Assignment №2

Almabekuly Aibek

Date: 13.10.2024

Almaty, 2024

Exercise 1: Installing Docker

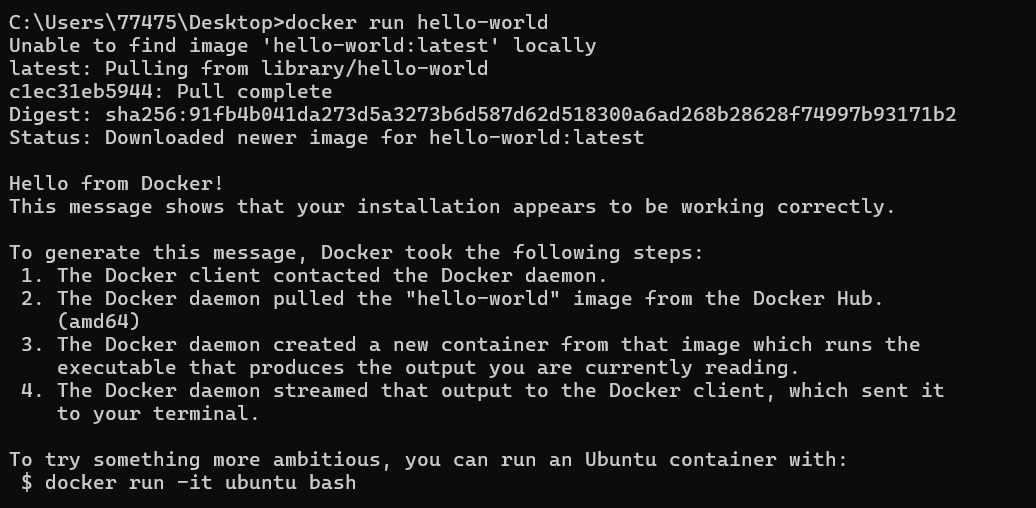
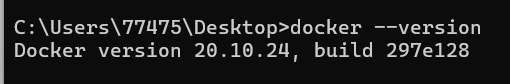
1. Objective: Install Docker on your local machine.
2. Steps:
   * Follow the installation guide for Docker from the official website, choosing the appropriate version for your operating system (Windows, macOS, or Linux).
   * After installation, verify that Docker is running by executing the command docker

--version in your terminal or command prompt.

* + Run the command docker run hello-world to verify that Docker is set up correctly.

1. Questions:
   * What are the key components of Docker (e.g., Docker Engine, Docker CLI)?
   * How does Docker compare to traditional virtual machines?
   * What was the output of the docker run hello-world command, and what does it signify?

Result:



Key components of Docker:

* Docker Engine, Docker CLI, Docker Daemon, Docker HUB Docker vs. traditional virtual machines:
* Docker uses containerization, sharing the host OS kernel, which makes it more lightweight and faster than traditional VMs that require full OS installations.

Output of docker run hello-world:

* The output confirms Docker is installed and working correctly by printing a message. It indicates the container successfully ran and connected to Docker Engine.

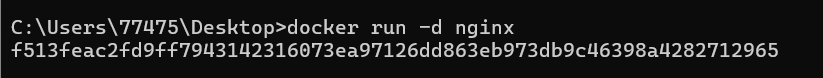
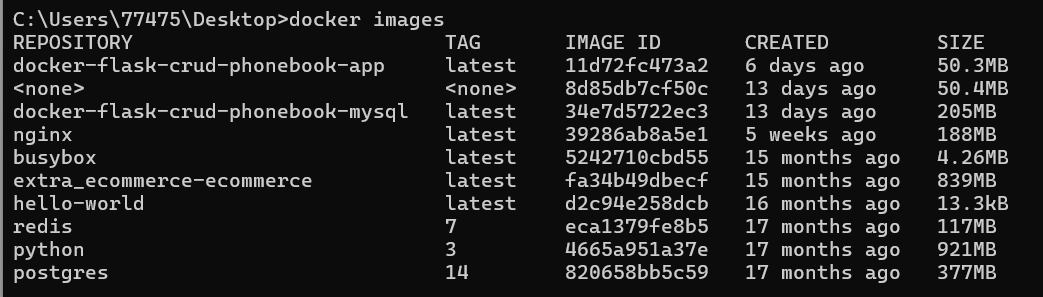
Exercise 2: Basic Docker Commands

1. Objective: Familiarize yourself with basic Docker commands.
2. Steps:
   * Pull an official Docker image from Docker Hub (e.g., nginx or ubuntu) using the command docker pull <image-name>.
   * List all Docker images on your system using docker images.
   * Run a container from the pulled image using docker run -d <image-name>.
   * List all running containers using docker ps and stop a container using docker stop

<container-id>.

1. Questions:
   * What is the difference between docker pull and docker run?
   * How do you find the details of a running container, such as its ID and status?
   * What happens to a container after it is stopped? Can it be restarted?

Result:





Difference between docker pull and docker run:

docker pull downloads an image from a registry.

docker run creates and starts a container from an image (it pulls the image if not already present).

How to find details of a running container (ID and status):

Use docker ps to see the container ID, status, and other details of running containers.

What happens after a container is stopped:

The container remains in a stopped state and can be restarted with docker start <container\_id>.

Exercise 3: Working with Docker Containers

1. Objective: Learn how to manage Docker containers.
2. Steps:
   * Start a new container from the nginx image and map port 8080 on your host to port 80 in the container using docker run -d -p 8080:80 nginx.
   * Access the Nginx web server running in the container by navigating to http://localhost:8080 in your web browser.
   * Explore the container's file system by accessing its shell using docker exec -it

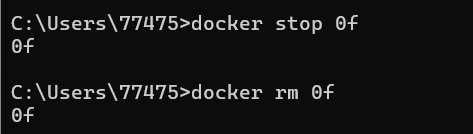
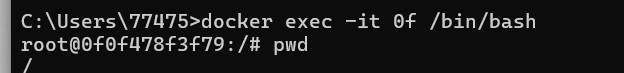
<container-id> /bin/bash.

* + Stop and remove the container using docker stop <container-id> and docker rm

<container-id>.

1. Questions:
   * How does port mapping work in Docker, and why is it important?
   * What is the purpose of the docker exec command?
   * How do you ensure that a stopped container does not consume system resources?

Result:





Port mapping in Docker:

Port mapping links a port on your computer to a port inside the container, allowing outside access to the services running inside the container. For example, with -p 8080:80, port 8080 on the host is mapped to port 80 inside the container.

Purpose of docker exec:

This command allows you to run another command inside a running container, like opening a terminal or running a script inside the container.

Stopping a container from using resources:

After stopping a container, you can delete it with docker rm to ensure it doesn’t take up any resources on the system.

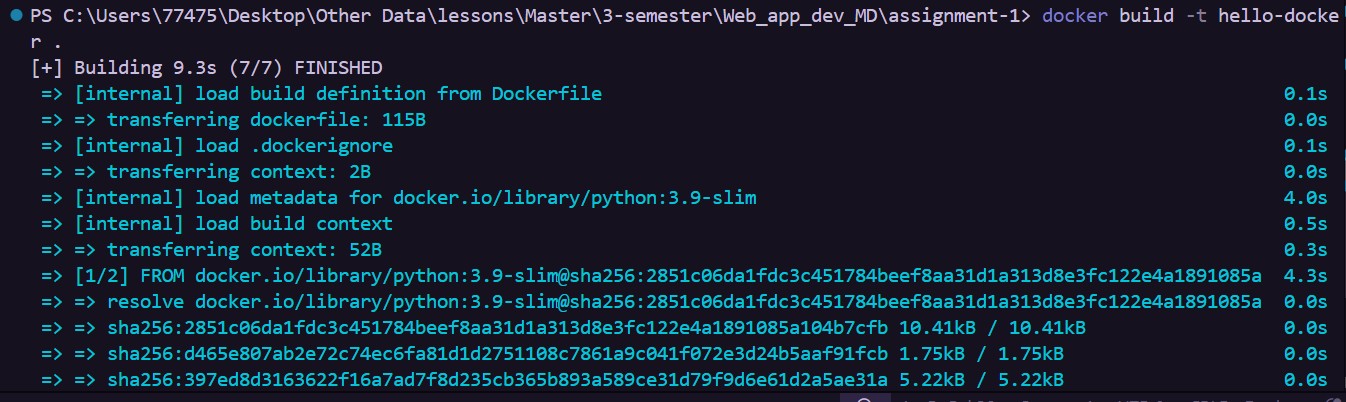
Dockerfile

Exercise 1: Creating a Simple Dockerfile

1. Objective: Write a Dockerfile to containerize a basic application.
2. Steps:
   * Create a new directory for your project and navigate into it.
   * Create a simple Python script (e.g., app.py) that prints "Hello, Docker!" to the console.
   * Write a Dockerfile that:
     + Uses the official Python image as the base image.
     + Copies app.py into the container.
     + Sets app.py as the entry point for the container.
   * Build the Docker image using docker build -t hello-docker ..
   * Run the container using docker run hello-docker.
3. Questions:
   * What is the purpose of the FROM instruction in a Dockerfile?
   * How does the COPY instruction work in Dockerfile?
   * What is the difference between CMD and ENTRYPOINT in Dockerfile?

Result:







Purpose of the FROM instruction:

The FROM instruction specifies the base image to use for building the container. It sets up the foundation upon which the rest of the image is built.

How the COPY instruction works:

The COPY instruction copies files or directories from your local machine into the container's filesystem. For example, COPY app.py /app.py copies app.py from your project directory into the container.

Difference between CMD and ENTRYPOINT:

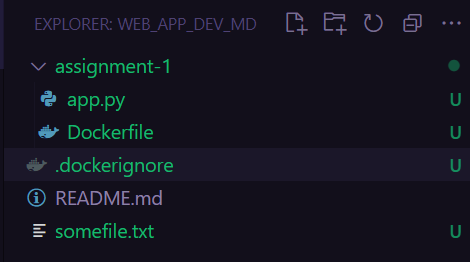
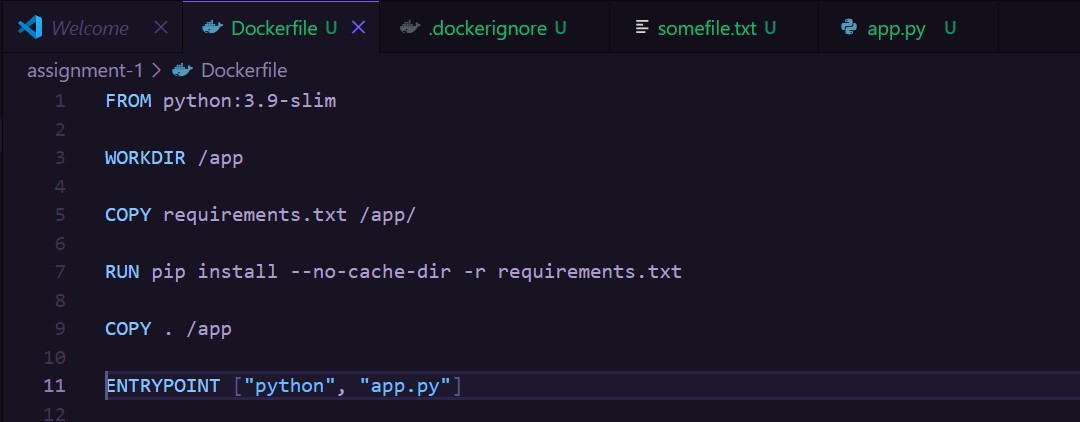
CMD defines the default command that runs when the container starts, but it can be overridden by passing a different command at runtime.

ENTRY POINT defines a fixed command that always runs when the container starts. It can't be overridden but can accept additional arguments.

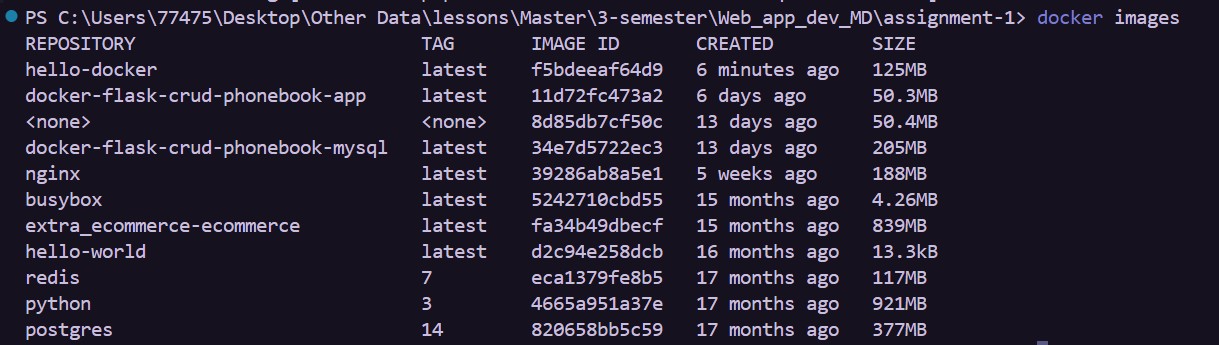
Exercise 2: Optimizing Dockerfile with Layers and Caching

1. Objective: Learn how to optimize a Dockerfile for smaller image sizes and faster builds.
2. Steps:
   * Modify the Dockerfile created in the previous exercise to:
     + Separate the installation of Python dependencies (if any) from the copying of application code.
     + Use a .dockerignore file to exclude unnecessary files from the image.
   * Rebuild the Docker image and observe the build process to understand how caching works.
   * Compare the size of the optimized image with the original.
3. Questions:
   * What are Docker layers, and how do they affect image size and build times?
   * How does Docker's build cache work, and how can it speed up the build process?
   * What is the role of the .dockerignore file?

Result:







What are Docker layers, and how do they affect image size and build times?

Docker layers are individual steps in the Dockerfile (like FROM, COPY, RUN). Each layer adds to the final image size. If a layer changes, all subsequent layers are rebuilt, impacting both image size and build time.

How does Docker's build cache work, and how can it speed up the build process?

Docker caches each layer during the build process. If a layer hasn’t changed, Docker uses the cached version instead of rebuilding it. This speeds up builds by only processing the layers that have been modified.

What is the role of the .dockerignore file?

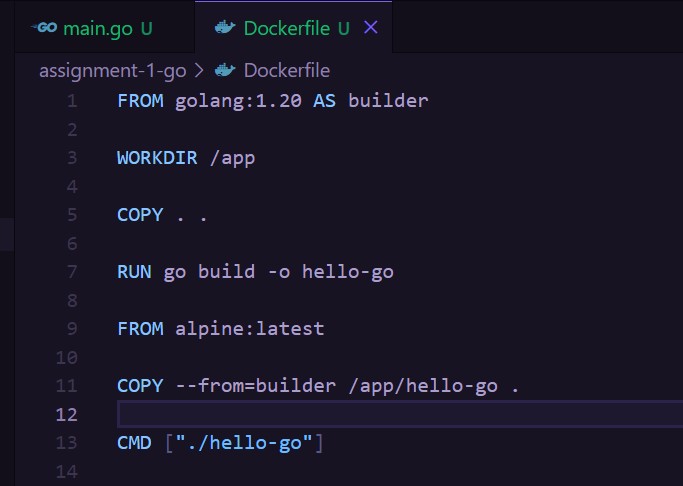
The .dockerignore file excludes specified files and directories from being copied into the image during the build. This helps reduce image size, prevent unnecessary files from being included, and speeds up the build process.

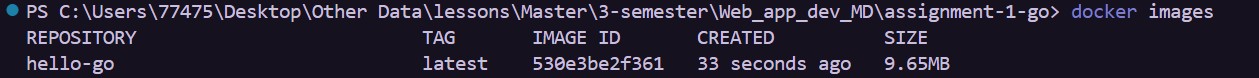
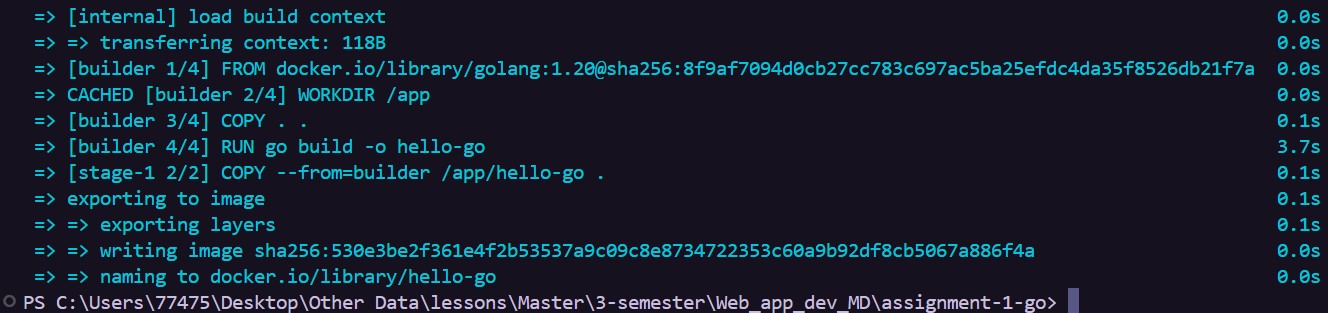
Exercise 3: Multi-Stage Builds

1. Objective: Use multi-stage builds to create leaner Docker images.
2. Steps:
   * Create a new project that involves compiling a simple Go application (e.g., a "Hello, World!" program).
   * Write a Dockerfile that uses multi-stage builds:
     + The first stage should use a Golang image to compile the application.
     + The second stage should use a minimal base image (e.g., alpine) to run the compiled application.
   * Build and run the Docker image, and compare the size of the final image with a single-stage build.
3. Questions:
   * What are the benefits of using multi-stage builds in Docker?
   * How can multi-stage builds help reduce the size of Docker images?
   * What are some scenarios where multi-stage builds are particularly useful?

Result:







Benefits of multi-stage builds:

They keep the Dockerfile organized and separate the build environment from the runtime environment.

Reducing image size:

They allow you to compile in a larger image and copy only the necessary files to a smaller image, removing unnecessary dependencies.

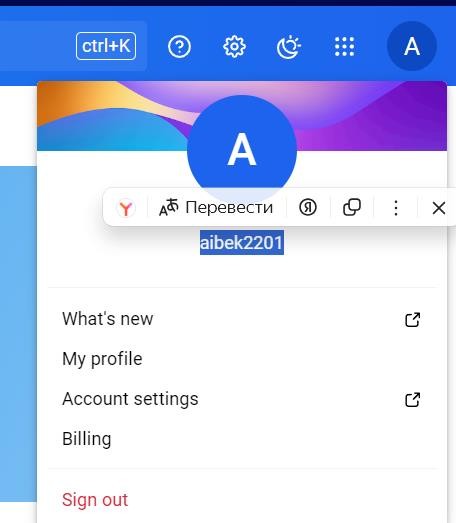
Useful scenarios:

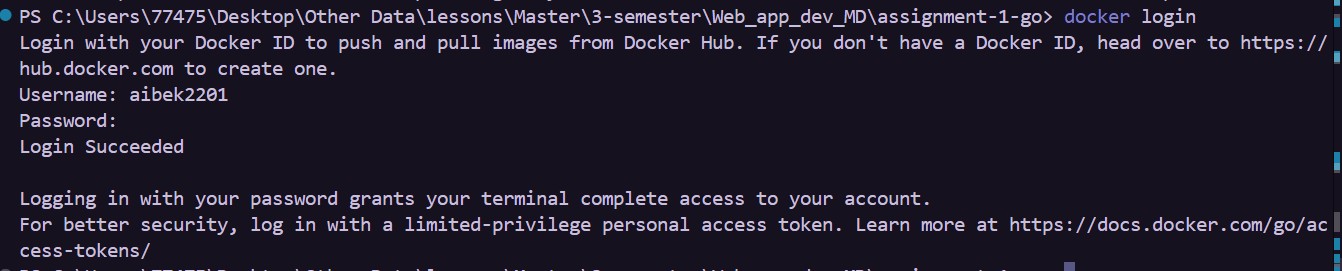
When building apps with heavy dependencies, compiling binaries, or creating lightweight images for deployment.

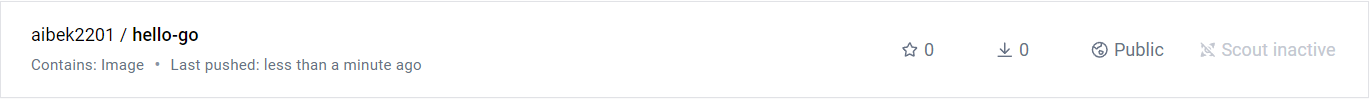
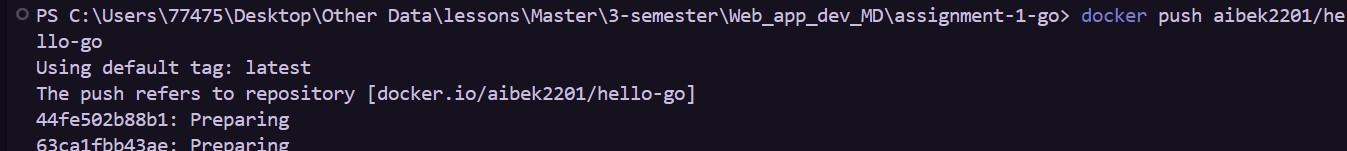
Exercise 4: Pushing Docker Images to Docker Hub

1. Objective: Learn how to share Docker images by pushing them to Docker Hub.
2. Steps:
   * Create an account on Docker Hub.
   * Tag the Docker image you built earlier with your Docker Hub username (e.g., docker tag hello-docker <your-username>/hello-docker).
   * Log in to Docker Hub using docker login.
   * Push the image to Docker Hub using docker push <your-username>/hello-docker.
   * Verify that the image is available on Docker Hub and share it with others.
3. Questions:
   * What is the purpose of Docker Hub in containerization?
   * How do you tag a Docker image for pushing to a remote repository?
   * What steps are involved in pushing an image to Docker Hub?

Result:







Purpose of Docker Hub:

To store and share Docker images.

docker tag <local-image> <your-username>/<repository-name>:<tag>

Steps to push an image to Docker Hub:

Log in with docker login. Tag the image.

Push the image using docker push <your-username>/<repository-name>:<tag>.