Assignment 1, Web Application Development

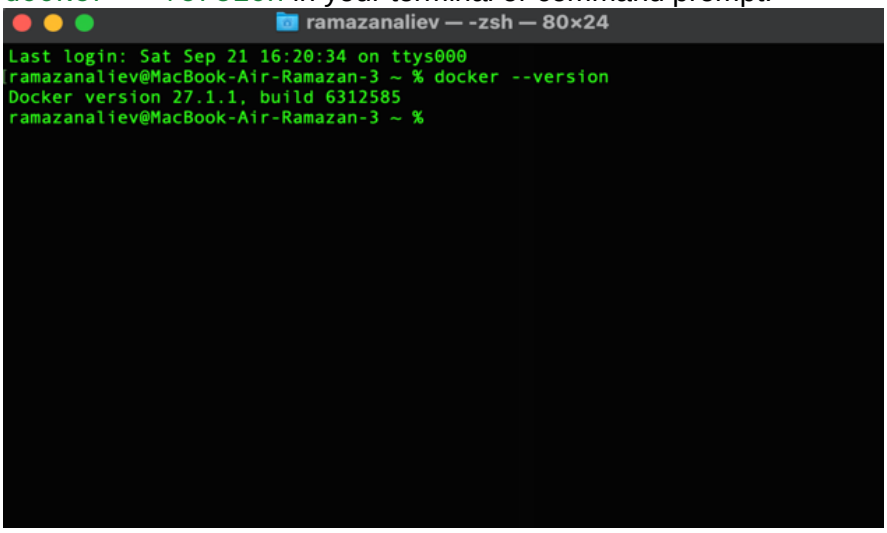
Student: Aliyev Ramazan

Intro to Containerization: Docker

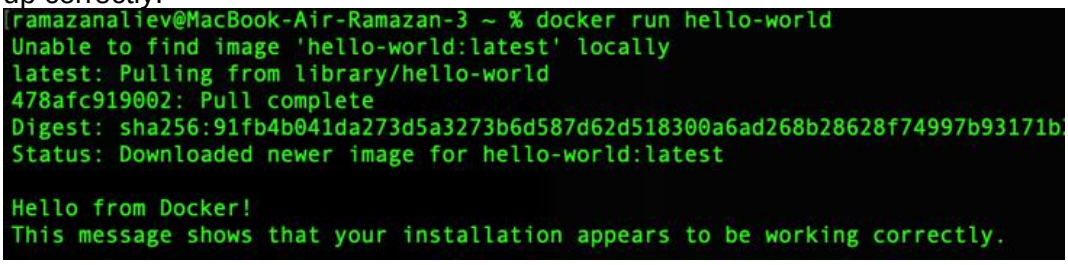
Git: https://github.com/rvmzik/Assignment_1_Web-.git

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)? Containers, images, Docker Hub, Dockerfile,
   - How does Docker compare to traditional virtual machines? Docker has containers when virtual machines don't have. Also virtual machines slower than docker
   - What was the output of the `docker run hello-world` command, and what does it signify?

Out: “Unable to find image” and then it pulls from library

Exercise 2: Basic Docker Commands

Pull an official Docker image from Docker Hub (e.g., `nginx` or `ubuntu`) using the command `docker pull <image-name>`.

```
ramazanaliev@MacBook-Air-Ramazan-3 ~ % docker pull nginx
Using default tag: latest
latest: Pulling from library/nginx
Digest: sha256:04ba374043ccd2fc5c593885c0eacddebabd5ca375f9323666f28dfd5a9710e3
Status: Image is up to date for nginx:latest
docker.io/library/nginx:latest
```

List all Docker images on your system using `docker images`.

```
ramazanaliev@MacBook-Air-Ramazan-3 ~ % docker images
REPOSITORY    TAG       IMAGE ID       CREATED          SIZE
nginx         latest    195245f0c792   5 weeks ago      193MB
hello-world   latest    ee301c921b8a   16 months ago    9.14kB
```

Run a container from the pulled image using `docker run -d <image-name>`.

```
ramazanaliev@MacBook-Air-Ramazan-3 ~ % docker run -d nginx
058f4c84678a56ea3c3a5c295e73583d0fcb1a327737094783831139ca68c195
```

List all running containers using `docker ps` and stop a container using `docker stop <container-id>`.

```
ramazanaliev@MacBook-Air-Ramazan-3 ~ % docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS
   PORTS     NAMES
058f4c84678a   nginx     "/docker-entrypoint.…"   38 seconds ago   Up 37 seconds
   80/tcp    determined_napier
```

```
ramazanaliev@MacBook-Air-Ramazan-3 ~ % docker stop 058f4c84678a
058f4c84678a
```

3. Questions:

- What is the difference between `docker pull` and `docker run`?
  I think docker run allows run the image in a new container when docker pull allows to take an image from docker hub.

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

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

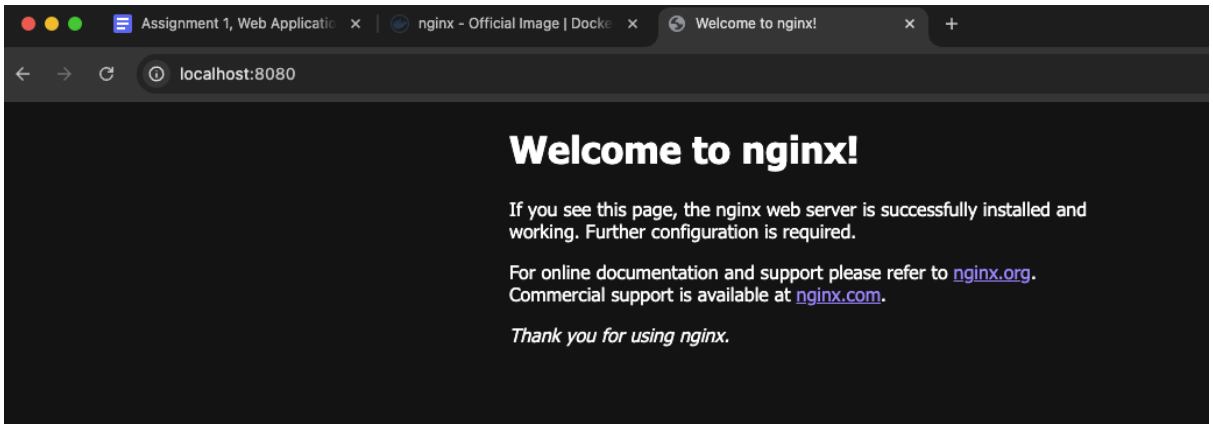
I think container will be stopped but saved. It can be restarted with command docker start but not automatically

Exercise 3: Working with Docker Containers

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`.

```
ramazanaliev@MacBook-Air-Ramazan-3 ~ % docker run -d -p 8080:80 nginx
43c4925b880f8b9dd4bdd0e76d281872479835487c0371f857ab9bc6ba5226e6
```

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`.

```
ramazanaliev@MacBook-Air-Ramazan-3 ~ % docker exec -it 43c4925b880f  /bin/bash
root@43c4925b880f:/#
```

```
root@43c4925b880f:/# ls
bin   dev                  docker-entrypoint.sh  home  media  opt   root  sbin  sys  usr
boot  docker-entrypoint.d  etc                   lib   mnt    proc  run   srv   tmp  var
```

```
root@43c4925b880f:/# exit
exit
```

Stop and remove the container using `docker stop <container-id>` and `docker rm <container-id>`.

```
ramazanaliev@MacBook-Air-Ramazan-3 ~ % docker stop 43c4925b880f
43c4925b880f
```

```
ramazanaliev@MacBook-Air-Ramazan-3 ~ % docker rm 43c4925b880f
43c4925b880f
```

3. **Questions**:
   - How does port mapping work in Docker, and why is it important?
     It's important because it allows server to be accessible from outside
   - What is the purpose of the `docker exec` command?

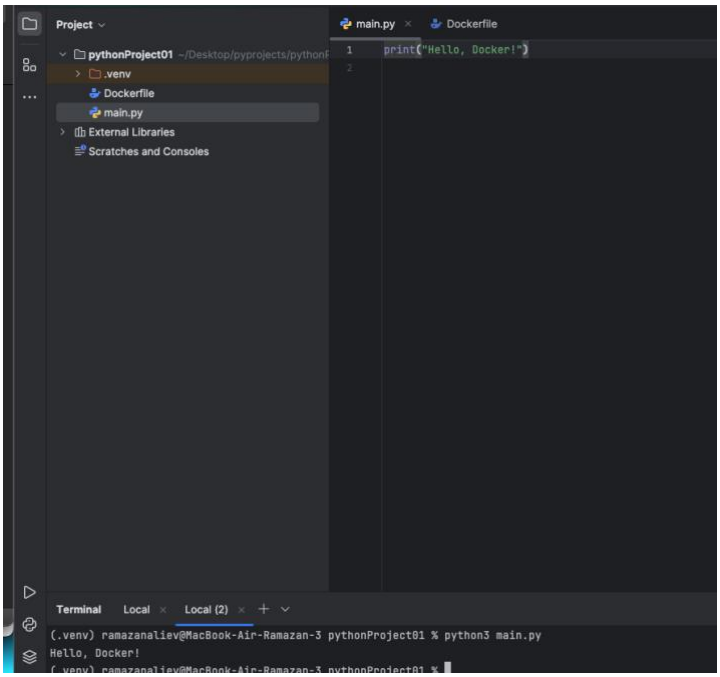
     To execute command while container is running
   - How do you ensure that a stopped container does not consume system resources?

     Docker ps command

Dockerfile

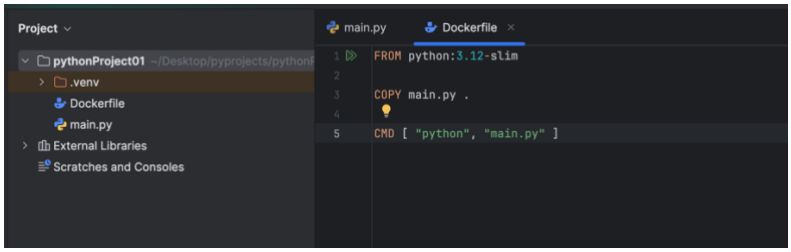
Exercise 1: Creating a Simple Dockerfile

1. **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.

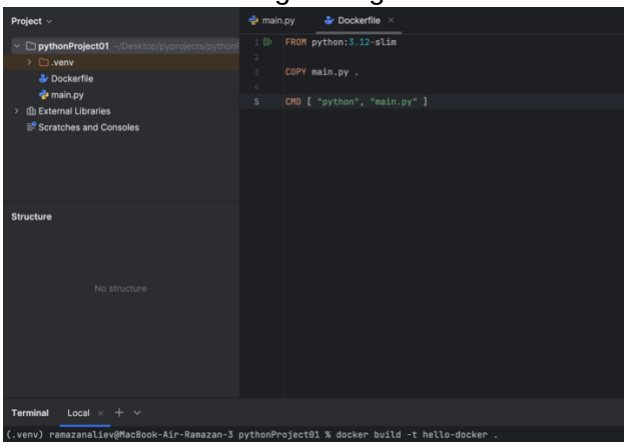


   -
   - Write a Dockerfile that:

- 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`.

```
(.venv) ramazanaliev@MacBook-Air-Ramazan-3 pythonProject01 % docker run hello-docker
Hello, Docker!
```

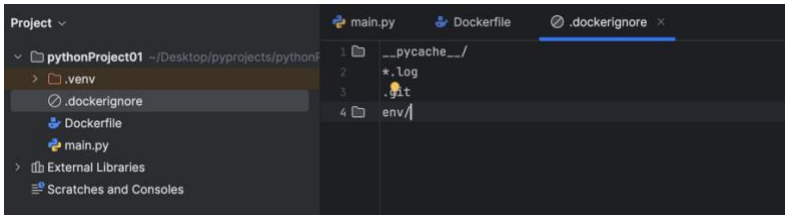
2. **Questions**:
   - What is the purpose of the `FROM` instruction in a Dockerfile?
     FROM instruction sets the base image for initialize the new image.

   - How does the `COPY` instruction work in Dockerfile? The copy instruction copies files and adds the to the filesystem of the container at a new path.

   - What is the difference between `CMD` and `ENTRYPOINT` in Dockerfile?
     ENTRYPOINT always will execute from the time when container is running when CMD is defines command that will execute from the time when container is running.

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.

```
(.venv) ramazanaliev@MacBook-Air-Ramazan-3 pythonProject01 % docker build -t hello-docker-new .
[+] Building 0.1s (7/7) FINISHED
```

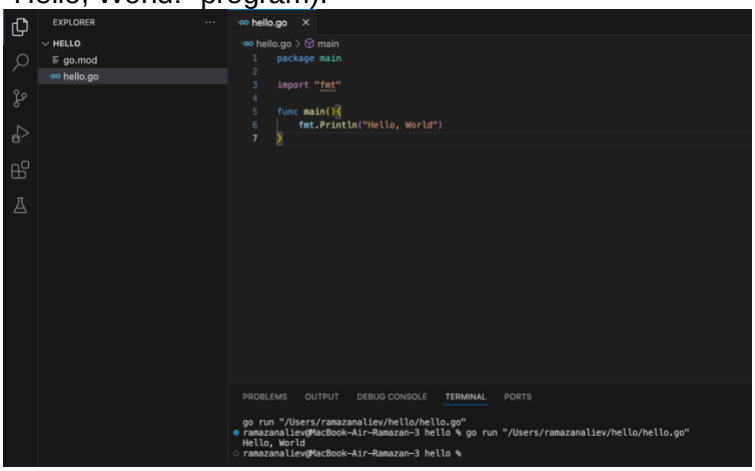
- Compare the size of the optimized image with the original.

```
(.venv) ramazanaliev@MacBook-Air-Ramazan-3 pythonProject01 % docker images
REPOSITORY          TAG         IMAGE ID       CREATED         SIZE
hello-docker-new    latest      7f10570287dd   2 hours ago     150MB
hello-docker        latest      7f10570287dd   2 hours ago     150MB
python              3.12-slim   c0cb1ac1ea30   12 days ago     150MB
```

3. **Questions**:
   - What are Docker layers, and how do they affect image size and build times?
     Images have docker layers and for every instruction there will be a layer. Also It's important for image size and build times because if there will be many layers then size of an image will be bigger
   - How does Docker's build cache work, and how can it speed up the build process?
     Cache work saves time because parts of the image will be remain
   - What is the role of the `.dockerignore` file? It needs to ignore some files to reduce image size.

Exercise 3: Multi-Stage Builds

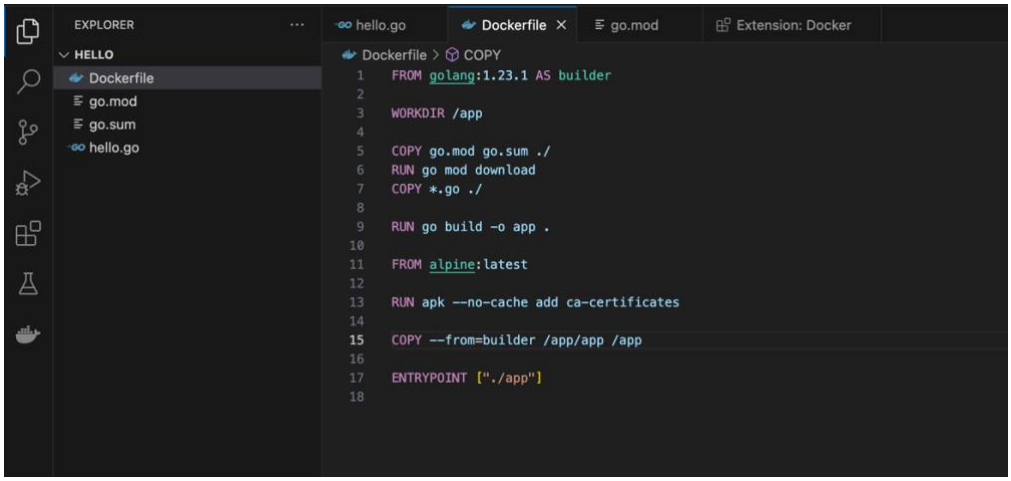
1. **Objective**: Use multi-stage builds to create leaner Docker images.
2. **Steps**:
   - Create a new project that involves compiling a simple Go application (e.g., a "Hello, World!" program).



   - Write a Dockerfile that uses multi-stage builds:
     - The first stage should use a Golang image to compile the application.
       From running first stage:

```
ramazanaliev@MacBook-Air-Ramazan-3 hello % docker run --rm hello-docker-go-first-stage
Hello, World
```

     - 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.

```
ramazanaliev@MacBook-Air-Ramazan-3 hello % docker images
REPOSITORY                    TAG       IMAGE ID       CREATED          SIZE
hello-docker-go-second-stage  latest    bd1b7926c085   19 seconds ago   11.6MB
hello-docker-go-first-stage   latest    3ac4c67db951   18 minutes ago   876MB
```

3. **Questions**:
   - What are the benefits of using multi-stage builds in Docker?
     To reduce size of an image, for example for the second stage
   - How can multi-stage builds help reduce the size of Docker images?
     As an example it can use Alpine image which can reduce size.
   - What are some scenarios where multi-stage builds are particularly useful? As an example I think it can be helpful when we use 2 programming language

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`).

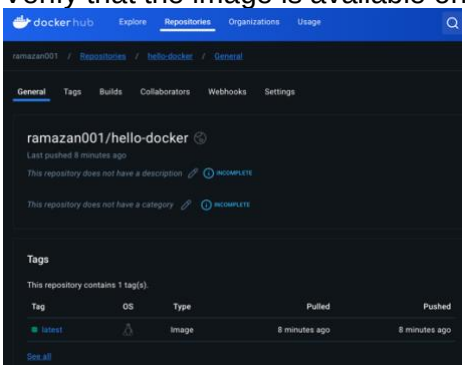
```
ramazanaliev@MacBook-Air-Ramazan-3 hello % docker tag hello-docker ramazan001/hello-docker
```

   - Log in to Docker Hub using `docker login`.

```
ramazanaliev@MacBook-Air-Ramazan-3 hello % docker login
Login Succeeded
```

   - Push the image to Docker Hub using `docker push <your-username>/hello-docker`.

```
ramazanaliev@MacBook-Air-Ramazan-3 hello % docker push ramazan001/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?
  To find and share docker images.
- How do you tag a Docker image for pushing to a remote repository?
  As an example: docker tag hello-docker username/hello-docker
- What steps are involved in pushing an image to Docker Hub? Login Docker Hub -> Tag -> docker login -> docker push