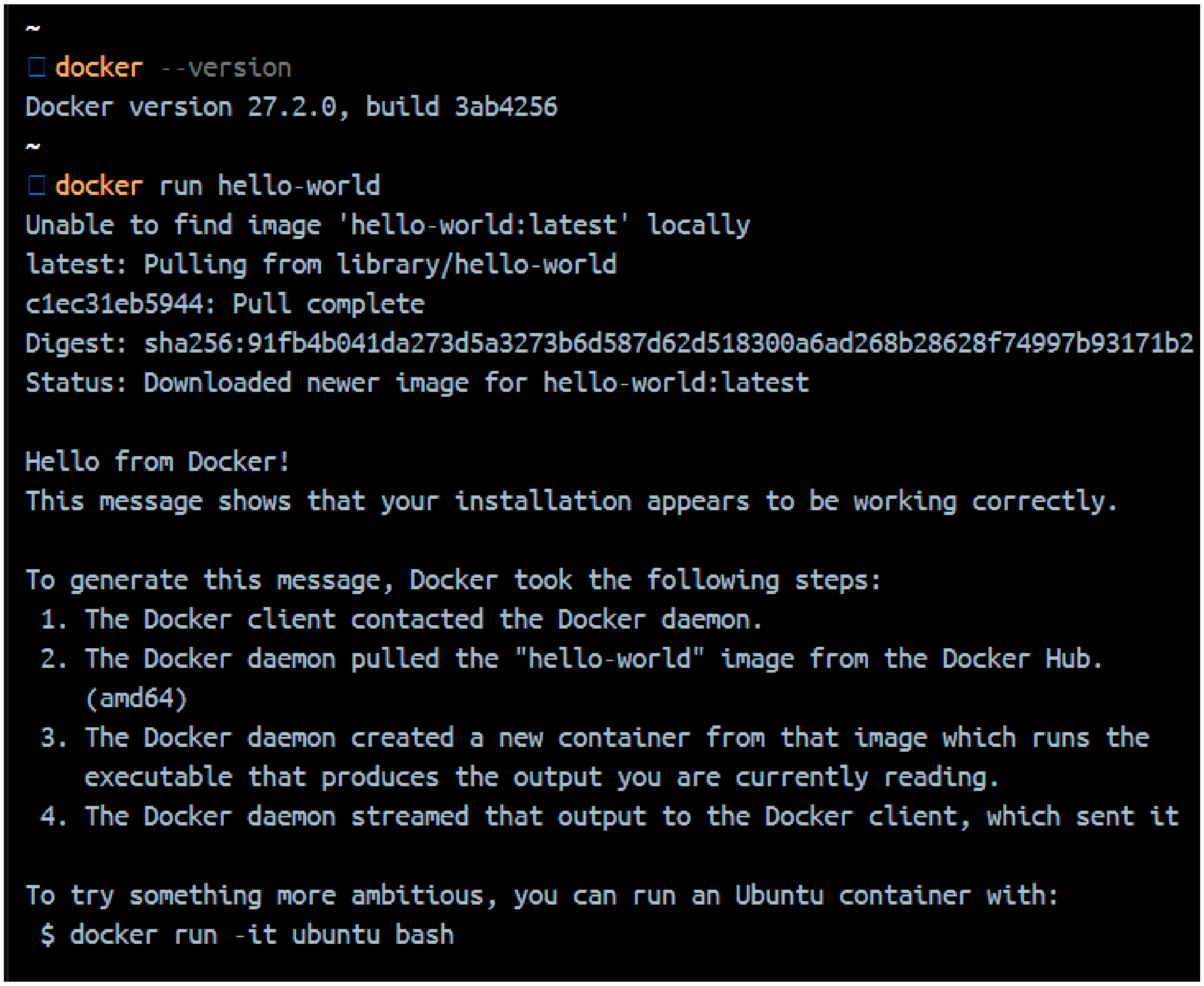
Assignment 1, Web Application Development

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/1FAIpQLSe0GyNdOYlvM1tX_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.
   - Run the command `docker run hello-world` to verify that Docker is set up correctly.



```
~
□ docker --version
Docker version 27.2.0, build 3ab4256
~
□ 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 try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash
```

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

Docker Engine is the core of whole Docker system responsible for functioning of components of Docker(Daemon, CLI, REST Api, Extensions)

Daemon(service) is responsible for background services of docker in host, receiving commands and manages docker components.

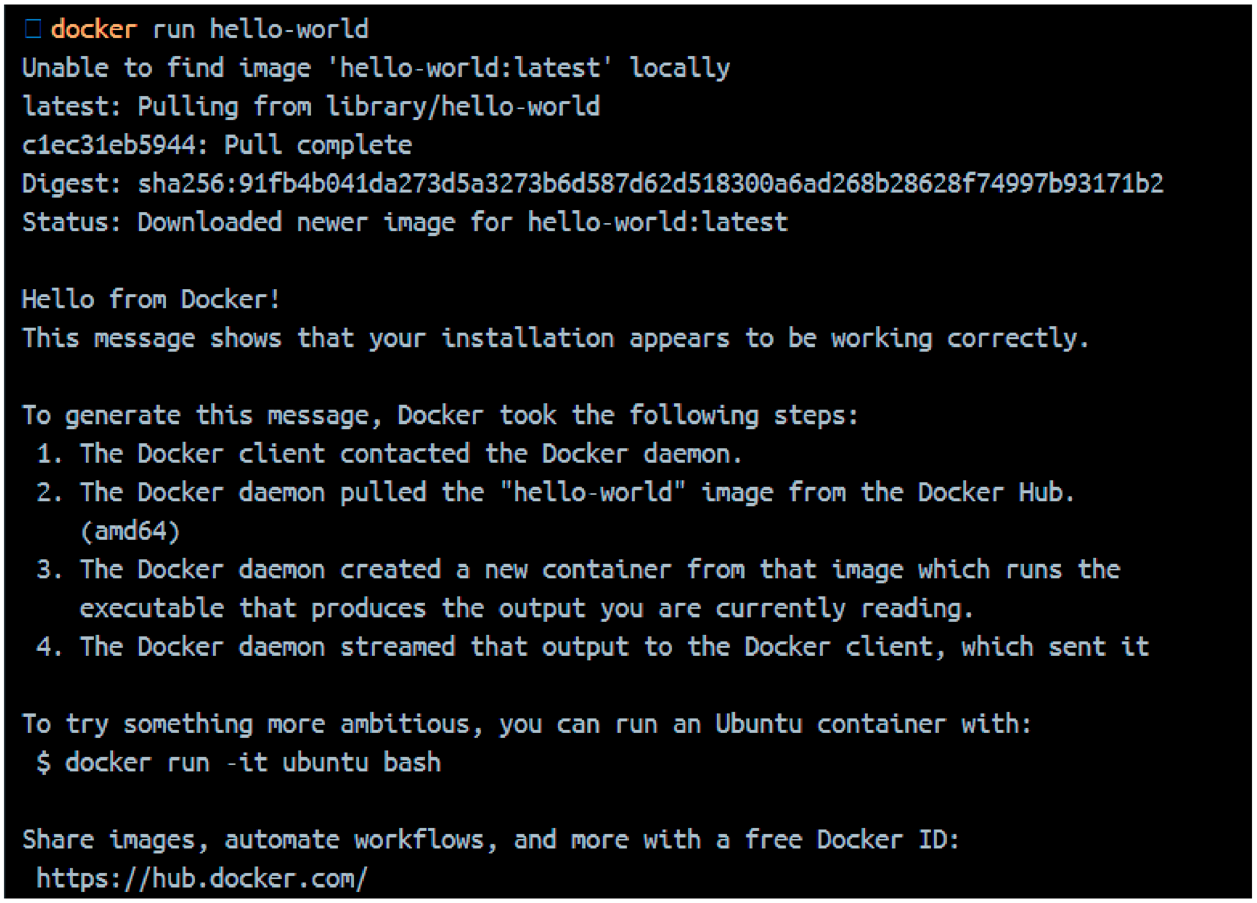
Client(CLI) is responsible for sending commands to docker, creating containers, running and etc.. Simple option for client is command line.

The client uses the docker engine api(rest api) to tell daemon what to do.

How does Docker compare to traditional virtual machines?

Virtual machines have whole guest operation systems. Containers have only app and its depencencies(so they more light and faster).

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

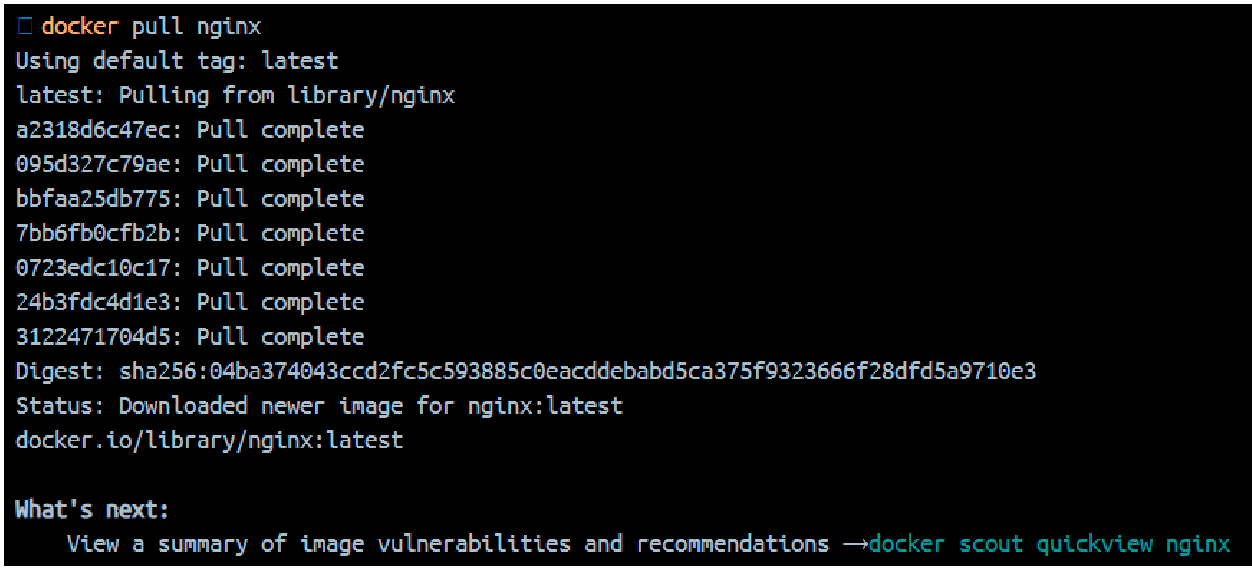


```
□ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
c1ec31eb5944: Pull complete
Digest: sha256:91fb4b041da273d5a3273b6d587d62d518300a6ad268b28628f74997b93171b2
Status: Downloaded newer image for hello-world:latest

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 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/
```

Docker client send command to daemon. Daemon pulled its image from docker hub. Then daemon created container from this image. This c container runs the executable and output showed in client.

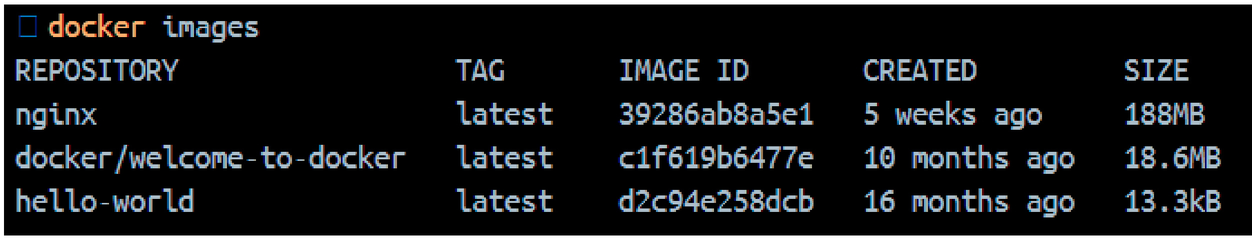
Exercise 2: Basic Docker Commands

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>`.



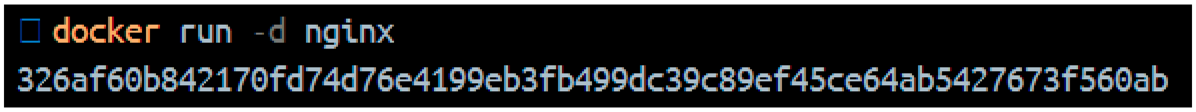
```
 docker pull nginx
Using default tag: latest
latest: Pulling from library/nginx
a2318d6c47ec: Pull complete
095d327c79ae: Pull complete
bbfaa25db775: Pull complete
7bb6fb0cfb2b: Pull complete
0723edc10c17: Pull complete
24b3fdc4d1e3: Pull complete
3122471704d5: Pull complete
Digest: sha256:04ba374043ccd2fc5c593885c0eacddebabd5ca375f9323666f28dfd5a9710e3
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`.



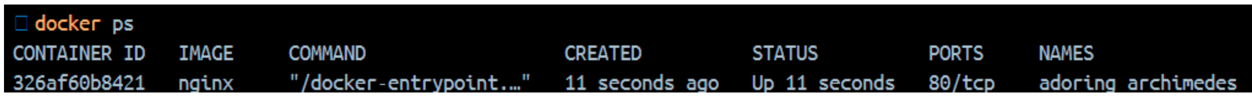
```
 docker images
REPOSITORY                 TAG       IMAGE ID       CREATED         SIZE
nginx                      latest    39286ab8a5e1   5 weeks ago     188MB
docker/welcome-to-docker   latest    c1f619b6477e   10 months ago   18.6MB
hello-world                latest    d2c94e258dcb   16 months ago   13.3kB
```

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

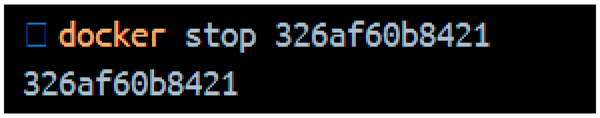


```
 docker run -d nginx
326af60b842170fd74d76e4199eb3fb499dc39c89ef45ce64ab5427673f560ab
```

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



```
 docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS     NAMES
326af60b8421   nginx     "/docker-entrypoint.…"   11 seconds ago   Up 11 seconds   80/tcp    adoring_archimedes
```



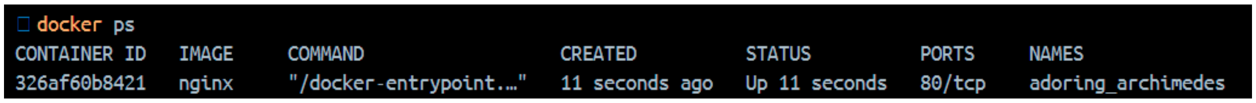
```
 docker stop 326af60b8421
326af60b8421
```

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

First we pull image, then we run the container based on this image in runtime.

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

We can run docker ps to list all running containers.



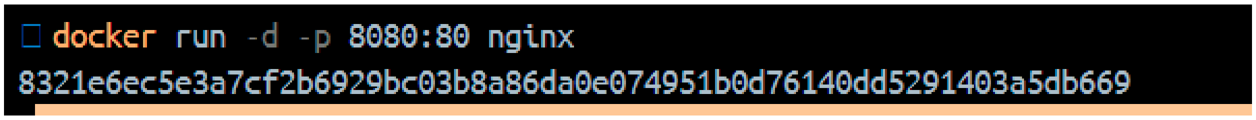
```
 docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS     NAMES
326af60b8421   nginx     "/docker-entrypoint.…"   11 seconds ago   Up 11 seconds   80/tcp    adoring_archimedes
```

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

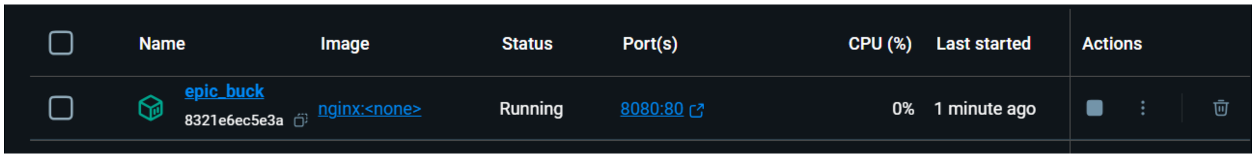
Just pause its processes but preserve state(configuration,files etc.). So we can restart container

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`.

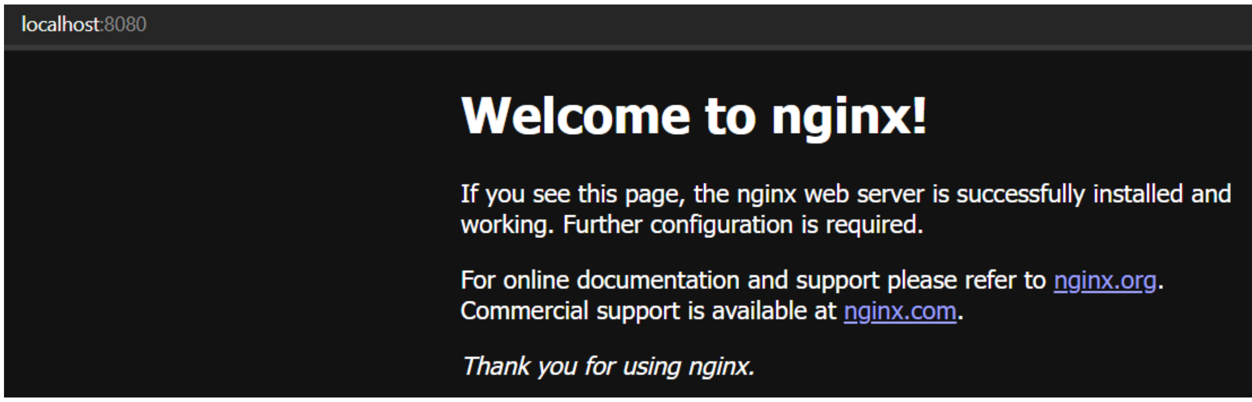


```
docker run -d -p 8080:80 nginx
8321e6ec5e3a7cf2b6929bc03b8a86da0e074951b0d76140dd5291403a5db669
```



	Name	Image	Status	Port(s)	CPU (%)	Last started	Actions
	epic_buck 8321e6ec5e3a	nginx:<none>	Running	8080:80	0%	1 minute ago	

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



localhost:8080

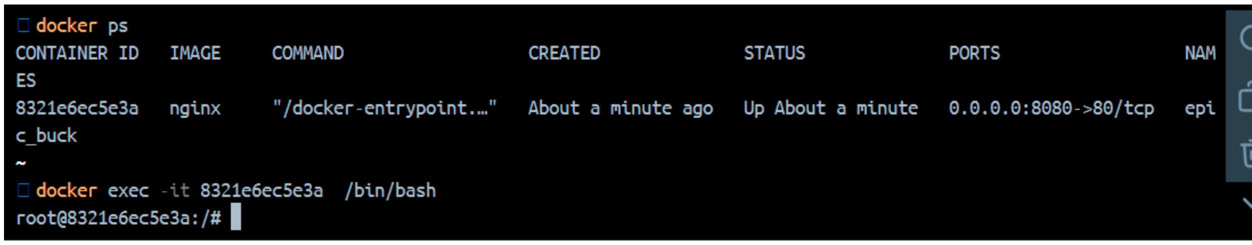
Welcome to nginx!

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.

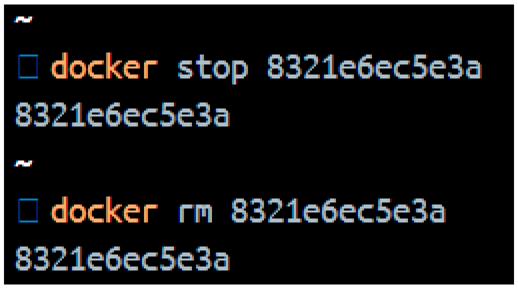
Thank you for using nginx.

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



```
docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED              STATUS              PORTS                  NAM
ES
8321e6ec5e3a   nginx     "/docker-entrypoint.…"   About a minute ago   Up About a minute   0.0.0.0:8080->80/tcp   epi
c_buck
~
docker exec -it 8321e6ec5e3a  /bin/bash
root@8321e6ec5e3a:/#
```

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



```
~
 docker stop 8321e6ec5e3a
8321e6ec5e3a
~
 docker rm 8321e6ec5e3a
8321e6ec5e3a
```

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

Through host ports we can access services of container by local host. Recommended that inner port and host port equal.

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

Runs commands in running container in default directory

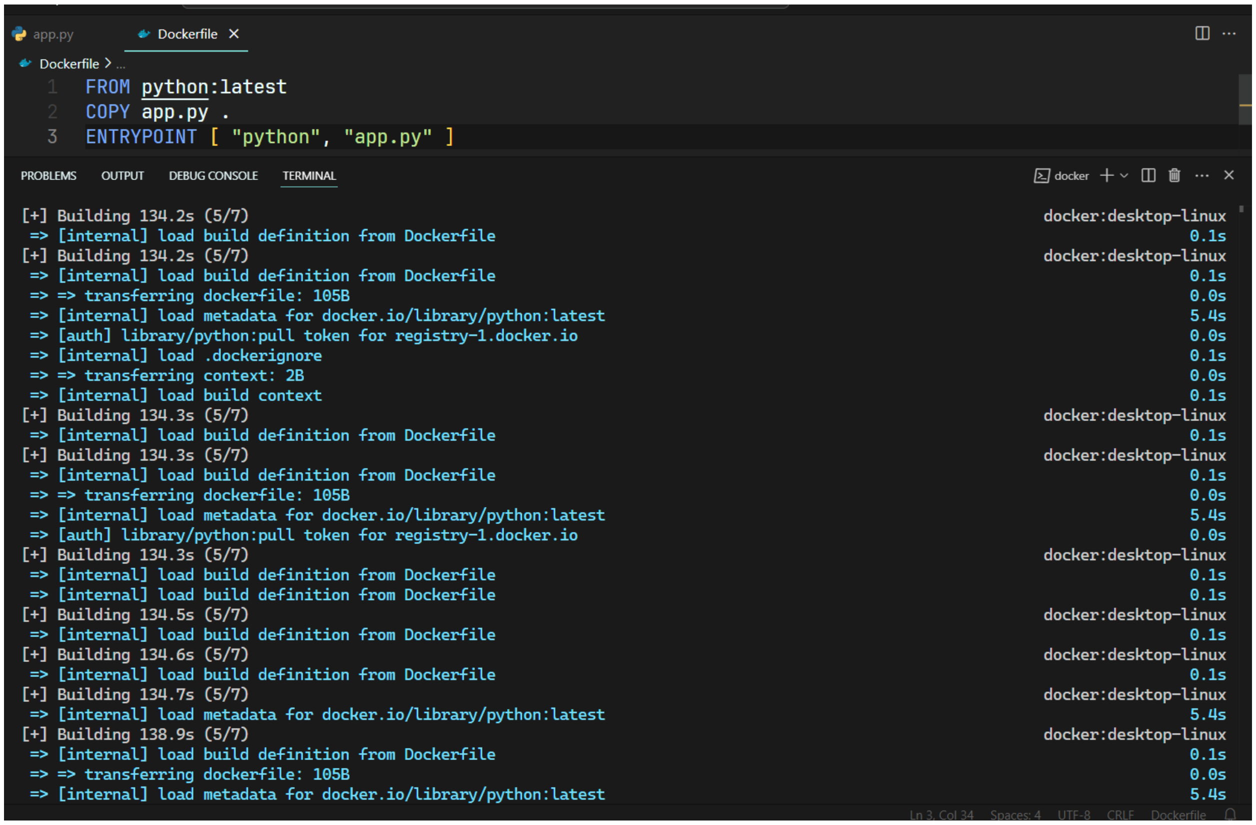
   - How do you ensure that a stopped container does not consume system resources?

We can run docker rm to remove container by id

Dockerfile

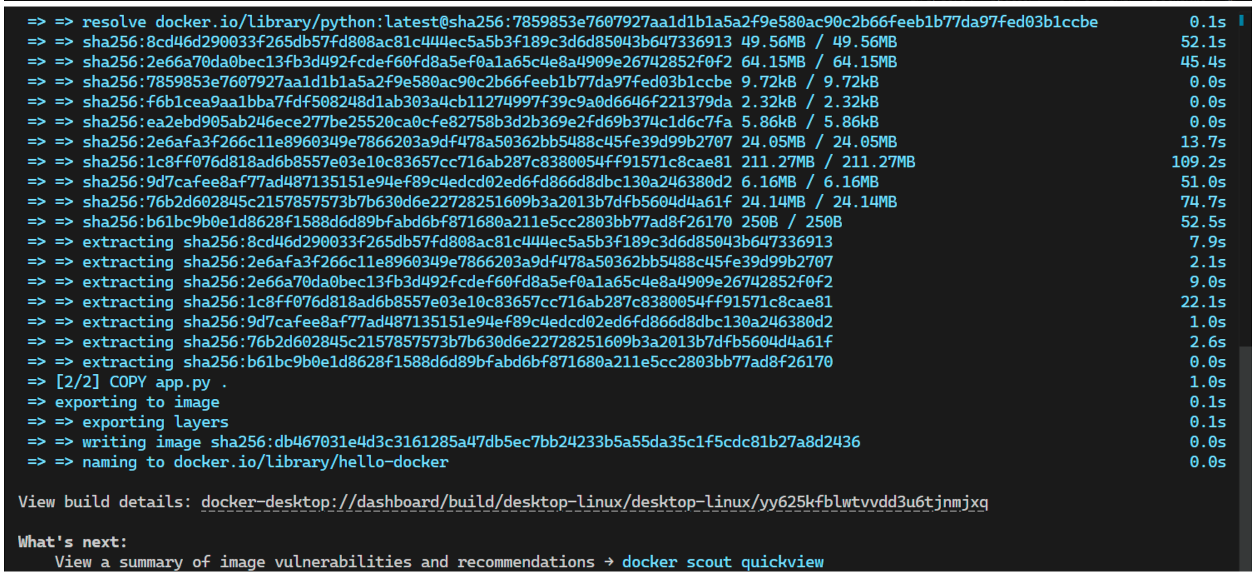
Exercise 1: Creating a Simple Dockerfile

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`.

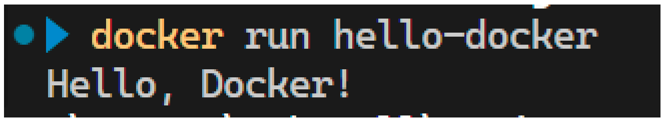


```dockerfile
FROM python:latest
COPY app.py .
ENTRYPOINT [ "python", "app.py" ]
```

```
[+] Building 134.2s (5/7)                                                            docker:desktop-linux
 => [internal] load build definition from Dockerfile                                                 0.1s
[+] Building 134.2s (5/7)                                                            docker:desktop-linux
 => [internal] load build definition from Dockerfile                                                 0.1s
 => => transferring dockerfile: 105B                                                                 0.0s
 => [internal] load metadata for docker.io/library/python:latest                                     5.4s
 => [auth] library/python:pull token for registry-1.docker.io                                        0.0s
 => [internal] load .dockerignore                                                                    0.1s
 => => transferring context: 2B                                                                      0.0s
 => [internal] load build context                                                                    0.1s
[+] Building 134.3s (5/7)                                                            docker:desktop-linux
 => [internal] load build definition from Dockerfile                                                 0.1s
[+] Building 134.3s (5/7)                                                            docker:desktop-linux
 => [internal] load build definition from Dockerfile                                                 0.1s
 => => transferring dockerfile: 105B                                                                 0.0s
 => [internal] load metadata for docker.io/library/python:latest                                     5.4s
 => [auth] library/python:pull token for registry-1.docker.io                                        0.0s
[+] Building 134.3s (5/7)                                                            docker:desktop-linux
 => [internal] load build definition from Dockerfile                                                 0.1s
 => [internal] load build definition from Dockerfile                                                 0.1s
[+] Building 134.5s (5/7)                                                            docker:desktop-linux
 => [internal] load build definition from Dockerfile                                                 0.1s
[+] Building 134.6s (5/7)                                                            docker:desktop-linux
 => [internal] load build definition from Dockerfile                                                 0.1s
[+] Building 134.7s (5/7)                                                            docker:desktop-linux
 => [internal] load metadata for docker.io/library/python:latest                                     5.4s
[+] Building 138.9s (5/7)                                                            docker:desktop-linux
 => [internal] load build definition from Dockerfile                                                 0.1s
 => => transferring dockerfile: 105B                                                                 0.0s
 => [internal] load metadata for docker.io/library/python:latest                                     5.4s
```



```
 => => resolve docker.io/library/python:latest@sha256:7859853e7607927aa1d1b1a5a2f9e580ac90c2b66feeb1b77da97fed03b1ccbe      0.1s
 => => sha256:8cd46d290033f265db57fd808ac81c444ec5a5b3f189c3d6d85043b647336913 49.56MB / 49.56MB                           52.1s
 => => sha256:2e66a70da0bec13fb3d492fcdef60fd8a5ef0a1a65c4e8a4909e26742852f0f2 64.15MB / 64.15MB                           45.4s
 => => sha256:7859853e7607927aa1d1b1a5a2f9e580ac90c2b66feeb1b77da97fed03b1ccbe 9.72kB / 9.72kB                              0.0s
 => => sha256:f6b1cea9aa1bba7fdf508248d1ab303a4cb11274997f39c9a0d6646f221379da 2.32kB / 2.32kB                              0.0s
 => => sha256:ea2ebd905ab246ece277be25520ca0cfe82758b3d2b369e2fd69b374c1d6c7fa 5.86kB / 5.86kB                              0.0s
 => => sha256:2e6afa3f266c11e8960349e7866203a9df478a50362bb5488c45fe39d99b2707 24.05MB / 24.05MB                           13.7s
 => => sha256:1c8ff076d818ad6b8557e03e10c83657cc716ab287c8380054ff91571c8cae81 211.27MB / 211.27MB                        109.2s
 => => sha256:9d7cafee8af77ad487135151e94ef89c4edcd02ed6fd866d8dbc130a246380d2 6.16MB / 6.16MB                             51.0s
 => => sha256:76b2d602845c2157857573b7b630d6e22728251609b3a2013b7dfb5604d4a61f 24.14MB / 24.14MB                           74.7s
 => => sha256:b61bc9b0e1d8628f1588d6d89bfabd6bf871680a211e5cc2803bb77ad8f26170 250B / 250B                                 52.5s
 => => extracting sha256:8cd46d290033f265db57fd808ac81c444ec5a5b3f189c3d6d85043b647336913                                   7.9s
 => => extracting sha256:2e6afa3f266c11e8960349e7866203a9df478a50362bb5488c45fe39d99b2707                                   2.1s
 => => extracting sha256:2e66a70da0bec13fb3d492fcdef60fd8a5ef0a1a65c4e8a4909e26742852f0f2                                   9.0s
 => => extracting sha256:1c8ff076d818ad6b8557e03e10c83657cc716ab287c8380054ff91571c8cae81                                  22.1s
 => => extracting sha256:9d7cafee8af77ad487135151e94ef89c4edcd02ed6fd866d8dbc130a246380d2                                   1.0s
 => => extracting sha256:76b2d602845c2157857573b7b630d6e22728251609b3a2013b7dfb5604d4a61f                                   2.6s
 => => extracting sha256:b61bc9b0e1d8628f1588d6d89bfabd6bf871680a211e5cc2803bb77ad8f26170                                   0.0s
 => [2/2] COPY app.py .                                                                                                       1.0s
 => exporting to image                                                                                                        0.1s
 => => exporting layers                                                                                                       0.1s
 => => writing image sha256:db467031e4d3c3161285a47db5ec7bb24233b5a55da35c1f5cdc81b27a8d2436                                0.0s
 => => naming to docker.io/library/hello-docker                                                                               0.0s

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/yy625kfblwtvvdd3u6tjnmjxq

What's next:
    View a summary of image vulnerabilities and recommendations → docker scout quickview
```



```
docker run hello-docker
Hello, Docker!
```

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

    The base image for our image

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

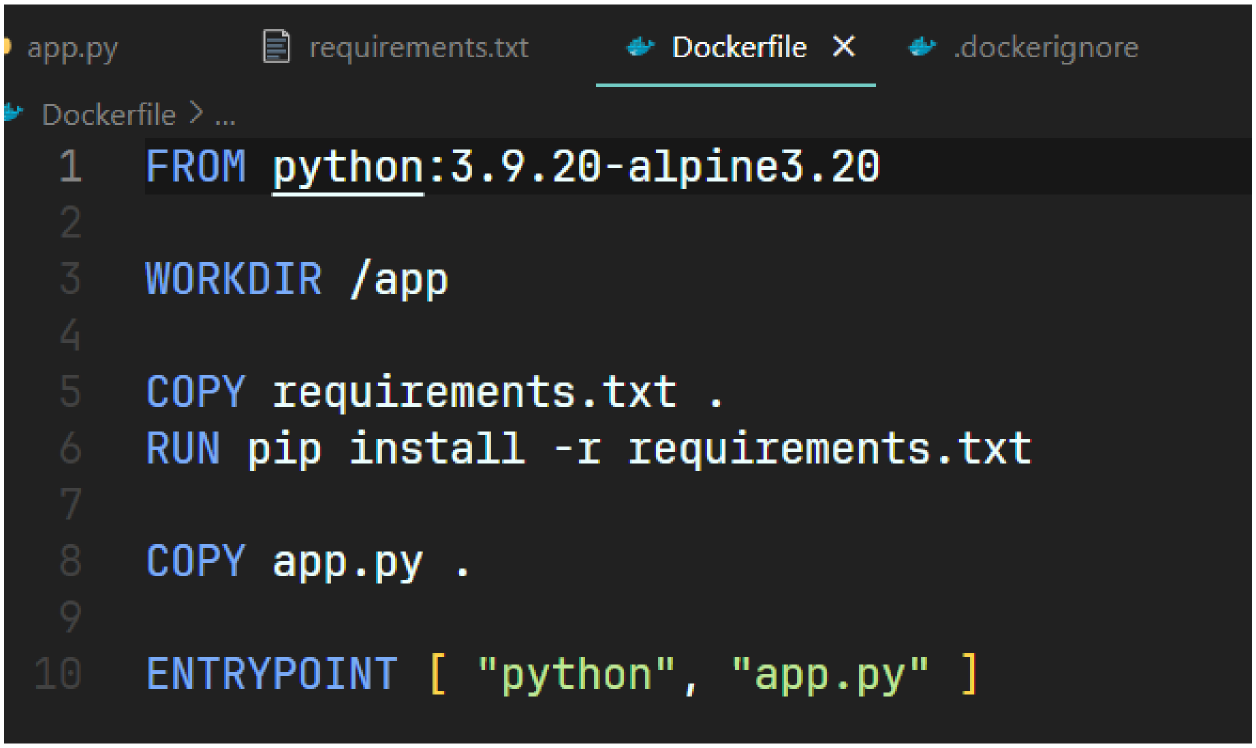
    Copy files or directories from our host machine to container.

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

Entry point we can define main first execute file for container. CMD like default arguments for our file, and they can be overrided.

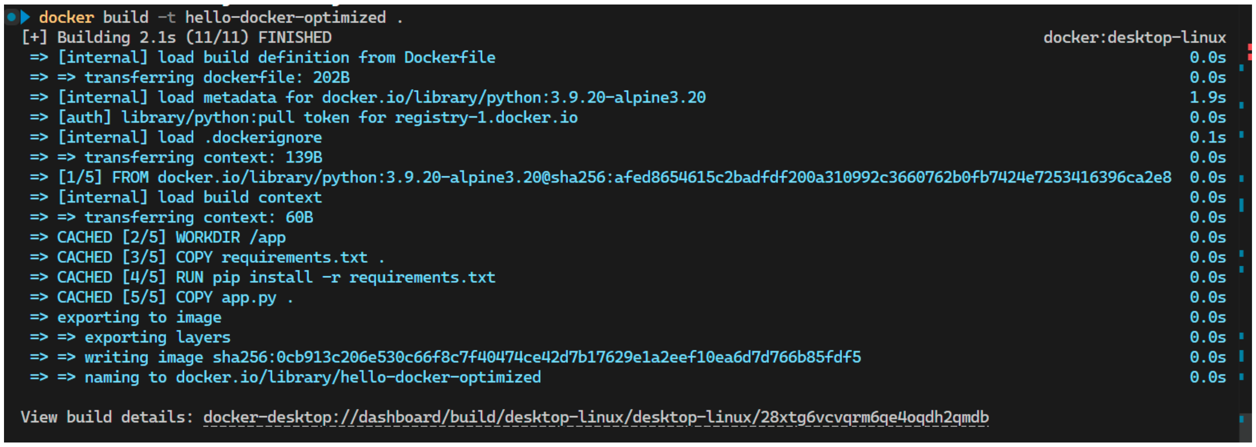
Exercise 2: Optimizing Dockerfile with Layers and Caching

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.



```
FROM python:3.9.20-alpine3.20

WORKDIR /app

COPY requirements.txt .
RUN pip install -r requirements.txt

COPY app.py .

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

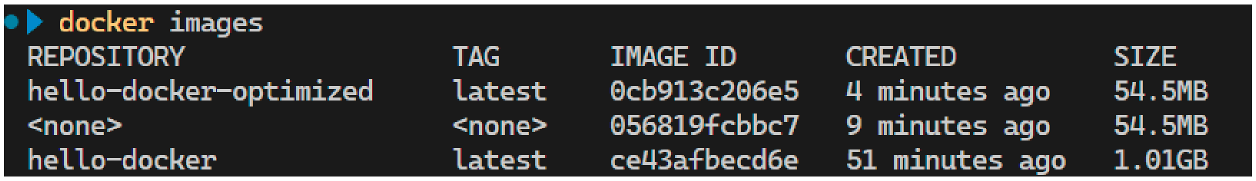
    - Rebuild the Docker image and observe the build process to understand how caching works.



```
docker build -t hello-docker-optimized .
[+] Building 2.1s (11/11) FINISHED                                                                    docker:desktop-linux
 => [internal] load build definition from Dockerfile                                                                  0.0s
 => => transferring dockerfile: 202B                                                                                  0.0s
 => [internal] load metadata for docker.io/library/python:3.9.20-alpine3.20                                           1.9s
 => [auth] library/python:pull token for registry-1.docker.io                                                         0.0s
 => [internal] load .dockerignore                                                                                     0.1s
 => => transferring context: 139B                                                                                     0.0s
 => [1/5] FROM docker.io/library/python:3.9.20-alpine3.20@sha256:afed8654615c2badfdf200a310992c3660762b0fb7424e7253416396ca2e8  0.0s
 => [internal] load build context                                                                                     0.0s
 => => transferring context: 60B                                                                                      0.0s
 => CACHED [2/5] WORKDIR /app                                                                                         0.0s
 => CACHED [3/5] COPY requirements.txt .                                                                              0.0s
 => CACHED [4/5] RUN pip install -r requirements.txt                                                                  0.0s
 => CACHED [5/5] COPY app.py .                                                                                        0.0s
 => exporting to image                                                                                                0.0s
 => => exporting layers                                                                                               0.0s
 => => writing image sha256:0cb913c206e530c66f8c7f40474ce42d7b17629e1a2eef10ea6d7d766b85fdf5                          0.0s
 => => naming to docker.io/library/hello-docker-optimized                                                             0.0s

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/28xtg6vcvqrm6qe4oqdh2qmdb
```

Docker caches each step of image building process. If step is not changed docker skips building this step and just uses cached result.

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



```
docker images
REPOSITORY               TAG       IMAGE ID       CREATED          SIZE
hello-docker-optimized   latest    0cb913c206e5   4 minutes ago    54.5MB
<none>                   <none>    056819fcbbc7   9 minutes ago    54.5MB
hello-docker             latest    ce43afbecd6e   51 minutes ago   1.01GB
```

I used alpine image of python. So this image very light.

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

   Docker images consist from layers. And each step in dockerfile creates a new layer. If step is not changed, it will build faster. And more efficient layering can reduce size of image.
   - How does Docker's build cache work, and how can it speed up the build process?

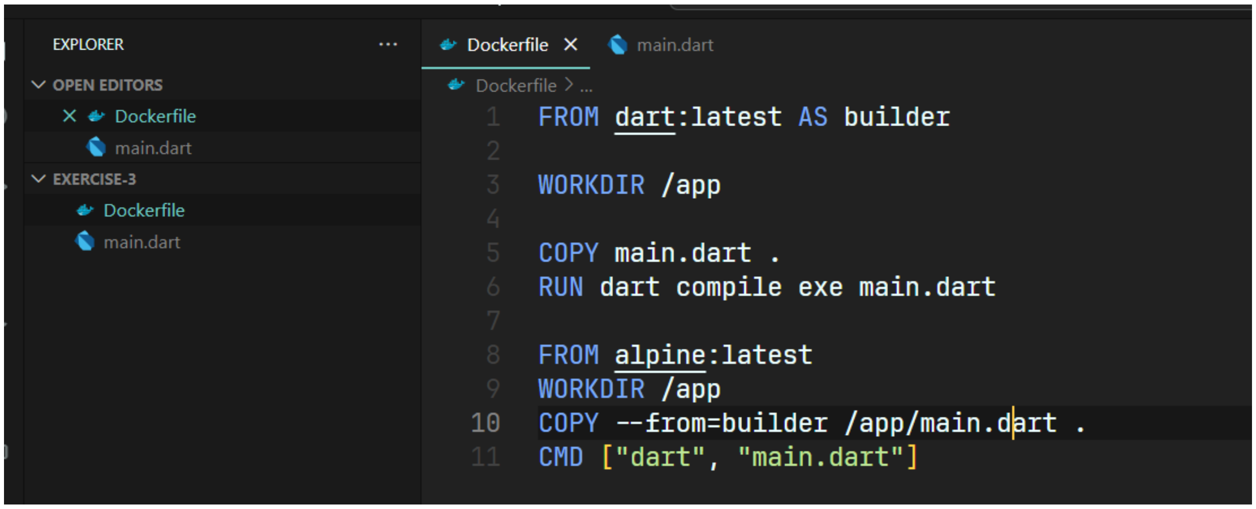
   Docker caches each step of the image building. If step not changed Docker skips rebuilding that step and uses the cached result.
   - What is the role of the `.dockerignore` file?

   It is like .gitignore, we specify which files and directories to exclude from build.

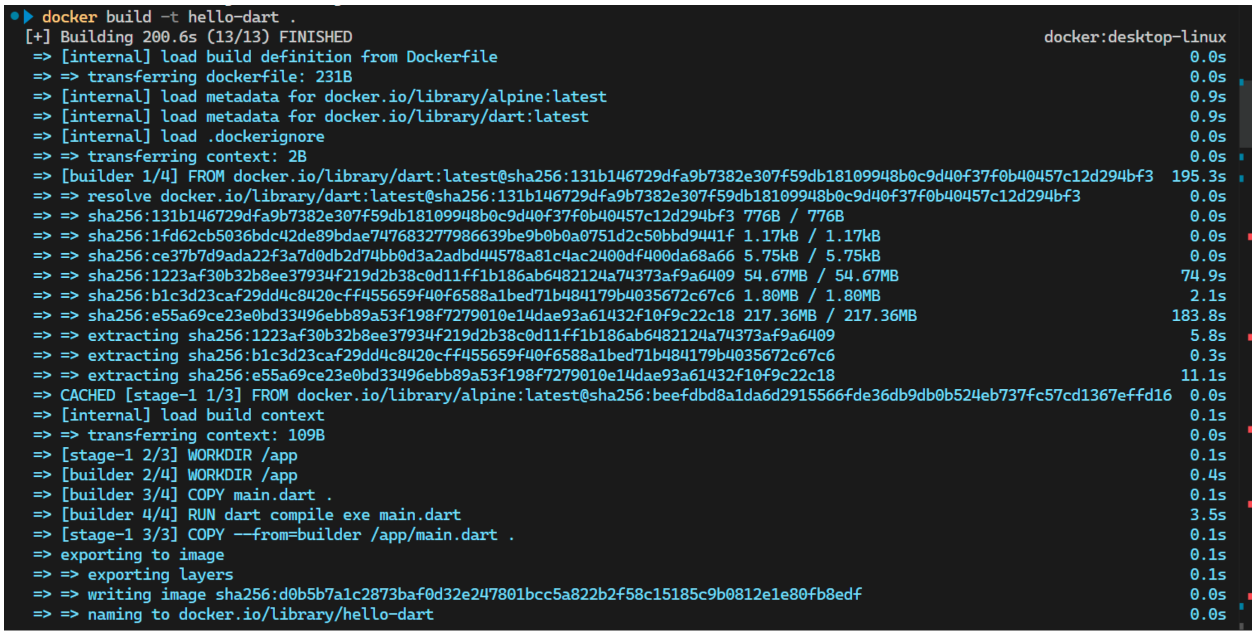
Exercise 3: Multi-Stage Builds

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.

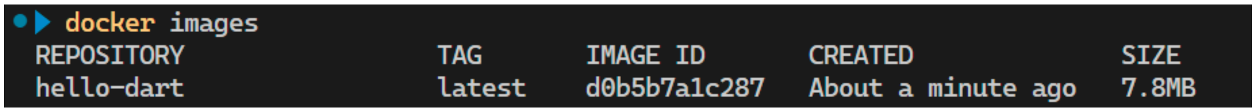
- Build and run the Docker image, and compare the size of the final image with a single-stage build.



```dockerfile
FROM dart:latest AS builder

WORKDIR /app

COPY main.dart .
RUN dart compile exe main.dart

FROM alpine:latest
WORKDIR /app
COPY --from=builder /app/main.dart .
CMD ["dart", "main.dart"]
```



```
docker build -t hello-dart .
[+] Building 200.6s (13/13) FINISHED                                                                              docker:desktop-linux
 => [internal] load build definition from Dockerfile                                                                             0.0s
 => => transferring dockerfile: 231B                                                                                             0.0s
 => [internal] load metadata for docker.io/library/alpine:latest                                                                 0.9s
 => [internal] load metadata for docker.io/library/dart:latest                                                                   0.9s
 => [internal] load .dockerignore                                                                                                0.0s
 => => transferring context: 2B                                                                                                  0.0s
 => [builder 1/4] FROM docker.io/library/dart:latest@sha256:131b146729dfa9b7382e307f59db18109948b0c9d40f37f0b40457c12d294bf3  195.3s
 => => resolve docker.io/library/dart:latest@sha256:131b146729dfa9b7382e307f59db18109948b0c9d40f37f0b40457c12d294bf3            0.0s
 => => sha256:131b146729dfa9b7382e307f59db18109948b0c9d40f37f0b40457c12d294bf3 776B / 776B                                       0.0s
 => => sha256:1fd62cb5036bdc42de89bdae747683277986639be9b0b0a0751d2c50bbd9441f 1.17kB / 1.17kB                                   0.0s
 => => sha256:ce37b7d9ada22f3a7d0db2d74bb0d3a2adbd44578a81c4ac2400df400da68a66 5.75kB / 5.75kB                                   0.0s
 => => sha256:1223af30b32b8ee37934f219d2b38c0d11ff1b186ab6482124a74373af9a6409 54.67MB / 54.67MB                                74.9s
 => => sha256:b1c3d23caf29dd4c8420cff455659f40f6588a1bed71b484179b4035672c67c6 1.80MB / 1.80MB                                   2.1s
 => => sha256:e55a69ce23e0bd33496ebb89a53f198f7279010e14dae93a61432f10f9c22c18 217.36MB / 217.36MB                             183.8s
 => => extracting sha256:1223af30b32b8ee37934f219d2b38c0d11ff1b186ab6482124a74373af9a6409                                        5.8s
 => => extracting sha256:b1c3d23caf29dd4c8420cff455659f40f6588a1bed71b484179b4035672c67c6                                        0.3s
 => => extracting sha256:e55a69ce23e0bd33496ebb89a53f198f7279010e14dae93a61432f10f9c22c18                                       11.1s
 => CACHED [stage-1 1/3] FROM docker.io/library/alpine:latest@sha256:beefdbd8a1da6d2915566fde36db9db0b524eb737fc57cd1367effd16  0.0s
 => [internal] load build context                                                                                                0.1s
 => => transferring context: 109B                                                                                                0.0s
 => [stage-1 2/3] WORKDIR /app                                                                                                   0.1s
 => [builder 2/4] WORKDIR /app                                                                                                   0.4s
 => [builder 3/4] COPY main.dart .                                                                                               0.1s
 => [builder 4/4] RUN dart compile exe main.dart                                                                                 3.5s
 => [stage-1 3/3] COPY --from=builder /app/main.dart .                                                                           0.1s
 => exporting to image                                                                                                           0.1s
 => => exporting layers                                                                                                          0.1s
 => => writing image sha256:d0b5b7a1c2873baf0d32e247801bcc5a822b2f58c15185c9b0812e1e80fb8edf                                     0.0s
 => => naming to docker.io/library/hello-dart                                                                                    0.0s
```



```
docker images
REPOSITORY                     TAG       IMAGE ID       CREATED              SIZE
hello-dart                     latest    d0b5b7a1c287   About a minute ago   7.8MB
```

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

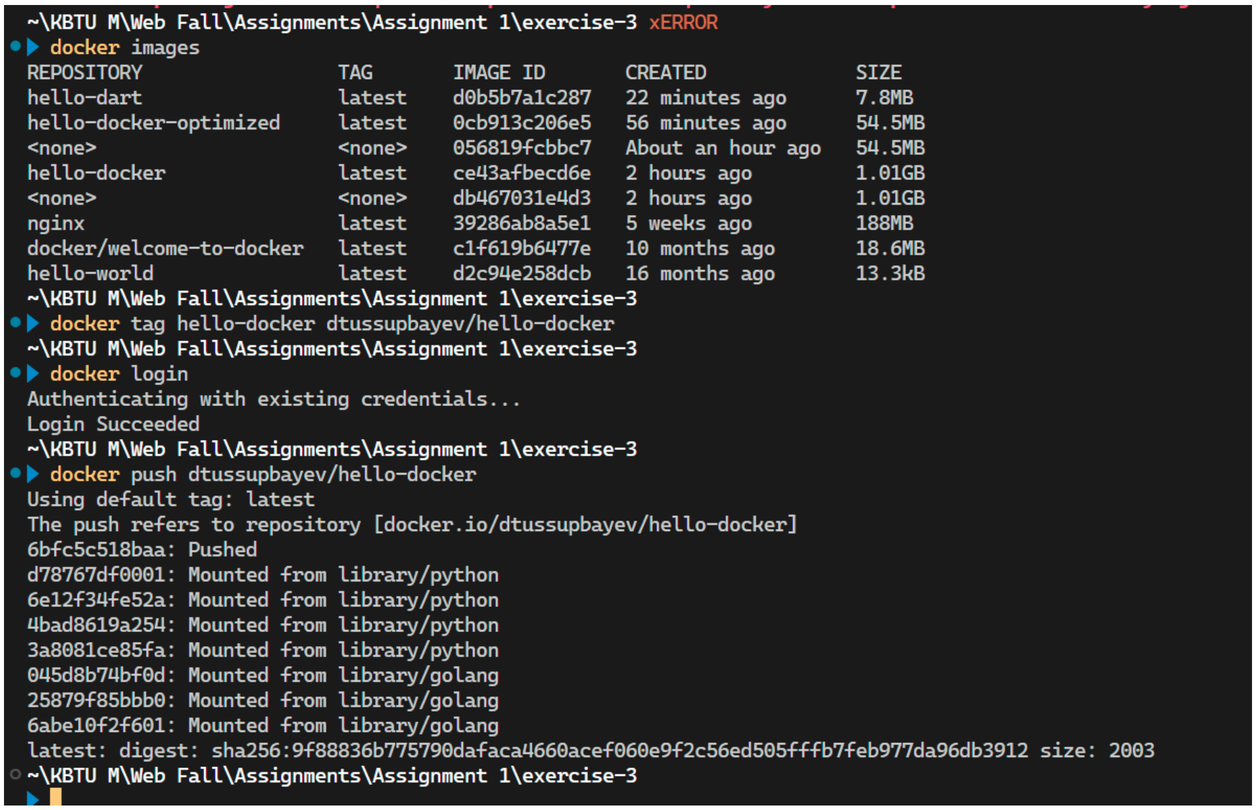
   Separating build and runtime stages, including only necessary components in the final image
   - How can multi-stage builds help reduce the size of Docker images?

   Avoid compilers, development dependencies
   - What are some scenarios where multi-stage builds are particularly useful?

   Language compile, build assets

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.



```
~\KBTU M\Web Fall\Assignments\Assignment 1\exercise-3 xERROR
 docker images
REPOSITORY                TAG       IMAGE ID       CREATED          SIZE
hello-dart                latest    d0b5b7a1c287   22 minutes ago   7.8MB
hello-docker-optimized    latest    0cb913c206e5   56 minutes ago   54.5MB
<none>                    <none>    056819fcbbc7   About an hour ago   54.5MB
hello-docker              latest    ce43afbecd6e   2 hours ago      1.01GB
<none>                    <none>    db467031e4d3   2 hours ago      1.01GB
nginx                     latest    39286ab8a5e1   5 weeks ago      188MB
docker/welcome-to-docker  latest    c1f619b6477e   10 months ago    18.6MB
hello-world               latest    d2c94e258dcb   16 months ago    13.3kB
~\KBTU M\Web Fall\Assignments\Assignment 1\exercise-3
 docker tag hello-docker dtussupbayev/hello-docker
~\KBTU M\Web Fall\Assignments\Assignment 1\exercise-3
 docker login
Authenticating with existing credentials...
Login Succeeded
~\KBTU M\Web Fall\Assignments\Assignment 1\exercise-3
 docker push dtussupbayev/hello-docker
Using default tag: latest
The push refers to repository [docker.io/dtussupbayev/hello-docker]
6bfc5c518baa: Pushed
d78767df0001: Mounted from library/python
6e12f34fe52a: Mounted from library/python
4bad8619a254: Mounted from library/python
3a8081ce85fa: Mounted from library/python
045d8b74bf0d: Mounted from library/golang
25879f85bbb0: Mounted from library/golang
6abe10f2f601: Mounted from library/golang
latest: digest: sha256:9f88836b775790dafaca4660acef060e9f2c56ed505fffb7feb977da96db3912 size: 2003
~\KBTU M\Web Fall\Assignments\Assignment 1\exercise-3
```

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?