Assignment 1: Your First Docker Commands

What you'll learn: Basic Docker commands - the bread and butter stuff

What to do:

1. Download the latest nginx image from Docker Hub
2. Start an nginx container in the background on port 8080
3. Check what containers are running
4. Stop that container
5. Delete the container
6. See what images you have on your machine

Show me: Your command history and some screenshots proving you did it

---

Assignment 2: Jump Inside a Container

What you'll learn: How to work inside a running container

What to do:

1. Start an Ubuntu container and jump into it with bash
2. Once you're inside:
   - Update the package list
   - Install curl and vim
   - Create a file at `/tmp/test.txt` with whatever content you want
3. Exit and restart the container
4. Check if your file is still there (spoiler: it won't be!)
5. Explain why your changes disappeared

Show me: Your explanation of why containers forget everything when you restart them

---

Assignment 3: Build Your First Docker Image

What you'll learn: How to write a Dockerfile from scratch

What to do:

1. Write a Dockerfile that:

    - Starts with Ubuntu 20.04
    - Updates all packages
    - Installs Python 3 and pip
    - Sets /app as the working folder
    - Copies a simple Python script that says "Hello Docker!"
    - Runs that script automatically when the container starts
2. Build your image and call it `my-python-app:v1`
3. Run a container from your new image

Show me: Your Dockerfile, the build output, and proof it's running

---

Assignment 4: Make Smaller, Smarter Images

What you'll learn: Multi-stage builds to create tiny images

What to do:

1. Create a Dockerfile for a Go app that uses two stages:
    - First stage: Build your Go program
    - Second stage: Copy just the final program to a tiny Alpine image
2. Build it both ways (single-stage and multi-stage)
3. Compare how big the images are
4. Explain why multi-stage builds are awesome

Show me: Your Dockerfile, size comparison, and your thoughts on why this matters

---

Assignment 5: Keep Your Data Safe with Volumes

What you'll learn: How to make sure your data doesn't disappear

What to do:

1. Create a volume named `mysql-data`
2. Run a MySQL container that uses this volume
3. Create a database with a table and some data in it
4. Stop and completely remove the container
5. Start a fresh MySQL container using the same volume
6. Check if your data is still there (it should be!)

Show me: All your commands and proof that the data survived

Assignment 6: Connect Your Local Files to Containers

What you'll learn: Bind mounts for live development

What to do:

1. Make a folder on your computer with an index.html file
2. Run nginx and connect your local folder to the container's web folder
3. Edit your index.html file on your computer
4. Refresh your browser - see the changes without restarting anything!
5. Explain the difference between volumes and bind mounts

Show me: Your setup and an explanation of when to use each

Assignment 7: Let Containers Talk to Each Other

What you'll learn: Docker networking basics

What to do:

1. Create a network called `my-network`
2. Start two containers (nginx and alpine) on this network
3. From inside the alpine container, ping the nginx container by name
4. Look at what's connected to your network
5. Explain how containers can find each other by name

Show me: Your network commands and proof that containers can communicate

Assignment 8: Build a Real Two-Part App

What you'll learn: Connecting multiple containers to work together

What to do:

1. Set up a network
2. Start a MySQL container on that network
3. Start a WordPress container on the same network
4. Make WordPress talk to MySQL

5. Open WordPress in your browser and set it up

Show me: A working WordPress site with its database connection

---

Assignment 9: Fix This Messy Dockerfile

What you'll learn: Writing clean, efficient Dockerfiles

What to do:

1. Look at this terrible Dockerfile:

```
FROM ubuntu
RUN apt-get update
RUN apt-get install -y python3
RUN apt-get install -y python3-pip
RUN apt-get install -y curl
RUN apt-get install -y git
COPY . /app
RUN pip3 install flask
CMD python3 /app/app.py
```

2. Rewrite it properly:
    - Put multiple RUN commands together
    - Use a specific version (not just "ubuntu")
    - Clean up after installing packages
    - Use WORKDIR instead of just /app everywhere
    - Add some useful labels
    - Make it cache-friendly
3. Build both and compare the sizes

Show me: Your improved Dockerfile and explain what you fixed

Assignment 10: Your First Docker Compose File

What you'll learn: Managing multiple containers easily

What to do:

1. Write a docker-compose.yml with:

    - An nginx web server on port 80
    - A PostgreSQL database
    - A volume so the database doesn't lose data
    - A network connecting everything
2. Start it all with `docker-compose up`
3. Clean everything up with `docker-compose down`

Show me: Your compose file and screenshots of it running

Assignment 11: Handle Passwords and Settings

What you'll learn: Working with environment variables securely

What to do:

1. Make a Dockerfile that uses environment variables
2. Build an image that prints out environment variables
3. Run it three different ways:
    - Pass variables directly with the -e flag
    - Use a .env file
    - Define them in docker-compose.yml
4. Show all three methods working

Show me: Your Dockerfile, commands, and output from each method

---

Assignment 12: Check if Containers Are Healthy

What you'll learn: Built-in health monitoring

What to do:

1. Write a Dockerfile with a HEALTHCHECK
2. The check should make sure a web service is responding
3. Run the container and watch its health status
4. Show what happens when the service fails the health check

Show me: Your Dockerfile with the health check and the monitoring output

---

Assignment 13: SSH Into Containers (Carefully!)

What you'll learn: Remote access to containers

What to do:

1. Create a Dockerfile that:
   - Installs SSH server
   - Sets up SSH to allow passwords
   - Creates a user with a password
   - Opens port 22
2. Build and run it
3. SSH into your running container
4. Talk about why SSH in containers can be risky

Show me: Your Dockerfile, proof of SSH working, and your security thoughts

---

Assignment 14: Control Container Resources

What you'll learn: Setting limits so containers don't hog everything

What to do:

1. Run a container with these limits:
   - Max 512MB of memory
   - Half a CPU core
   - At least 100MB memory reserved
2. Watch it with `docker stats`
3. Make a docker-compose.yml with limits for several services
4. Show what happens when a container hits its limits

Show me: Your commands and monitoring screenshots

Assignment 15: Share Your Images Online

What you'll learn: Using Docker Hub to share images

What to do:

1. Create a custom image (can be simple)

2. Tag it properly for Docker Hub
3. Push it to Docker Hub (make a free account first)
4. Delete your local copy
5. Pull it back down from Docker Hub
6. Document everything you did

Show me: Step-by-step documentation and your Docker Hub image link

---

Assignment 16: Fix Broken Containers

What you'll learn: Debugging when things go wrong

What to do:

1. Make a container that fails on purpose
2. Use `docker logs` to see what went wrong
3. Use `docker exec` to poke around inside a running container
4. Use `docker inspect` to see all the container details
5. Fix the problems and write up what you did

Show me: The problem, how you found it, and how you fixed it

Assignment 17: Build a Complete App Stack

What you'll learn: Orchestrating multiple services

What to do: Create a docker-compose.yml that runs:

1. A Node.js web app
2. Redis for caching
3. MongoDB for data
4. Nginx as a reverse proxy
5. Custom networks connecting them
6. Volumes for keeping data
7. Environment variables for config
8. Health checks for everything

Show me: Your complete compose file with notes explaining each part

---

Assignment 18: Try Different Network Types

What you'll learn: When to use which network mode

What to do:

1. Test containers with:
    - Bridge network (the normal one)
    - Host network (shares your computer's network)
    - None network (totally isolated)
2. Document what's different:
    - How isolated are they?
    - Do you need port mapping?
    - Which is faster?
3. Explain when you'd use each type

Show me: A comparison table and real-world examples

Assignment 19: Make Flexible Build Configs

What you'll learn: Using build arguments for different builds

What to do:

1. Write a Dockerfile that accepts build arguments
2. Use arguments to:
    - Pick different base images
    - Pass in config values at build time
    - Set the app version
3. Build the same Dockerfile multiple ways with different arguments
4. Add build arguments to a docker-compose.yml too

Show me: Your Dockerfile with ARG instructions and different builds

---

Assignment 20: Deploy a Full Application

What you'll learn: Everything together in one project!

What to do: Build a complete three-tier app:

1. **Frontend**: React or Angular app
2. **Backend**: An API (Node.js, Python, or Java - your choice)
3. **Database**: PostgreSQL or MongoDB

You need:

- Dockerfiles for each part
- A docker-compose.yml that runs everything
- Volumes so data sticks around
- Networks connecting the pieces
- Environment variables for settings
- Health checks
- Resource limits
- Full documentation
- Step-by-step setup instructions

Show me: Everything - all files, a README, an architecture diagram, and proof it works (screenshots or video)

What to Submit

For every assignment, give me:

1. All your files (Dockerfiles, docker-compose.yml, etc.)
2. The commands you ran (or a script)
3. Screenshots showing it works
4. Write-up explaining what you did
5. Any problems you hit and how you solved them