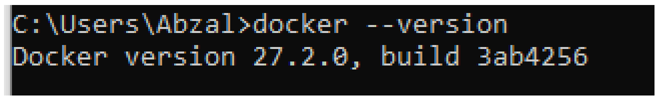
Assignment 1, Web Application Development

Put all deliverables into github repository in your profile. Share link to google form 24 hours before defense. Defend by explaining deliverables and answering questions.
Deliverables: report in pdf
Google form:
https://docs.google.com/forms/d/e/1FAIpQLSe0GyNdOYIvM1tX_I_CtlPod5jBf-ACLGdHYZq1gVZbUeBzIg/viewform?usp=sf_link

Intro to Containerization: Docker

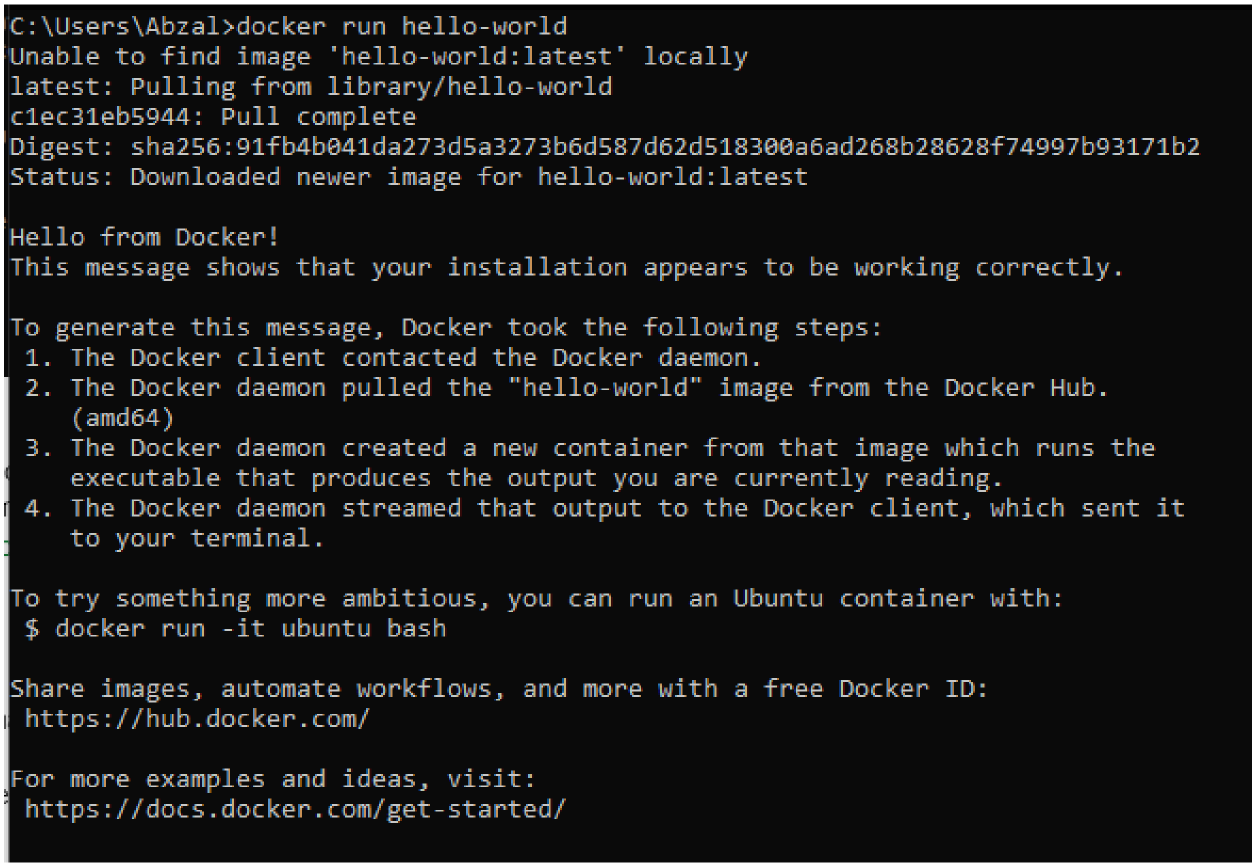
Exercise 1: Installing Docker

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.



```
C:\Users\Abzal>docker --version
Docker version 27.2.0, build 3ab4256
```

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



```
C:\Users\Abzal>docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
c1ec31eb5944: Pull complete
Digest: sha256:91fb4b041da273d5a3273b6d587d62d518300a6ad268b28628f74997b93171b2
Status: Downloaded newer image for hello-world:latest

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/
```

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

Docker consists of 3 main components: The Docker Engine (Server, REST API, CLI), Docker Images, Dockerfile.

- How does Docker compare to traditional virtual machines?

The main difference between Docker and virtual machines (VMs) is how they are set up. VMs have both a host OS and a separate guest OS inside each VM, so you can run different operating systems. Docker containers run on the same host OS, sharing it across all containers, making them lighter and faster.

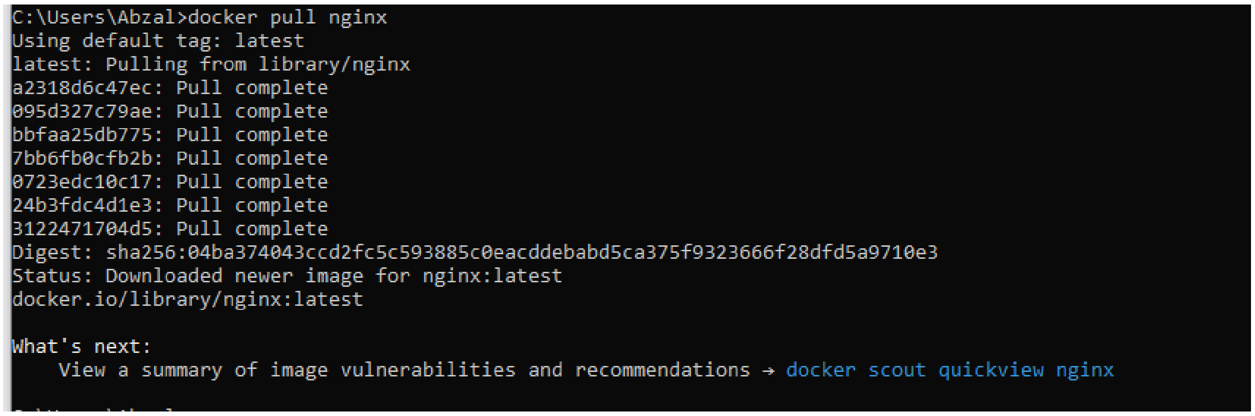
- What was the output of the `docker run hello-world` command, and what does it signify?

It tells us that our installation of Docker is correct.

Exercise 2: Basic Docker Commands

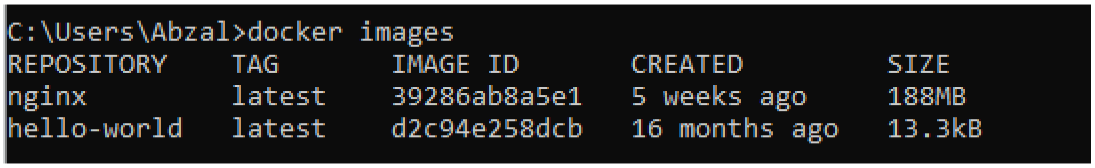
1. **Objective**: Familiarize yourself with basic Docker commands.
2. **Steps**:

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



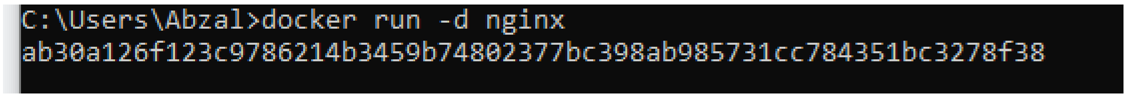
```
C:\Users\Abzal>docker pull nginx
Using default tag: latest
latest: Pulling from library/nginx
a2318d6c47ec: Pull complete
095d327c79ae: Pull complete
bbfaa25db775: Pull complete
7bb6fb0cfb2b: Pull complete
0723edc10c17: Pull complete
24b3fdc4d1e3: Pull complete
3122471704d5: Pull complete
Digest: 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
```

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



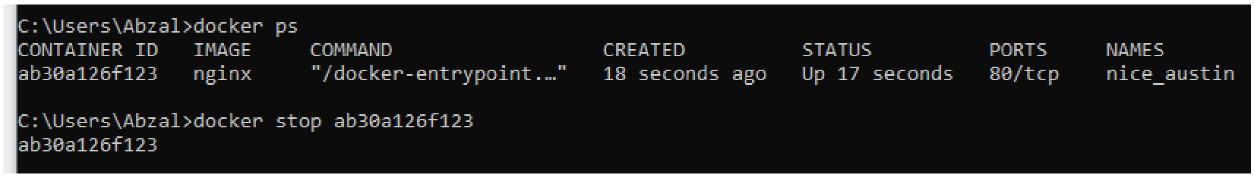
```
C:\Users\Abzal>docker images
REPOSITORY    TAG       IMAGE ID       CREATED         SIZE
nginx         latest    39286ab8a5e1   5 weeks ago     188MB
hello-world   latest    d2c94e258dcb   16 months ago   13.3kB
```

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



```
C:\Users\Abzal>docker run -d nginx
ab30a126f123c9786214b3459b74802377bc398ab985731cc784351bc3278f38
```

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



```
C:\Users\Abzal>docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS     NAMES
ab30a126f123   nginx     "/docker-entrypoint.…"   18 seconds ago   Up 17 seconds   80/tcp    nice_austin

C:\Users\Abzal>docker stop ab30a126f123
ab30a126f123
```

3. **Questions**:
   - What is the difference between `docker pull` and `docker run`?

docker run will run an instance of a container, to do that it will pull all needed images in the background if they are not in the local storage.

docker pull will download the image from the docker registry.

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

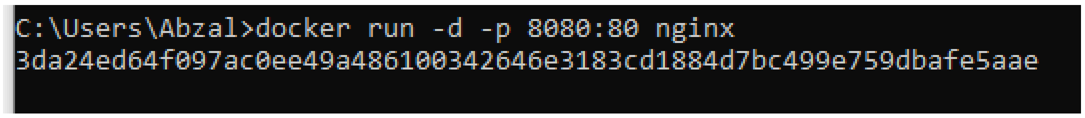
For that you need to run **docker ps** command.

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

After stopping a container, all the processes will stop, but they can be started again using **docker start <container name>.**

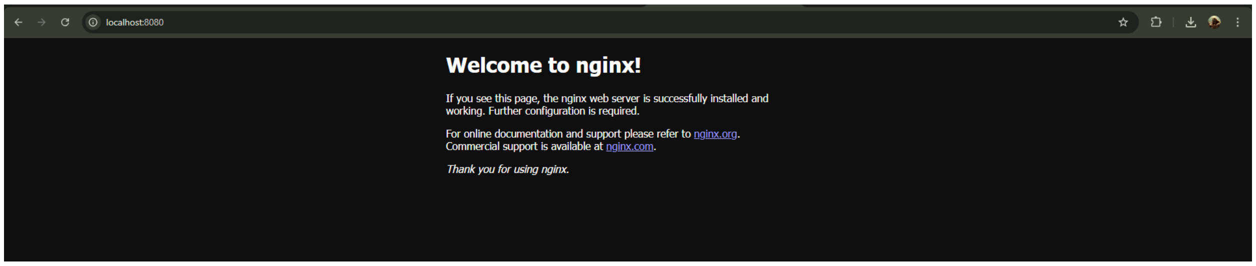
Exercise 3: Working with Docker Containers

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`.



```
C:\Users\Abzal>docker run -d -p 8080:80 nginx
3da24ed64f097ac0ee49a486100342646e3183cd1884d7bc499e759dbafe5aae
```

- Access the Nginx web server running in the container by navigating to `http://localhost:8080` in your web browser.



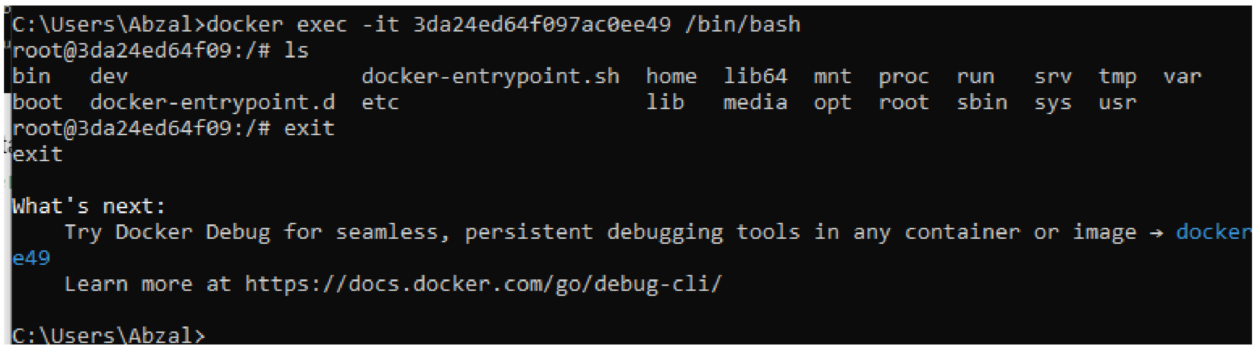
Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

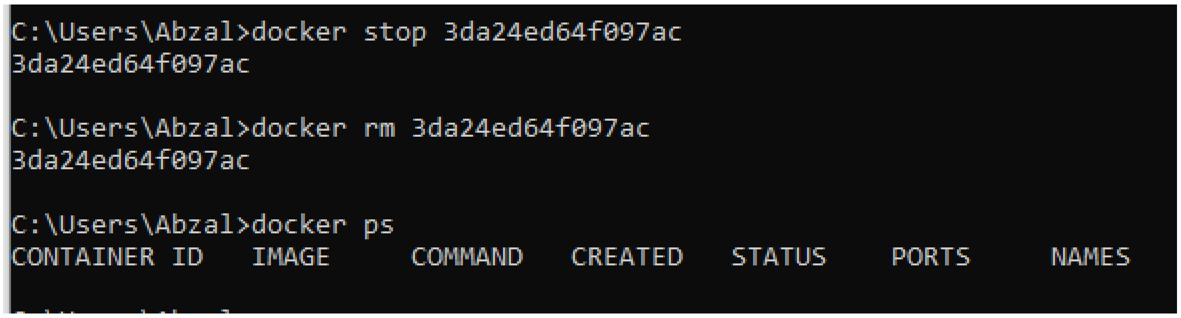
Thank you for using nginx.

- Explore the container's file system by accessing its shell using `docker exec -it <container-id> /bin/bash`.



```
C:\Users\Abzal>docker exec -it 3da24ed64f097ac0ee49 /bin/bash
root@3da24ed64f09:/# ls
bin   dev                   docker-entrypoint.sh  home  lib64  mnt  proc  run   srv  tmp  var
boot  docker-entrypoint.d   etc                   lib   media  opt  root  sbin  sys  usr
root@3da24ed64f09:/# exit
exit

What's next:
    Try Docker Debug for seamless, persistent debugging tools in any container or image → docker
e49
    Learn more at https://docs.docker.com/go/debug-cli/

C:\Users\Abzal>
```

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



```
C:\Users\Abzal>docker stop 3da24ed64f097ac
3da24ed64f097ac

C:\Users\Abzal>docker rm 3da24ed64f097ac
3da24ed64f097ac

C:\Users\Abzal>docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
```

3. **Questions**:
   - How does port mapping work in Docker, and why is it important?

Port mapping allows us to connect a container's internal port with computer ports, allowing external access to the services inside the container. It is important because it allows external access, keeps containers isolated and maintains flexibility in a sense that you can run multiple containers on different ports.

   - What is the purpose of the `docker exec` command?

docker exec command is used to run commands inside a running container without stopping or restarting the container. It allows us to interact with the container and perform tasks.

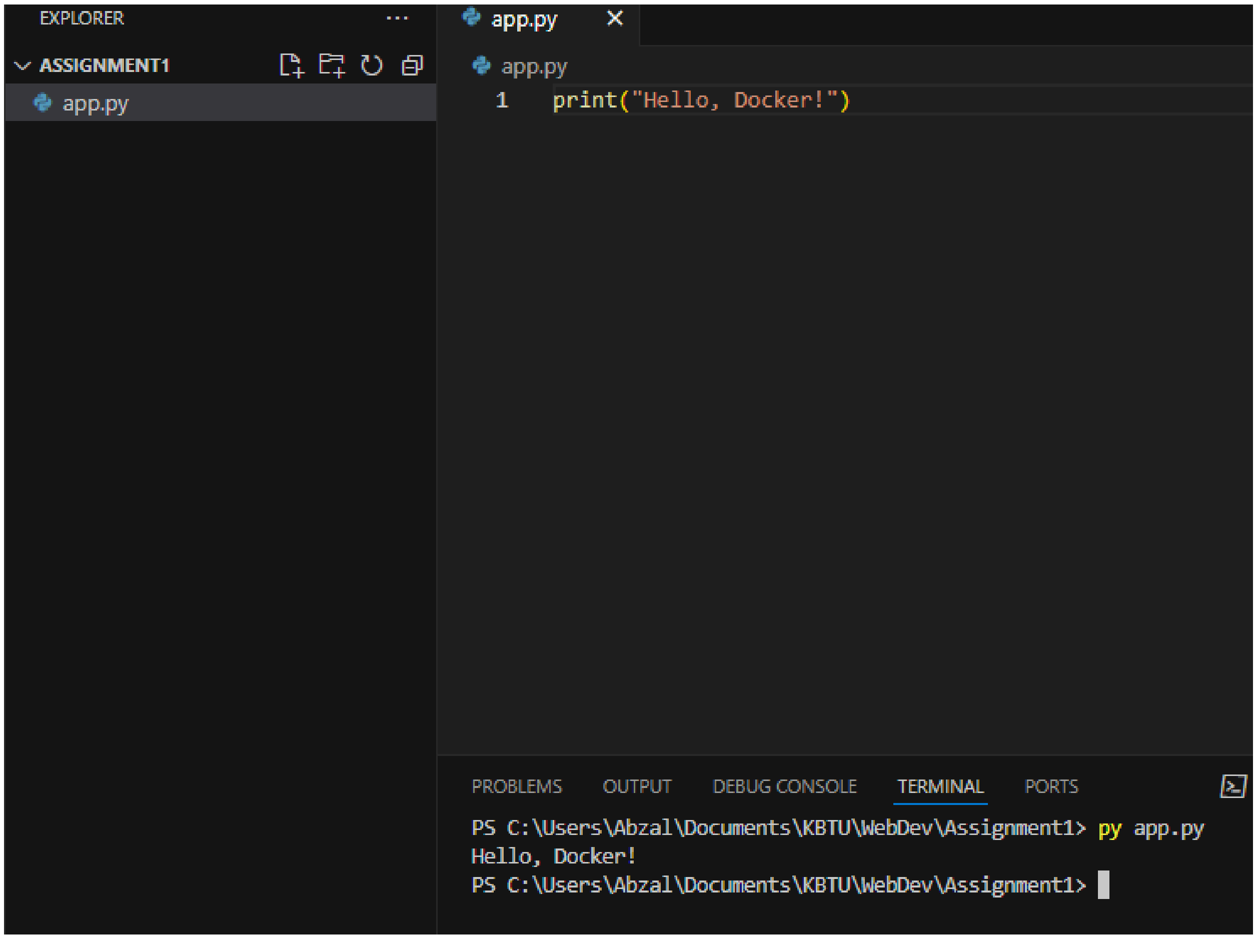
   - How do you ensure that a stopped container does not consume system resources?

For that you need to run **docker rm <container id>** command.

Dockerfile

Exercise 1: Creating a Simple Dockerfile

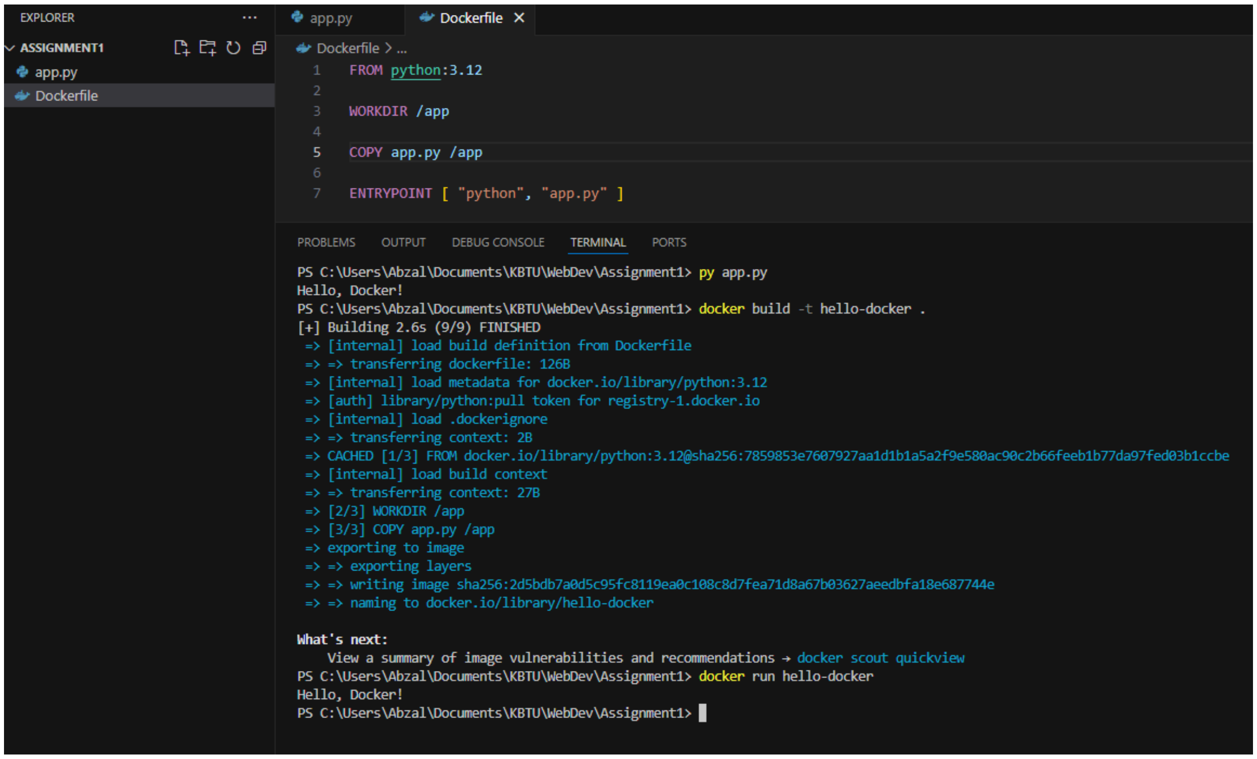
1. **Objective**: Write a Dockerfile to containerize a basic application.
2. **Steps**:
   - Create a new directory for your project and navigate into it.
   - Create a simple Python script (e.g., `app.py`) that prints "Hello, Docker!" to the console.



```python
print("Hello, Docker!")
```

```
PS C:\Users\Abzal\Documents\KBTU\WebDev\Assignment1> py app.py
Hello, Docker!
PS C:\Users\Abzal\Documents\KBTU\WebDev\Assignment1>
```

- 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`.



```dockerfile
FROM python:3.12

WORKDIR /app

COPY app.py /app

ENTRYPOINT [ "python", "app.py" ]
```

```
PS C:\Users\Abzal\Documents\KBTU\WebDev\Assignment1> py app.py
Hello, Docker!
PS C:\Users\Abzal\Documents\KBTU\WebDev\Assignment1> docker build -t hello-docker .
[+] Building 2.6s (9/9) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 126B
 => [internal] load metadata for docker.io/library/python:3.12
 => [auth] library/python:pull token for registry-1.docker.io
 => [internal] load .dockerignore
 => => transferring context: 2B
 => CACHED [1/3] FROM docker.io/library/python:3.12@sha256:7859853e7607927aa1d1b1a5a2f9e580ac90c2b66feeb1b77da97fed03b1ccbe
 => [internal] load build context
 => => transferring context: 27B
 => [2/3] WORKDIR /app
 => [3/3] COPY app.py /app
 => exporting to image
 => => exporting layers
 => => writing image sha256:2d5bdb7a0d5c95fc8119ea0c108c8d7fea71d8a67b03627aeedbfa18e687744e
 => => naming to docker.io/library/hello-docker

What's next:
    View a summary of image vulnerabilities and recommendations → docker scout quickview
PS C:\Users\Abzal\Documents\KBTU\WebDev\Assignment1> docker run hello-docker
Hello, Docker!
PS C:\Users\Abzal\Documents\KBTU\WebDev\Assignment1>
```

3. **Questions**:
   - What is the purpose of the `FROM` instruction in a Dockerfile?

It uses the official Python image as a base.

   - How does the `COPY` instruction work in Dockerfile?

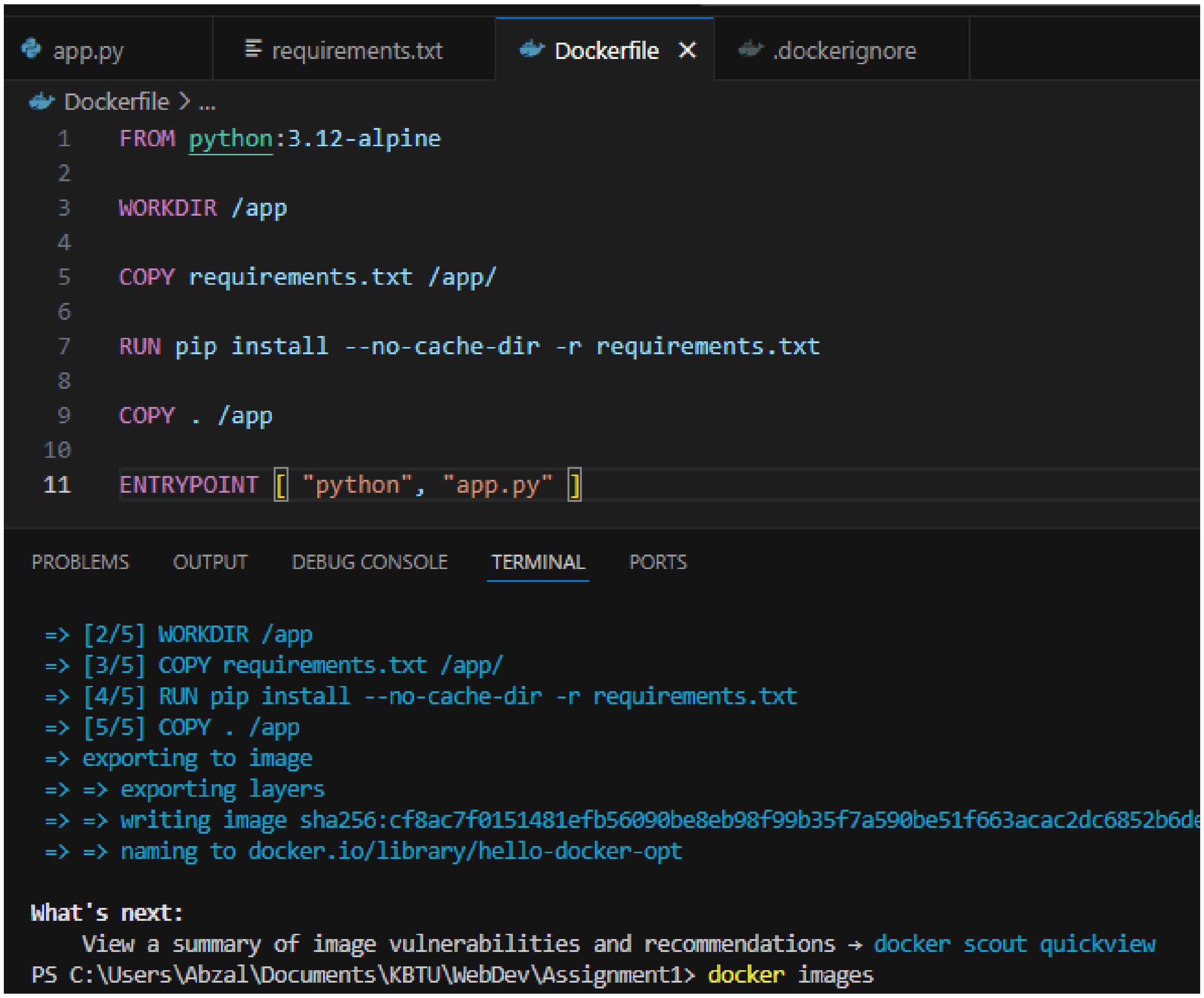
It copies the app.py file from the local machine to the container at the /app destination.

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

CMD sets a default command to run when the container starts, but it can be easily overridden by specifying a different command at runtime. **ENTRYPOINT** defines a command that always runs, and any additional arguments provided at runtime are passed to it. Both can be used together, with ENTRYPOINT setting the main command and CMD providing default arguments.

Exercise 2: Optimizing Dockerfile with Layers and Caching

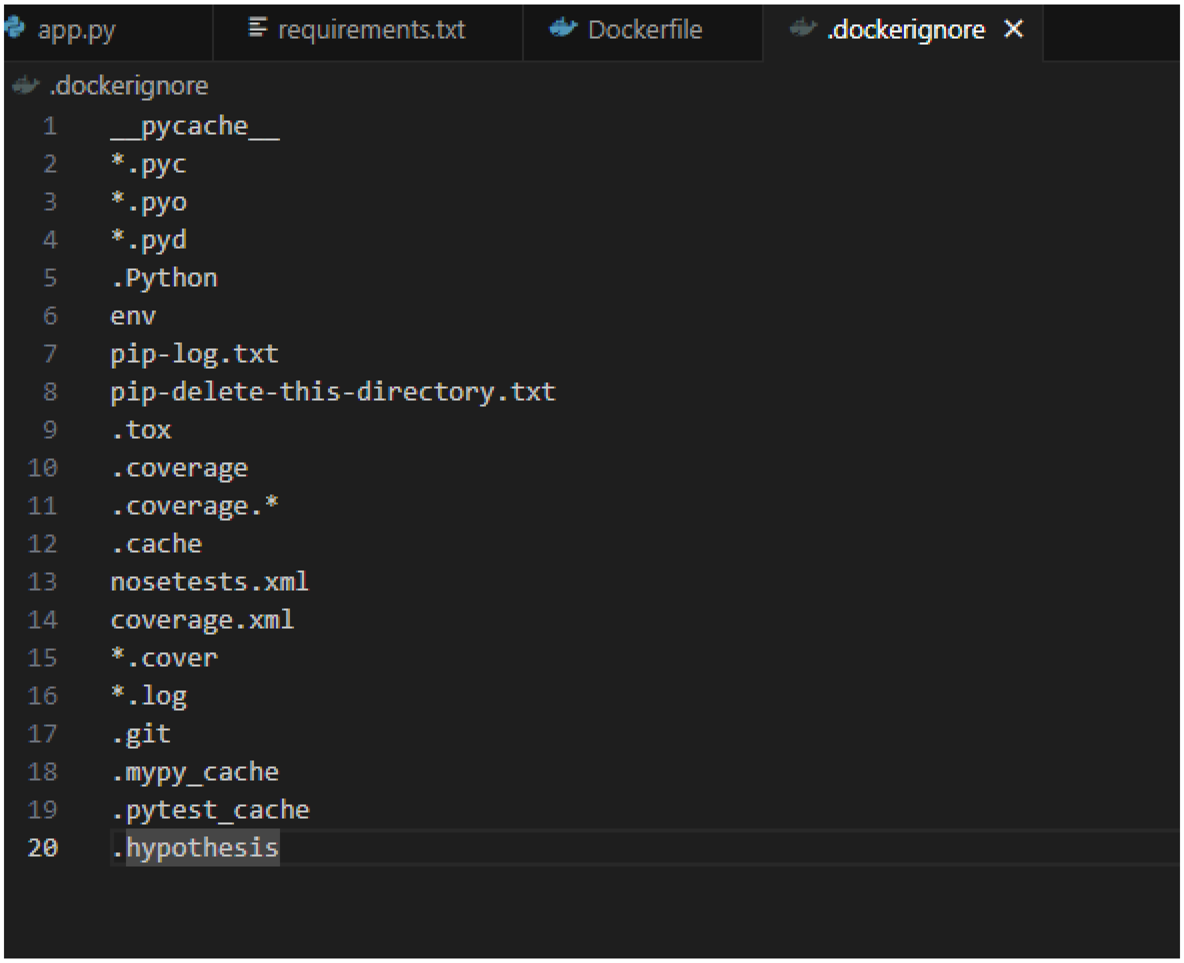
1. **Objective**: Learn how to optimize a Dockerfile for smaller image sizes and faster builds.
2. **Steps**:
   - Modify the Dockerfile created in the previous exercise to:
     - Separate the installation of Python dependencies (if any) from the copying of application code.



```dockerfile
FROM python:3.12-alpine

WORKDIR /app

COPY requirements.txt /app/

RUN pip install --no-cache-dir -r requirements.txt

COPY . /app

ENTRYPOINT [ "python", "app.py" ]
```

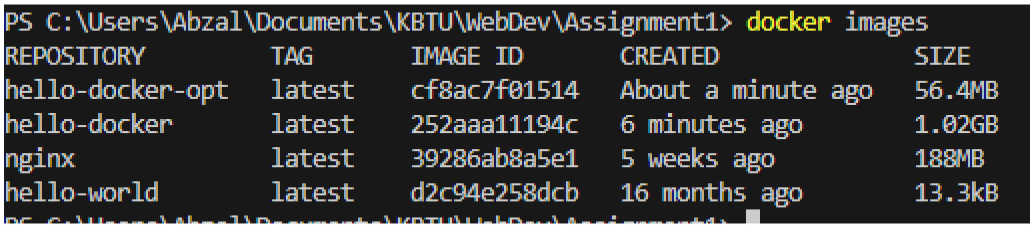
```
 => [2/5] WORKDIR /app
 => [3/5] COPY requirements.txt /app/
 => [4/5] RUN pip install --no-cache-dir -r requirements.txt
 => [5/5] COPY . /app
 => exporting to image
 => => exporting layers
 => => writing image sha256:cf8ac7f0151481efb56090be8eb98f99b35f7a590be51f663acac2dc6852b6de
 => => naming to docker.io/library/hello-docker-opt

What's next:
    View a summary of image vulnerabilities and recommendations → docker scout quickview
PS C:\Users\Abzal\Documents\KBTU\WebDev\Assignment1> docker images
```

- Use a `.dockerignore` file to exclude unnecessary files from the image.



```
__pycache__
*.pyc
*.pyo
*.pyd
.Python
env
pip-log.txt
pip-delete-this-directory.txt
.tox
.coverage
.coverage.*
.cache
nosetests.xml
coverage.xml
*.cover
*.log
.git
.mypy_cache
.pytest_cache
.hypothesis
```

- Rebuild the Docker image and observe the build process to understand how caching works.
- Compare the size of the optimized image with the original.



```
PS C:\Users\Abzal\Documents\KBTU\WebDev\Assignment1> docker images
REPOSITORY         TAG       IMAGE ID       CREATED              SIZE
hello-docker-opt   latest    cf8ac7f01514   About a minute ago   56.4MB
hello-docker       latest    252aaa11194c   6 minutes ago        1.02GB
nginx              latest    39286ab8a5e1   5 weeks ago          188MB
hello-world        latest    d2c94e258dcb   16 months ago        13.3kB
```

3. **Questions**:
   - What are Docker layers, and how do they affect image size and build times?

Docker images are built from multiple layers, and each layer corresponds to a command in the Dockerfile. Image size and build times increase with more and larger layers.

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

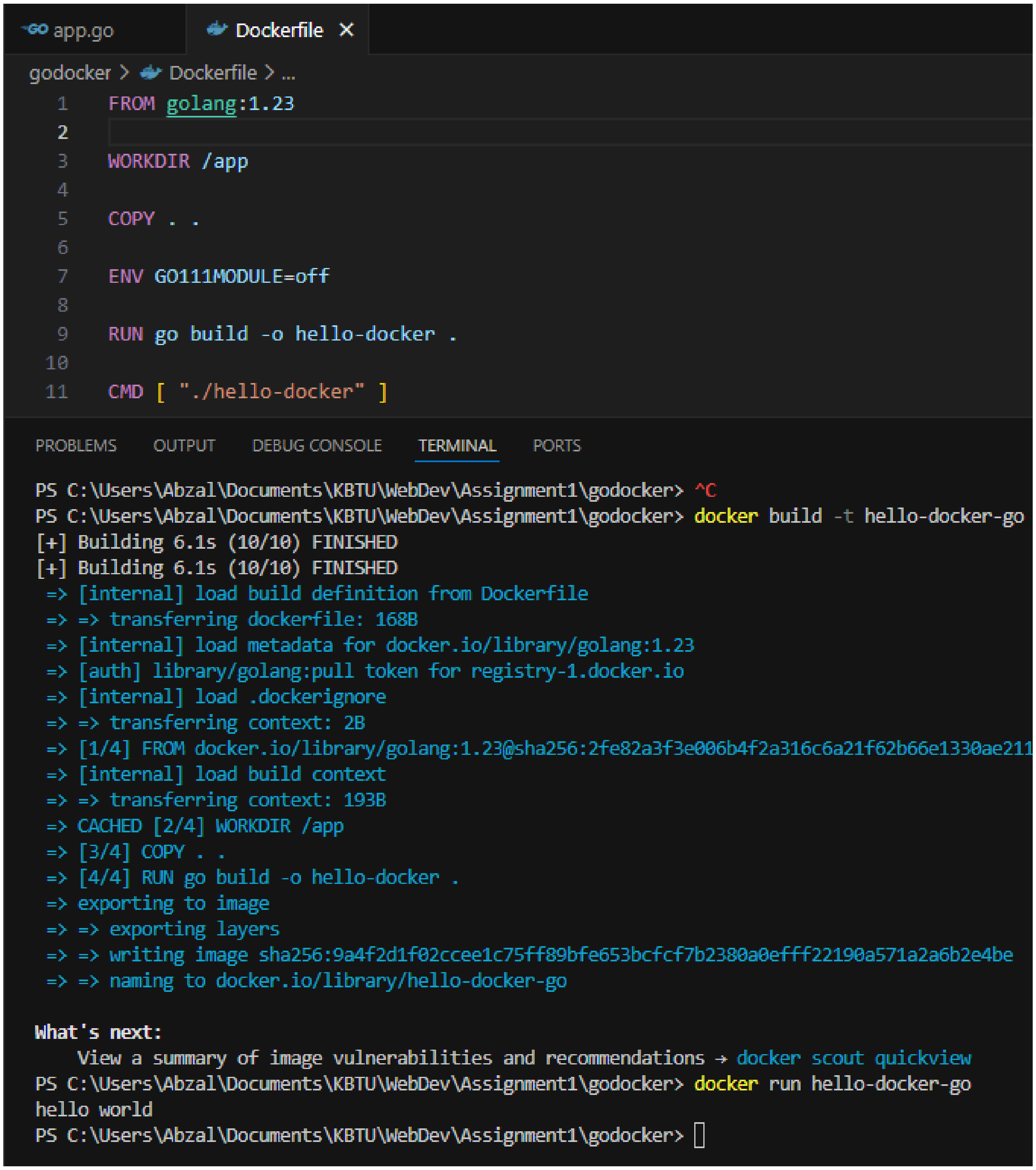
Docker's build cache saves the results of each step during the image build. It speeds up future builds by reusing unchanged steps, so only the parts that have changed are rebuilt, making the process faster.

- What is the role of the `.dockerignore` file?

The .dockerignore file tells Docker which files and directories to ignore when building an image.

Exercise 3: Multi-Stage Builds

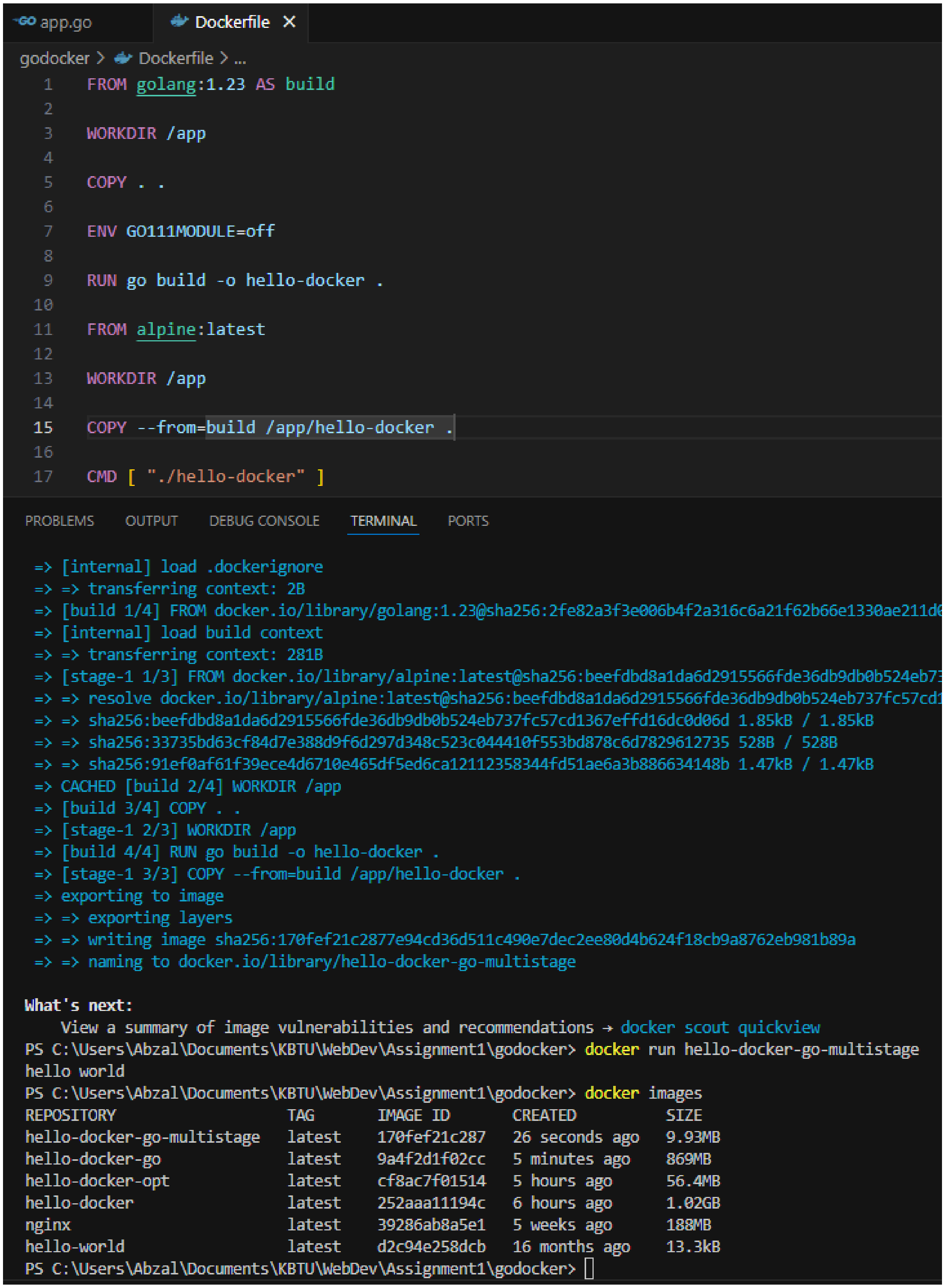
1. **Objective**: Use multi-stage builds to create leaner Docker images.
2. **Steps**:
   - Create a new project that involves compiling a simple Go application (e.g., a "Hello, World!" program).



```
FROM golang:1.23

WORKDIR /app

COPY . .

ENV GO111MODULE=off

RUN go build -o hello-docker .

CMD [ "./hello-docker" ]
```

```
PS C:\Users\Abzal\Documents\KBTU\WebDev\Assignment1\godocker> ^C
PS C:\Users\Abzal\Documents\KBTU\WebDev\Assignment1\godocker> docker build -t hello-docker-go
[+] Building 6.1s (10/10) FINISHED
[+] Building 6.1s (10/10) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 168B
 => [internal] load metadata for docker.io/library/golang:1.23
 => [auth] library/golang:pull token for registry-1.docker.io
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [1/4] FROM docker.io/library/golang:1.23@sha256:2fe82a3f3e006b4f2a316c6a21f62b66e1330ae211
 => [internal] load build context
 => => transferring context: 193B
 => CACHED [2/4] WORKDIR /app
 => [3/4] COPY . .
 => [4/4] RUN go build -o hello-docker .
 => exporting to image
 => => exporting layers
 => => writing image sha256:9a4f2d1f02ccee1c75ff89bfe653bcfcf7b2380a0efff22190a571a2a6b2e4be
 => => naming to docker.io/library/hello-docker-go

What's next:
    View a summary of image vulnerabilities and recommendations → docker scout quickview
PS C:\Users\Abzal\Documents\KBTU\WebDev\Assignment1\godocker> docker run hello-docker-go
hello world
PS C:\Users\Abzal\Documents\KBTU\WebDev\Assignment1\godocker> 
```

- 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.



```dockerfile
FROM golang:1.23 AS build

WORKDIR /app

COPY . .

ENV GO111MODULE=off

RUN go build -o hello-docker .

FROM alpine:latest

WORKDIR /app

COPY --from=build /app/hello-docker .

CMD [ "./hello-docker" ]
```

```
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [build 1/4] FROM docker.io/library/golang:1.23@sha256:2fe82a3f3e006b4f2a316c6a21f62b66e1330ae211d
 => [internal] load build context
 => => transferring context: 281B
 => [stage-1 1/3] FROM docker.io/library/alpine:latest@sha256:beefdbd8a1da6d2915566fde36db9db0b524eb7
 => => resolve docker.io/library/alpine:latest@sha256:beefdbd8a1da6d2915566fde36db9db0b524eb737fc57cd
 => => sha256:beefdbd8a1da6d2915566fde36db9db0b524eb737fc57cd1367effd16dc0d06d 1.85kB / 1.85kB
 => => sha256:33735bd63cf84d7e388d9f6d297d348c523c044410f553bd878c6d7829612735 528B / 528B
 => => sha256:91ef0af61f39ece4d6710e465df5ed6ca12112358344fd51ae6a3b886634148b 1.47kB / 1.47kB
 => CACHED [build 2/4] WORKDIR /app
 => [build 3/4] COPY . .
 => [stage-1 2/3] WORKDIR /app
 => [build 4/4] RUN go build -o hello-docker .
 => [stage-1 3/3] COPY --from=build /app/hello-docker .
 => exporting to image
 => => exporting layers
 => => writing image sha256:170fef21c2877e94cd36d511c490e7dec2ee80d4b624f18cb9a8762eb981b89a
 => => naming to docker.io/library/hello-docker-go-multistage

What's next:
    View a summary of image vulnerabilities and recommendations → docker scout quickview
PS C:\Users\Abzal\Documents\KBTU\WebDev\Assignment1\godocker> docker run hello-docker-go-multistage
hello world
PS C:\Users\Abzal\Documents\KBTU\WebDev\Assignment1\godocker> docker images
REPOSITORY                 TAG       IMAGE ID       CREATED          SIZE
hello-docker-go-multistage latest    170fef21c287   26 seconds ago   9.93MB
hello-docker-go            latest    9a4f2d1f02cc   5 minutes ago    869MB
hello-docker-opt           latest    cf8ac7f01514   5 hours ago      56.4MB
hello-docker               latest    252aaa11194c   6 hours ago      1.02GB
nginx                      latest    39286ab8a5e1   5 weeks ago      188MB
hello-world                latest    d2c94e258dcb   16 months ago    13.3kB
PS C:\Users\Abzal\Documents\KBTU\WebDev\Assignment1\godocker> 
```

3. **Questions**:
    - What are the benefits of using multi-stage builds in Docker?

Multi-stage builds in Docker reduce image size, improve security, speed up builds through caching, and make Dockerfiles more organized by separating build and runtime environments.

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

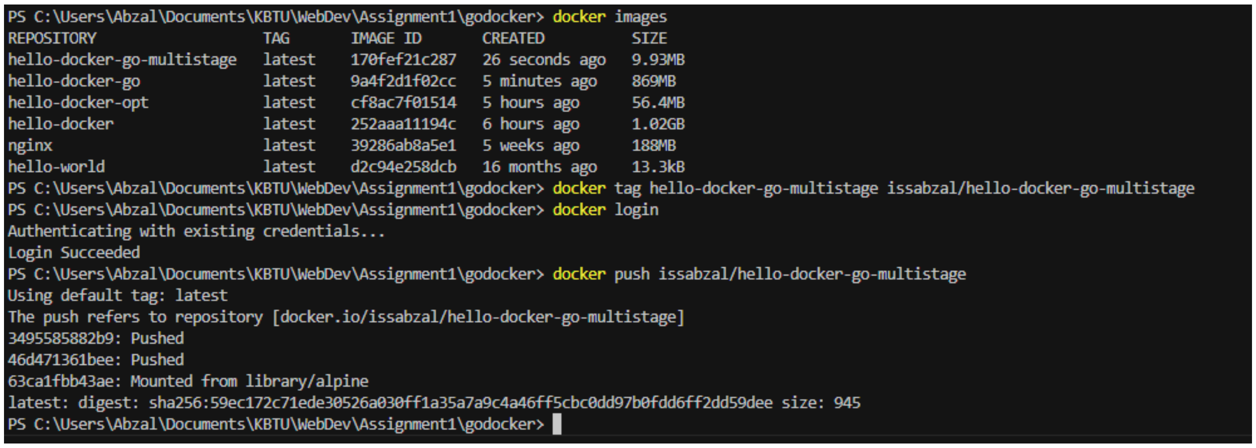
Multi-stage builds reduce Docker image size by keeping only the important files in the final image and removing everything else used during the build process.

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

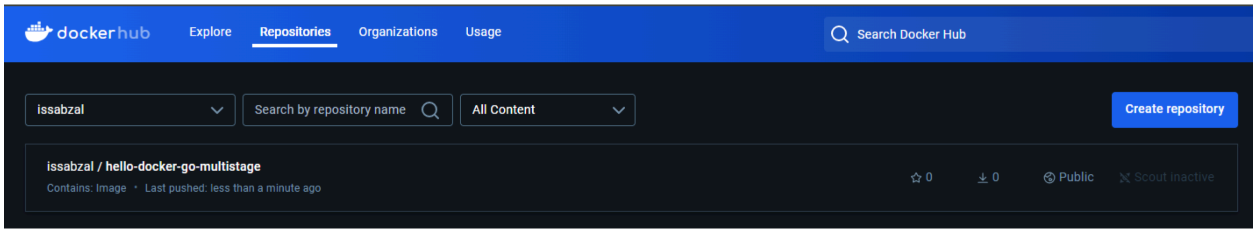
Multi-stage builds are useful when optimizing image size, improving security by excluding build tools, and separating environments for building, testing, and running applications.

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.



```
PS C:\Users\Abzal\Documents\KBTU\WebDev\Assignment1\godocker> docker images
REPOSITORY                 TAG       IMAGE ID       CREATED          SIZE
hello-docker-go-multistage latest    170fef21c287   26 seconds ago   9.93MB
hello-docker-go            latest    9a4f2d1f02cc   5 minutes ago    869MB
hello-docker-opt           latest    cf8ac7f01514   5 hours ago      56.4MB
hello-docker               latest    252aaa11194c   6 hours ago      1.02GB
nginx                      latest    39286ab8a5e1   5 weeks ago      188MB
hello-world                latest    d2c94e258dcb   16 months ago    13.3kB
PS C:\Users\Abzal\Documents\KBTU\WebDev\Assignment1\godocker> docker tag hello-docker-go-multistage issabzal/hello-docker-go-multistage
PS C:\Users\Abzal\Documents\KBTU\WebDev\Assignment1\godocker> docker login
Authenticating with existing credentials...
Login Succeeded
PS C:\Users\Abzal\Documents\KBTU\WebDev\Assignment1\godocker> docker push issabzal/hello-docker-go-multistage
Using default tag: latest
The push refers to repository [docker.io/issabzal/hello-docker-go-multistage]
3495585882b9: Pushed
46d471361bee: Pushed
63ca1fbb43ae: Mounted from library/alpine
latest: digest: sha256:59ec172c71ede30526a030ff1a35a7a9c4a46ff5cbc0dd97b0fdd6ff2dd59dee size: 945
PS C:\Users\Abzal\Documents\KBTU\WebDev\Assignment1\godocker>
```



issabzal / hello-docker-go-multistage
Contains: Image • Last pushed: less than a minute ago
☆ 0 ⤓ 0 Public Scout inactive

3. **Questions**:
    - What is the purpose of Docker Hub in containerization?

The purpose of Docker Hub in containerization is to provide a centralized platform for storing, sharing, and distributing Docker images.

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

For that use **docker tag hello-docker <your-username>/hello-docker** command to tag.

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

For that use **docker login** command first to authenticate then use and **docker push <your-username>/hello-docker** command to push to a remote repository.