NucleusTeq

🧩 Docker Lab Exercises

📄 Submission Requirements

You must submit a **single PDF file** that includes the following:

- **Commands Used**
  A list of all Docker commands you executed, presented in the correct order.

- **Output Evidence**
  Provide the following for each command:

  - **Screenshots** of your terminal clearly showing both the command and its output.

  - *(Optional)* Copy-pasted output directly below each command for additional clarity.

- **Short Answer**
  Write 2–3 sentences in your own words comparing **named volumes** and **bind mounts**, including when and why you might use each.

🔹 Exercise 1: Getting Started with Containers

Objective: Learn how to run and inspect 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`.

Checkpoint:
What happens if a container doesn't run in detached mode? What if ports aren't mapped?

🔹 Exercise 2: Working with Container State

Objective: Modify containers and commit custom images.

- 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`.

🔹 Exercise 3: Build Custom Images Using Dockerfile

Objective: Write Dockerfiles and build your own image.

- 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`.

🔹 Exercise 4: Sharing Images

Objective: Push images to Docker Hub.

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

Reflection:
Why is image tagging important, and what sort of tagging strategy can we use?

🔹 Exercise 5: Data Persistence with Volumes

Objective: Use volumes to persist container data.

- 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`.

Compare:
Named volume vs bind mount. Pros/cons?

🔹 Exercise 6: Container Networking Basics

Objective: Set up communication between containers.

- 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.

Explore:
View IPs with `docker inspect`. Try without a custom network—what's different?

🔹 Exercise 7: Building a Two-Tier App

Objective: Deploy a small web + DB stack without Compose.

- 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.

Extension:
Add a volume to persist the DB data.

🔹 Exercise 8: Docker Compose Basics

Objective: Use Compose to simplify multi-container apps.

- 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.

🔹 Exercise 9: Healthchecks and Best Practices

Objective: Make robust, production-like Dockerfiles.

- 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`.

🔹 Exercise 10: Debugging, Cleanup & Troubleshooting

Objective: Learn to manage resources and solve issues.

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

🔹 Capstone Project: Dockerizing FastAPI + Postgres

Objective: Dockerize your existing FastAPI application and connect it to a Postgres/mysql database using Docker Compose. Create a `Dockerfile` for FastAPI, configure a Postgres/Mysql container with necessary environment variables, and set up networking in `docker-compose.yml` to enable communication between the two services. Test the setup by running both containers and ensuring the FastAPI app interacts correctly with Postgres/Mysql.