**Build image:**docker build -t app .

**Tag image for Docker Hub:**docker tag app 23bd1a6796/sampleapp:1.0

**Login to Docker Hub:**docker login

**Push image to Docker Hub:**docker push 23bd1a6796/sampleapp:1.0

**Run container (new):**docker run --name sampleapp-container 23bd1a6796/sampleapp:1.0

**See running containers:**docker ps

**See all containers (including stopped):**docker ps -a

**Start existing container**:docker start sampleapp-container

**Start and attach to see output**:docker start -ai sampleapp-container

**See container logs**:docker logs sampleapp-container

**Remove container:**docker rm sampleapp-container

**Remove all stopped containers (optional):**docker container prune

**Pull MongoDB image:**

docker pull mongo:latest

**Verify downloaded images:**

docker images

**Run container (new):**

docker run --name mymongo -p 27017:27017 -d mongo:latest

**See running containers:**

docker ps

**See all containers (including stopped):**

docker ps -a

**Connect to MongoDB shell inside container:**

docker exec -it mymongo mongo

**Stop container:**

docker stop mymongo

**Start existing container:**

docker start mymongo

**Start and attach to see output:**

docker start -ai mymongo

**See container logs:**

docker logs mymongo

**Remove container:**

docker rm mymongo

**Remove all stopped containers (optional):**

docker container prune

**1 Tag your local image for Docker Hub**

docker tag mongo:latest <your\_dockerhub\_username>/mongoapp:1

*Replace <your\_dockerhub\_username> with your Docker Hub username.*

**2 Login to Docker Hub**

docker login

*Enter your Docker Hub username and password to authenticate.*

**3 Push the image to Docker Hub**

docker push <your\_dockerhub\_username>/mongoapp:1

**1 Build images defined in docker-compose.yml**

docker-compose build

*Build all services’ images defined in the Compose file.*

**2 Start services (new containers)**

docker-compose up

*Start and attach to all containers as per docker-compose.yml.*

**3 Start services in background (detached mode)**

docker-compose up -d

*Run all containers in background without attaching.*

**4 See running containers**

docker-compose ps

*List containers started by Compose.*

**5 View logs of containers**

docker-compose logs

*Shows combined logs of all services.*

**6 Follow logs in real-time**

docker-compose logs -f

*Keep streaming logs as they appear.*

**7 Stop and remove containers, networks, volumes**

docker-compose down

*Stops and deletes all containers and resources created by Compose.*

**8 Stop services without removing resources**

docker-compose stop

*Stops containers but keeps volumes and networks intact.*

**9 Restart services**

docker-compose restart

*Restarts running services.*

**10 Remove stopped containers only**

docker-compose rm

1. **Create a new folder compose-lab**

Inside it, create a file docker-compose.yml with the following content:

1.Created a “docker-compose.yml” file and

\* docker-compose -f docker-compose.yml up -d

2.Search in browser

\* localhost:8080

3.Stop the container

\* docker-compose -f docker-compose.yml down

Here’s a basic example of a Docker Compose file that runs WordPress and MySQL together:

version: '3.8' # Docker Compose file format version

services:

wordpress: # WordPress service

image: wordpress:latest

ports:

- "8080:80" # Map port 80 of the container to port 8080 of the host

environment:

WORDPRESS\_DB\_HOST: db:3306 # Database host

WORDPRESS\_DB\_USER: wordpress

WORDPRESS\_DB\_PASSWORD: wordpress

WORDPRESS\_DB\_NAME: wordpress

depends\_on:

- db # Ensures the db service starts first

db: # MySQL service

image: mysql:5.7

environment:

MYSQL\_ROOT\_PASSWORD: rootpassword

MYSQL\_DATABASE: wordpress

MYSQL\_USER: wordpress

MYSQL\_PASSWORD: wordpress

**1.Define and run multiple interdependent services**

**Task:**

1. **Create a new folder compose-lab**

Inside it, create a file docker-compose.yml with the following content:

version: "3.9"

services:

web:

image: nginx:latest

ports:

- "8080:80"

db:

image: postgres:15

environment:

POSTGRES\_USER: demo

POSTGRES\_PASSWORD: demo

POSTGRES\_DB: demo\_db

**2.Write and interpret docker-compose.yml files**

**Task:**

1. **Modify docker-compose.yml to add a Redis cache:**

redis:

image: redis:alpine

**II. Add a depends\_on so web waits for Redis:**

web:

image: nginx:latest

ports:

- "8080:80"

depends\_on:

- redis

**4. Networking and persistent storage**

📂 compose-lab/docker-compose.yml

version: "3.9"

networks:

app-net:

volumes:

db-data:

services:

web:

image: nginx:latest

ports:

- "8080:80"

networks:

- app-net

depends\_on:

- db

db:

image: postgres:15

environment:

POSTGRES\_USER: demo

POSTGRES\_PASSWORD: demo

POSTGRES\_DB: demo\_db

volumes:

- db-data:/var/lib/postgresql/data

networks:

- app-net

Step I – Create Flask app (app.py)

from flask import Flask

app = Flask(\_\_name\_\_)

@app.route("/")

def home():

return "Hello from Flask + Docker!"

if \_\_name\_\_ == "\_\_main\_\_":

app.run(host="0.0.0.0", port=5000)

Step II – Create Dockerfile

FROM python:3.10-slim

WORKDIR /app

COPY app.py /app/

RUN pip install flask

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

Step III – Update docker-compose.yml

version: "3.9"

services:

web:

build: . # build from Dockerfile in current folder

ports:

- "5000:5000" # expose Flask app

depends\_on:

- db

db:

image: postgres:15

environment:

POSTGRES\_USER: demo

POSTGRES\_PASSWORD: demo

POSTGRES\_DB: demo\_db

**1 Initialize a new Git repository**

git init

*Creates a new Git repository in the current folder.*

**2 Check repository status**

git status

*Shows changed, staged, and untracked files.*

**3 Add files to staging area**

git add <filename>

*Stage a specific file for commit.*

git add .

*Stage all changed files for commit.*

**4 Commit changes**

git commit -m "Your commit message"

*Save staged changes with a descriptive message.*

**5 Check commit history**

git log

*Shows all commits in the repository.*

**6 Connect to a remote repository (GitHub/GitLab)**

git remote add origin <repository\_url>

*Links local repo to remote repository.*

**7 Verify remote connection**

git remote -v

*Shows the remote repository URLs.*

**8 Push changes to remote repository**

git push -u origin main

*Uploads local commits to the remote branch (main).*

If your branch is master, replace main with master.

**9 Pull changes from remote repository**

git pull origin main

*Fetches and merges changes from remote to local repo.*

**10 Clone a remote repository**

git clone <repository\_url>

*Download a remote repository to your local system.*

**11 Create a new branch**

git branch <branch\_name>

*Creates a new branch locally.*

**12 Switch to another branch**

git checkout <branch\_name>

*Switch to a different branch.*

**13 Create and switch to a new branch in one step**

git checkout -b <branch\_name>

**14 Merge another branch into current branch**

git merge <branch\_name>

*Merge changes from another branch into current branch.*

**15 Delete a local branch**

git branch -d <branch\_name>

**16 View differences**

git diff

*Shows changes that are not staged.*

git diff --staged

*Shows changes that are staged for commit.*

**17 Undo last commit (soft, keep changes staged)**

git reset --soft HEAD~1

**18 Undo last commit (hard, discard changes)**

git reset --hard HEAD~1