Assignment 1, Web Application Development, Akhmetov R.R.

Intro to Containerization: Docker
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.

3. **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?

ramazan — -zsh — 80×24

```
Last login: Sun Jul 21 13:49:29 on console
(base) ramazan@MacBook-Pro-Ramazan ~ % docker --version
Docker version 27.1.1, build 6312585
(base) ramazan@MacBook-Pro-Ramazan ~ %
```

ramazan — -zsh — 80×24

```
(base) ramazan@MacBook-Pro-Ramazan ~ % docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/

(base) ramazan@MacBook-Pro-Ramazan ~ %
```

- What are the key components of Docker (e.g., Docker Engine, Docker CLI)?

Docker Engine is the main engine that runs and manages containers, Docker CLI is the command line that allows you to execute commands to work with Docker. Docker Image - this can be called a kind of template or image based on which containers are created. Docker Container is an environment in which applications are launched with all the necessary components, such as code and libraries.

- How does Docker compare to traditional virtual machines?

Virtual machines provide full virtualization, including emulation of the entire operating system, which is resource-intensive. Docker containers, on the other hand, isolate applications at the process level using the host operating system kernel. This makes them lighter and faster to run than virtual machines.

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

This command starts a test container that prints "Hello from Docker!" This indicates that Docker is installed and configured correctly, and that containers are running successfully.

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>.
3. **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?

```
(base) ramazan@MacBook-Pro-Ramazan ~ % docker pull nginx
Using default tag: latest
latest: Pulling from library/nginx
a2318d6c47ec: Pull complete
095d327c79ae: Pull complete
bbfaa25db775: Pull complete
7bb6fb0cfb2b: Pull complete
0723edc10c17: Pull complete
24b3fdc4d1e3: Pull complete
3122471704d5: Pull complete
Digest: sha256:04ba374043ccd2fc5c593885c0eacddebabd5ca375f9323666f28dfd5a9710e3
Status: Downloaded newer image for nginx:latest
docker.io/library/nginx:latest

What's next:
    View a summary of image vulnerabilities and recommendations → docker scout quickview nginx
(base) ramazan@MacBook-Pro-Ramazan ~ % docker images
REPOSITORY    TAG       IMAGE ID       CREATED         SIZE
nginx         latest    39286ab8a5e1   5 weeks ago     188MB
hello-world   latest    d2c94e258dcb   16 months ago   13.3kB
(base) ramazan@MacBook-Pro-Ramazan ~ % docker run -d nginx
b4df8b9caf0bb47121cee16f044d66ffbbda2c411c2045236229f6e3e4a61f52
(base) ramazan@MacBook-Pro-Ramazan ~ % docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS     NAMES
b4df8b9caf0b   nginx     "/docker-entrypoint.…"   25 seconds ago   Up 24 seconds   80/tcp    goofy_wu
(base) ramazan@MacBook-Pro-Ramazan ~ % docker stop b4df8b9caf0b
b4df8b9caf0b
(base) ramazan@MacBook-Pro-Ramazan ~ %
```

- What is the difference between docker pull and docker run?

The docker pull command downloads an image from a Docker repository to your local computer.
The docker run command uses the image to create and run a container. For example: if the image is not downloaded locally, the command will automatically docker pull.

- How do you find the details of a running container, such as its ID and status?

The docker ps command will show a list of all running containers with their ID, status, name, etc. For more detailed information about a container, you can use docker inspect containerID.

- What happens to a container after it is stopped? Can it be restarted?

After stopping, the container remains in the system in a stopped state. Its data and settings are saved. The container can be restarted with the docker start containerID command.

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>.
3. **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?

```
(base) ramazan@MacBook-Pro-Ramazan ~ % docker run -d -p 8080:80 nginx
02a7d313f8b4464fb1df31a732202071035c48f549b2e29456c0796ef7290b01
(base) ramazan@MacBook-Pro-Ramazan ~ % docker images
REPOSITORY    TAG       IMAGE ID       CREATED         SIZE
nginx         latest    39286ab8a5e1   5 weeks ago     188MB
hello-world   latest    d2c94e258dcb   16 months ago   13.3kB
(base) ramazan@MacBook-Pro-Ramazan ~ % docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED              STATUS              PORTS                   NAMES
02a7d313f8b4   nginx     "/docker-entrypoint.…"   About a minute ago   Up About a minute   0.0.0.0:8080->80/tcp   unruffled_jepsen
(base) ramazan@MacBook-Pro-Ramazan ~ % docker exec -it 02a7d313f8b4 /bin/bash
root@02a7d313f8b4:/# whoami
root
root@02a7d313f8b4:/# close
bash: close: command not found
root@02a7d313f8b4:/# exit
exit

What's next:
    Try Docker Debug for seamless, persistent debugging tools in any container or image → docker debug 02a7d313f8b4
    Learn more at https://docs.docker.com/go/debug-cli/
(base) ramazan@MacBook-Pro-Ramazan ~ % docker stop 02a7d313f8b4
02a7d313f8b4
(base) ramazan@MacBook-Pro-Ramazan ~ % docker rm 02a7d313f8b4
02a7d313f8b4
(base) ramazan@MacBook-Pro-Ramazan ~ %
```

- How does port mapping work in Docker, and why is it important?

Port mapping allows us to forward ports on the host machine to ports inside the container. This allows you to interact with the application running in the container from the outside network. For example, the -p 8080:80 command maps port 8080 on the host to port 80 in the container, where the web server can be running. This will allow you to access the containers from the outside.

- What is the purpose of the docker exec command?

The docker exec command allows you to execute commands inside an already running container. For example, you can connect to the container terminal with the docker exec -it containerID /bin/bash command.

- How do you ensure that a stopped container does not consume system resources?

To completely remove a container and free up resources, use the docker rm containerID command.

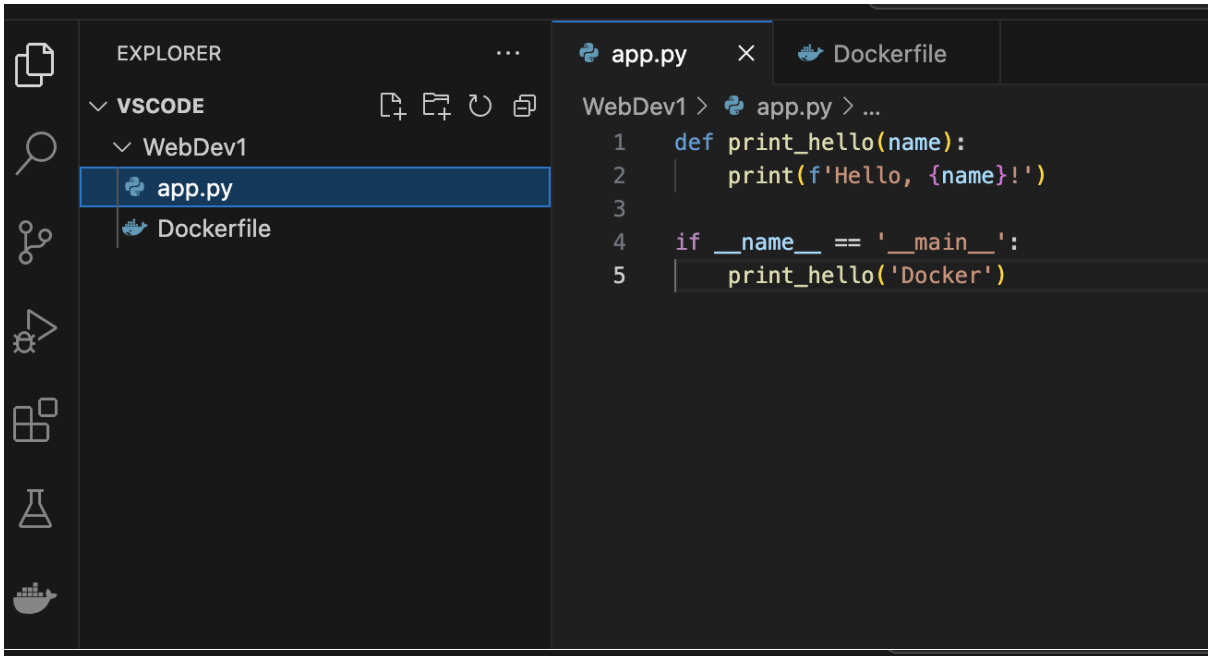
Dockerfile

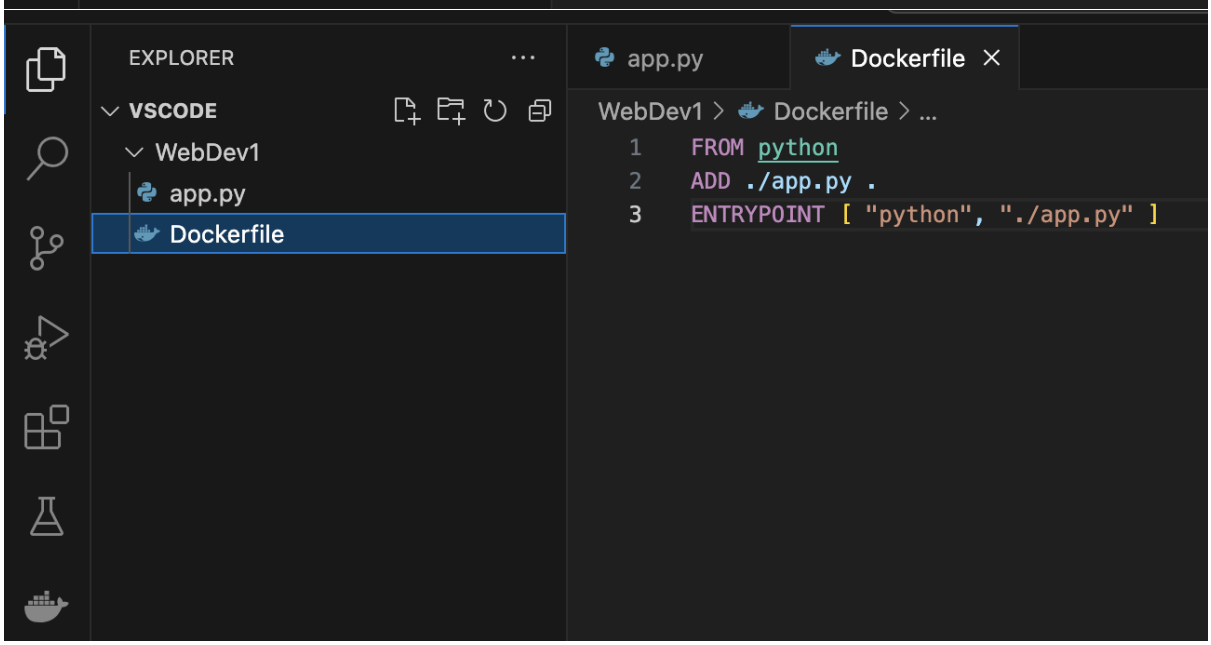
Exercise 1: Creating a Simple Dockerfile

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?





```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS                                                    Python - WebDev1

(base) ramazan@MacBook-Pro-Ramazan WebDev1 % docker build -t hello-docker .
[+] Building 310.0s (8/8) FINISHED                                                                      docker:desktop-linux
 => [internal] load build definition from Dockerfile                                                                     0.0s
 => => transferring dockerfile: 136B                                                                                     0.0s
 => [internal] load metadata for docker.io/library/python:latest                                                         4.8s
 => [auth] library/python:pull token for registry-1.docker.io                                                            0.0s
 => [internal] load .dockerignore                                                                                        0.0s
 => => transferring context: 2B                                                                                          0.0s
 => [internal] load build context                                                                                        0.0s
 => => transferring context: 176B                                                                                        0.0s
 => [1/2] FROM docker.io/library/python:latest@sha256:7859853e7607927aa1d1b1a5a2f9e580ac90c2b66feeb1b77da97fed03b1ccbe   302.1s
 => => resolve docker.io/library/python:latest@sha256:7859853e7607927aa1d1b1a5a2f9e580ac90c2b66feeb1b77da97fed03b1ccbe    0.0s
 => => sha256:ea2ebd905ab246ece277be25520ca0cfe82758b3d2b369e2fd69b374c1d6c7fa 5.86kB / 5.86kB                           0.0s
 => => sha256:2e6afa3f266c11e8960349e7866203a9df478a50362bb5488c45fe39d99b2707 24.05MB / 24.05MB                         59.4s
 => => sha256:7859853e7607927aa1d1b1a5a2f9e580ac90c2b66feeb1b77da97fed03b1ccbe 9.72kB / 9.72kB                           0.0s
 => => sha256:f6b1cea9aa1bba7fdf508248d1ab303a4cb11274997f39c9a0d6646f221379da 2.32kB / 2.32kB                           0.0s
 => => sha256:8cd46d290033f265db57fd808ac81c444ec5a5b3f189c3d6d85043b647336913 49.56MB / 49.56MB                       115.3s
 => => sha256:2e66a70da0bec13fb3d492fcdef60fd8a5ef0a1a65c4e8a4909e26742852f0f2 64.15MB / 64.15MB                       122.8s
 => => sha256:1c8ff076d818ad6b8557e03e10c83657cc716ab287c8380054ff91571c8cae81 211.27MB / 211.27MB                      275.0s
 => => sha256:9d7cafee8af77ad487135151e94ef89c4edcd02ed6fd866d8dbc130a246380d2 6.16MB / 6.16MB                          128.3s
 => => extracting sha256:8cd46d290033f265db57fd808ac81c444ec5a5b3f189c3d6d85043b647336913                                6.7s
 => => extracting sha256:2e6afa3f266c11e8960349e7866203a9df478a50362bb5488c45fe39d99b2707                                1.8s
 => => sha256:76b2d602845c2157857573b7b630d6e22728251609b3a2013b7dfb5604d4a61f 24.14MB / 24.14MB                        160.0s
 => => extracting sha256:2e66a70da0bec13fb3d492fcdef60fd8a5ef0a1a65c4e8a4909e26742852f0f2                                7.8s
 => => sha256:b61bc9b0e1d8628f1588d6d89bfabd6bf871680a211e5cc2803bb77ad8f26170 250B / 250B                              130.2s
 => => extracting sha256:1c8ff076d818ad6b8557e03e10c83657cc716ab287c8380054ff91571c8cae81                                18.6s
 => => extracting sha256:9d7cafee8af77ad487135151e94ef89c4edcd02ed6fd866d8dbc130a246380d2                                 3.3s
 => => extracting sha256:76b2d602845c2157857573b7b630d6e22728251609b3a2013b7dfb5604d4a61f                                 4.3s
 => => extracting sha256:b61bc9b0e1d8628f1588d6d89bfabd6bf871680a211e5cc2803bb77ad8f26170                                 0.0s
 => [2/2] ADD ./app.py .                                                                                                 2.7s
 => exporting to image                                                                                                   0.1s
 => => exporting layers                                                                                                  0.0s
 => => writing image   docker-desktop://dashboard/build/desktop-linux/desktop-                                           0.0s
 => => naming to doc   linux/h1fbqjsozpajq8mckun19b9ta (cmd + click)                                                     0.0s

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/h1fbqjsozpajq8mckun19b9ta

What's next:
    View a summary of image vulnerabilities and recommendations → docker scout quickview
(base) ramazan@MacBook-Pro-Ramazan WebDev1 % docker run hello-docker
Hello, Docker!
(base) ramazan@MacBook-Pro-Ramazan WebDev1 %
```

- What is the purpose of the FROM instruction in a Dockerfile?

The FROM construct specifies the base image from which the new Docker image will be created.

- How does the COPY instruction work in Dockerfile?

COPY copies files and folders from the host machine to the container file system.

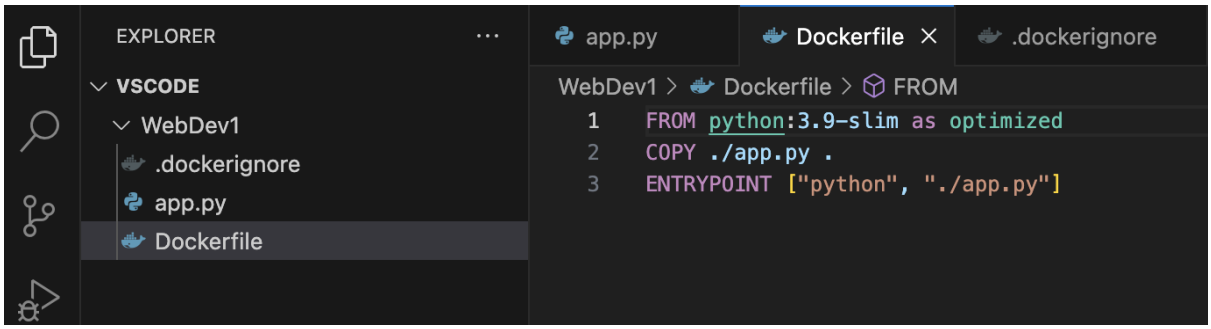
- What is the difference between CMD and ENTRYPOINT in Dockerfile?

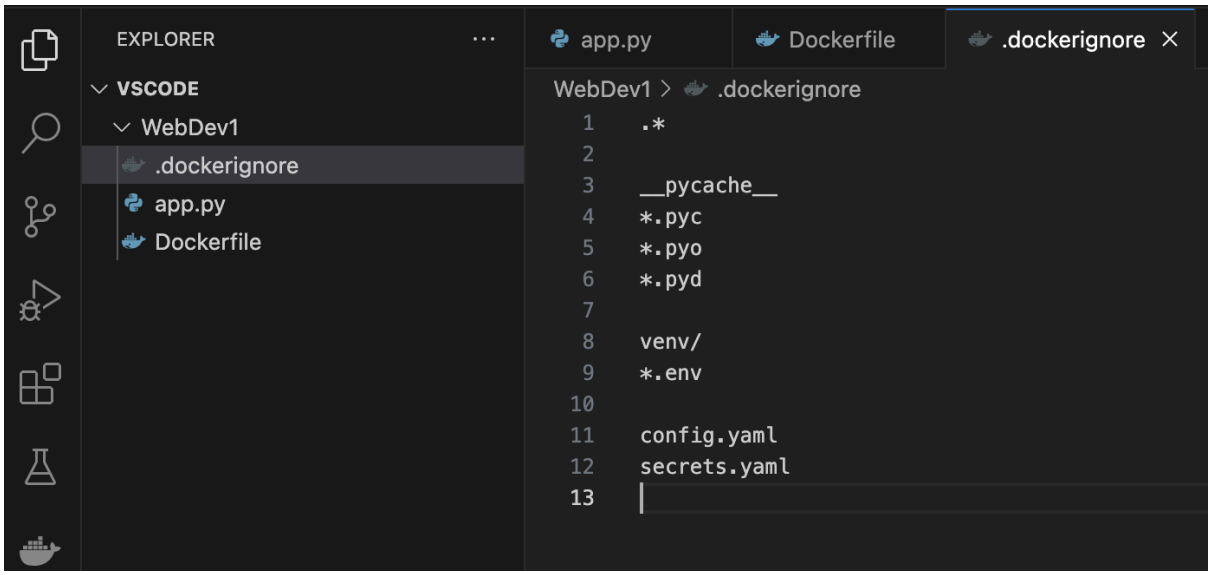
The CMD construct specifies the default command to be executed when the container is started. The ENTRYPOINT construct specifies a command or executable that will be run each time the container is started.

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?






```
(base) ramazan@MacBook-Pro-Ramazan WebDev1 % docker build -t hello-docker-optimized .
[+] Building 19.8s (8/8) FINISHED                                    docker:desktop-linux
```

```
ramazan — -zsh — 80×24
(base) ramazan@MacBook-Pro-Ramazan ~ % docker images
REPOSITORY                TAG       IMAGE ID       CREATED          SIZE
hello-docker-optimized    latest    f906eda08dd9   2 minutes ago    125MB
hello-docker              latest    b07561556bb7   35 minutes ago   1.01GB
nginx                     latest    39286ab8a5e1   5 weeks ago      188MB
hello-world               latest    d2c94e258dcb   16 months ago    13.3kB
(base) ramazan@MacBook-Pro-Ramazan ~ %
```

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

Layers are individual levels in a Docker image, each corresponding to a single instruction in the Dockerfile. Layers allow Docker to store and reuse common files, which reduces the overall size of the image.

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

Docker uses a cache to store the results of previous build steps. If the Dockerfile has not changed and previous layers have not been modified, Docker can use the cached layers instead of rebuilding them. This speeds up the build process because reusable layers do not require instructions to be executed again, saving time and resources.

- What is the role of the .dockerignore file?

The .dockerignore file is used to specify files and folders that should not be included in Docker builds. Excluding these files reduces the size of the final image.

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?

Multi-stage builds allow you to split the build process into multiple stages, which limits control dependencies and minimizes the number of intermediate files in the traditional style.

- How can multi-stage builds help reduce the size of Docker images?

Allow you to include only the necessary files from the assembly, excluding unnecessary dependencies and temporary files, which significantly reduces the size of the final image.

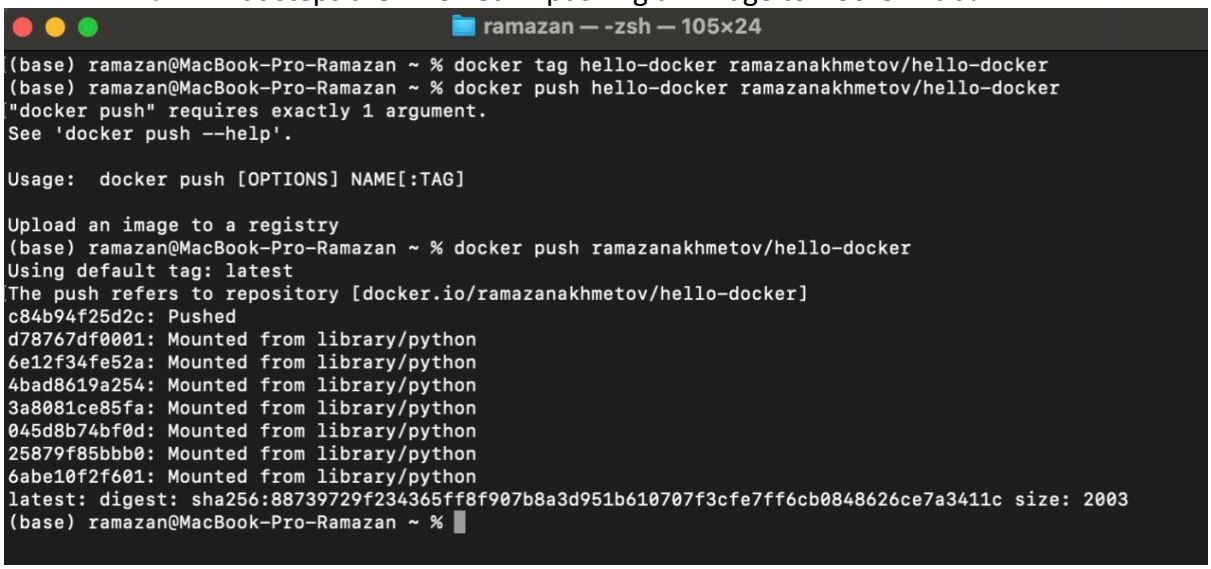
- What are some scenarios where multi-stage builds are particularly useful?

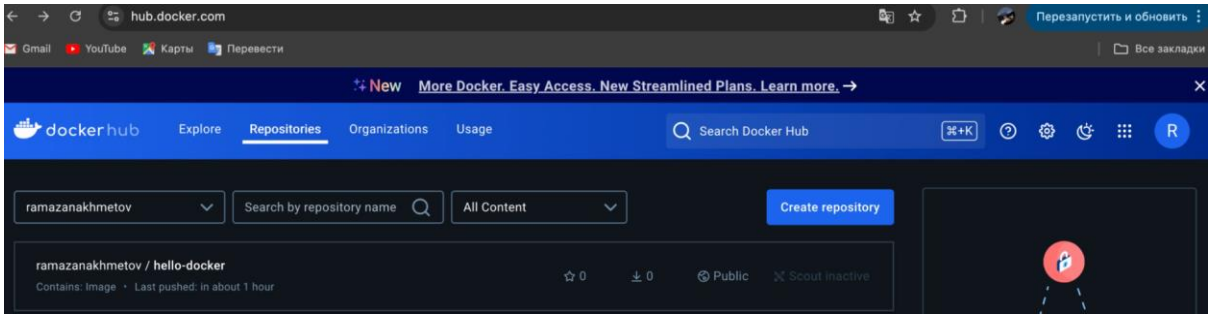
Useful when building complex applications with many dependencies.

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?





- What is the purpose of Docker Hub in containerization?

Docker Hub is a cloud service for storing and distributing Docker images, allowing developers to share and use images in their projects.

- How do you tag a Docker image for pushing to a remote repository?

An image is tagged using the docker tag <image_name> < your-username >:<tag> command, where < your-username > is the name of the remote repository and <tag> is the version or tag of the image.

- What steps are involved in pushing an image to Docker Hub?

First, log in to Docker Hub using docker login, then use the docker push < your-username >:<tag> command to push the tagged image to the repository.