Sr. No.	Date	Particulars	Page No.	Remarks
1.		Docker & its need.		
2.		Docker images & containers.		
3.		Important docker commands.		
4.		Port Mapping & & setting env. variables		
5.		Docker vs VM.		
6.		Troubleshooting containers.		
7.		Using con. to build node app.		
8.		Dockerization of Node.js app (Dockerfile).		
9.		Docker compose :		
		a. Services.		
		b. Port Mapping.		
		c. Env variables.		
		d. Volumes.		
10.		Publishing to DockerHub.		
11.		Layering in docker images.		
12.		Volume Mounting.		
13.		DOCKER NETWORKING.		
		a. Default & Custom networks.		
		b. network drivers :- Bridge, Host, null.		
		c. Using custom network for multi-container apps.		
14.		QUICK REF.	LAST PAGE.	

6/7/25.

DevOps.

Date: _____ Page No. _____

Topic: _____

→ Why DevOps?

- A Tech Company has two teams :-

i.) Development team ,

ii.) Operations team.

These two teams Contribute to a successful release of a software

i.) Development team :-

- Developers develops the code.

- This code is then sent to the operations team.

Date: _____ Page No. _____

Topic: _____

ii) Operations team :-

- Persons in this team tests & perform various operations on this code.
- provides feedback to the developers team in case of a bug.
- If not, then upload it onto the testing server or production servers as per the requirements.

PROBLEM !

- The code sent by developer to the operations, did not run!
- problem can be of environment dependencies, installed packages & their versions, can be anything.

Quote : "It runs on my machine, Why not on yours ?".

Date: _____ Page No. _____

Topic: _____

This problem was solved by using DevOps!

→ ## WHAT is DevOps:-

- Devops is a idealogy & methodology which improves the collaboration between developers & operations team using various automation tools.

- These automation tools are implemented using various stages which are a part of the Devops lifecycle.

- DevOps LIFECYCLE:-

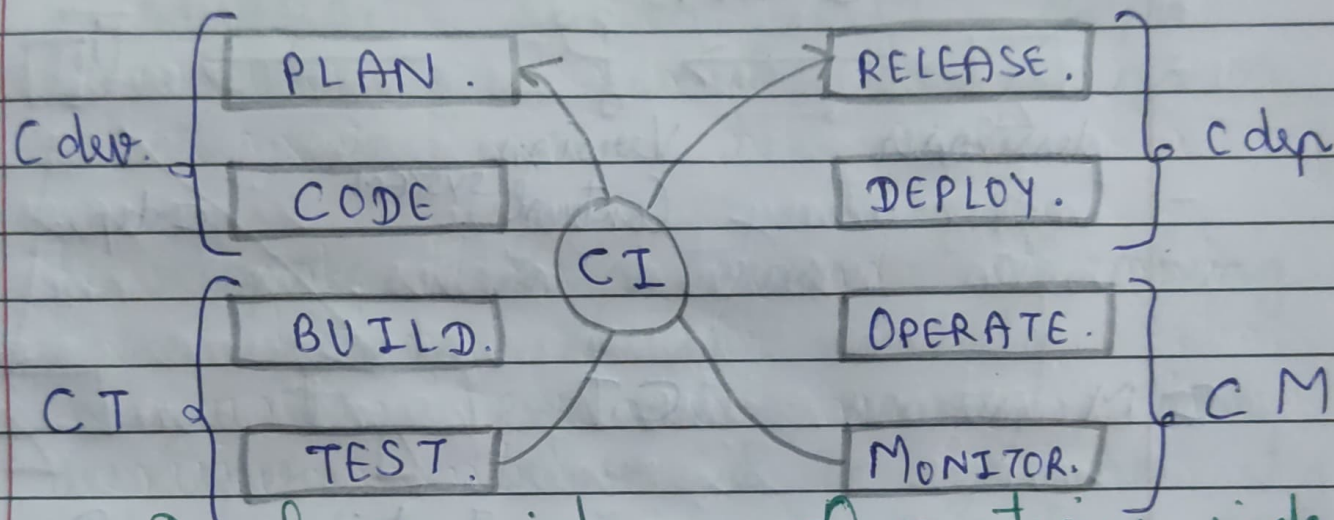
- PLAN,
- CODE,
- Build,
- test,
- Release,
- Deploy,
- Operate, &
- Monitor.

Date: _____ Page No. _____

Topic: _____

→ How DevOps Works?

- The devops lifecycle divides the SDLC lifecycle into the following Stages [5] :-



Developers side → Operations side.

→ Integrating.

- 5 Stages :-

C dev → Continuous Development.

C T → Continuous Testing.

C I → " Integration.

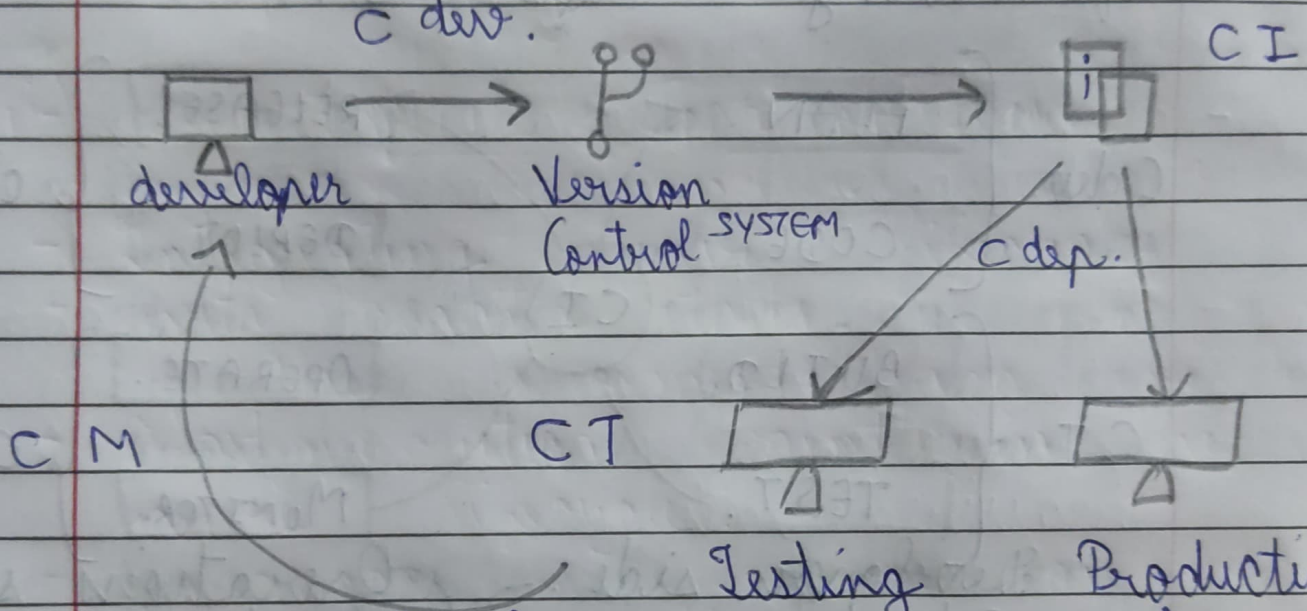
C dep → " Deployment.

C M → " Monitoring.

Date: _____ Page No. _____

Topic: _____

- Why continuous because it keeps on going on & on & on.
- It never stops.



AUTOMATED CI/CD PIPELINE.

- developer does CD & pushes the code onto the version Control System.
- then the CI tools are used for deploying these code.

Date: _____ Page No._____

Topic: _____

- On Testing Server if successful then, then it (code) is deployed onto the production server.

- either on testing or on production server CM keeps monitoring about the bugs or error & keeps the log.

- Developers keeps on checking these logs & if and bug or error comes, then developers solve it & pushes it onto the version control SYSTEM. continuous.

- And, the cycle conitnous.

i.] CONTINUOUS DEVELOPMENT :-

- We pushes our code onto the VCS (version control system) like Git or SVN (subversion).

Date: _____ Page No. _____

Topic: _____

- tools like Ant, Maven, Gradle for building / packing the code into an executable file that can be forwarded to the QAs for testing.

ii) CONTINUOUS INTEGRATION :-

- Integrating your lifecycles with each others.
- Most critical point in the whole lifecycle.
- And, therefore the key in automating the whole DevOps process.

iii) C dep :-

- deploying our code onto the particular server be it testing or production server or any other.

Date: _____ Page No. _____

Topic: _____

- installing software(s), building them as well as deploying onto the server (it is automated as well).

- key processes in this stage are Configuration Management, Virtualization & Containerization.

IV. CT :-

- This stage deals with automated testing of the application pushed by the developer.

- error (YES) → message → integration tool → notifies further → developer.

- error (NO) → message → integration tool → pushes the build → production server.

build → means converted to an executable file, & this file is then run.

Date: _____ Page No. _____

Topic: _____

V.] C M :-

- The stage continuously monitors the deployed application for bugs or crashes.
- ~~A~~ setup to collect user feedback as well.
- The collected data is then sent to the development team (devs) to improve the application.
- feedbacks like error - logs or version - updates are collected.
- All these stages are AUTOMATED using several TOOLS.

08/07/25.

Date: _____ Page No. _____

Topic: _____

GIT

- Git is the most popular tool among all the DVCS tools (Distributed Version Control System).
- used for tracking changes in computer files & coordinating work on those files among multiple people.
- Primarily used for source code management in Software development, but it can be used to keep track of changes in any set of files.
- Git is a Software, falling under DVCS tool.

09/07/25

Date: _____ Page No. _____

Topic: _____

⇒ How GIT Works :-

- following tasks are common, when working with git.

1.] Creating Repository :-

- Create a git repository using "git init".
- this initialize a git repository for your project on the local system.

- mkdir project
- cd project/
- touch 1.txt & touch 2.txt
- ls
- clear
- ls
- git init
- Initialized empty Git Repository in ..../project/.git/

Date: _____ Page No._____

Topic: _____

2.) Making Changes :-

→ track files status using "git status", whether the files are being tracked by git or not.

- ls
- git status
  - On branch master.
  - No Commits yet.
  - Untracked files:
    .....
    .....
  - use git add <file> to track.

→ No files are being tracked, to track a file use "git add <file>" or for tracking all files are "git add .".

- ls
- git add 1.txt 2.txt or
- git add .
- git status

Date: _____ Page No. _____

Topic: _____

Now our files are in staging state, files are not yet SAVED in git. to do so use commit.

→ Once the files or changes have been staged, we are ready to commit them in our repository using "git commit -m " message here type mano ".

-m ⇒ is for typing the commit message.

- ls
- git commit -m "Commiting 1.txt & 2.txt."
- git status
On branch master
nothing to commit, working tree clean.
- touch 3.txt.
Untracked file is now 3.txt.

Date: _____ Page No. _____

Topic: _____

- cat >> 2.txt
  Hello
  Modified the file 2.txt that was already committed.
- git status
Not staged to Commit:
  modified : 2.txt.
Untracked files:
  3.txt.
- the only way to commit them are to first add, then stage, & then Commit.

- git add 3.txt
- git status
- git commit -m "added 3.txt & modified 2.txt"
- git status
  working tree clean.

Date: _____ Page No. _____

Topic: _____

3. SYNCING REPOSITORIES :-

→ How we'll make it available to the world ?

- Once everything is ready on our local, we can ~~stat~~ start pushing our changes to the remote repository.

- Copy your ~~rep~~ repository link & paste it in ~~to~~ the command:

- git remote add origin "<URL to repository>"

- Go to github.com sign up / login > click new repository > give repository name, make it public > create repository.

- copy the HTTPS or SSH (Secure shell) link and paste it in the Command.

Date: _____ Page No. _____

Topic: _____

→ to push the changes to your repository, enter

- git push origin <branch_name>
- in our case branch_name is master.
- git push origin master.
- it can also request you to enter github username & password, provide it.

- BOOM! go & check the github repo. changes & files are reflected / pushed over there.

→ // git clone ||.

- if we want to download the remote repo. to our local system, use "git clone <URL>".
- this will create a folder with the repo. name & downloads all the files into this folder.
- Ex :- demo-repo.

Date: _____ Page No. _____

Topic: _____

- go to github copy the url of any repo you want to clone.

- git clone https://github.com/.....
  ↳ demo-repo/ # added.
  ↳ check using "ls".

- if we clone this and create s.txt, then commit it > push it → done Successful even without using the remote Command.

- because when cloning the repo. it copy the parameters as well as to what my origin repo. was.

- file added from clone repo. will not be in our local repo. but it will be in the github repo, from where we can pull it using.
- git pull origin ~~ma~~ master.

Date: _____ Page No. _____

Topic: _____

- local repo. will not get automatically updated, those using "pull" command.

- git pull <URL of link>

- this command only works in the (inside) ~~the~~ an initialized git repo..

- used to pull the latest changes, that might have pushed to the remote repo..

⇒ GIT PULL:-

it updates the local repository, from remote repo..

⇒ GIT CLONE:-

it copies / clones the whole repo. from remote to local.

Date: _____ Page No._____

Topic: _____

4.] PARALLEL DEVELOPMENT :-

- we saw, how to work on git.
- Imagine, multiple developers working on the same project or repo..
- To handle the workspace of multiple developers, we use branches.
- To create a branch from an existing branch, use :-

```
git branch <name of new branch>
```

- to delete a branch, use :-

```
git branch -D <branch-name>
```

- to list all branches, use :-

```
git branch
```

Date: _____ Page No. _____

Topic: _____

- create & switch to that branch :- use "checkout".

- LAST Commit of master, becomes FIRST Commit of our new branch.

- git checkout <branch-name>

- Branches are isolated from each other.
- to Combine them we use "merge" Command.

- to see which branch we are in :-
  git branch
  * feature1 ✓ * denotes it.
    master
    branch1
- git checkout master #to go from current branch to master branch.

- No Branches are interfering with each other.

Date: _____ Page No. _____

Topic: _____

- Ex :-

- create a branch feature1.
- check all branches 2 right now :-
- * master
  feature1
- go to feature1 (git checkout feature1).
- * feature1
  master
- do ls in both initially same files.
- create 6.txt in feature1 & add it, commit it & push it to the feature1 branch.
  - git add .
  - git commit -m "added 6.txt from feature1 branch"
  - git push origin feature1
  - new branch created & pushed file 6.txt as well from feature1 to feature1.
- check "git status", working tree clean. on branch feature1
- git checkout ls.
- total 6 files (5 branched from master & 1 created).

11/07/23.

Date: _____ Page No. _____

Topic: _____

- git checkout master.
- ls
- total 5 files that were already there, (6.txt is NOT added).
- check status
on branch master,
working tree clean.

⇒ #

- to check from github select on branch choose master (only 5 files visible), select branch choose feature/ (total 6 files visible.)

→ || git log ||

- checks every commit detail in the repo..
- to check the history of master branch :-
- ONLY the LOGS of master.
- git checkout master
- git branch → (to Confirm).
- git log.

:

History of everything in master from 1st file to latest commit/changes.

Date: _____ Page No. _____

Topic: _____

Logs

- to check in feature1 branch :-

- git checkout feature1
- git branch # optional.
- git log

- Logs of master branch as well as logs of this branch (feature1).

- Because feature1 was created from master branch, so it will have all the logs of master branch as ~~well~~ well.

- ~~LOGS of MASTER branch~~ is

- MASTER branch shows LOGs of ONLY MASTER branch &

- FEATURE1 branch created from master branch will have LOGs of master branch as well as LOGs of feature1 branch itself.

Date: _____ Page No. _____

Topic: _____

|| git stash. ||

- to avoid dangling files moving between branches.

- Ex:- In our case we have 2 branches:-
master → 1 to 5.txt files (5).
feature1 → 1 to 6.txt files (6).

- created 7.txt in feature1.
- ls → total 7 files.
- changing to master branch.
- ls → total 6 files (6.txt) is reflected here.

⇒ This is known as dangling file.

- On working in a production level environment this may get create confusion, to avoid this we use 'git stash' & 'git stash pop'.

Date: _____ Page No. _____

Topic: _____

- Ex :- modifying 5.txt in feature1 :

Then, check in master branch. the changes will be reflected in the master branch.

- this is not staged (git add .), from feature1.

- then if we change & check in master the changes are reflected even after staging the files.

- ~~to avoid this use :-~~

- stowing :-
  M 5.txt. Modified.
  A 7.txt. Added.

- to avoid this :-

- git stash
- saved working directory & index.

Date: _____ Page No. _____

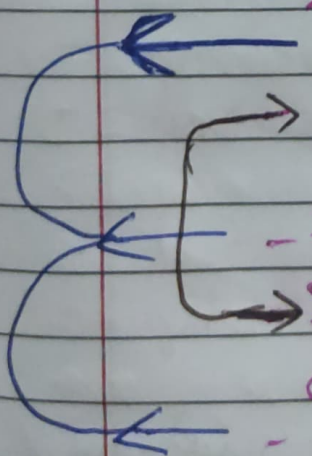
Topic: _____

- git stash (if used again).
- No Local changes to Save.

- ls (in feature1 branch).
  - total 6 files.
  - No Changes in 5 (even if we have changed).

- ls (in master branch)
  - total 5 files.
  - No changes in 5 (even if we have changed).

- ~~To~~ The main reason to use stash is,
  - while working in a branch,
  - we got to do some imp. work in the another branch.
  - the first branch is Not Completed.
  - So, we use stash & checkout to another branch - finish Work.
  - Go back to first branch & use "git stash pop" to resume our works from there.



Date: _____ Page No. _____

Topic: _____

- git stash pop

  Dropped refs/stash@{0} (.....).

- "Saves Our Work without Committing the Code?"

  - Use 'git stash'.

- to stash your untracked files, type :-

  - Use 'git stash -u'.

- git stash.

- git stash -u.

- git stash pop.

C-ID → COMMIT-ID.

Date: _____ Page No. _____

Topic: _____

//git revert <commit-id>//.

- Helps you in reverting a Commit, to a previous version.
- 'git revert <COMMIT-ID>'
- <COMMIT-ID> Can be obtained from the "git log" Command.
- ls - in feature1 $ → total 7 files.
- git log - copy the id you want to revert the Commit.
- git revert <C-ID> - it will ask for the Confirmation. In VIM - press ESC > :wq > enter.
- Successfully reverted.
- ls - to check or use.
- git log - shows the message of reverted with C-ID.
- Copy this C-ID & enter :- git checkout C-ID. → to enter the reverted message & see what was reverted.

Date: _____ Page No. _____

Topic: _____

- in place of branch name you will see the C-ID of reverted commit.

- to exit that checkout press :-
  git checkout feature1

- ~~that~~ this command will take us to the feature1 branch that now has 6 files & the old content of 5.txt (this was the changes in the reverted C-ID that's way).

- ls - to check.
  cat > 5.txt - to check the content

- Reverting means Undoing the changes done in that C-ID.

// git diff //

- check the difference B/W two versions of a file / two commits.

Date: _____ Page No. _____

Topic: _____

- git diff <C-ID of version X>
  <C-ID of version Y>

- C-ID → get it from → git log.

- git diff Read .

Read → ~~for~~ latest Commit of Current HEAD (branch)
. → Current changes untracked/ unstaged.

- git diff db785..... 567899fdez...

db785... → changes from this C-ID to
567899fdez... → to this C-ID.

GIT COMMANDS

Date: _____ Page No. _____

Topic: _____

- git init
- git status
- git add .
- git commit -m "message"
- git remote add origin <URL-link>
- git push origin master
- git clone <URL - link>
- git pull <URL- link>
- git branch <branch-name> [-D]
- git checkout <branch-name>
- git log
- "git stash" && "git stash pop"
- git revert <C-ID>

11/07/25

DOCKER.

Date: _____ Page No. _____

Topic: _____

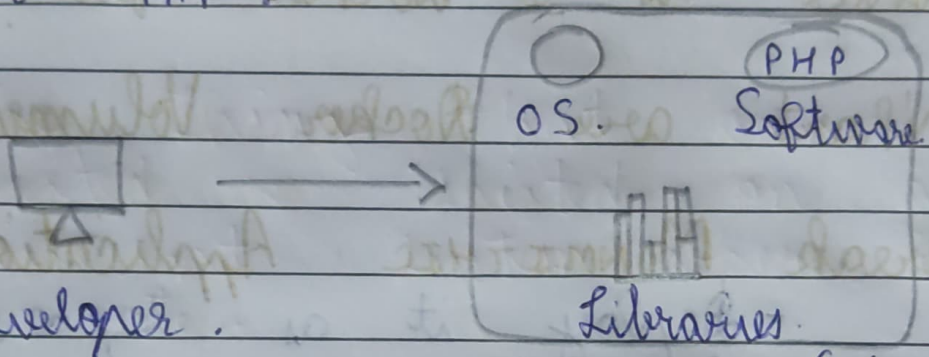
- Introduction to Docker.
- Common Docker Commands.
- What is a Docker file?
- What are Docker Volumes?
- Break MONOLITHIC Application.
  - Deploy it as separate Containers on your infrastructure as Micro-Services.
- What is Docker Compose.
- What is Docker orchestration?
- Deploy application using docker SWARM.

Date: _____ Page No. _____

Topic: _____

- Problems before docker.

- Imagine you are a developer, & you are creating a website on PHP.



- Program is running well in developer's machine.

- Problem occurs when this program is given to someone else.

- tester tests/checks this program.

- The Ops guy has to run this file on his system

Date: _____ Page No. _____

Topic: _____

- He needs to replicate the environment of developers.
  - the same OS, same PHP, same libraries
  - PHP versions may differ as there are many of them.
  - the Ops Guy will have to have the same exact version.
  - Hence, the Code didn't ran on the OPS Guy machine.
  - he says the code is faulty, but actually it ran on developers machine.

- It can work with VM's but the size & versions of VM can cause a problem, & it was Complex as well.

- this was the ~~problem~~ problem before the docker came in picture.

Date: _____ Page No. _____

Topic: _____

- we needed a wrapper that contains or bundles all the things (os, software, libraries) in one single thing as VM.

- The thing that can be lifted & shifted on a different system where it can execute.

- Something smaller, easily portable, lightweight, that can be given to someone for testing.

- If we get a wrapper / container / an environment that wraps this things. So, that the developer environment & testing / staging environment & as well as the production env. all becomes the same after using that wrapper thing.

09/05/25.

Date: _____ Page No. _____

Topic: _____

⇒ docker pull hello-world → fetching the image.
image-name.

⇒ docker images → see all the images.

⇒ docker run hello-world
image-NAME.
↓
creating the container.

Images have [image-name, tag, image-id, created, size].

Containers have [name [automatically created a random name], container-id, Image-name, ports

daemon (DAEMON).

⇒ Docker ~~demon~~ is the core, the most important part of docker desktop.

Date: _____ Page No_____

Topic: _____

⇒ "pull" pulls the docker image, & "run" creates the containers.

⇒ While directly "run" is used to first fetch (pull) the image then creates the container.

⇒ docker run -it ubuntu
↳ interactive shell.
to run it in a

interactive shell will be different from the normal shell / cmd.

⇒ docker start | stop container-name or container-id.
↙
to start or stop an existing containers

⇒ docker rmi image-name
↳ removes the image.
This image should have no containers. then only then image will be deleted. (irrespective of running or stopped container)

Date: _____ Page No. _____

Topic: _____

⇒ docker rm container_name.
↳ removes the container.

② ⟹ docker run -d \
--name mongo-express \
-p 8081:8081 \
--network mongo-network \
-e ME_CONFIG_MONGODB_ADMINUSERNAME=admin
-e ME_"_"_ADMINPASSWORD=1234 \
-e "_"_"_URL:"mongodb://admin:1234@
mongo:27017" \

mongo-express.

① ⟹ docker run -d \
-p 27017:27017 \
--name mongo \
--network mongo-network \
-e MONGO_INITDB_ROOT_USERNAME=admin \
-e "_"_"_PASSWORD=1234 \

mongo.

Date: _____ Page No. _____

Topic: _____

⇒ docker network ls

⇒ docker network create network-name

⇒ git clone URL .

⇒ docker run -d --name mongo \
-e MONGO-INITDB-ROOT-USERNAME = admin \
-e "-"-"-"-PASSWORD = password \
--network mongo-~~express~~ network \
mongo

⇒ docker run -d --name mongo-express \
--link mongo \
-e ME-CONFIG-MONGODB-SERVER = mongo \
-e "-"-"-"-ADMINUSERNAME = admin \
-e "-"-"-"-ADMINPASSWORD = password \
-p 8081:8081 \
--network mongo-~~express~~ network \
mongo-express

⇒ docker ps ⇒ then go to LOCALHOST:8081.
⇒ sign up ⇒ ~~sign~~ up will be asked ⇒ admin & pass.

Date: _____ Page No. _____

Topic: _____

⇒ docker run -d --name mongo-express \
--link mongo \
-e ME_CONFIG_MONGODB_SERVER = mongo \
-e "_ "_BASICAUTH_USERNAME = admin \
-e "_ "_ "_PASSWORD = password \
-e "_ "_MONGODB_ADMINUSERNAME = admin \
-e "_ "_ "_ADMINPASSWORD = password \
-p 8081:8081 \
--network mongo-network \
mongo-express.

⇒ changing the MONGO_URL in server.js code.

const MONGO_URL = "mongodb://admin:pass
@localhost:27017/?authSource=admin";

Date: _____ Page No. _____

Topic: _____

⇒ docker-compose is nothing but a .yaml.

.yaml ⇒ yet another markup language.

⇒ dc.yaml :- [containers are written as services].

⇒ INDENTATION is really IMPORTANT.

```
version : "3.8"

services :
    mongo :
        image : mongo
        container_name : mongo
        ports :
        - 27017:27017
        environment :
            MONGO-INITDB-ROOT-USERNAME: admin
            MONGO-INITDB-ROOT-PASSWORD : pass
        volumes :
        - HOST PATH: /data/db
    mongo-express :
        image : mongo-express
        ports :
        - 8081 : 8081
        environment :
```

Date: _____ Page No. _____

Topic: _____

ME_CONFIG_MONGODB_ADMINUSERNAME: admin

ME_CONFIG_MONGODB_ADMINPASSWORD: pass

no need "_" ← ME_CONFIG_MONGODB_URL: mongodb://

admin:pass@mongo:27017/

⇒ to actually create containers from the .yaml file we use 2 commands.

⇒ docker compose commands :-

docker compose -f filename.yaml up -d

docker compose -f filename.yaml down

Date: _____ Page No_____

Topic: _____

⇒ Dockerfile.

⇒ DOCKERIZING OUR APP :- a text file.

⇒ testapp
↓
docker images
↓
container

DOCKERIZING means converting our app into docker image then to our ~~running~~ running containers.

⇒ FROM <base image>

WORKDIR <path>

COPY <host-path> <image-path>

RUN <COMMAND>

CAN BE MULTIPLE. ENV <name> <value>

EXPOSE <port-numbers>

only (1) CMD ["<COMMAND>", "<arg1>"].
Ex :- node server.js.

⇒ MULTIPLE RUN command but only 1 CMD command.

Date: _____ Page No. _____

Topic: _____

⇒ Ex :-

```
FROM node
ENV MONGO-DB-USERNAME = admin
    MONGO-DB-PWD = 123 pass
RUN mkdir -p testapp
COPY . /testapp
// RUN npm install → if there is no node-modules.
CMD ["node", "/testapp/server.js"]
```

⇒ docker build (-t) testapp:1.0 . (dot.)

(-t) → tag ; testapp:1.0 → tag

⇒ to verify run :-

docker images

Date: _____ Page No. _____

Topic: _____

⇒ to run the container in the interactive way (-it).

docker run -it testapp:1.0 bash

Container name. bash/sh.

⇒ to publish our docker image on the hub.docker.com follow the steps:-

⇒ ★ container to docker file (docker-compose) ★

PUBLISHING DOCKER IMAGES.

STEP 1:- sign up at hub.docker.com.

STEP 2:- my profile → create repository.
Ex:- (testapp).

STEP 3:- create docker image with the same name of username/repo.
Ex:- sharmajiyush10718/testapp

STEP 4:- docker run -t sha./.yy ⊙→ dot.
image name now.

STEP 5:- now push it and before pushing sign in in the

Date: _____ Page No. _____

Topic: _____

terminal with the same account that we signed in in the hub.docker.com.

STEP 6 :- LOGIN in the terminal :-
docker logout (if logged in, then)
docker login.

i.) docker login -u <username>, sharmayush10718
then enter password.

ii.) activation link & give it the above mentioned one-time device authentication password.
Ex :- MZDN-QWDK.
open it in the same browser.

STEP 7 :- click continue, the CONFIRM.
"LOGIN SUCCEEDED" in terminal.

STEP 8 :- dock push username/repo image name image name now.

STEP 9 :- go & check. Uploaded on hub.docker.com
delete the code. no need to worry.

STEP 10 :- docker push image name } ready to be pulled.

Date: _____ Page No _____

Topic: _____

⇒ Docker Volumns :-

persistent :- permanent / data is not lost.

non-persistent :- data is lost when restarted / temporarily.

⇒ Volumns are persistent data storage for containers.

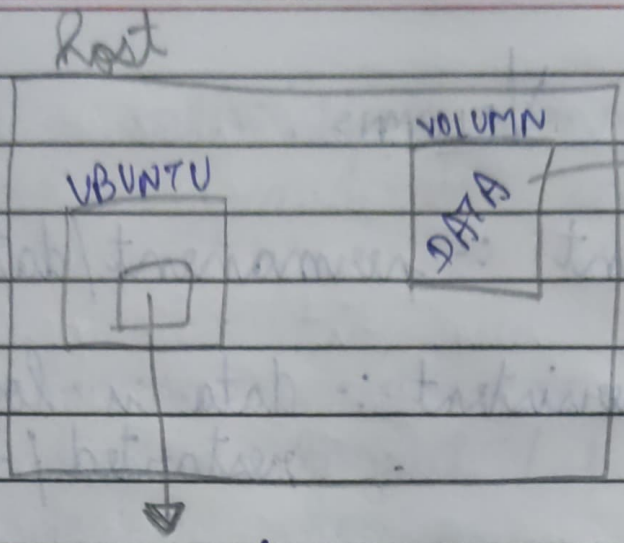
⇒ Multiple volumns can also be mapped to only 1 storage on host.

⇒ The data is not lost when one restarts the container when it is mapped to the host machine.

⇒ Even if we delete the whole container the data is still accessed from the host machine.

Date: _____ Page No. _____

Topic: _____



⇒ -v Root-path : container-path

absolute path.

⇒ docker run -it -v /D:/Piyush/docker/volumns:/test/data ubuntu

⇒ cd test/data

⇒ touch abc.txt ⇒ it will also be created on volumns folder on desktop.

⇒ Generally when the containers are stopped & restarted the data / files / DB within them is ~~also~~ lost.

Date: _____ Page No. _____

Topic: _____

⇒ Now, stop & restart them.

⇒ the data/files/DB will not be lost.

⇒ docker stop/start con-name

⇒ docker exec -it con-name bin/bash.
bin/sh.

⇒ to check with deleted delete con, images, everything.

⇒ and now run:

⇒ docker compose -f mongodb.yaml up.

⇒ when this is completed the host path has few files meaning the volumes has been created.
to add volumes in .yaml file.

⇒ volumes:
- host-path: ~~con-path~~ /data/db

Date: _____ Page No. _____

Topic: _____

⇒ docker compose -f mongodb.yaml down

⇒ refresh the DB website.

⇒ docker compose -f mongodb.yaml up.

⇒ The old data will be there even if we have deleted the containers & Has installed them again.

⇒ more volumes commands:-

docker volume ls

docker volume create vol-name.

docker volume rm vol-name.

↓

named_volume

* path is not given to these volumes.

Date: _____ Page No. _____

Topic: _____

⇒ docker automatically creates volumes at:-

WINDOWS:- C:\ProgramData\Docker\volumes.

MAC/LINUX:- /var/lib/docker/volumes.

⇒ the volumes that we create are isolated volumes that are not attached to any of the containers.

⇒ to connect we can use the ~~3 ways~~ & 2 flags:-

⇒ docker run -v VOL_NAME:CON_DIR. [NAMED_VOLUMES]. — docker handles it.

temporary storage.

⇒ docker run -v MOUNT_PATH. [ANONYMOUS_VOLUMES]. — con_dir.

⇒ docker run -v host_dir:con_dir. [BIND MOUNT]. — because of absolute path the OS handles it.

~~--vol~~ --volume (-v)
--mount

Date: _____ Page No. _____

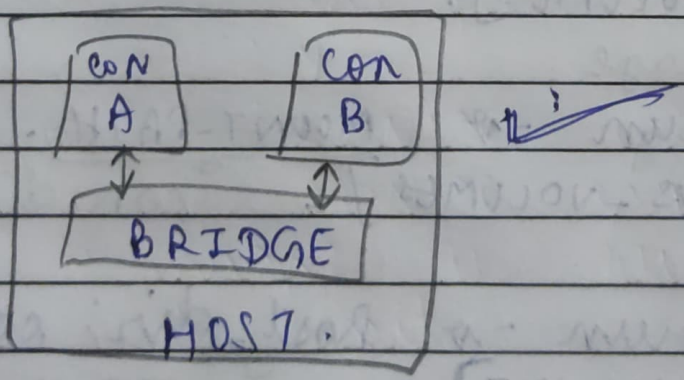
Topic: _____

⇒ docker volume ~~purne~~ prune

used to remove unused volumes.
targets anonymous volumes.
volumes that are no longer being used by any container.

⇒ DOCKER NETWORK :-

by default there are 3, bridge, host, null.
network drivers not specified then, it will by default take bridge driver.



27/07/25.

Date: _____ Page No. _____

Topic: _____

- Saving changes to a Container :-

⇒ docker commit <container-ID> <name-for-image-to-give>

⇒ docker commit b5597895769 test

⇒ new image is created using this Command (test).

⇒ test is seen under docker images.

- To Remove all Running Containers shortcut is :-

to stop them. ⇐ stop

docker rm -f $(docker ps -a -q)

force ← ↑ quiet

X. for all Container

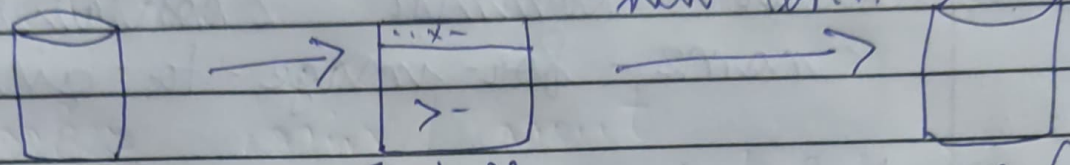
use $(docker ps -q) → for all Containers ~~only~~.

⇒ "Get me the ID's of all Containers (running or NOT), & then force-remove each of them."

Date: _____ Page No. _____

Topic: _____

⇒ Ex :-



- docker run -it -d ubuntu (abc).

docker exec -it abc bash
root@abc :/# . . . .
: apt-get update
: apt-get install apache2
: y
: service apache2 status
: service apache2 start
⇒ apach2 is running.
: exit.

docker ps # return its ID.
docker commit abc <username/image-name>
⇒ naming rule :- dharmapiyush10718/apache

Date: _____ Page No. _____

Topic: _____

docker images # ...../apache. ✓
docker ps # abc !! ID !!
docker rm -f abc.

clear / cls.

external port (LOCAL MACHINE)
docker run -it -p 82:80 -d
sharmapiyush10718/apache
internal port (DOCKER)

docker ps # XYZ ID.

docker exec -it XYZ bash
~~service~~
root@XYZ:/# . . . . . .
: service apache2 start.
error means not installed. else
it will be up & running.
IN OUR CASE :- up & running.
⇒ check on port 82.
⇒ localhost:82.
⇒ up & running inside a Container not inside a Machine
: exit
⇒
⇒ docker ps # XYZ.
⇒ docker stop XYZ. #refresh the page, &
it WON'T LOAD!...

Date: _____ Page No. _____

Topic: _____

STEP 4. PUSH ⟹ HUB.DOCKER.COM.

docker login / logout
docker push con-id (abc).

⇒ we have pushed our container onto the dockerhub.

⇒ local to remote push.

⇒ remote to local (pull).