**Docker Lab Exercises Submission**

**Exercise 1: Getting Started with Containers**

* Run hello-world, nginx, and alpine containers.
* Use docker ps and docker ps -a to inspect states.
* Explore docker run flags: --rm, -it, -d, -p.
* Use docker exec and docker logs.

**Commands Used**

* docker version
* docker compose version
* docker run hello-world
* docker run nginx
* docker run alpine
* docker ps
* docker images
* docker run -d --name mynginx -p 8080:80 nginx
* # uname -a
* # ls / and exit
* docker run --rm -it --name myapline alpine sh

**Screenshots / Output Evidence**A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A black screen with white text

AI-generated content may be incorrect.

A black screen with white text

AI-generated content may be incorrect.

**Exercise 2: Working with Container State**

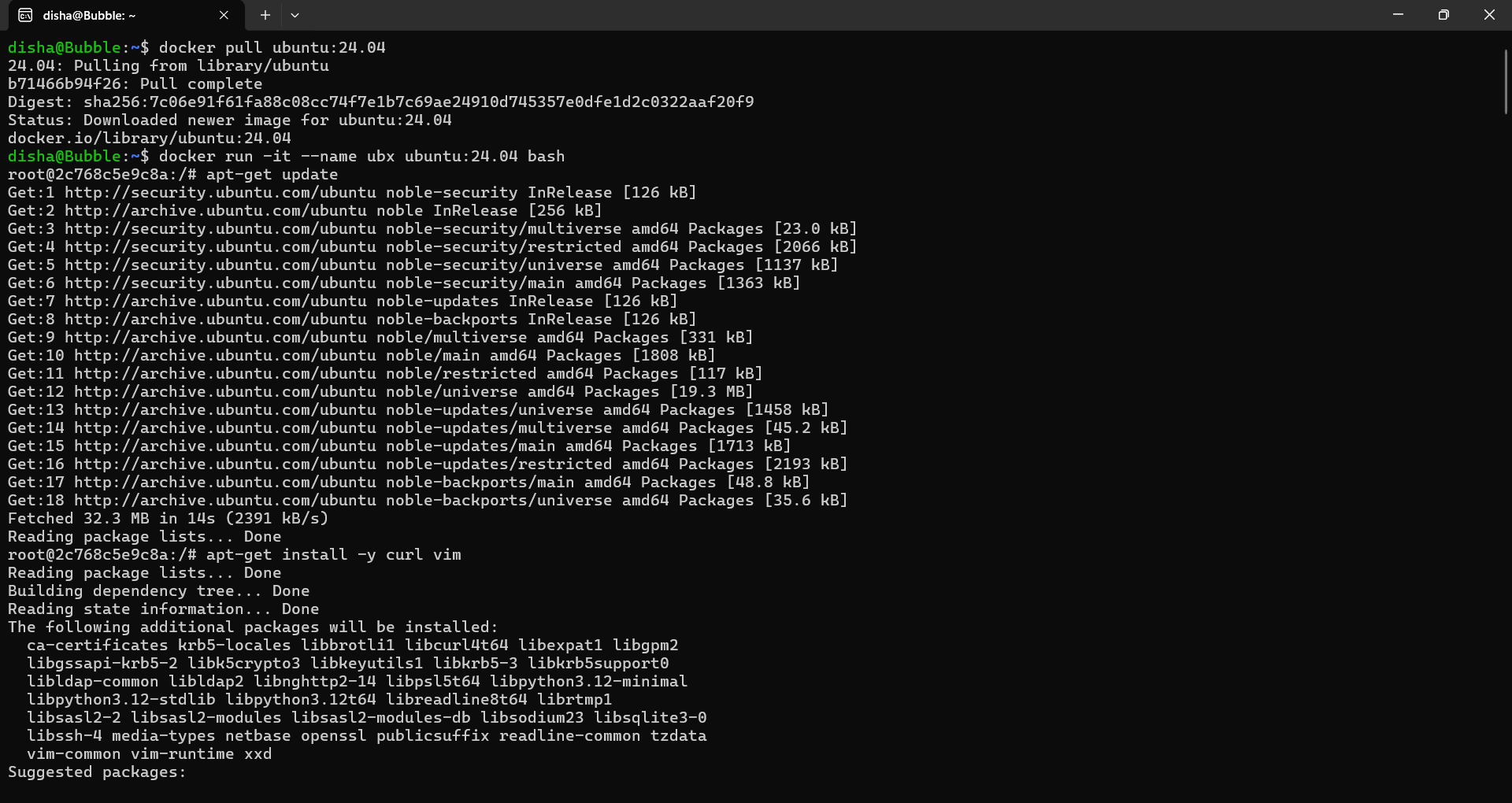
Run an Ubuntu container, install curl and vim.

* Exit and commit the image as ubuntu-tools.
* Run a new container from the committed image.
* Tag the image and list it with docker images.

**Commands Used**

* docker pull ubuntu:24.04
* docker run -it --name ubx ubuntu:24.04 bash
* docker commit -m “Add curl and vim” -a “Disha Bundela” ubx ubuntu-tools:latest
* docker images
* docker run -t –name tools ubuntu-tools:latest bash
* # curl –version
* #vim version | head -n 3
* docker tag ubuntu-tools:v1
* docker images

## Screenshots / Output Evidence



A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**Exercise 3: Build Custom Images Using Dockerfile**

* Create a simple Node or Python web server.
* Write a Dockerfile to copy the code and expose a port.
* Add metadata using LABEL and set CMD or ENTRYPOINT.
* Build and run the image. Test with curl.

**Commands Used**

* cd /mnt/c/Users/Disha/oneDrive/Documents/DOCKER-ASSIGNMENT
* mkdir excercise03
* cd excercise03
* docker build -t flask-app:v1
* docker run -d --name myflask -p 5000-5000 flask-app:v1
* curl <http://localhost:5000>
* docker ps
* docker logs myflask
* docker exec -myflask sh
* docker stop myflask
* docker images

**Screenshots / Output Evidence**

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**Exercise 4: Sharing Images**

* Create a Docker Hub account.
* Tag your custom image.
* Push it to Docker Hub.
* Pull it from Docker Hub.

**Commands Used**

* docker login
* docker images
* docker push disha0509/flask-app:v1
* docker pull disha0509/flask-app:v1
* docker run -d -p 5000:5000 disha0509/flask-app:v1
* curl <http://localhost:5000>
* docker stop disha0509/flask-app:v1

**Screenshots / Output Evidence**

A screenshot of a computer program

AI-generated content may be incorrect.

**Exercise 5: Data Persistence with Volumes**

* Launch a busybox container with a named volume.
* Insert sample data.
* Stop, remove, and relaunch to verify persistence.
* Try bind mount using -v $(pwd)/data:/data.

**Commands Used**

* mkdir excercise05
* cd excercise05
* docker volume create mydata
* docker run -it –-name volcontainer -v mydata:/data busybox sh
* # echo “Hello Docker Volumes!” > /data/hello.txt
* # cat /data/hello/txt and exit
* docker run -it –-name volcontainer2 -v mydata:/data busybox sh
* # cat /data/hello.txt and exit
* mkdir data
* docker run -it --name bindcontainer -v ${PWD}\data:/data busybox sh
* # echo “Hello Bind Mount!” > /data/bind.txt
* # ls /data
* # cat /data/bind.txt and exit
* docker rm volcontainer2 bindcontainer

**Screenshots / Output Evidence**

A screenshot of a computer

AI-generated content may be incorrect.

**Exercise 6: Container Networking Basics**

* Start an nginx container and a busybox container.
* Create a user-defined bridge network.
* Attach both containers to the network.
* From busybox, use wget or curl to access nginx.

**Commands Used**

* mkdir excercise06
* cd excercise06
* docker network create mynet
* docker network ls
* docker run -d --name web --network mynet nginx
* docker run -d --name client --network mynet busybox sh
* # wget -qO-http://web

**Screenshots / Output Evidence**

A screenshot of a computer program

AI-generated content may be incorrect.

**Exercise 7: Building a Two-Tier App**

* Manually run a Python/Flask app container and a Postgres container.
* Use environment variables to configure the connection.
* Verify the app connects to the DB and serves content.

**Commands Used**

* mkdir excercise07
* cd exercise07
* mkdir flask\_postgres
* cd flask\_postgres
* docker build -t flask-app-dv:v
* docker images
* docker network create appnet
* docker network ls
* docker run -d --name pgdb \ --network appnet \ -e POSTGRES\_USER=user \ -e POSTGRES\_PASSWORD=’Anjali#123’ \ -e POSTGRES\_DB=dockerdb \ postgres:13
* docker ps
* docker exec -it pgdb psql -U user -d dockerdb psql
* # \l and \du and \q
* docker run -d --name flaskweb \ --network appnet \ -e DB\_HOST=pgdb \ -e DB\_NAME=dockerdb \ -e DB\_USER=user \ -e DB\_PASS=’Anjali#123’ \ -p 5000:5000 \ flask-app-db:v
* docker ps
* docker logs flaskweb
* curl <http://localhost:5000>
* docker stop flaskweb pgdb
* docker network rm appnet

**Screenshots / Output Evidence**

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**Exercise 8: Docker Compose Basics**

* Write a docker-compose.yml for FastAPI + Postgres.
* Use docker compose up, inspect logs and containers.
* Add health checks and environment variables.
* Use depends\_on, restart policies.

**Commands Used**

* mkdir excercise08
* cd excercise08
* mkdir fastapi\_postgres
* cd fastapi\_postgres
* docker compose up --build -d
* docker compose ps
* docker compose logs -f
* curl <http://localhost:8000/>
* curl <http://localhost:8000/users>
* docker compose down

**Screenshots / Output Evidence**

A screenshot of a computer

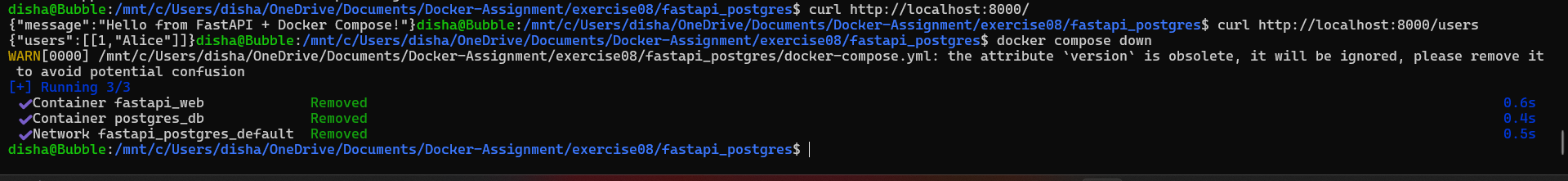
AI-generated content may be incorrect.

A computer screen with white text

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.



A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**Exercise 9: Healthchecks and Best Practices**

* Add HEALTHCHECK instruction to your Dockerfile.
* Use ENTRYPOINT vs CMD appropriately.
* Minimize layers and image size (e.g., using alpine).
* Inspect container health via docker inspect.

**Commands Used**

* mkdir excercise09
* cd excercise09
* mkdir fastapi\_health
* cd fastapi\_heath
* docker build -t fastapi-health:v1 .
* docker images
* docker run -d --name fastapi\_prod -p 8000:8000 fastapi-health:v1
* docker ps
* docker inspect --format=’{{json .State.Health}}’ fastapi\_prod
* curl <http://localhost:8000/>
* docker stop fastapi\_prod
* docker rm fastapi\_prod
* docker rmi fastapi-health:v1

**Screenshots / Output Evidence**

A screenshot of a computer program

AI-generated content may be incorrect.

A computer screen shot of a black screen

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**Exercise 10: Debugging, Cleanup & Troubleshooting**

* Run containers with bad commands or missing ports.
* Clean up unused images, containers, volumes with:

**Commands Used**

* docker run --name bad-container ubuntu invalid-command
* docker ps -a
* docker logs bad-container
* docker run -d --name web-server -p 8080:80 nginx
* docker ps -a
* docker stop <cont\_id>
* docker rm <cont\_id>

**Screenshots / Output Evidence**

A screenshot of a computer program

AI-generated content may be incorrect.

A screen shot of a computer

AI-generated content may be incorrect.