Table of Contents

[**Multi-layer Builds** 7](#_Toc199672853)

[**Scenario-Based Questions** 7](#_Toc199672854)

[**Q1: You want to create a Dockerfile for a Python web app. It should:** 7](#_Toc199672855)

[**Q2: What is the difference between COPY and ADD?** 8](#_Toc199672856)

[**Q3: How can you ensure that the container runs only after the database service is ready?** 8](#_Toc199672857)

[**Q4: Why use ENTRYPOINT instead of CMD?** 8](#_Toc199672858)

[**Q5: How to build and run a Docker image?** 8](#_Toc199672859)

[**Shebang & Script Execution** 11](#_Toc199672860)

[**How to Run a Script** 11](#_Toc199672861)

[**Variables in Bash** 11](#_Toc199672862)

[**Define Variables** 11](#_Toc199672863)

[**Use Variables** 11](#_Toc199672864)

[**Read User Input** 11](#_Toc199672865)

[**Command Substitution ($(command))** 11](#_Toc199672866)

[**Environment Variables** 11](#_Toc199672867)

[**if Statements** 11](#_Toc199672868)

[**if-else Statement** 11](#_Toc199672869)

[**elif (Else If) Statement** 11](#_Toc199672870)

[**Comparing Strings** 11](#_Toc199672871)

[**For Loop** 11](#_Toc199672872)

[**While Loop** 11](#_Toc199672873)

[**Until Loop (Runs Until Condition is True)** 11](#_Toc199672874)

[**Loop Over Files** 11](#_Toc199672875)

[**Basic Function** 11](#_Toc199672876)

[**Function with Multiple Arguments** 12](#_Toc199672877)

[**Returning Values** 12](#_Toc199672878)

[**Read a File Line by Line** 12](#_Toc199672879)

[**Write to a File** 12](#_Toc199672880)

[**Check If a File Exists** 12](#_Toc199672881)

[**Run a Command in Background** 12](#_Toc199672882)

[**Kill a Process** 12](#_Toc199672883)

[**Check Running Processes** 12](#_Toc199672884)

[**Run Script in Debug Mode** 12](#_Toc199672885)

[**Enable Debugging Inside Script** 12](#_Toc199672886)

[**Check if a Package is Installed** 12](#_Toc199672887)

[**Find and Delete Files Larger Than 100MB** 12](#_Toc199672888)

[**Monitor Log File in Real-Time** 12](#_Toc199672889)

[**Find and Replace Text in a File** 12](#_Toc199672890)

[**Extract Column from CSV** 12](#_Toc199672891)

[**Q1: How do you create a script that automatically backs up a directory every hour?** 12](#_Toc199672892)

[**Q4: How do you write a script that checks system memory usage and alerts if usage exceeds 90%?** 12](#_Toc199672893)

[**Q5: How do you automate user creation in Linux using a Bash script?** 12](#_Toc199672894)

[Docker Compose Config: 18](#_Toc199672895)

[Solution for Question #1: Bash Script to Update config.yaml 23](#_Toc199672896)

[update\_config.sh (Bash Script): 23](#_Toc199672897)

[How to Use the Script 23](#_Toc199672898)

[Solution for Question #2: Docker Compose for Microservices (MySQL + Nginx) 23](#_Toc199672899)

[Docker Compose File (docker-compose.yml) 23](#_Toc199672900)

[Steps to Run the Containers 23](#_Toc199672901)

[Solution for Question #3: GitHub Actions for Node.js Project 23](#_Toc199672902)

[.github/workflows/ci.yml 23](#_Toc199672903)

**Docker Commands:**

* Docker build –file [custom docker file name] –t <image-name>:tag . (akhri dot context ka hai kay files kahan say milni hai)
* docker run -p 127.0.0.1:8000:8000 test:latest ( -p actually port define karta hai kay kis port par chalani hai and –d agr laga daeyn to wo as a background service par chalay gi without attached terminal)
* docker build –target build –t hello . **ham yeh wali command use kar sktay hain multistaged main kisi specific point par build ko stop karnay kay liyeh.**
* docker build –f –Dockerfile –no-cache –target stage2 .
* docker login –u username –p password (only to login for docker hub)
* To map Docker image port at some port run → sudo docker run -p port:port myfirst:v0.0.1
* To push docker image to docker hub we first tag image →
* sudo docker tag first-app:v0.0.1 syhaiderali/first-app:v0.0.1
* sudo docker push syhaiderali/first-app:v0.0.1

**Docker Build and Buildx + BuildKit:**

Buildx -> Front end

Buildkit -> Backend is a daemon process which takes Docker file, outputs, Caching and exporting Mechanism.

Build X to

**DockerFile:**

**Example:**

FROM node

WORKDIR app/

COPY . . (Second dot represents present directory)

RUN npm install

CMD [“npm”,”start”]

**FROM:**

FROM [--platform=<platform>] [<image>:tag] OR [<image>@<digest>] [AS <name>]

[--platform=$BUILDPLATFORM] use with CROSS Compiler to Targeted system.

**ARG and FROM using ARG For MultiStaged Builds:**ARG CODE\_VERSION=latest  
FROM base:${CODE\_VERSION}  
CMD /code/run-app  
FROM extras:${CODE\_VERSION}  
CMD /code/run-extras

* **ARG ki Value agr from say pehlay declared huwi hai to wo First From kay baad use nahi ho skti neechay di gai example main hamaeyn ARG ki value ko FORM Build kay ander access karna hai to then hamaeyn usko re initialize karna paray ga**   
    
  ARG VERSION=latest  
  FROM busybox:$VERSION

ARG VERSION

RUN echo $VERSION > image\_version

* **Multiple Instructions run karnay kay liyeh ham Dockerfile Run ka syntax kuch aysay use karaeyn gaey pehla method Heredoc(HereDocument kay sath hai)**RUN <<EOF

apt-get update

apt-get install -y curl

EOF

**2nd Method escape kay sath hai jis main line kay end par ham \ backslash use kartay hain linux ki multiple commands kay liyeh.**RUN apt-get update && \

apt-get install -y curl && \

rm -rf /var/lib/apt/lists/\*  
**Windows kay conflict kay liyeh ham `` back tick use kar sktay hain.**

# escape=`

FROM mcr.microsoft.com/windows/servercore

RUN echo Hello `

&& echo World  
  
**DockerFile basic Python Code ka Syntax jo Official para huwa hai:  
# syntax=docker/dockerfile:1**

FROM ubuntu:22.04

**# install app dependencies**

RUN apt-get update && apt-get install -y python3 python3-pip

RUN pip install flask==3.0.\*

**# install app**

COPY hello.py /

**# final configuration**

ENV FLASK\_APP=hello

EXPOSE 8000

CMD ["flask", "run", "--host", "0.0.0.0", "--port", "8000"]

Pehli line main basically dependency check ki ja rahi build kay liyeh konsa syntax use karna parsing kay liyeh.  
**ENV FLASK\_APP=hello** :: yeh aik example hai kay kaisay ap nay env variable ko aik environment main set karna hai.

**MultiStaged build:**

# syntax=docker/dockerfile:1

FROM golang:1.23

WORKDIR /src

COPY <<EOF ./main.go

package main

import "fmt"

func main() {

fmt.Println("hello, world")

}

EOF

RUN go build -o /bin/hello ./main.go

FROM scratch

COPY --from=0 /bin/hello /bin/hello

CMD ["/bin/hello"]

MultiStaged builds main ham har aik build ko AS say name kar sktay hain like this ->

**FROM golang:1.233 AS build1**

Last COPY --from=0 /bin/hello /bin/hello ( 0 yahan suggest karta hai pehli build ko but ab ham nay name day diya hai to ham ab aysay karaeyn gaey **COPY –from=build1 /path**)

**Ham external image bhi stage kay tor par use kar sktay hain aysay:**COPY --from=nginx:latest /etc/nginx/nginx.conf /nginx.conf

**Ham pichli build ko bhi agli main use kar sktay hain like this**FROM builder as build 1

**:: BuildKit just wohi builds banaey ga jis par builds depend kar rahi hogi.**

# syntax=docker/dockerfile:1

FROM ubuntu AS base

RUN echo "base"

FROM base AS stage1

RUN echo "stage1"

FROM base AS stage2

RUN echo "stage2"

**Run Cached**

The cache for RUN instructions isn't invalidated automatically between builds. Suppose you have a step in your Dockerfile to install curl

FROM alpine:3.21 AS install

RUN apk add curl

This doesn't mean that the version of curl in your image is always up-to-date. Rebuilding the image one week later will still get you the same packages as before. To force a re-execution of the RUN instruction, you can:

Make sure that a layer before it has changed

Clear the build cache ahead of the build using docker builder prune

Use the --no-cache or --no-cache-filter options

The --no-cache-filter option lets you specify a specific build stage to invalidate the cache for:

docker build --no-cache-filter install .

**Build Secrets Cached**

Build secrets

The contents of build secrets are not part of the build cache. Changing the value of a secret doesn't result in cache invalidation.

If you want to force cache invalidation after changing a secret value, you can pass a build argument with an arbitrary value that you also change when changing the secret. Build arguments do result in cache invalidation.

FROM alpine

ARG CACHEBUST

RUN --mount=type=secret,id=TOKEN,env=TOKEN \

some-command ...

TOKEN="tkn\_pat123456" docker build --secret id=TOKEN --build-arg CACHEBUST=1 .

Properties of secrets such as IDs and mount paths do participate in the cache checksum, and result in cache invalidation if changed.

**SIR’S DOCKER AND DOCKER COMPOSE FILE’s PROJECT**

**Github WorkFlows CI/CD pipline:**

**buildandpush.yml**

name: Docker Build and Push

on:

push:

branches:

- main

jobs:

build:

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v4

- name: Login to Docker Hub

run: echo ${{ secrets.DOCKERHUB\_PASSWORD }} | docker login -u ${{ secrets.DOCKERHUB\_USERNAME }} --password-stdin

- name: Build the docker image

run: docker build -t firstapp:latest .

- name: Tag the image

run: docker tag firstapp:latest jawadc/firstapp:latest

- name: Push the image

run: docker push jawadc/firstapp:latest

**Explanation:** Uper wali file main ham github main khudh ba khudh docker image ki build bannay and us ko docker hub par push karnay ki automation kar rahay hain. Syntax “**on**” bata raha hai kay yeh file kab chalay gi - > jab main push ho ga, dosra tag “**jobs:**” ka hai jo basically yeh bataey ga kay ap nay konsay konsay jobs karnay hain -> aik concept yahan github secrets ka bhi ajata hai jo kay ham github main actions -> secrets main ja kar ham day sktay hain. Password kay liyeh ham docker hub say ham aik token generate karaeyn gaey and “**DOCKER\_PASSWORD**” ki value main wo token rakh daeyn gaey.

**docker-compose.yml**

name: First App System

services:

app:

build:

context: ./

ports:

- 3000:3000

volumes:

- ./:/app

- /app/node\_modules/

db:

image: mongo:latest

redis:

image: redis:latest

Description: name tag helps to give the name to container, Services includes what we are going to provide in the container, app main ham apni jo build banaeyn gaey wo aey gi and **db and redis**, app tag kay parallel scope main chalay gi means to say app ,db and redis yeh teeno 3 containers banaeyn gaey aik sab container kay ander.

**DevOps Cheat-Sheet**

**Dockerfile**

|  |  |  |
| --- | --- | --- |
| **Command** | **Meaning** | **Example** |
| FROM | Specifies the base image | FROM node:18 |
| LABEL | Adds metadata to the image | LABEL maintainer="John Doe" |
| WORKDIR | Sets the working directory inside the container | WORKDIR /app |
| COPY | Copies files from host to container | COPY . /app |
| ADD | Like COPY, but can extract archives | ADD myfile.tar.gz /app/ |
| RUN | Runs a command during image build | RUN npm install |
| CMD | Default command executed when container starts | CMD ["node", "server.js"] |
| ENTRYPOINT | Like CMD, but more rigid (doesn’t get overridden) | ENTRYPOINT ["python", "app.py"] |
| EXPOSE | Declares a port to be used | EXPOSE 3000 |
| ENV | Sets environment variables | ENV PORT=8080 |
| ARG | Defines build-time variables | ARG VERSION=1.0 |
| VOLUME | Creates a mount point for persistent data | VOLUME /data |
| USER | Switch to a different user in container | USER node |
| HEALTHCHECK | Defines a health check command | `HEALTHCHECK CMD curl --fail [http://localhost](http://localhost/) |

**Multi-layer Builds**

FROM node:18 AS builder

WORKDIR /app

COPY . .

RUN npm install && npm run build

FROM node:18

WORKDIR /app

COPY --from=builder /app/dist .

CMD ["node", "index.js"]

**Use a .dockerignore file** to avoid copying unnecessary files:

node\_modules

.git

.env

**Minimize layers**: Combine RUN commands:

RUN apt-get update && apt-get install -y curl

**Use specific image versions** (avoid latest tag):

“FROM python:latest” instead of “FROM python:3.9”

**Scenario-Based Questions**

**Q1: You want to create a Dockerfile for a Python web app. It should:**

* Use Python 3.9
* Install dependencies from requirements.txt
* Set working directory as /app
* Expose port 5000
* Run app.py when the container starts

**Answer:**

FROM python:3.9

WORKDIR /app

COPY requirements.txt .

RUN pip install -r requirements.txt

COPY . .

EXPOSE 5000

CMD ["python", "app.py"]

**Q2: What is the difference between COPY and ADD?**

**Answer:**

* COPY is used to copy files from the host to the container.
* ADD does the same but can also extract archives (.tar.gz files).

**Q3: How can you ensure that the container runs only after the database service is ready?**

**📌 Answer:**  
Use HEALTHCHECK to wait for the DB:

HEALTHCHECK CMD curl --fail http://db:5432 || exit 1

**Q4: Why use ENTRYPOINT instead of CMD?**

**📌 Answer:**

* ENTRYPOINT ensures that the main process **cannot be overridden** when running docker run.
* CMD can be easily **overridden** with command-line arguments.

**Q5: How to build and run a Docker image?**

docker build -t myapp .

docker run -p 5000:5000 myapp

**Example of My Dockerfile:**

# Use a lightweight Node.js base image

**FROM node:alpine**

# Set the working directory inside the container actually defines /app as the working directory inside the container

**WORKDIR /app**

# Copy package.json and package-lock.json first (for better caching)

**COPY package\*.json ./**

# Install all dependencies (including dotenv)

**RUN npm install**

# Copy the rest of the application

**COPY . .**

# Expose the application port

**EXPOSE 3000**

# Define the command to run the app

**CMD ["npm", "start"]**

**Linux Commands**

|  |  |  |
| --- | --- | --- |
| Command | Usage | Example |
| ls | List directory contents. | ls -l (List in long format) |
| cd | Change the current directory. | cd /home/user (Change to /home/user directory) |
| pwd | Print the current working directory. | pwd (Outputs /home/user) |
| mkdir | Create a new directory. | mkdir new\_folder (Creates a directory named new\_folder) |
| rmdir | Remove an empty directory. | rmdir empty\_folder (Removes empty\_folder) |
| rm | Remove files or directories. | rm file.txt (Deletes file.txt) |
|  |  | rm -r folder (Recursively deletes folder and its contents) |
| cp | Copy files or directories. | cp file.txt /backup/ (Copies file.txt to /backup/) |
|  |  | cp -r folder /backup/ (Recursively copies folder to /backup/) |
| mv | Move or rename files or directories. | mv file.txt /new\_location/ (Moves file.txt to /new\_location/) |
|  |  | mv old\_name.txt new\_name.txt (Renames old\_name.txt to new\_name.txt) |
| touch | Create an empty file or update the timestamp of an existing file. | touch new\_file.txt (Creates new\_file.txt) |
| cat | Display the contents of a file. | cat file.txt (Displays the contents of file.txt) |
| more | Display file contents one screen at a time. | more large\_file.txt (Displays large\_file.txt page by page) |
| less | Similar to more, but allows backward navigation. | less large\_file.txt (Displays large\_file.txt with navigation) |
| head | Display the first few lines of a file. | head -n 10 file.txt (Displays the first 10 lines of file.txt) |
| tail | Display the last few lines of a file. | tail -n 10 file.txt (Displays the last 10 lines of file.txt) |
| grep | Search for a pattern in a file. | grep "error" log.txt (Searches for the word "error" in log.txt) |
| find | Search for files and directories. | find /home -name "\*.txt" (Finds all .txt files in /home) |
| chmod | Change file permissions. | chmod 755 script.sh (Gives read, write, execute to owner, and read/execute to others) |
| chown | Change file ownership. | chown user:group file.txt (Changes ownership of file.txt to user:group) |
| ps | Display information about running processes. | ps aux (Lists all running processes) |
| top | Display real-time system processes. | top (Shows real-time system stats and processes) |
| kill | Terminate a process by PID. | kill 1234 (Terminates process with PID 1234) |
| tar | Archive files. | tar -cvf archive.tar folder/ (Creates archive.tar from folder/) |
|  |  | tar -xvf archive.tar (Extracts archive.tar) |
| gzip | Compress files. | gzip file.txt (Compresses file.txt to file.txt.gz) |
| gunzip | Decompress files. | gunzip file.txt.gz (Decompresses file.txt.gz to file.txt) |
| ssh | Connect to a remote server securely. | ssh user@192.168.1.1 (Connects to 192.168.1.1 as user) |
| scp | Securely copy files between local and remote systems. | scp file.txt user@192.168.1.1:/remote/path/ (Copies file.txt to remote) |
| wget | Download files from the web. | wget https://example.com/file.zip (Downloads file.zip) |
| curl | Transfer data from or to a server. | curl -O https://example.com/file.zip (Downloads file.zip) |
| df | Display disk space usage. | df -h (Shows disk usage in human-readable format) |
| du | Display directory space usage. | du -sh folder/ (Shows the size of folder/ in human-readable format) |
| uname | Display system information. | uname -a (Shows all system information) |
| history | Display command history. | history (Shows a list of previously executed commands) |
| alias | Create a shortcut for a command. | alias ll='ls -la' (Creates an alias ll for ls -la) |
| echo | Display a message or variable value. | echo "Hello, World!" (Prints "Hello, World!") |
| export | Set environment variables. | export PATH=$PATH:/new/path (Adds /new/path to PATH) |
| man | Display the manual page for a command. | man ls (Shows the manual for the ls command) |
| sudo | Execute a command with superuser privileges. | sudo apt update (Updates package lists with superuser privileges) |
| apt | Package management for Debian-based systems. | sudo apt install nginx (Installs nginx) |
| systemctl | Manage system services. | sudo systemctl start nginx (Starts the nginx service) |
| journalctl | View system logs. | journalctl -u nginx (Shows logs for the nginx service) |

|  |  |  |
| --- | --- | --- |
| Command | Usage | Example |
| basename | Extracts filename from a path. | basename /home/user/file.txt → file.txt |
| dirname | Extracts directory path from a full path. | dirname /home/user/file.txt → /home/user |
| which | Shows the full path of a command. | which python → /usr/bin/python |
| whereis | Finds the binary, source, and manual page for a command. | whereis ls → ls: /bin/ls /usr/share/man/man1/ls.1.gz |
| whoami | Displays the current logged-in user. | whoami → user |
| who | Shows who is logged into the system. | who |
| id | Displays user and group IDs. | id user |
| groups | Shows groups a user belongs to. | groups user |
| cut | Extracts specific columns from input. | `echo "one,two,three" |
| awk | Pattern scanning and text processing. | `echo "one two three" |
| sed | Stream editor for modifying files. | sed 's/old/new/g' file.txt |
| xargs | Constructs command lines from input. | `echo "file1 file2" |
| tee | Reads input and writes to both stdout and a file. | `echo "Hello" |
| diff | Compares two files line by line. | diff file1.txt file2.txt |
| cmp | Compares two files byte by byte. | cmp file1.txt file2.txt |
| stat | Displays detailed file information. | stat file.txt |
| touch | Updates file timestamps or creates a new file. | touch newfile.txt |
| wc | Counts words, lines, characters in a file. | wc -l file.txt |
| sort | Sorts lines of text files. | sort file.txt |
| uniq | Filters duplicate lines. | uniq file.txt |
| tr | Translates or deletes characters. | `echo "hello" |
| split | Splits large files into smaller ones. | split -b 10M largefile |
| paste | Merges lines from multiple files. | paste file1.txt file2.txt |
| tac | Displays file contents in reverse order. | tac file.txt |
| df -i | Shows inode usage. | df -i |
| fuser | Identifies processes using a file. | fuser file.txt |
| lsof | Lists open files. | `lsof |
| watch | Repeats a command at regular intervals. | watch -n 2 free -m |
| uptime | Shows system uptime. | uptime |
| vmstat | Reports system performance. | vmstat 2 5 |
| iostat | Displays CPU and disk usage statistics. | iostat -x 2 5 |
| htop | Interactive process monitoring. | htop |
| ionice | Sets disk I/O scheduling priority. | ionice -c3 command |
| nice | Starts a process with a given priority. | nice -n 10 command |
| renice | Changes priority of a running process. | renice 5 -p 1234 |
| strace | Traces system calls of a process. | strace -p 1234 |
| tcpdump | Captures network packets. | tcpdump -i eth0 |
| netstat | Displays network connections. | netstat -tulnp |
| ss | Newer alternative to netstat. | ss -tulnp |
| iptables | Configures firewall rules. | sudo iptables -L |
| ufw | Manages uncomplicated firewall. | sudo ufw allow 22 |
| mount | Mounts a filesystem. | sudo mount /dev/sdb1 /mnt |
| umount | Unmounts a filesystem. | sudo umount /mnt |
| blkid | Displays block device information. | blkid |
| lsblk | Shows block devices in a tree format. | lsblk |
| fdisk | Partition management. | sudo fdisk /dev/sda |
| mkfs | Creates a new filesystem. | sudo mkfs.ext4 /dev/sdb1 |
| fsck | Checks and repairs filesystem. | sudo fsck -y /dev/sdb1 |
| du -sh \* | Displays disk usage per directory. | du -sh \* |
| crontab -e | Edits cron jobs for scheduling. | crontab -e |
| at | Schedules one-time tasks. | `echo "ls -l" |

**Bash, Shell Scripting**

**Shebang & Script Execution**

Every Bash script **must** start with a **shebang (#!)** to define the interpreter:

#!/bin/bash # This script runs with Bash shell

**How to Run a Script**

chmod +x script.sh # Give execute permissions

./script.sh # Run the script

**Variables in Bash**

**Define Variables**

name="Alice"

age=25

❌ **No spaces around =**

name = "Alice" # ❌ WRONG!

**Use Variables**

echo "Hello, my name is $name and I am $age years old."

**Read User Input**

read -p "Enter your name: " user\_name

echo "Welcome, $user\_name!"

**Command Substitution ($(command))**

current\_date=$(date)

echo "Today's date is $current\_date"

**Environment Variables**

echo "Home directory: $HOME"

echo "Current user: $USER"

**Conditional Statements**

**if Statements**

if [ $age -gt 18 ]; then

echo "You are an adult."

fi

**if-else Statement**

if [ $age -ge 18 ]; then

echo "You can vote."

else

echo "You are too young to vote."

fi

**elif (Else If) Statement**

if [ $age -lt 13 ]; then

echo "You are a child."

elif [ $age -lt 20 ]; then

echo "You are a teenager."

else

echo "You are an adult."

fi

**Comparing Strings**

if [ "$name" == "Alice" ]; then

echo "Hello Alice!"

fi

**Loops**

**For Loop**

for i in 1 2 3 4 5; do

echo "Number: $i"

done

**While Loop**

count=1

while [ $count -le 5 ]; do

echo "Iteration $count"

((count++))

done

**Until Loop (Runs Until Condition is True)**

count=1

until [ $count -gt 5 ]; do

echo "Iteration $count"

((count++))

done

**Loop Over Files**

for file in \*.txt; do

echo "Processing $file..."

done

**Functions in Bash**

**Basic Function**

greet() {

echo "Hello, $1!"

}

greet "Alice"

📝 **$1 refers to the first argument passed to the function.**

**Function with Multiple Arguments**

sum() {

echo "Sum: $(($1 + $2))"

}

sum 5 10

**Returning Values**

multiply() {

echo $(($1 \* $2))

}

result=$(multiply 4 5)

echo "Multiplication result: $result"

**File Handling**

**Read a File Line by Line**

while IFS= read -r line; do

echo "$line"

done < file.txt

**Write to a File**

echo "Hello World" > output.txt # Overwrites file

echo "Appended line" >> output.txt # Appends to file

**Check If a File Exists**

if [ -f "file.txt" ]; then

echo "File exists."

fi

**Process Management**

**Run a Command in Background**

./long\_process.sh &

**Kill a Process**

kill $(pidof process\_name)

**Check Running Processes**

ps aux | grep process\_name

**Debugging Bash Scripts**

**Run Script in Debug Mode**

bash -x script.sh

**Enable Debugging Inside Script**

set -x # Start debugging

echo "Debugging mode enabled"

set +x # Stop debugging

**Useful One-Liners**

**Check if a Package is Installed**

dpkg -l | grep package\_name

**Find and Delete Files Larger Than 100MB**

find /path/to/dir -type f -size +100M -exec rm -rf {} \;

**Monitor Log File in Real-Time**

tail -f /var/log/syslog

**Find and Replace Text in a File**

sed -i 's/old-text/new-text/g' file.txt

**Extract Column from CSV**

cut -d ',' -f2 data.csv

**Scenario-Based Questions**

**Q1: How do you create a script that automatically backs up a directory every hour?**

✅ **Answer:**

#!/bin/bash

src="/home/user/documents"

dest="/backup/documents\_$(date +%F\_%T).tar.gz"

tar -czf "$dest" "$src"

echo "Backup completed: $dest"

➡ **Then, schedule it with cron:**

crontab -e

0 \* \* \* \* /path/to/backup\_script.sh

**Q2: How do you check if a website is online using Bash?**

✅ **Answer:**

#!/bin/bash

URL="https://example.com"

if curl -s --head --request GET $URL | grep "200 OK" > /dev/null; then

echo "Website is online"

else

echo "Website is down"

fi

**Q3: How do you create a script that renames all .txt files in a directory by adding \_backup to their name?**

✅ **Answer:**

#!/bin/bash

for file in \*.txt; do

mv "$file" "${file%.txt}\_backup.txt"

done

**Q4: How do you write a script that checks system memory usage and alerts if usage exceeds 90%?**

✅ **Answer:**

#!/bin/bash

mem\_usage=$(free | awk '/Mem/{printf "%.2f", $3/$2 \* 100}')

if (( $(echo "$mem\_usage > 90" | bc -l) )); then

echo "ALERT: Memory usage at ${mem\_usage}%!"

fi

**Q5: How do you automate user creation in Linux using a Bash script?**

✅ **Answer:**

#!/bin/bash

read -p "Enter new username: " new\_user

sudo useradd -m $new\_user

echo "User $new\_user created successfully."

**Q6: How do you write a script that finds and deletes all log files older than 7 days?**

✅ **Answer:**

#!/bin/bash

find /var/log -name "\*.log" -type f -mtime +7 -exec rm -f {} \;

echo "Deleted all log files older than 7 days."

**Q7: How do you create a script that monitors CPU usage and sends an alert if usage exceeds 80%?**

✅ **Answer:**

**#!/bin/bash**

**cpu\_usage=$(top -bn1 | grep "Cpu(s)" | awk '{print $2 + $4}')**

**threshold=80.0**

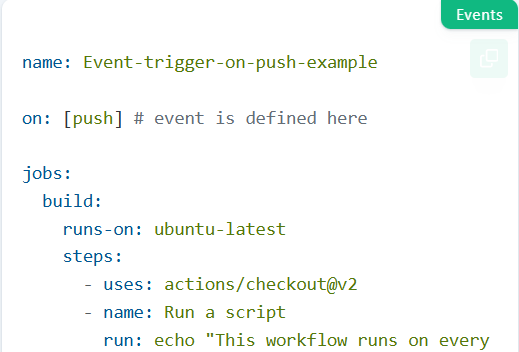
**if (( $(echo "$cpu\_usage > $threshold" | bc -l) )); then**

**echo "ALERT: CPU usage is at ${cpu\_usage}%!" | mail -s "High CPU Usage Alert" user@example.com**

**fi**

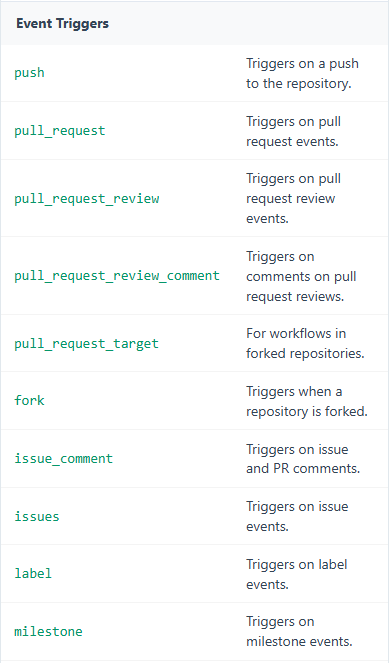
**CI/CD Pipelines**

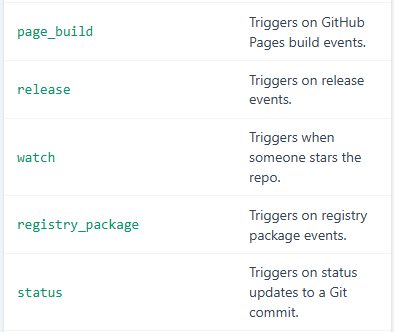
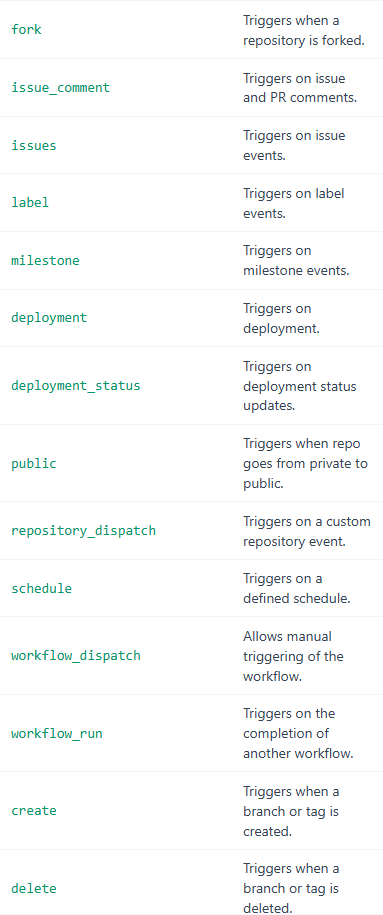
****

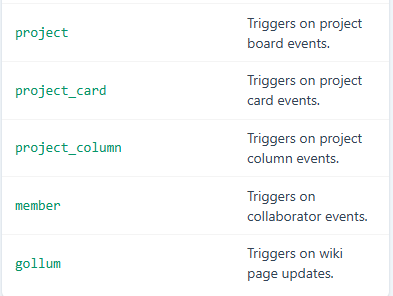
****

****

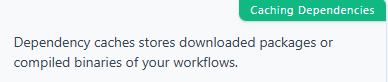
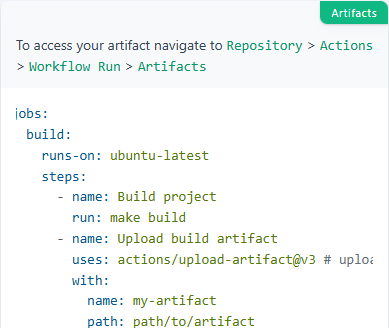
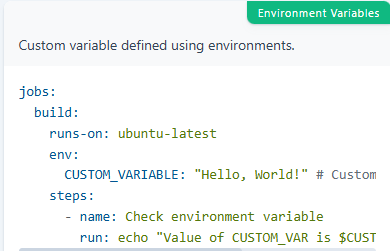
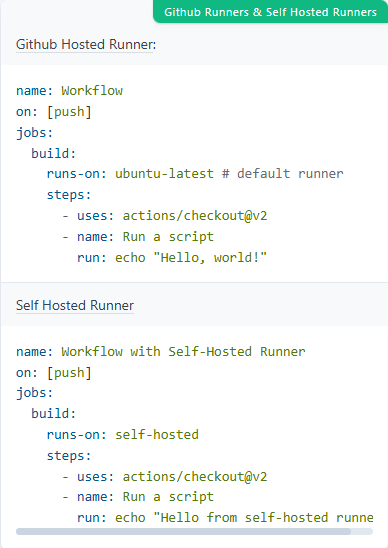
**Event Triggers**



****

****

****

**jobs:**

**build:**

**runs-on: ubuntu-latest**

**steps:**

**- uses: actions/checkout@v2**

**- name: Cache dependencies**

**uses: actions/cache@v2 # stores downloaded packages or compiled binaries**

**with:**

**path: |**

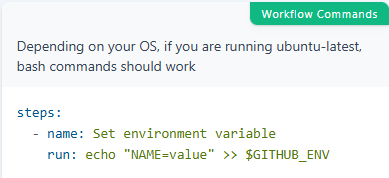
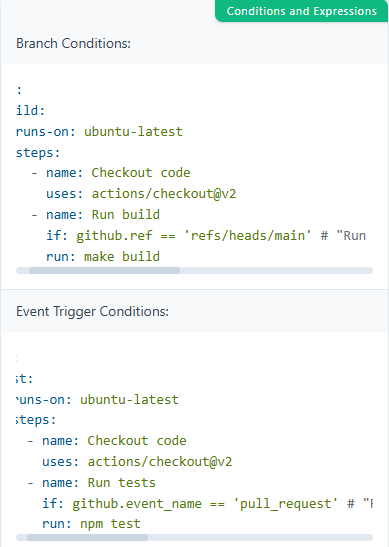
**path/to/dependencies**

**another/path**

**key: ${{ runner.os }}-deps-${{ hashFiles('\*\*/lockfile') }} # hash of the dependency lock file is generated in the OS**

**- name: Install dependencies**

**run: install-command**

****

name: Docker Build and Push   #Defines the name of the workflow

on:

    push:

        branches:

            - main

jobs:

    buildnpush:

        runs-on: ubuntu-latest

        steps:

        - uses: actions/checkout@v4

        - name: Docker Hub Login

          run: echo ${{ secrets.DOCKERHUB\_PASSWORD }} | docker login -u ${{ secrets.DOCKERHUB\_USERNAME }} --password-stdin

        - name: Build the Docker Image

          run: docker build -t first\_project:latest .

        - name: Tag the image

          run: docker tag first\_project:latest ${{ secrets.DOCKERHUB\_USERNAME }}/first\_project:latest

        - name: Push the image

          run: docker push ${{ secrets.DOCKERHUB\_USERNAME }}/first\_project:latest

**Docker-Compose.yml**

**BASIC COMMANDS:**

# Build or rebuild services

docker compose build

# Creates containers for a service.

docker compose create

# Start services

docker compose start

# Restart service containers

docker compose restart

# Create and start containers

docker compose up

# Run a one-off command on a service.

docker compose run

# Pause services

docker compose pause

# Unpause services

docker compose unpause

# Stop services

docker compose stop

# Removes stopped service containers

docker compose rm

# Stop and remove containers, networks

docker compose down

# Force stop service containers.

docker compose kill

# Execute a command in a running container.

docker compose exec

**Check Status COMMANDS:**

# List running compose projects

docker compose ls

# List containers

docker compose ps

# Print the public port for a port binding.

docker compose port

# Receive real time events from containers.

docker compose events

# View output from containers

docker compose logs

# Display the running processes

docker compose top

**Images:**

# Pull service images

docker compose pull

# Push service images

docker compose push

# List images used by the created containers

docker compose images

**File Operations:**

# Copy files/folders from a service container to the local filesystem

docker compose cp <service>:<src-filepath> <dst-filepath>

# Copy files/folders from the local filesystem to a service container.

docker compose cp <src-filepath> <service>:<dst-filepath>

**Others:**

# Show the Docker Compose version information

docker compose version

# Converts the compose file to platform’s canonical format

docker compose convert

## Docker Compose Config:

**An Basic Example:**

version: '3.9'

services:

hello-world:

image: hello-world:latest

**Spec: Build**[:](https://sweworld.net/cheatsheets/docker_compose/#build)

services:

web:

# Build from Dockerfile

build: .

# Build arguments.

args:

APP\_HOME: app

# Build from custom Dockerfile

build:

context: ./dir

dockerfile: Dockerfile.dev

# Build image.

image: debian

image: ubuntu

image: ubuntu:20.04

#### **Network**

services:

web:

# Set container network mode.

network\_mode: "host"

network\_mode: "none"

network\_mode: "service:[service name]"

# Define the networks that service containers are attached to.

networks:

- some-network

- other-network

# Expose container ports.

ports:

- "3000"

- "3000-3005"

- "8000:8000"

- "9090-9091:8080-8081"

- "49100:22"

- "127.0.0.1:8001:8001"

- "127.0.0.1:5000-5010:5000-5010"

- "6060:6060/udp"

# Define dns server.

dns: 8.8.8.8

# Define custom DNS search domains to set on container network interface configuration.

dns\_search: example.com

# List custom DNS options to be passed to the container’s DNS resolver.

dns\_opt:

- use-vc

- no-tld-query

# Defines a network link to containers in another service.

links:

- db

- db:database

- redis

#### **Environment Variable**

services:

web:

# Define environment variables.

environment:

RACK\_ENV: development

SHOW: "true"

USER\_INPUT:

COMPOSE\_PROJECT\_NAME: "foo"

# Define environment variables from file.

env\_file: .env

env\_file:

- ./a.env

- ./b.env

#### **Commands in docker-compose:**

services:

web:

# Start up command, which overrides the image default command.

command: echo "I'm running ${COMPOSE\_PROJECT\_NAME}"

# Start up command in the list form, which overrides the image default command.

entrypoint:

- php

- -d

- zend\_extension=/usr/local/lib/php/extensions/no-debug-non-zts-20100525/xdebug.so

- -d

- memory\_limit=-1

- vendor/bin/phpunit

#### **Labels**

services:

web:

# Container label meta data.

labels:

com.example.description: "Accounting webapp"

com.example.department: "Finance"

com.example.label-with-empty-value: ""

#### **Logging**

services:

web:

# Define logging.

logging:

driver: syslog

options:

syslog-address: "tcp://192.168.0.42:123"

#### **Dependencies**

services:

web:

build: .

# Define startup and shutdown dependencies between services.

depends\_on:

- db

- redis

redis:

image: redis

db:

image: postgres

**Sir wali File**

name: First App System

services:

  app:

    build:

      context: .

      dockerfile: Dockerfile

    ports:

      - 3000:3000

    volumes:

      - ./:/app

      - /app/node\_modules

  db:

    image: mongo:latest

**Apni File**

services:

  app:

    build: .

    ports:

      - "3000:3000"

    depends\_on:

      - mongo

    environment:

      - MONGO\_URI=mongodb://mongo:27017/mydatabase

      - DB\_NAME=my\_database

    networks:

      - my\_own\_network

  mongo:

    image: mongo

    container\_name: my\_mongo\_container

    ports:

      - "27017:27017"      environment:

      MONGO\_INITDB\_DATABASE: mydatabase

    networks:

      - my\_own\_network

networks:

  my\_own\_network:  # Define a custom network for communication between services

volumes:

  mongo-data:  # Define a named volume for MongoDB data persistence

**2 Containers on same Network**

# Create a custom network named 'my\_own\_network'

docker network create my\_own\_network

docker run -d \ # Run the container in detached mode

--name my\_mongo\_container \ # Assign a custom name to the container

--network my\_own\_network \ # Connect the container to 'my\_own\_network'

-p 27017:27017 \ # Expose MongoDB on port 27017

-e MONGO\_INITDB\_DATABASE=mydatabase \ # Set the initial database name

mongo # Use the official MongoDB image

docker build -t my\_app . # Build the Docker image for the app

docker run -d \

--name my\_app\_container \ # Assign a custom container name for the app

--network my\_own\_network \ # Connect to the same network as MongoDB

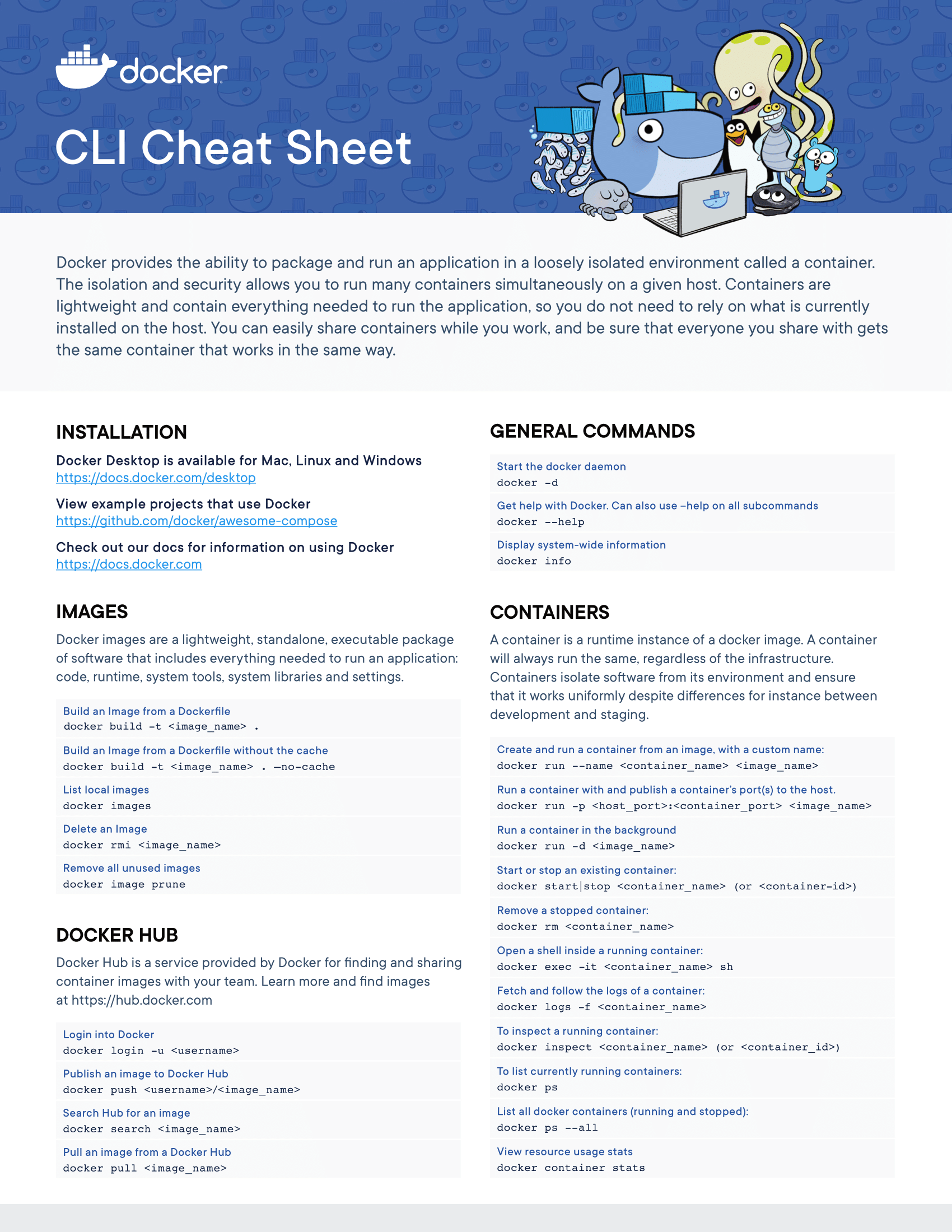
-p 3000:3000 \ # Map port 3000 of the container to the host

--env-file .env \ # Use the environment variables from the .env file

my\_app # Run the app using the built image

docker ps # List running containers

docker network inspect my\_own\_network # Check which containers are connected to the network



**Quiz#01 Section A**

Question 1: You are tasked with automating the deployment of a multi-service application using Docker.

A PostgreSQL database (postgres:12) to host port 5437.

* A frontend service (frontend:6.0) mapped to host port 80.
* A backend service (backend:2.1) mapped to host port 8080.

Part A: Write a Bash script:

* Pull the Docker images.
* Run all containers in detached mode with the specified port mappings.
* Ensure container names are db, frontend, and backend, respectively.

Part B: Convert the above script into a Docker Compose file (docker-compose.yml) to orchestrate the services. Include:

* Service definitions with images, container names, and ports.
* A volume named db\_data mounted at /var/lib/postgresql/data for the PostgreSQL service.

Part C: Design a GitHub Actions CI/CD pipeline (deploy.yml) to:

* Trigger on push to the main branch or pull\_request targeting main.

Perform the following jobs in sequence:

1. Check out the repository.
2. Builds and tests your app (frontend + backend in one repo) using npm install, build, and test.
3. Deploy the Docker Compose stack only if tests pass and the event is a push to main.

**Solution:**

PART A:

#! /bin/bash

docker pull postgres:12

docker pull frontend:6.0

docker pull backend:2.1

docker run -d --name db -p 5437:5437 postgres:12

docker run -d --name frontend -p 80:80 frontend:6.0

docker run -d --name backend -p 8080:8080 backend:2.1

PART B:

name: Part B of quiz

services:

db:

image: postgres:12

container\_name: db

ports:

- 5437:5437

volumes:

- db\_data: /var/lib/PostgreSQL/data

frontend:

image: frontend:6.0

container\_name: frontend

ports:

- 80:80

backend:

image: backend:2.1

container\_name: backend

ports:

- 8080:8080

volumes:

db\_data:

PART C:

name: Deploy and Test

on:

push:

branches:

- main

pull\_request:

branches:

- main

jobs:

buildntest:

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v4

- uses: actions/setup-node@v2

with:

node-version: '18'

- name: Installing Dependencies

run: npm install

- name: Building the Node Application

run: npm run build

- name: Testing Application

run: npm test

deploy:

runs-on: ubuntu-latest

needs: buildntest

if: github.event\_name == 'push' && github.ref == 'refs/heads/main'

steps:

- name: Checkout repository

uses: actions/checkout@v2

- name: Deploy Docker Compose stack

run: docker-compose up -d

**Quiz#01 Section B**

# **Solution for Question #1: Bash Script to Update** config.yaml

### update\_config.sh **(Bash Script):**

#!/bin/bash

# Check if the user provided an argument

if [ $# -ne 1 ]; then

echo "Usage: ./update\_config.sh <AUTH\_KEY>"

exit 1

fi

# Store the argument in a variable

AUTH\_KEY=$1

sed -i "s/REPLACE\_WITH\_AUTH\_KEY/$AUTH\_KEY/g" config.yaml

echo "✅ Authentication key updated successfully in config.yaml!"

### **How to Use the Script**

1. **Make it executable:**
2. chmod +x update\_config.sh
3. **Run the script with an authentication key:**
4. ./update\_config.sh ZXCVBNM123

✅ **What Happens?**

* It replaces "REPLACE\_WITH\_AUTH\_KEY" with "ZXCVBNM123" inside config.yaml

# **Solution for Question #2: Docker Compose for Microservices (MySQL + Nginx)**

### **Docker Compose File (**docker-compose.yml**)**

version: '3.8'

networks:

app\_network: # Define custom network

driver: bridge

services:

db\_container:

image: mysql:latest

container\_name: db\_container

environment:

MYSQL\_ROOT\_PASSWORD: rootpass

MYSQL\_DATABASE: app\_db

ports:

- "3306:3306"

networks:

- app\_network

web\_container:

image: nginx:latest

container\_name: web\_container

ports:

- "80:80"

volumes:

- ./nginx.conf:/etc/nginx/nginx.conf # Mount nginx config file

networks:

- app\_network

depends\_on:

- db\_container

### **Steps to Run the Containers**

1. **Create & navigate to the project directory:**
2. mkdir microservices-app && cd microservices-app
3. **Create docker-compose.yml & nginx.conf file.**
4. **Run the services using Docker Compose:**
5. docker-compose up -d

# **Solution for Question #3: GitHub Actions for Node.js Project**

### .github/workflows/ci.yml

name: Node.js CI/CD Workflow

on:

push:

branches:

- main

pull\_request:

branches:

- main

jobs:

build-and-test:

runs-on: ubuntu-latest # Use the latest Ubuntu OS

steps:

- name: Checkout Repository

uses: actions/checkout@v4 # Fetch code from GitHub

- name: Setup Node.js

uses: actions/setup-node@v3

with:

node-version: "18" # Use Node.js v18

- name: Install Dependencies

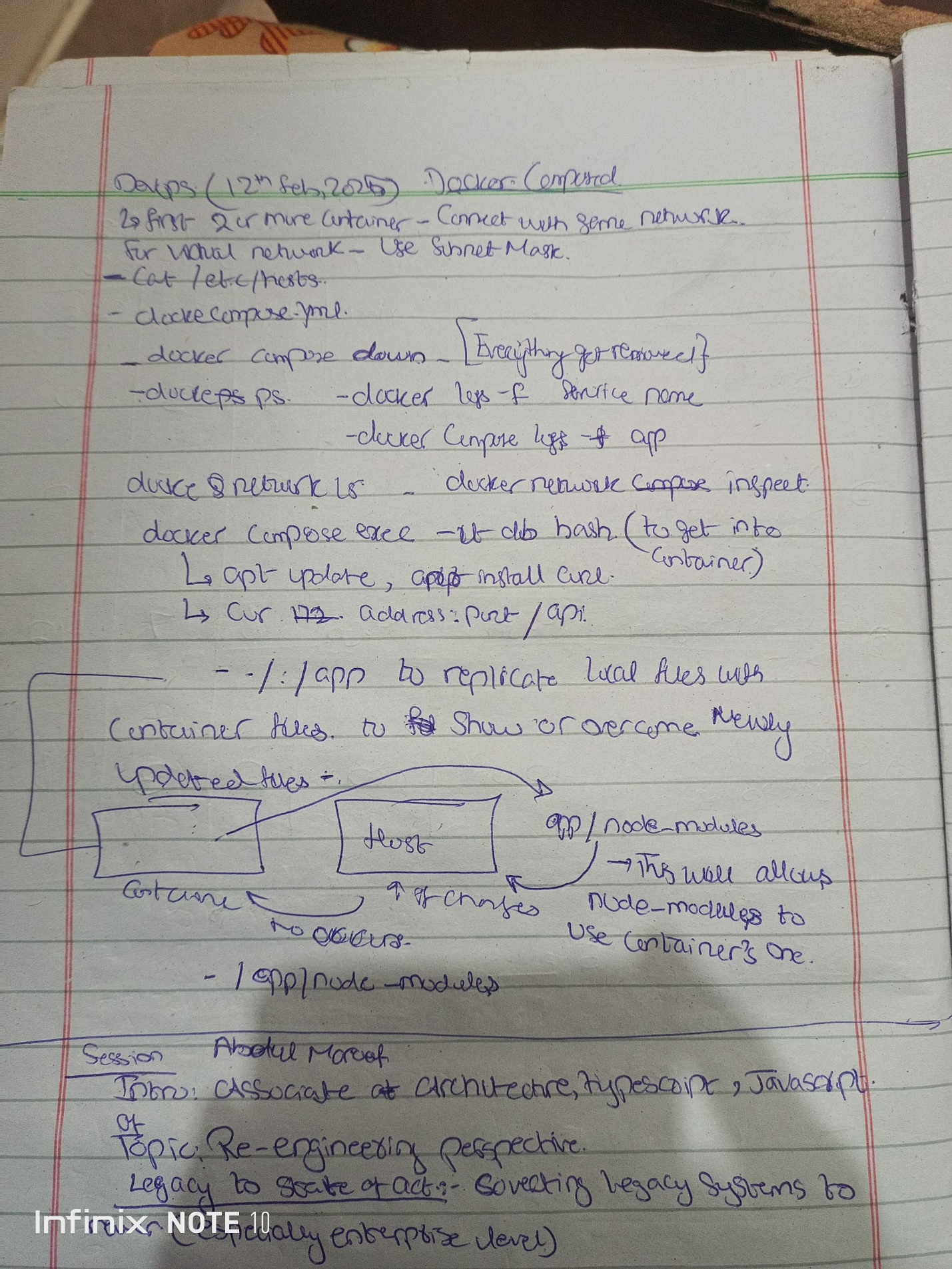
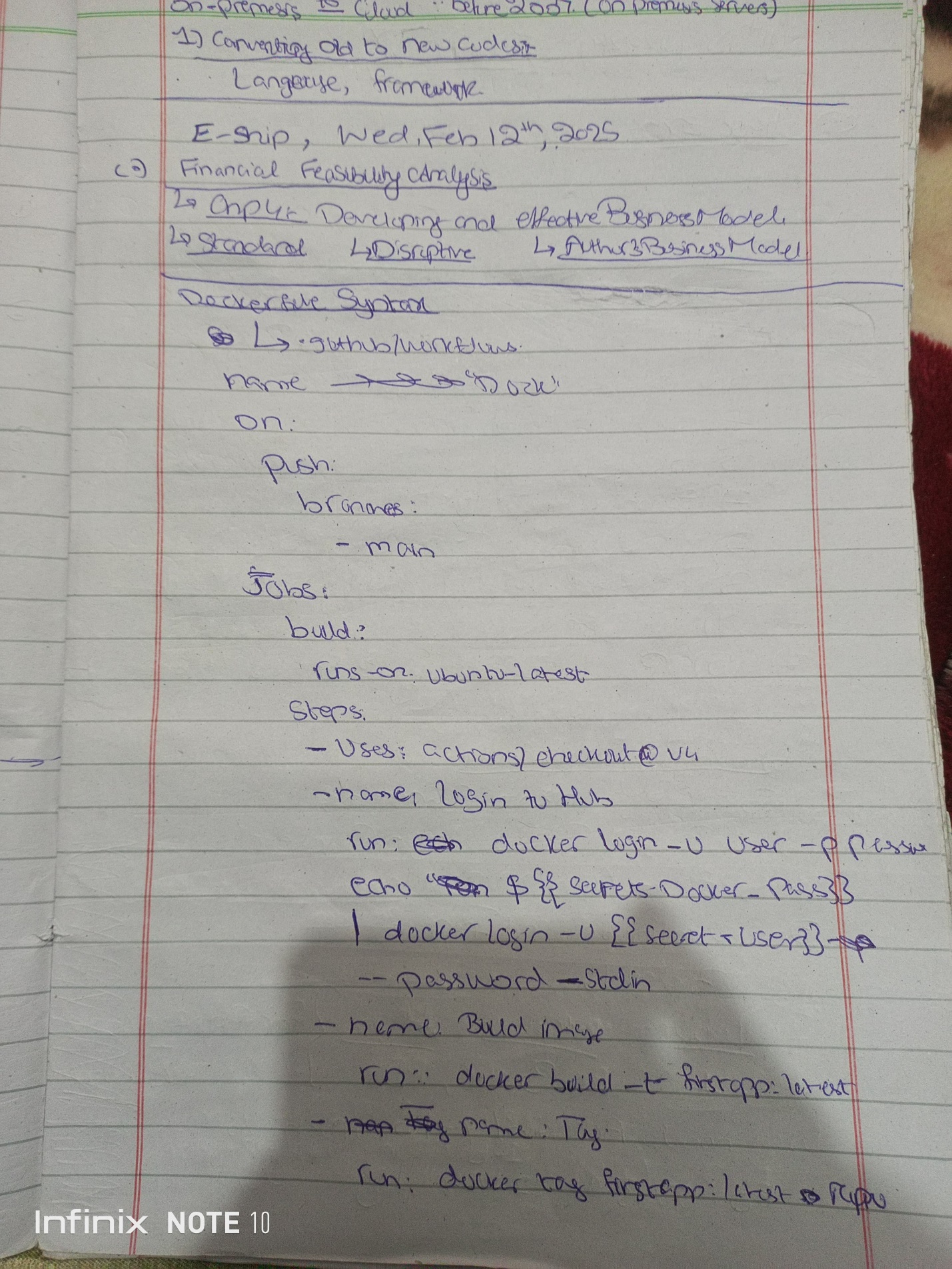
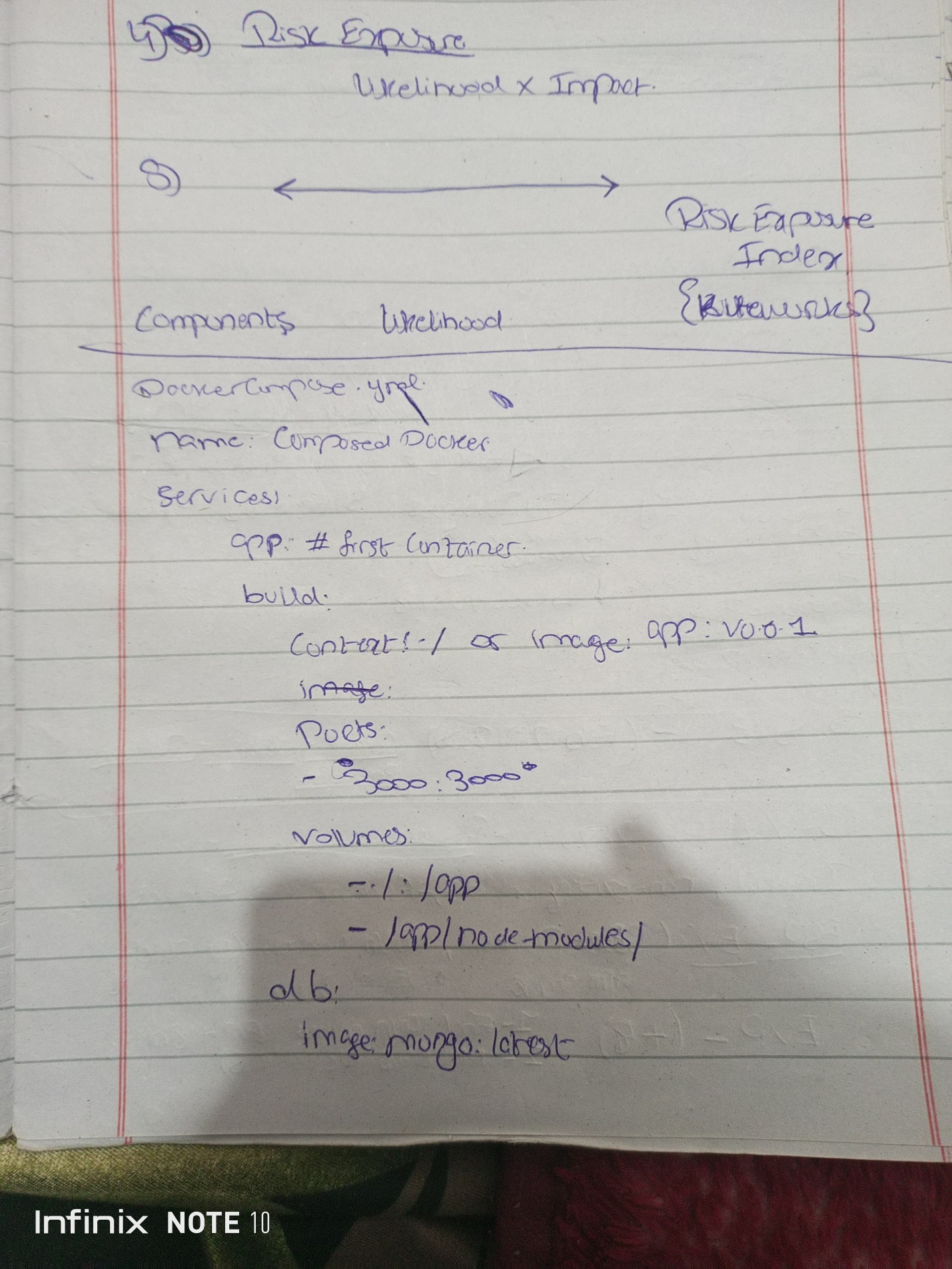
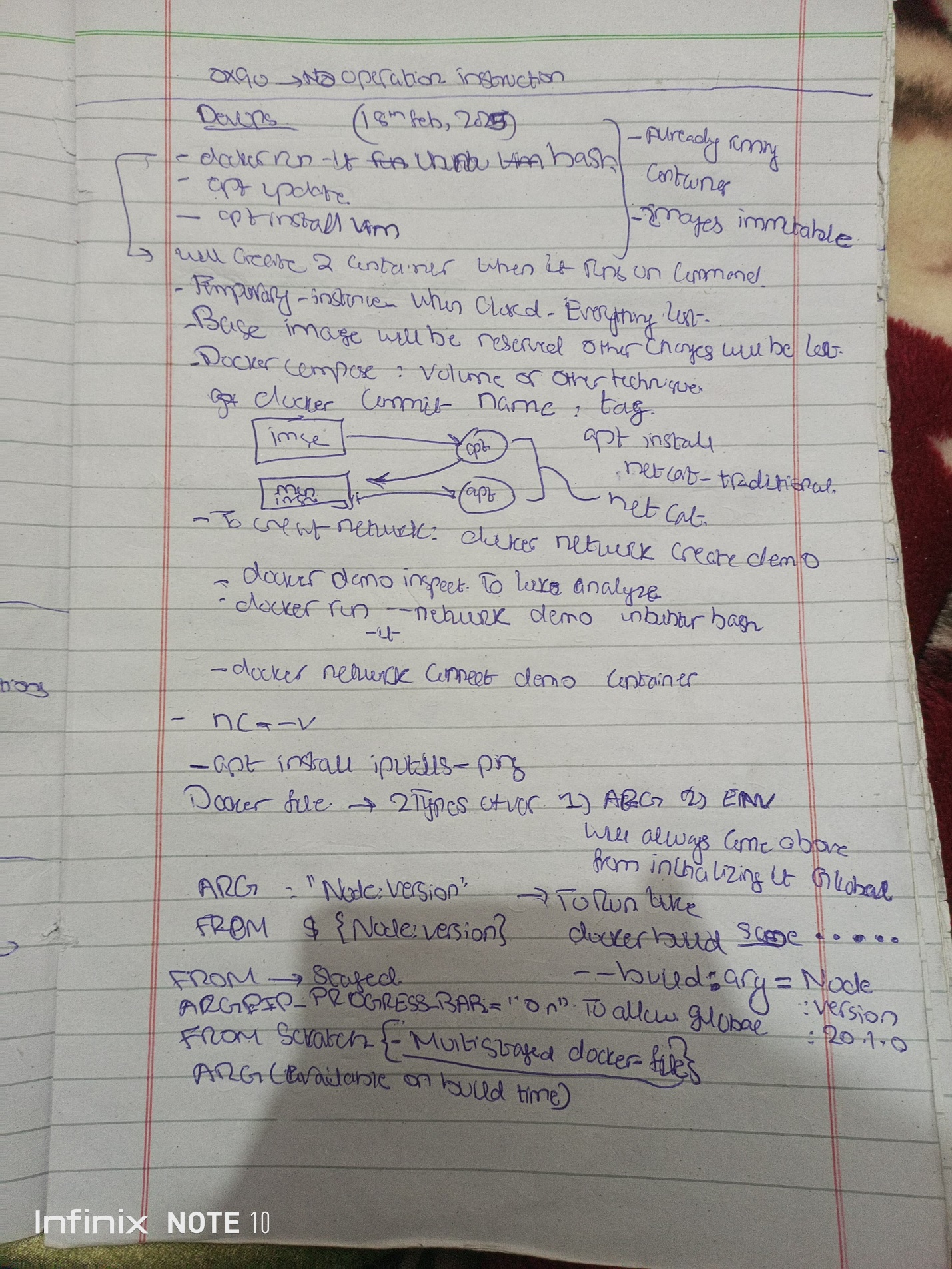
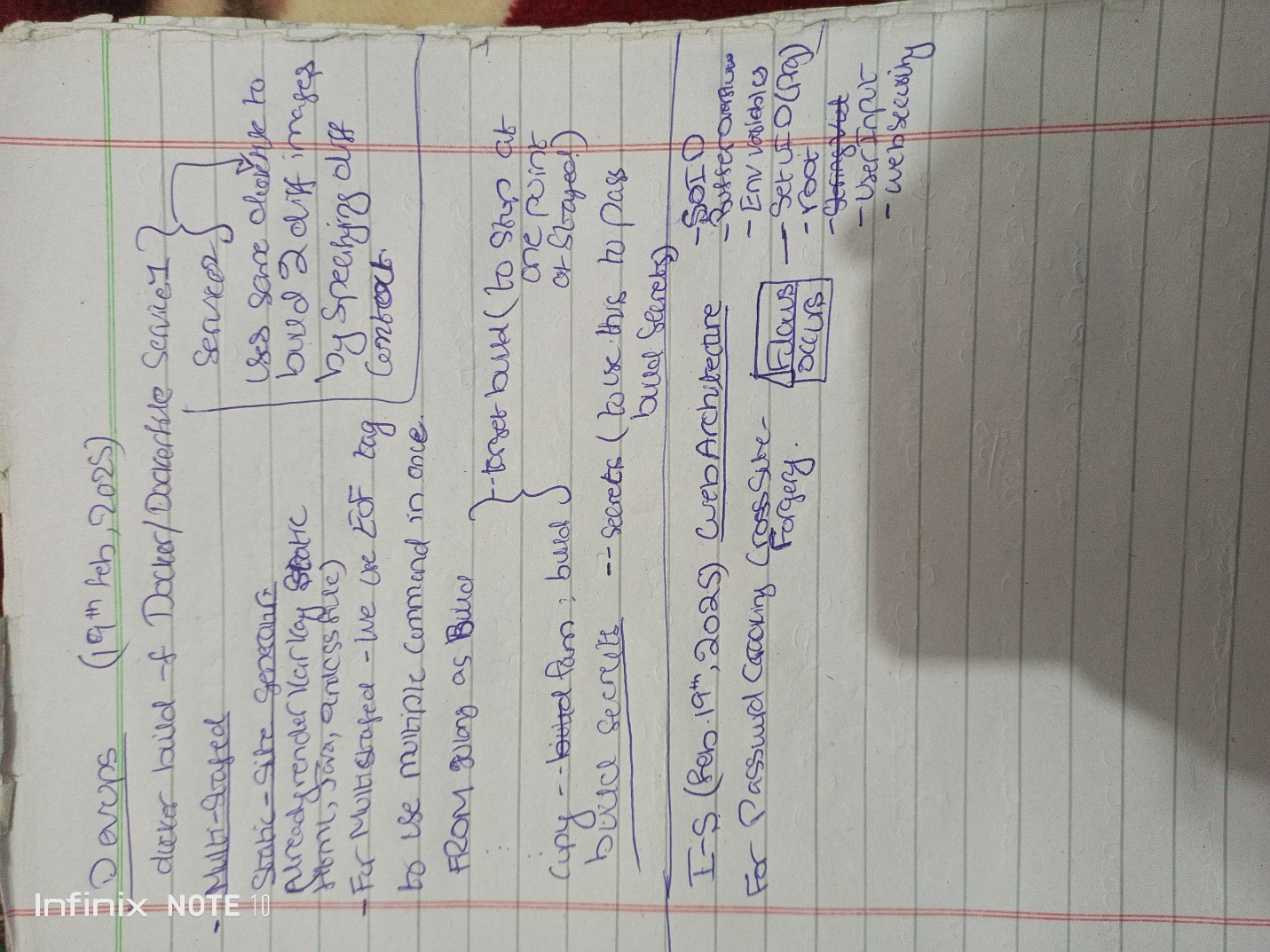
run: npm install # Install project dependencies

- name: Build Project

run: npm run build # Build the project

- name: Run Tests

run: npm run test # Run tests to validate changes

****