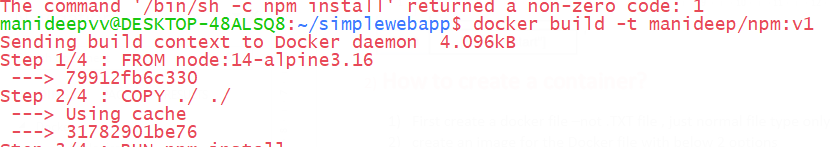
FROM node:14-alpine

//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.  
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
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



1. 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

1. 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

1. 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

1. 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)

1. 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

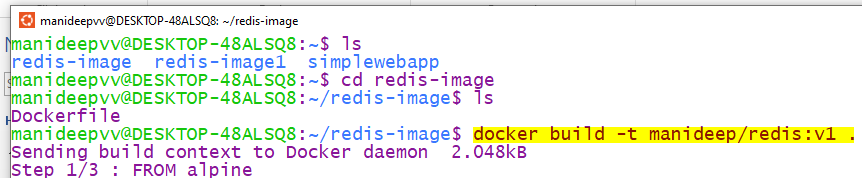
here if u see, in second line “build” was written after giving 2 spaces

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

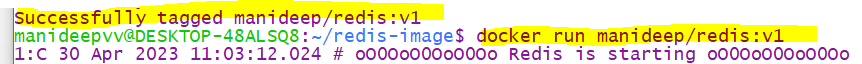
Creating image from docker file and running it

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



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

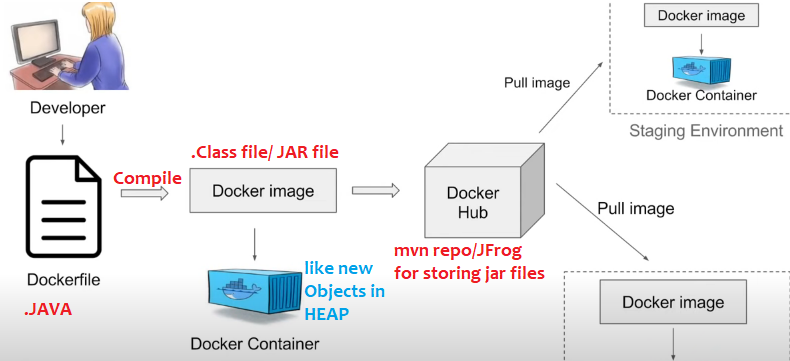
Dot . represents the current directory where the docker file is residing

1. *To run the image use below command*

docker run <tagged image name>

docker run coreiq/npm:v1

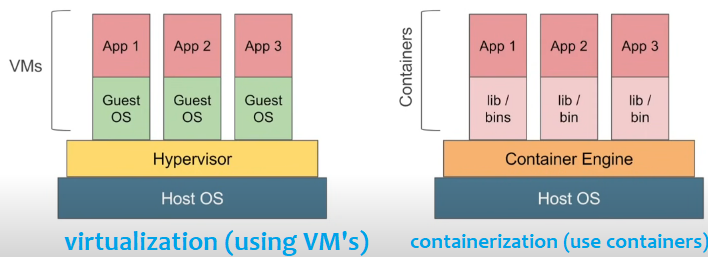
## **Docker flow**



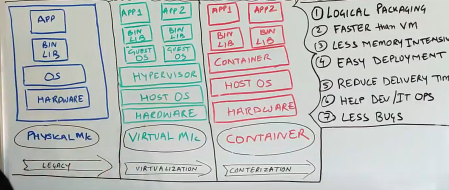
Here to run the application, if we need tomcat, JDK then we have to mention those as dependencies in docker file

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:

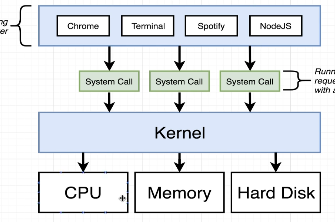
• The pid namespace: Process isolation (PID: Process ID). // each container running on separate port

• 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).

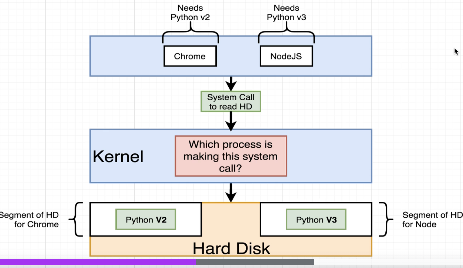


**Generally, if any application wants to interact with hardware (HD,RAM..)they must go via kernel only**

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

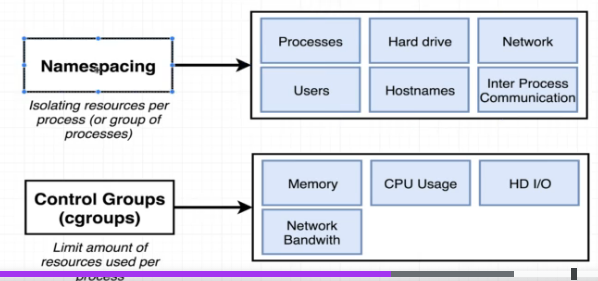
* With name spacing we can isolate resource per each process or for each application
* We can limit the resources for each and every process ,chrome only this segment of hard drive,

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

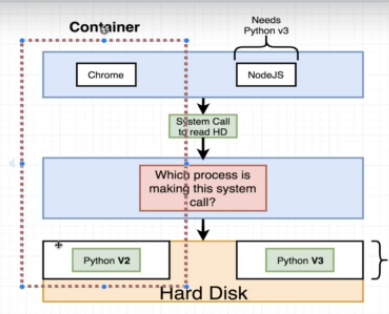
In isolated environments

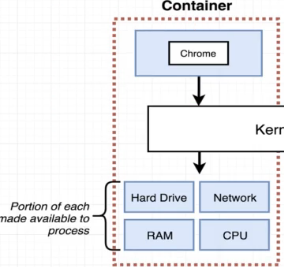
Containers are meant for isolated environments

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

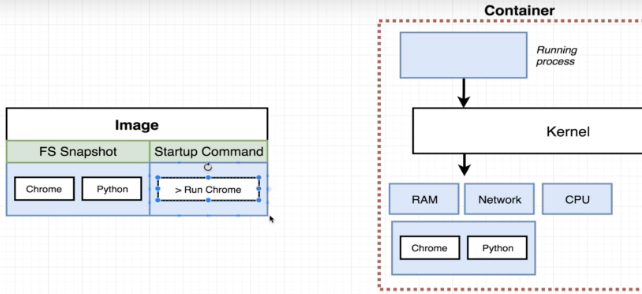


In left side,This entire set of running process and little portion of hardware is called a container

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

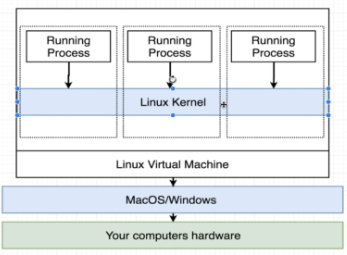


In the image , if u keep chrome, when u create a container from it, then you will be assigned with set of system resources like HDD,RAM,CPU..

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

## Sample image names with tags

Jenkins:latest

Nginx:latest

nginx:stable-perl

|  |  |
| --- | --- |
| nginx:latest |  |
| nginx:stable-perl |  |
| jenkins/Jenkins:latest |  |
| tomee | /this is tomcat image official name, the port might be 80 |
| Ubuntu |  |
| mysql:5 |  |
| bibinwilson/docker-kubectl:latest | In this image kubectl software is also installed |
| stephengrider/multi-client |  |

# Docker commands

Create and running a container from that image

Docker run=Docker create+ Docker Start

Basic commands

1. When u opened a shell inside a running container- to exit from a container command prompt type “Ctrl+PQ”
2. How to check whether docker is running or not? Just type “docker version”

Check docker is running or not

Just type “docker version”

Or type “docker info”

How to start a docker service –“service docker stop”

|  |  |
| --- | --- |
|  |  |
| Restart a docker service | sudo service docker restart ↵ |

If u want to just update inside a docker container then try –“sudo apt-get update”

Installing curl in ur Ubuntu container

“apt update && apt install curl -y”

Here apt is advance packaging tool, it is the packaging manager for linux

When u want to do “curl 172.17.0.3”means when u want to connect to the another container,if this curl command didn’t work

Then u have to install curl in your container using above command

2)Installing vi/visual editor

“apt update && apt install vim -y”

This vi editor is to open and edit it

1. Installing ping in the container

“apt update && apt install iputils-ping -y“

1. sh vs bash

Some times, in ubuntu or in someother conatiners,bash wont be there, only shell will be present

Always prefer working with bash

4)install git

“apt update && apt install git –y”

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

### Download an image

If u want multiple continers, then prefer busybox,ngingx

docker pull nginx:latest

docker pull jenkins/Jenkins:latest

docker pull nginx:stable-perl

docker pull tomee //this is tomcat image official name

docker pull Ubuntu

docker pull mysql:5

docker pull bibinwilson/docker-kubectl:latest

the above image “docker-kubectl” is a special image where kubectl is already installed

image names:-

busybox contains tiny versions of many unix utilities into single small executable

wordpress,redis,busybox

No need to pull the image explicitly, when u use “docker run <image-name>” then it will download automatically

More about images

===================

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

### 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

### Create container ,run =create+start

Tip: always remember –when u create container-decide what is the name and modes

If u are giving double hyphen then it means after double hyphen param some arg u are passing

1. “–-name <container-name>” N-name a container (--name <container-name>)
2. “-d” D-run in detached mode (--d ) or
3. “-it” I- run in interactive /console mode if necessary (-it)-if u start in interactive mode container wont die immdly

U can interact with container

1. “–p”-port (-p s/m port :docker port ex:- -p 8080:80//80 means where the server is running in that docker cont)
2. “–P” (capital p for automatic port mapping)
3. “–e“ for passing environment variable
4. “–-link < target container name> “-old -obselete
5. “--network <network obj name>” to connect your container to a network,by def bridge network
6. <any startup command name>;-- docker run –name n1 nginx /bin/bash
7. “-v <copy to folder>:<copy from folder>” –its docker volumes
8. “-—volumes-from <existing container name>”-to copy the existing docker cont volumes

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 --name j3 -it -P jenkins/jenkins:latest

1)scenario start a tomcat container from image tome & name it as webserver,perform port mapping & run in detached mode

docker run -–name webserver-p 8080:80 -d

docker run –-name

-p for automatic port maping

docker run <image-name>

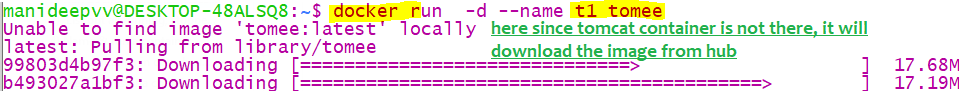
Ex:- docker run redis

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

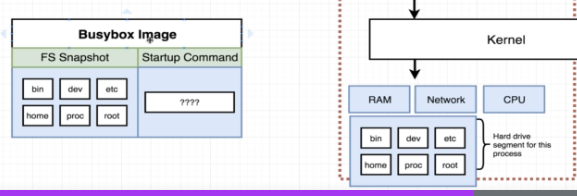
// whereas **echo hi there is** the command that will be executed after creating the container



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

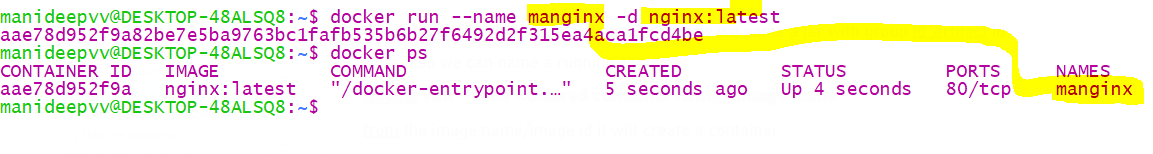
Similarly , we can name a running container, means we can give to a name running space

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



#### Run with Container modes- detached, interactive modes

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 –it <image-name>

Interactive mode

If u start the container in interactive mode, container wont die, and u will get a console of that

If u want to exit from that , then press “ctrl+PQ”, u can come out of it and container will be keep on running

some times ,if the image is not server/Jenkins, container will die immediately, to avoid that use “-it”

interactive mode, so that container will be present for longer time

docker run -d nginx:latest

## it will create container using that image



Sample commands

docker run -d --name u1 ubuntu:latest

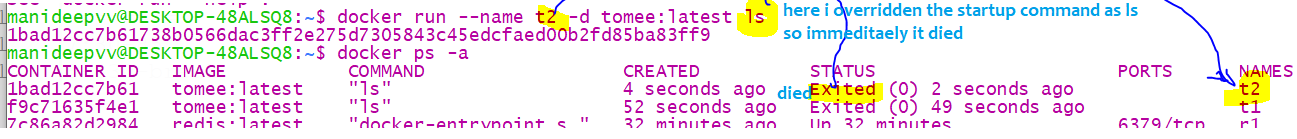
docker run -d --name n1 nginx:latest

#### **3) Overriding the start-up command**

Generally in docker files we have [CMD[“start”]], u can over ride that command with this

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

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

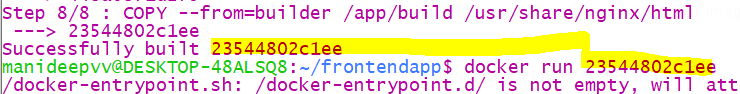


#### **4) Diff b/n docker run & docker exec**

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>



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

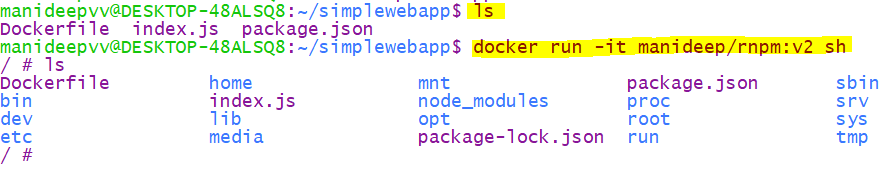
Way 1:- docker run <image id>

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



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

docker run 858b47be28e9 npm run test

docker run -it 858b47be28e9 npm run test

#### 5) Passing environment variables

docker run –name <container-name> –it –e varname=var pass <image-name>

docker run –name n1 –it –e mysqlpass=abcd123 anotherkey=anotherValue

#### Exposing a port of a docker container

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

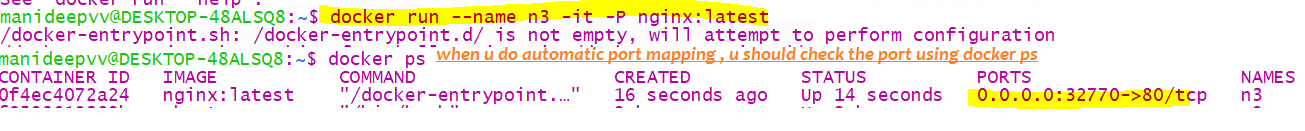


Here since we are mapping our windows port 8000 with docker container port 3000

Automatic port mapping

Use “–P” flag

A port will be automatically chosen



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

#### linking with a running container

docker run –-name <b2-container-name> –-link <b1-container-name> <image:tag>

if u want connection from b2 to b1 , 1st create b2 and then link with b1 while b1 is getting created

#### Linking with a network

By default all containers are connected to bridge network, they are pingable only via Ip address

U can create a custom network which is a child of (bridge,host)

docker run –-name <container-name> -it –-network <new network name> <containerName:version>

sample commands

====================

1. Connect with custom Bridge network

docker run --name b3 -it --network my-net busybox:latest

docker run --name b1 -it --network my-net busybox:latest

docker run –-name Cont1 –d –-network bridge\_sample nginx:latest

in the above my busybox container is connected to custom network

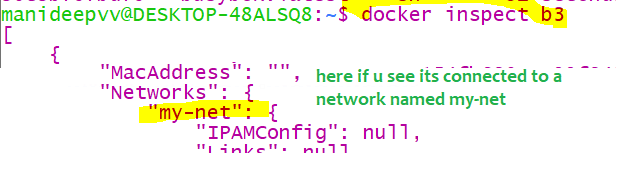
named “my-net ”which is a child of bridge network

2 options to verify either inspect container or inspect network, here I have inspected my container

1. Connect with host network

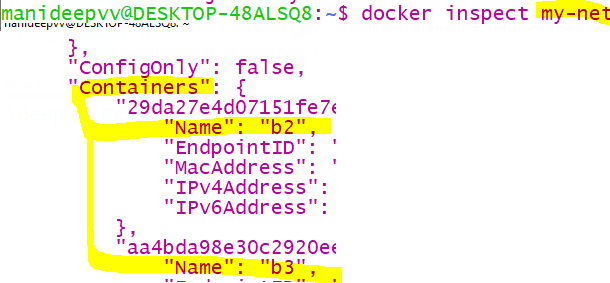
docker run --name n2 -it --network host nginx:latest

Inspect a container



Inspect a network

If you inspect a network then also u can see containers attached to it



#### Docker volumes

1. To see the volumes list

docker volume ls

1. Create un named volumes/anonymous volumes

docker run --name n2 -it -v /data01 nginx:latest

docker run -it --name n2 -v /data01 nginx /bin/bash

here “/bin/bash” is the startup command which will be executed when nginx container is getting started

docker run –name <container-name> -it –v <target to store>:<source folder to copy from>

docker run –name <container-name> -it –v <source folder to copy from>

containers are ephimeral

here all the files present inside a container will be stored externally in docker host, that’s the main motto, so that even if container died

files stored on server will be protected

1. Create named volumes

docker run --name n1 -it -v myvol:/etc ubuntu

### 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

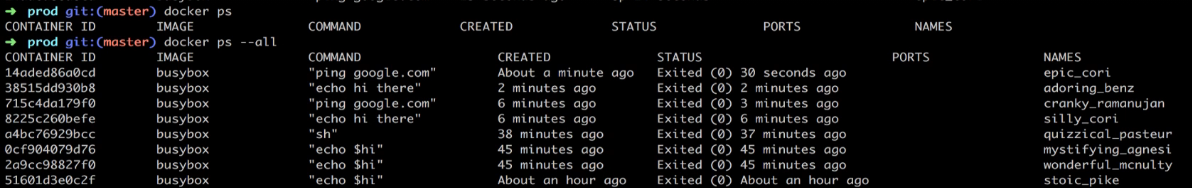
// this “docker-compose” will work only when we have a file called “docker-compose” in current directory

//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……



1. 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

5)To exit from a container

Ctrl+PQ

If u start the container in interactive mode ,means whenever u used –it flag

Like “docker run –name u1 –it Ubuntu:latest” or “docker exec -it u4 bash” then u will get the container console

If u want to get back from that container console, then type “Ctrl+PQ”

### 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> <container-id/name> <container-id/name>//

docker stop n1 n2

docker stop $(docker ps) –means stop all running containers

u can stop multiple containers at once

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

Dangling image- when image related container is running then if we try to remove that image,it will be untagged/dangling

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?

**F**irst 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

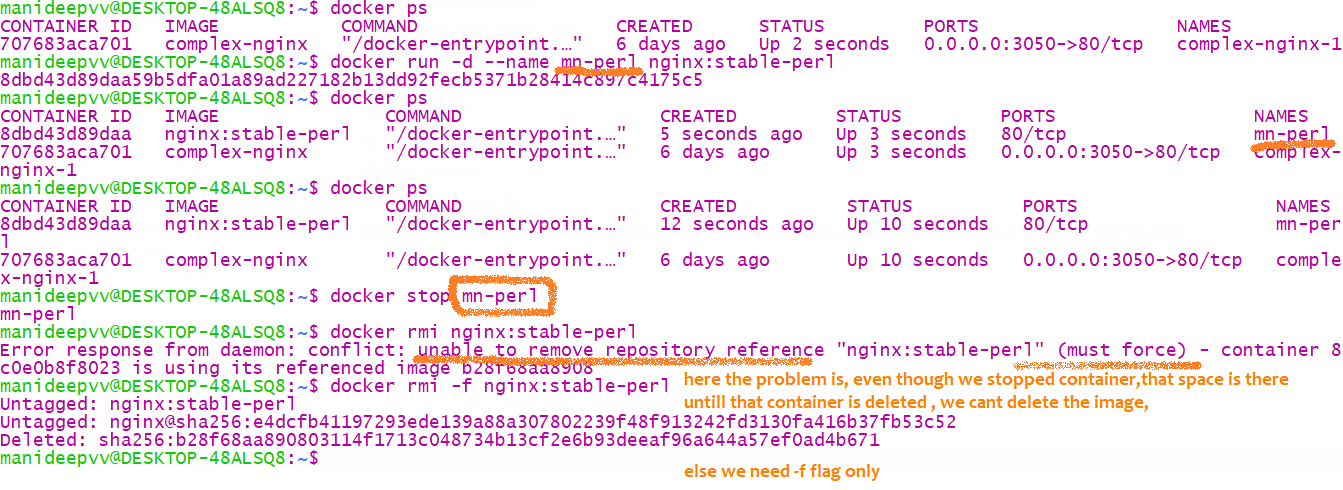
rmi-means remove image

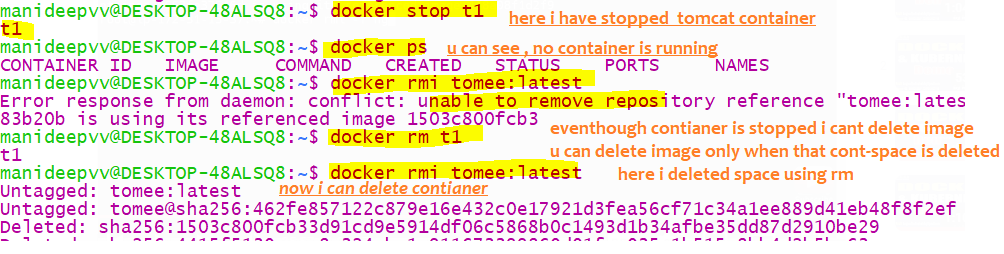
it will remove that single image ,

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

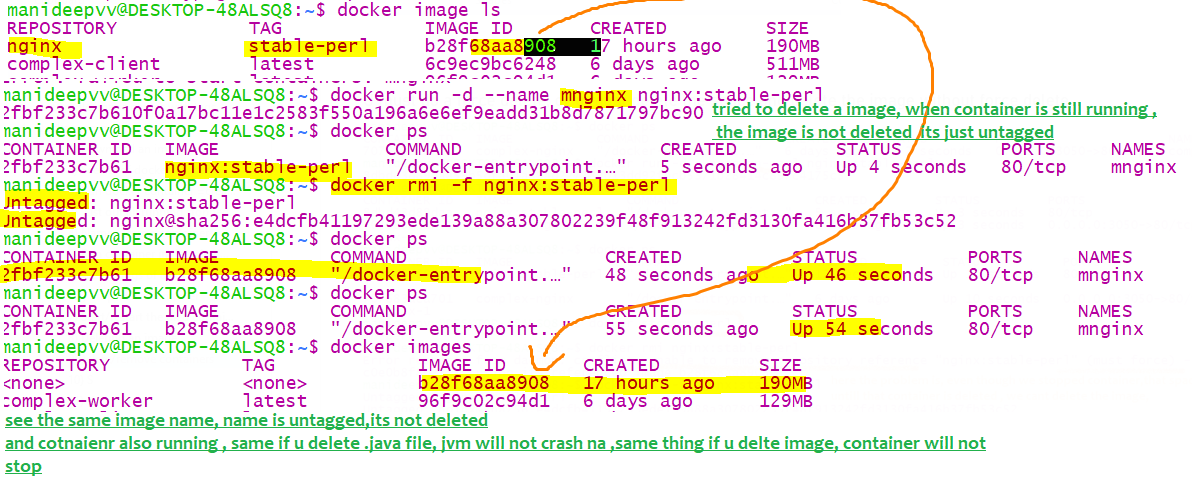




Delete an image when container is running

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

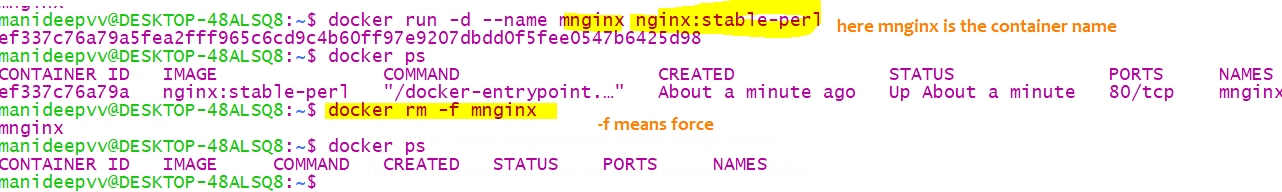
This will delete only space/container-image wont be deleted,

**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

**Deleting all containers**

docker rm –f $(docker ps -aq)

**Deleting all running containers**

Here the docker ps will give u the list of running containers, that list we have to pass to docker rm –f (container name list)

docker rm -f $(docker ps)

**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) creating network

docker network create <new network name>

* by default this network will be created from bridge network

docker network ls

to see list of available networks

### 11) inspect all objects

docker inspect <any inspectable object>

docker inspect <container-name/image-name/network object>

docker network inspect <container-name>

docker volume inspect <volume name>

|  |  |
| --- | --- |
| //all docker objects are inspectabl  Inspect a image | docker inspect –-type=image tomcat  docker inspect nginx |
| Inspect a network | docker network ls // u will get list of network objects  docker network inspect <<network-name>>  docker network inspect bridge |
| Inspect a volume | docker volume inspect <volume-name>  docker volume inspect manivol |
| Inspect an image | docker image inspect <<image-name>>  docker image inspect nginx |
| Inspect a container | docker inspect <any-container-name>  //to display the size  docker inspect -s n1 |

### 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

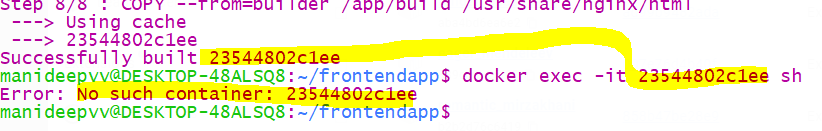
this exec command will not create any additional container like docker run

,it will get the shell of a running container,whereas docker run command will create a new 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



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

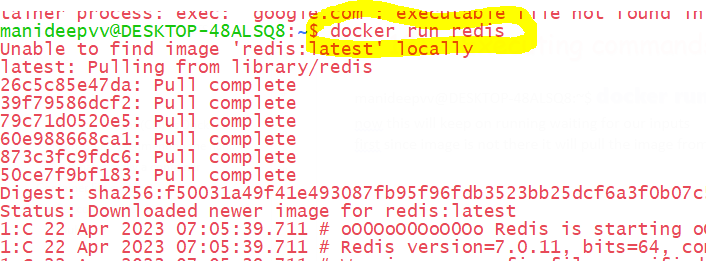
-it means connect my terminal to the input stream/interactive mode

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

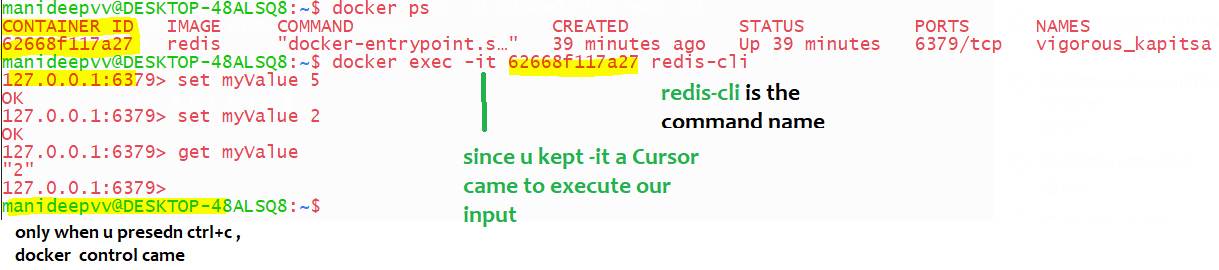


See now this window is continoulsy running and redis is continuously running , now it wont take any inputs

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



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)-it/ Open the cmnd prompt of a container

docker exec –it <container-id> sh

docker exec –it <container-id> bash

generally, shell is basic, if bash is not available ,minimum shell will be available

docker exec –it n1 bash

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

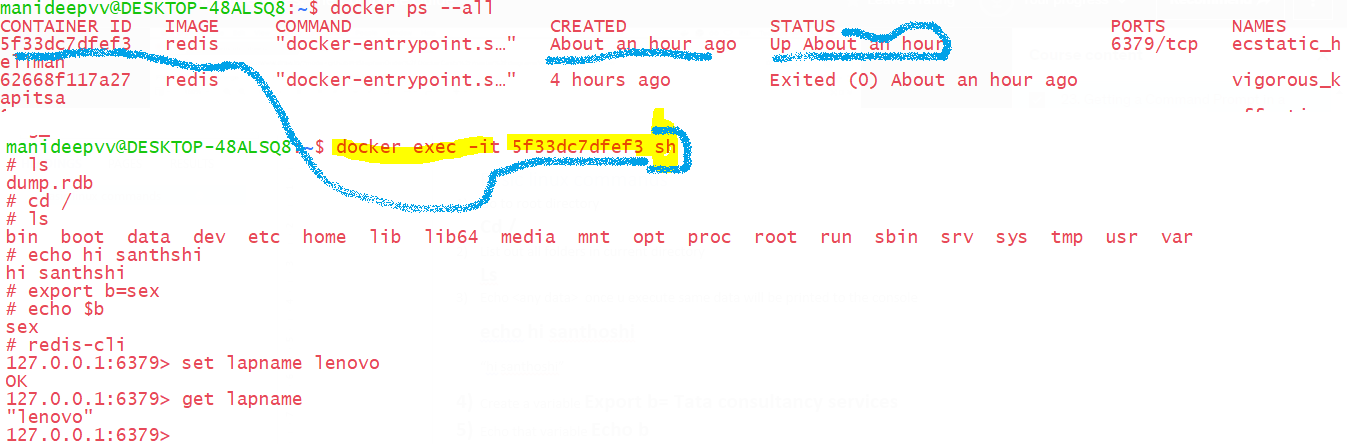
**if u want to exit from that mode/from that console type “ctrl+pq”simultaneously**

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”



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

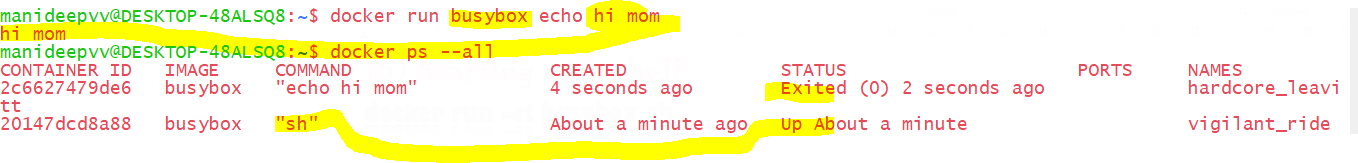
when u run the container like “docker run busybox echo hi there”,

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**



### 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

16.2)history- See intermediate images

docker history <image name-version>

### **17)List of networks**

“docker network ls”

To see the list of available networks

For like every container we have a container id , for Every network will have a network id,

18) Docker attach

“docker exec –it <image name> sh”

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



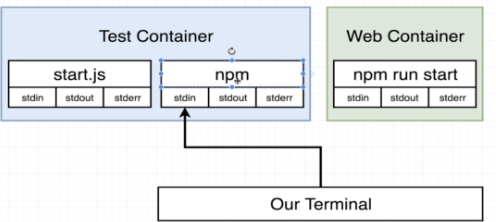
With docker attach we can forward a command from our terminal to the running container ,but why we need it,

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

19) Create image from cotainer

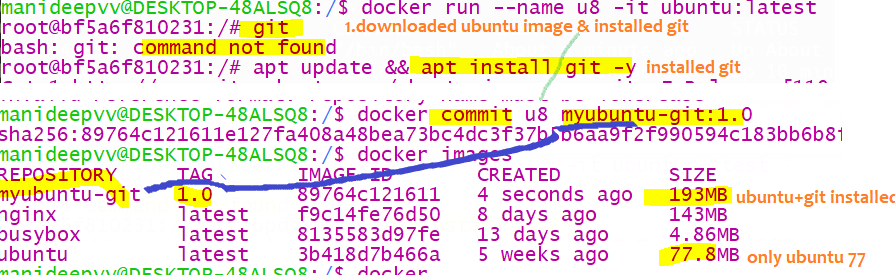
docker commit <container-name> <desired image name:version/tag>

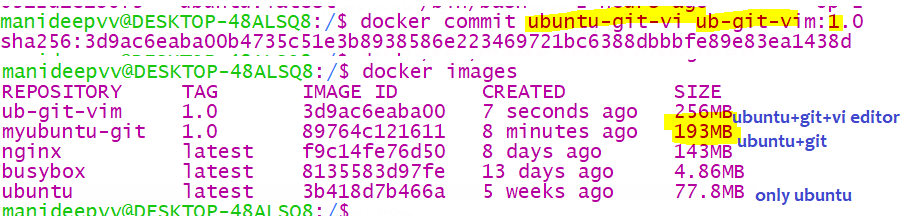
ex: docker commit ubuntu-git-vi ub-git-vim:1.0

We can create an image from our running container, with this commit command, image will be created,it will not be pushed to hub , this docker commit command we wont use in realtime

Same like we took a photo of our current situation

Committing a container to image



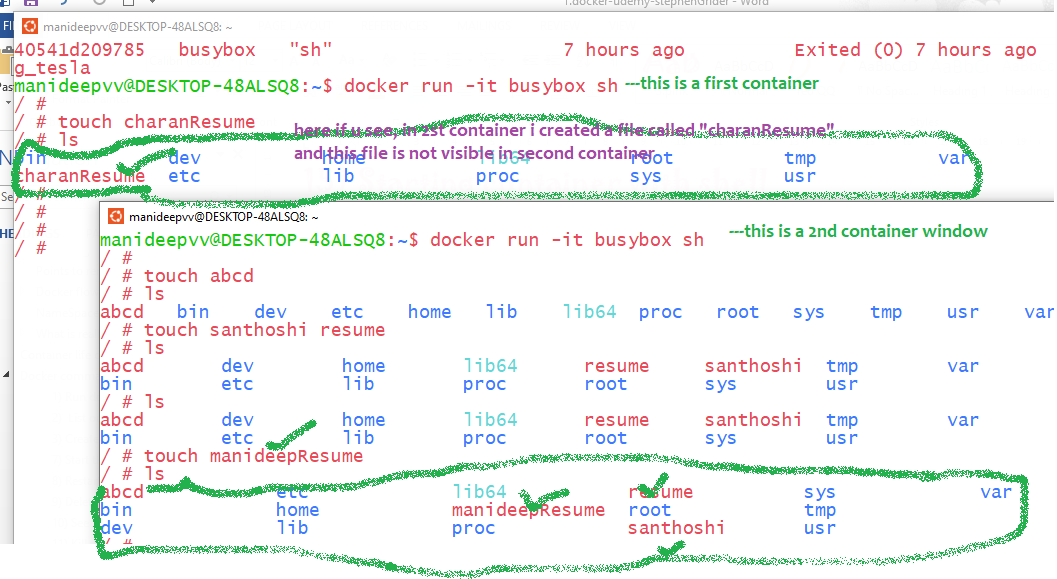


## Containers Isolation

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

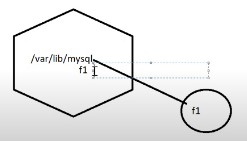


Docker volumes

Volume is nothing but some directory on the linux box

Containers are ephemeral-means temporary, if the program is generating some files, if container dies then those

Files on the server are not gone.. so that if insert them in container it will be pushed to docker those folders/volumes

here even if the container is deleted also these files on the disk are not gone

Data stored on Volumes are ***reusable and are shareable(between 2 containers)*** even when the ***container is stopped***.

docker run –name <container-name> -it –v <target to store>:<source folder to copy from>

docker run –name <container-name> -it –v <source folder to copy from>

docker run --name n2 -it -v /data01 nginx:latest

same like int x=10; //here right side 10 will be stored on left side variable

lly, right side is the source folder where data will be copied FROM

here inside the server WE HAVE a folder called “data01” the files /content from this folder will be copy to

some other named folder/un named folder

* Data volumes can be ***reused*** and ***shared*** amongst containers
* Data volume changes are ***independent*** of the image update
* Even when containers are ***deleted***, data volumes ***persist***
* Data volumes get ***initialized*** when we ***create a container***

CRUD volumes

|  |  |
| --- | --- |
| Create volumes | //always first u have to tell what u are creating,volume/network… so it is  “docker volume create” instead of “~~docker create volume~~” like “docker network create”  docker volume create --name tcsvol |
| Create a new container with volume | docker run --name n1 -it -v myvol:/etc Ubuntu  docker run -v /dbdata --name dbstore2 ubuntu /bin/bash |
| Using existing container volumes | //1st create a container with named volume-if it doesn’t exist, that vol will be created automatically  docker run --name n1 -it -v manivol:/etc nginx  docker run --name n2 -d --volumes-from n1 nginx |
| Inspect the volume | docker volume inspect new\_volume  if u don’t use keyword volume –if anyother object is also having same name,then it will print that object info,  docker network inspect new\_network\_host |
| List all the volume | docker volume ls  we can list networks like “docker network ls” |
| To delete unused volumes | docker volume prune |
| To delete a volume | docker volume rm <volume name>  docker volume rm tcsvol |

un-named volumes

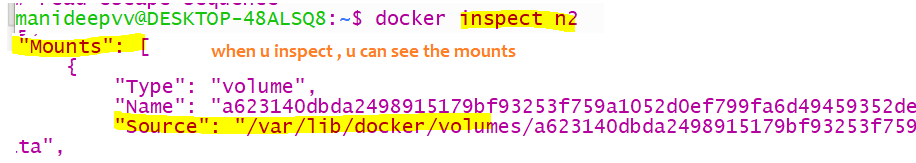
1. Un named volumes means the data will be copied to un named folder,only if u inspect the container u will know

To which folder the data was copied to, that to if container is deleted, u cant see where it was copied to

Volumes are stored in linux machine-ouside of a container,so that it will present even if container dies

If u want to know where the data was copied to , we have to get it from

docker inspect <container-name/id>



1. Named volumes

docker run --name n1 -it -v myvol:/etc Ubuntu

if left side or right side folder ,if whichever folder is not there it will be created.

Bidirectional sync will be there b/n those 2 folders, if u create a file in left side host folder,it will reflect in container folder

docker run --name u2 -it -v myhostv1:/etc ubuntu:latest

//in Ubuntu etc-many folders will be there

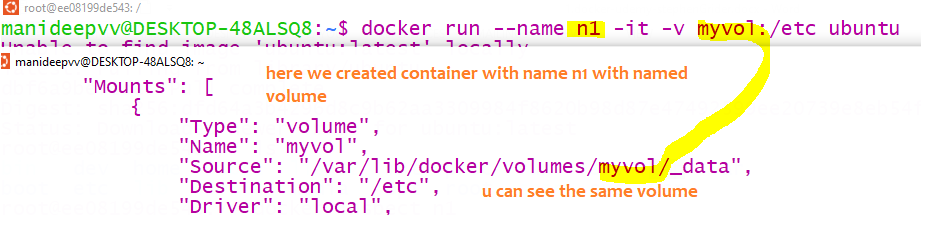
docker run --name n3 -it -v my-v1:/etc nginx:latest

if the above target folder ”my-v1” is present then data will be stored in “/var/lib/docker/volumes/<ur folder name>”,

else it will be created & stored

whatever is there in the “myvol” it will be synced to the etc folder, therefore bi directional sync will happen

if u want to know where the volume is created in the host,use docker inspect <container-name>



Example--2



1. Host volumes

Simple-if u give slash a separate folder will be created inside host machine, instead of creating a folder inside

“var/lib/docker/volumes/your folder name”

In realtime, if u want to copy some files to container then use host volumes, bec of bidirectional sync, and named folders

The only difference between named & host volume is “a slash”

|  |  |
| --- | --- |
| Named volume  docker run --name n3 -it -v my-v1:/etc nginx:latest  here no slash , means not a directory , only a folder inside var/lib/docker/volumes | Ur folder will be created under  “var/lib/docker/volumes/<your folder name>” |
| Host volume  docker run –-name c1 -it –v /data02:v1 ubuntu bash  here slash is there left side- because a directory will be created inside host | Folder will be directly created in host,  **NOT inside**  /var/lib/docker/volumes |
|  |  |

Host machine-machine where docker is installed

If u created a volume (folder) in host machine ,in that folder If u create a file that same file will be synced into ur container

Volume,

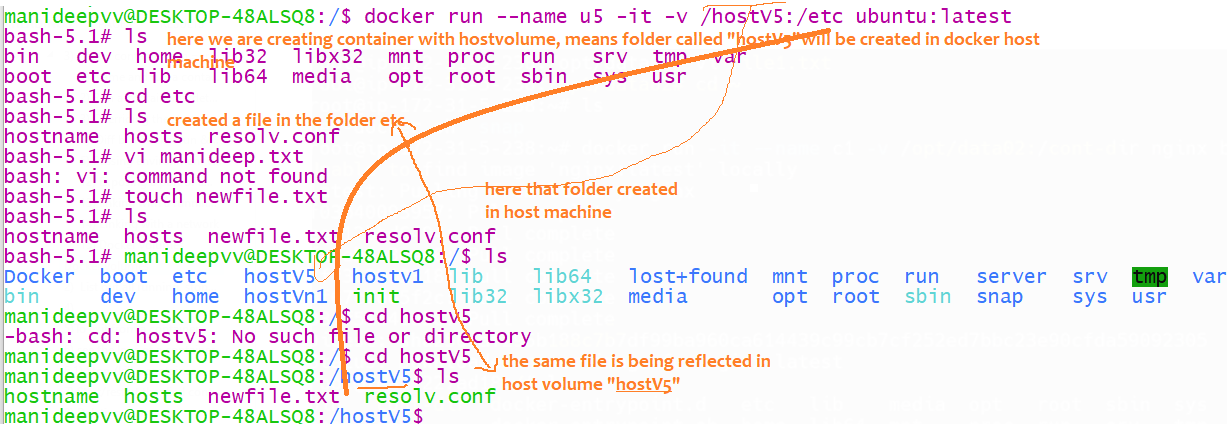
lets say in host machine created a folder called /data02 and in container u created a folder called v1 ,if u map both

then, if u created a file in host machine folder /data02 it will be reflected in container folder v1

Remember in the below command for the host volume if slash is not there,then that volume will be created in

“var/lib/docker/volumes”, if slash is there then that will be created in home folder data02

docker run –-name c1 -it –v /data02:v1 ubuntu bash



If in the below command if the slash is not there, then that folder will be created in

Host only but in “var/lib.docker/volumes”

If slash is there it will be created in the front, not in that folder

#### Shareable volumes

Shareable volumes means, whatever the volume the earlier container Is using the same volume will be used by this

Container also

example

docker run --name u6 -it -v /**hostv6**:/etc ubuntu:latest

docker run –-name <<desired container-name>> -it –volumes-from <<existing container-name>> <image-name>

docker run --name u7 -it --volumes-from u6 ubuntu:latest

now u7 container will also have same volumes/folders which was maintained by u6,

now u7 will use the same host volume “**hostv6**”,and sync will happen from etc-hostv6 bidirectionally

example-2

docker run –-name c1 -it –v /data02:v1 ubuntu bash

docker run –name c2 –it –-volumes-from c1 ubuntu

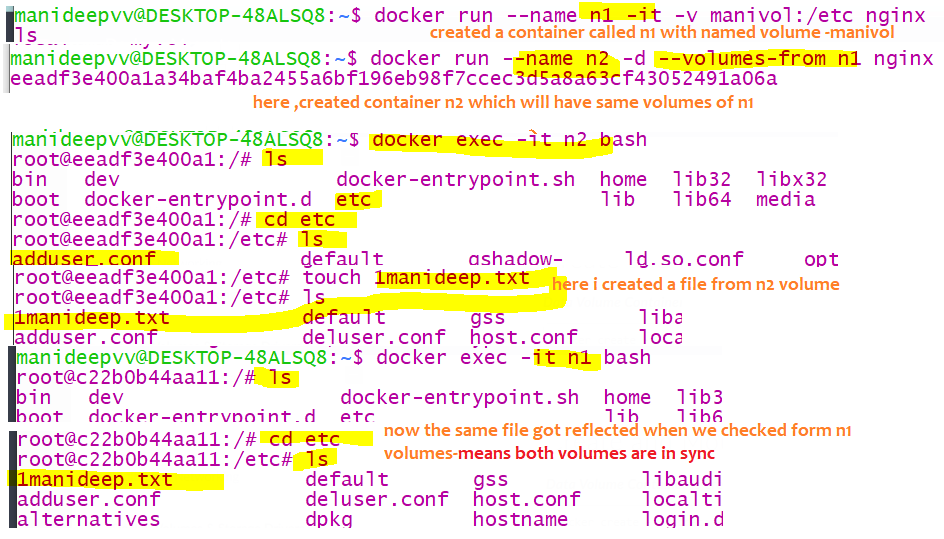
ex:-3

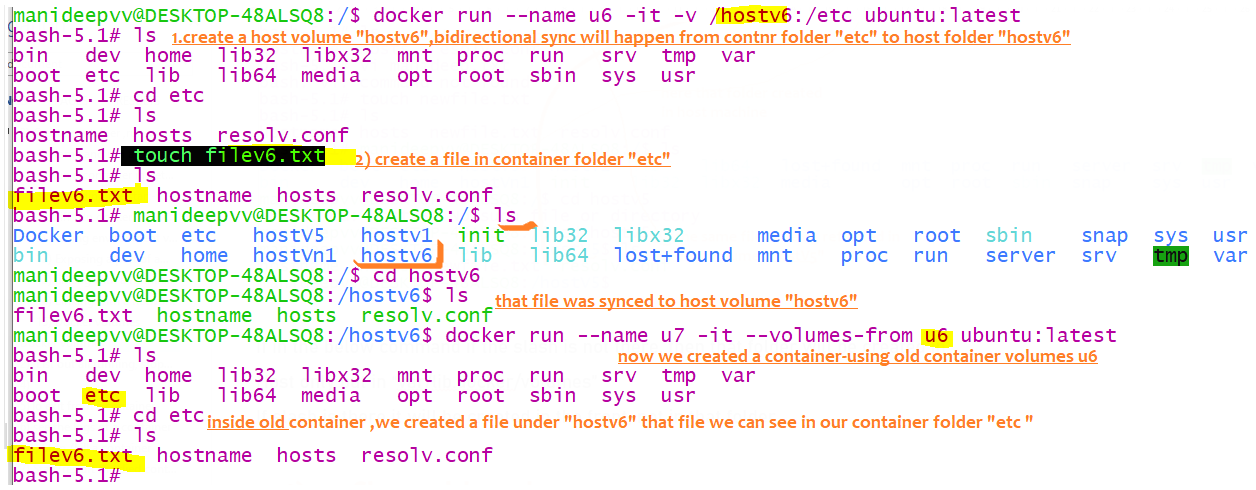
docker run --name n1 -it -v manivol:/etc nginx

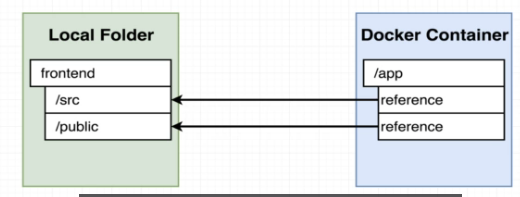
docker run --name n2 -d --volumes-from n1 nginx

here a container called c2 will be created where as it will also use same volumes of c1 having below

1. This container also will use data02 as the host volume
2. This container also will have an internal folder called v1,







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

## 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**

docker build –f <docker file name> -t <desired name to tag> .

docker build –t my-image:1.0 –no-cache .//means create the image without using any cache

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

1)Write all these commands in a “Dockerfile” which doesn’t have any extension,

2)u can create using “touch Dockerfile” or u can give any other name or u can type “vi Dockerfile”

It will create the file , edit it there itself , after that, click escape+caps ZZ, so that file will be saved and exited

3) command to create the image is

docker build –f <docker file name> -t <desired name to tag> .

if the instructions are in file named “Dockerfile” then no need to mention file name also, u can give as

“docker build –t <give the desired image/jar file name> .”

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"]

*Sample docker file*

Write all instructions in file named “Dockerfilev1”

FROM Ubuntu:latest

Maintainer Name=Manideep//to specify the author of the docker file

Label type=practice

RUN apt update//install git—RUN instruction is used to execute linux command inside container

RUN apt install git –y

CMD[“date”] //startup command-here the date command is executed when the container is started

Imp:-

Here the container start up command is date, we can override that startup command also

Docker run –-name c1 –it Ubuntu:latest sh

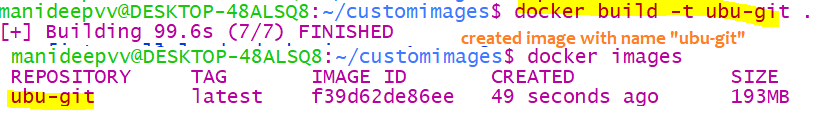
Now when the container is getting started, then date command wont get executed, “sh” only gets executed

CMD command will be overridden by docker run command

Build the image

docker build –f <docker file name> -t <desired name to tag> .

docker build –f dockerfilev1 –t Ubun-git:1.0

**

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,

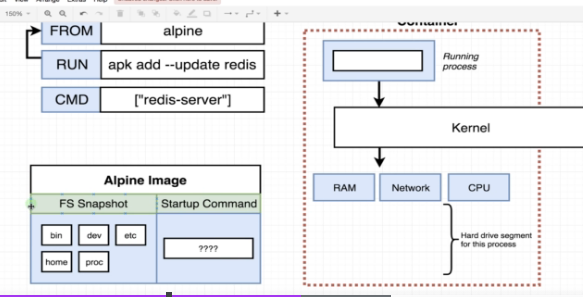
It will give the docker file to docker CLI, it will create a image from the given docker file



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

#### Sample commands

1. RUN instruction is used to execute linux command inside container
2. Difference between entrypoint and CMD commands

Both are for same purpose only, which are executed during container startup

Whereas, when both commands are present, CMD command will be passed as argument to ENTRYPOINT

Ex:-sample docker file

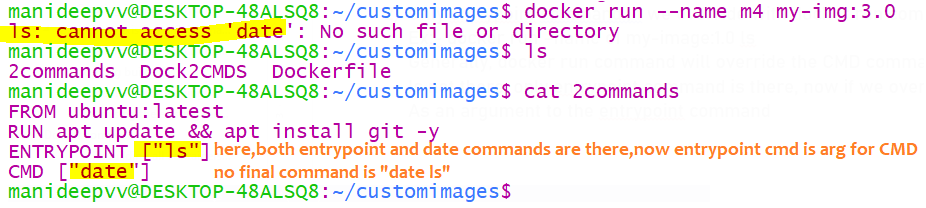
FROM Ubuntu:latest

RUN apt update && apt install git -y

ENTRYPOINT [‘’date’]

CMD [“ls”]

//now the container startup command will be “date ls”



1. Only entrypoint command ,if we try to override with docker run command then what will happen?

Then, the docker run cmnd will be passed as argument to entry point command

Ex:- docker run –name c1 my-image:1.0 ls

Sample docker file

===========

FROM Ubuntu:latest

RUN apt update && apt install git -y

ENTRYPOINT [‘’date’]

--final commands is “date ls”

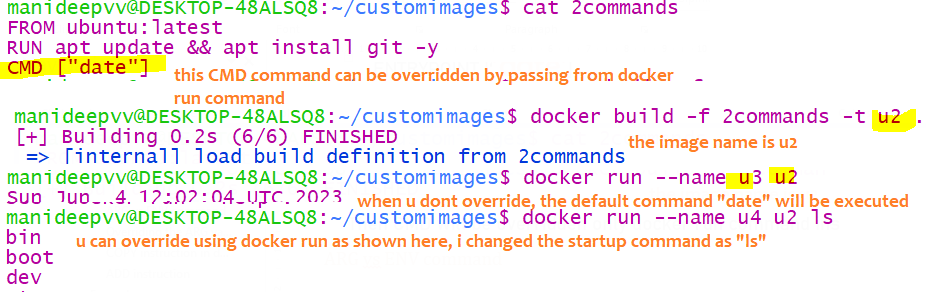
Generally, docker run command will override the CMD command, whereas in docker file if CMD command

Is not there, only entrypoint command is there, now if we over ride, then that RUN command will be passed

As an argument to the entrypoint command

1. If only CMD command is there along with docker run command,

Then CMD will be overridden only docker run command instruction will be present



##### ARG vs ENV command

1. What is ARG command in Docker file?

**Disadvantage**: The variable u created using ARG will be available only during image creation process only.

It will not be available in side container, bec it wont be set to cont system variables

When u create the container, and when u echo like “echo $variablename” that wont be available

**advantage** here is, u can pass the values from docker run command

,whereas u cant override the variables declared via ENV

Declared via ENV

Example

=========

FROM Ubuntu:latest

RUN apt update

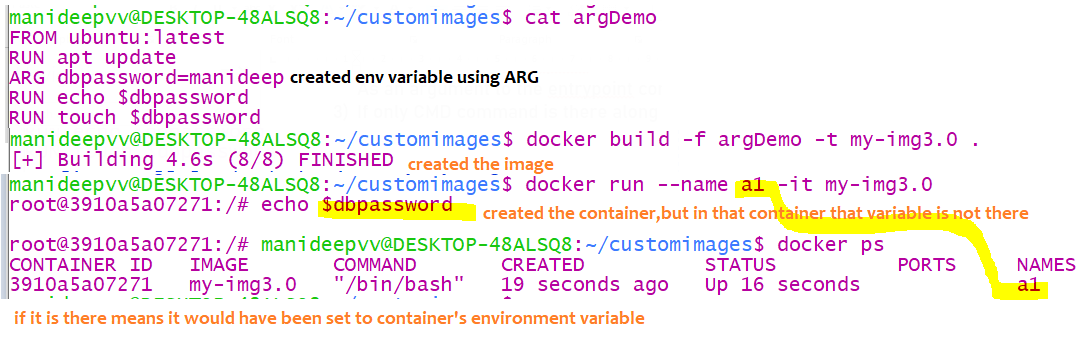
ARG dbpassword=Manideep //here we are creating a variable ,but it wont be set to containers env variable

//this variable scope is only till image,now a

RUN echo $dbpassword //remember any linux command if u want to run, u should use RUN only

RUN touch $dbpassword //a file called Manideep will be created—all these run commands will be executed

// while building the image



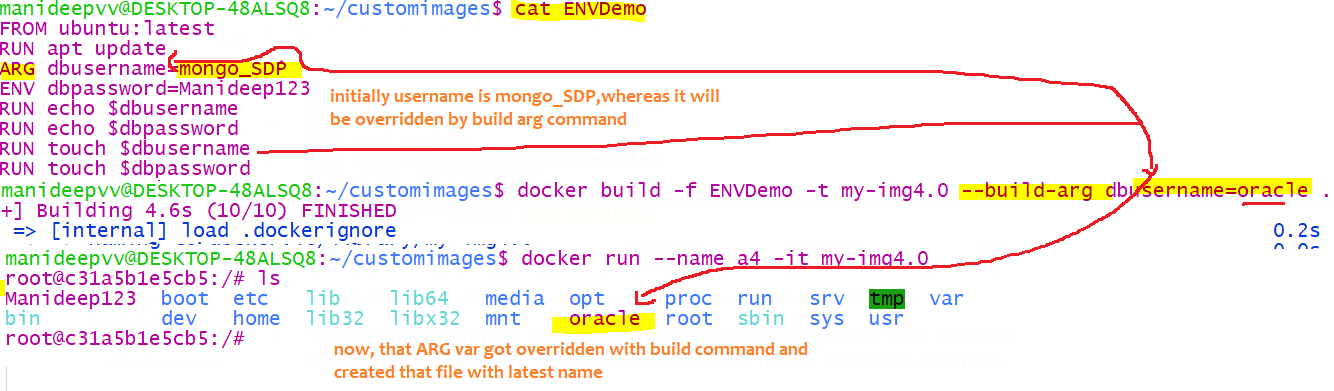
Overriding the ARG variable name

=======================

You can dynamically pass some values to ARG variables from docker build command

docker build –f <file name> -t <tag with desired image name > --buid-arg env =prod .

means here instead of hardcoding we can pass diff variable form build command



1. What is ENV command inside a dockerfile?

**Advantage:-**When u created a variable using ENV command those variables and values will be set to containers

environment variables,hence even after container is created the values will be present inside the container

**disadvantage:-**valuesto this variables cannot be passed from docker run command,but u can set the values

**from ARG to ENV**

7Sample Docker file

FROM Ubuntu:latest

RUN apt update

ARG dbusername=mongo\_SDP //this var,value will not be set to environment variables

ENV dbpassword=Manideep123 //this var,value both will be set to containers environment variables

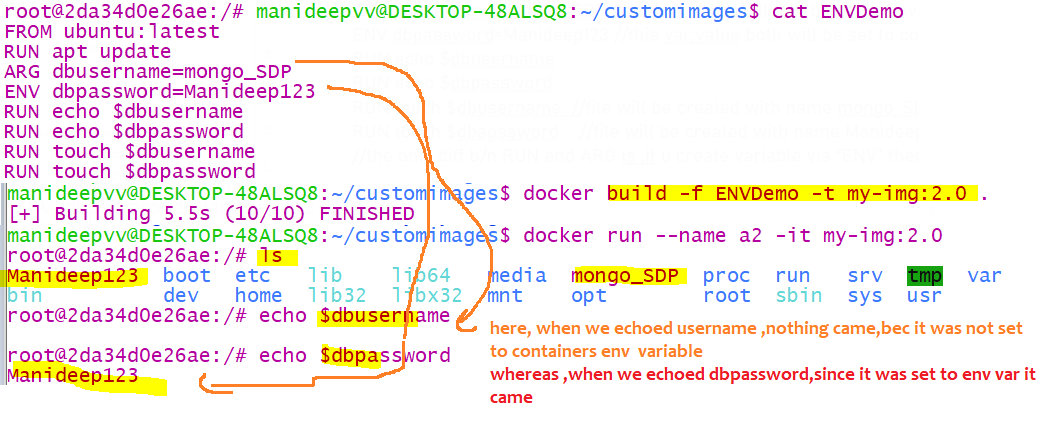
RUN echo $dbusername

RUN echo $dbpassword

RUN touch $dbusername //file will be created with name mongo\_SDP

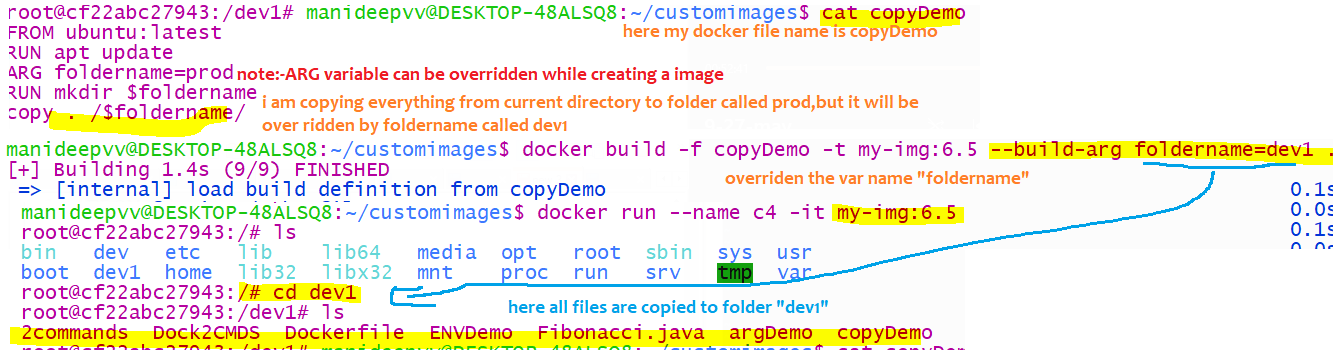
RUN touch $dbapssword //file will be created with name Manideep123

//the only diff b/n RUN and ARG is ,if u create variable via “ENV” then that will be set to containers sys variable



COPY instruction in docker file

Copy command is to copy the files from current folder to docker container



ADD instruction

Copy command can only copy the file

Copy <source folder> <destination folder>

Whereas the AD command can download the file form remote repository and copy to specified directory

Exposing a port

In docker file we can expose a port using

EXPOSE 9000

#### **Examples**

1. In the docker file, if you provide 2 CMD[] instructions, only last one will be executed

Sample Docker file

=============

FROM Ubuntu:latest

RUN apt update

CMD [“date”]

CMD [“ls”,”-lrt”]

//here only ls-lrt will be executed



1. Overriding the CMD instruction

In the above image, u gave 2 CMD-which are container startup commands,

u can override that also while running the container

docker run –name c1 –if <imagename:version> <any startup command>

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,

1)like first from base image a container will be created using docker run

2) and commands will be executed in that container ,using RUN..

3) and image will be created from the running container using docker commit internally..same cycle repeats

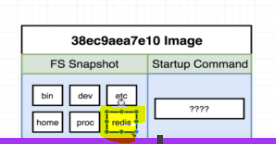
4) again commands will be executed in that container create image🡪create container🡪run the commands…

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

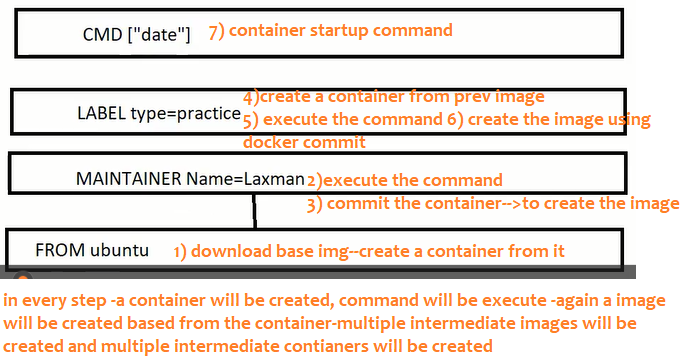
Doubt:-

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



This is done for caching purposes,

Next time if u create the image with same instruction, it will be fetched from cache

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

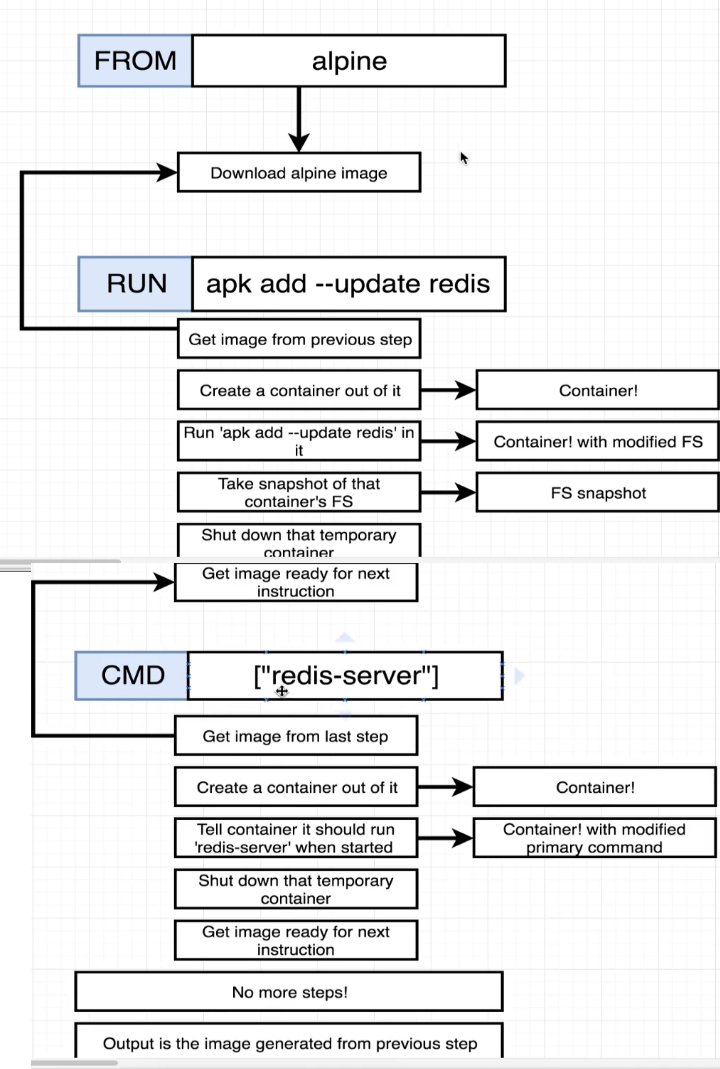
intermediate images will be cached for future purposes

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

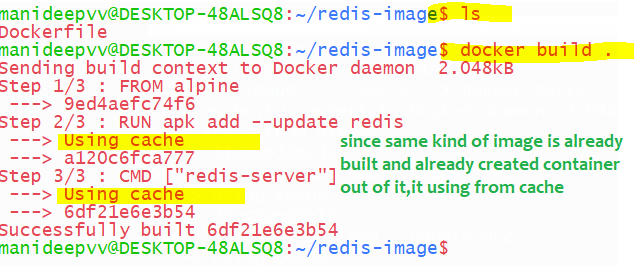
We don’t need to do it manually, automatically the intermediate steps images will get cached.

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

#### building without using cache memory

docker build –t my-image:1.0 -–no-cache .//means create the image without using any cache

docker build -f Dock2CMDS -t i3 --no-cache .

this is required, because, lets say u have the git installed, some intermediate images would have already go t created

after 6 months also it will use the same cache,

while creating a image, if u want to avoid using cache, we have to use “–-no-cache “ option

### 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 .

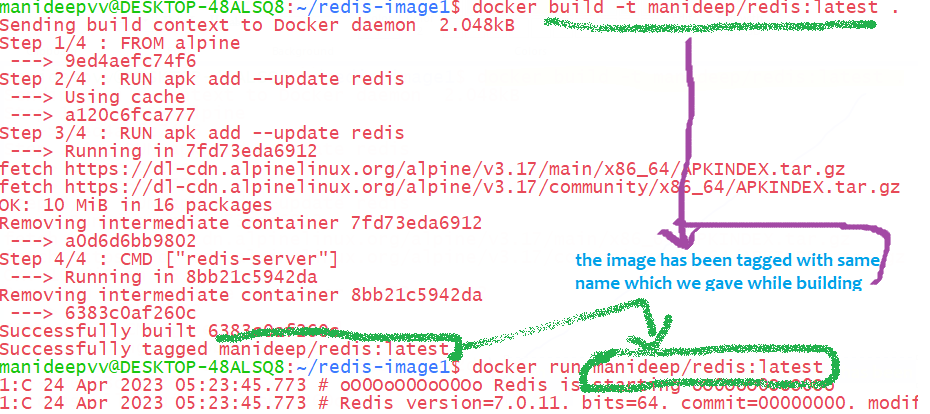
1. in the above dot at the end specifies the direct of folder/files where the docker file is available to use for the build
2. we should specify where the docker file is present, whereas dot represents the current directory
3. –t stands for tag

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

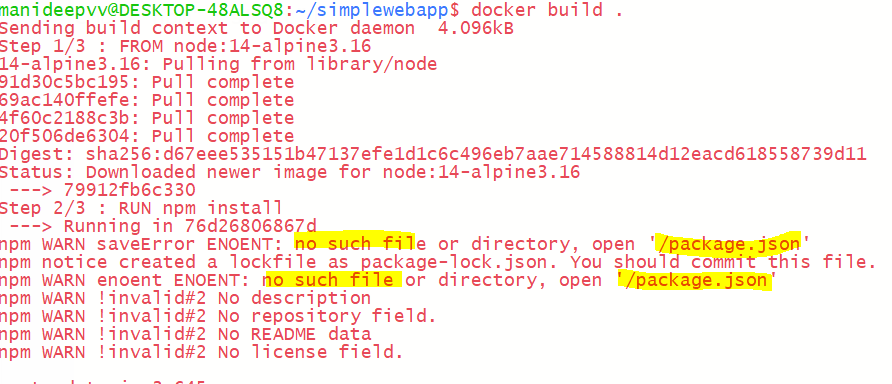
Steps of creating this container

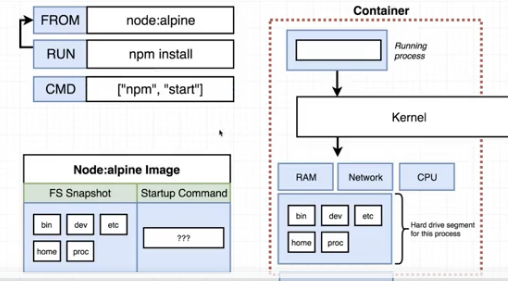
1)Base image [node:14-alpine] will be pulled and as usual, since all s’w are mentioned in the image ,container will be created from that image in HDD

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”

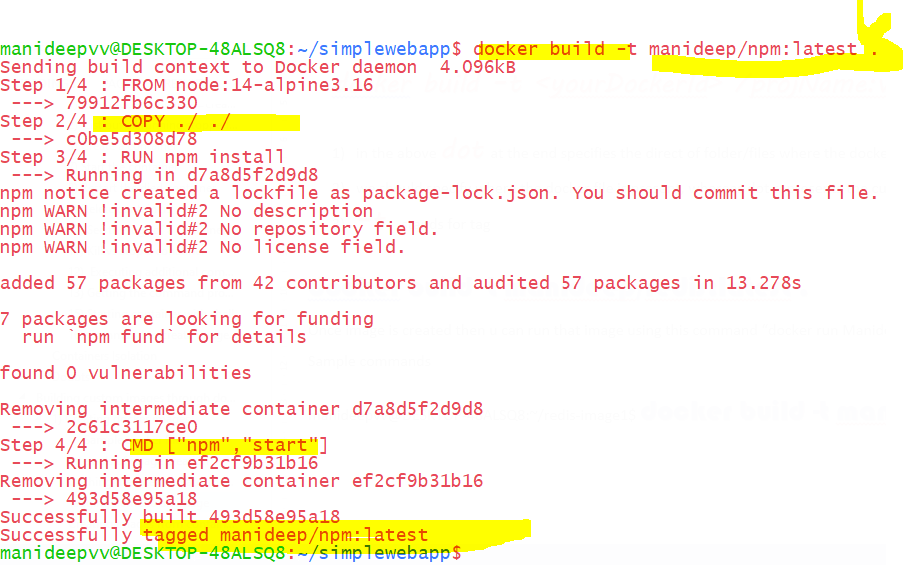
1. Touch dockerfile and edit it with below contents

FROM node:14-alpine3.16

COPY ./ ./

RUN npm install

CMD ["npm","start"]



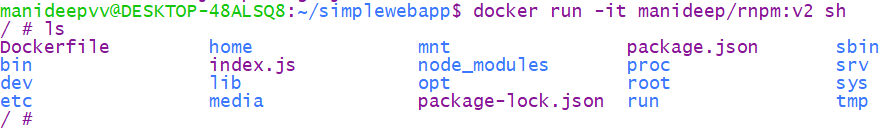
Now after copying those 2 files, and I created image and tagged this name, it built successfully

And u can run the container with below tagged image name

docker run Manideep/npm

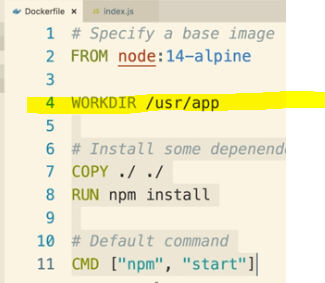
The problem with the above approach is it will copy those required files into the root directory

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

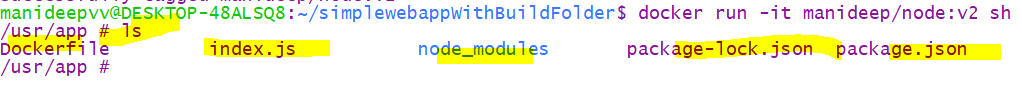
COPY ./ ./

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

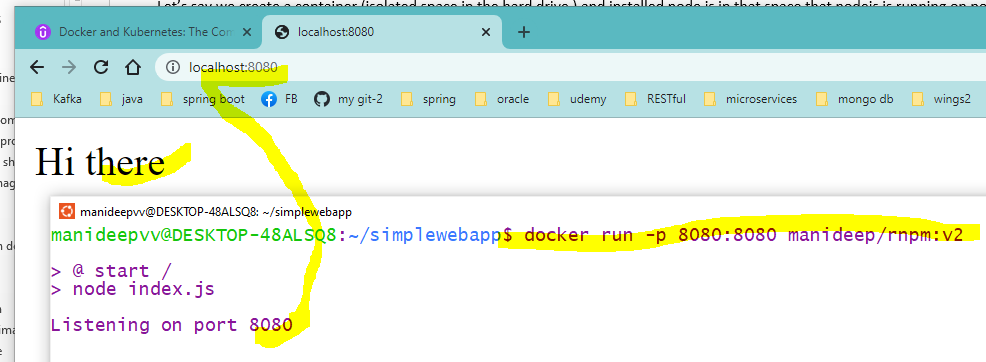
And from browser u can’t run hit the port 8080. Because that nodejs is running in an isolated space and this is a public port

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 |

Best practices

1) Avoid storing data in containers.

Use Volumes to store data if needed, since a container can be stopped or application running can be re-version-ed.

2) Avoid creating large images which is difficult to manage across multiple containers.

3) Always create multi-layered Image.

It is easier to manage and distribute across multiple containers.

4) Always use Dockerfile to create an image.

This process is reproducible and changes to this file can be tracked through any version control.

5)Tag images with correct version.

Avoid generic tags like 'Latest' to the image.

6) Always run one process per containers.

This will be easier to manage, trouble shoot issues and update processes individually.

7)Use Environment variables to store credentials.

Do not store directly in the image.

8) Communicate between containers using environmental variables.

Do not use IP address, since it changes for every container restart.

9) Keep Container interactions minimal.

Do not accept any connection on exposed ports through any network interface.

10) Use tools to monitor and identify vulnerabilities in images.

11) Wherever feasible, Run containers in Read -only mode.