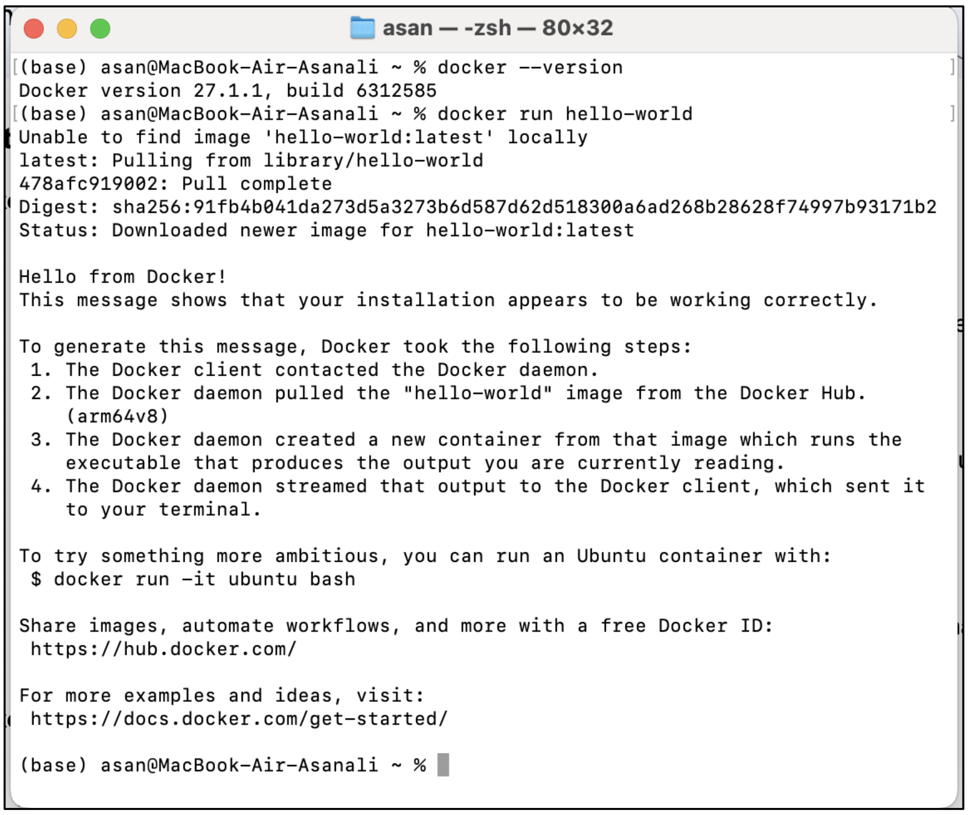
Web Application Development

Bitanov Asanali 23MD0392

Assignment 1

Intro to Containerization: Docker

Exercise 1. Installing Docker.



```
asan — -zsh — 80×32
(base) asan@MacBook-Air-Asanali ~ % docker --version
Docker version 27.1.1, build 6312585
(base) asan@MacBook-Air-Asanali ~ % docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
478afc919002: 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.
    (arm64v8)
 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) asan@MacBook-Air-Asanali ~ %
```

Fig 1. Installing Docker

Questions:

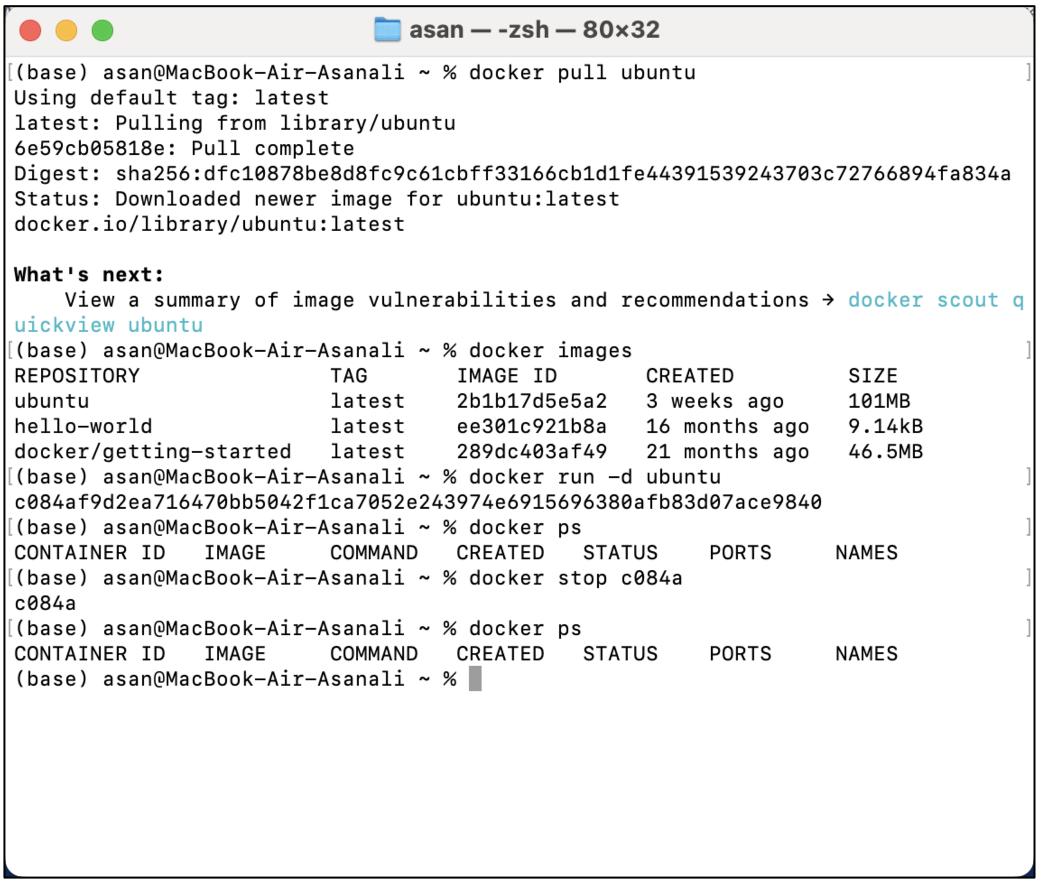
What are the key components of Docker (e.g., Docker Engine, Docker CLI)?
The key components of Docker are Docker Engine, Docker Images, Docker Containers, Dockerfile, Docker Hub, Docker Compose, Docker Volumes, Docker Network, and Docker Registry.

How does Docker compare to traditional virtual machines?
Docker is more lightweight compared to VMs as it requires fewer computing resources.

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

The command's output is shown in the figure below (Figure 1). It says there weren't any image names "hello-world" on the computer, so it downloaded it from the Docker Hub.

Exercise 2. Basic Docker Commands.



```
asan — -zsh — 80×32
(base) asan@MacBook-Air-Asanali ~ % docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
6e59cb05818e: Pull complete
Digest: sha256:dfc10878be8d8fc9c61cbff33166cb1d1fe44391539243703c72766894fa834a
Status: Downloaded newer image for ubuntu:latest
docker.io/library/ubuntu:latest

What's next:
    View a summary of image vulnerabilities and recommendations → docker scout q
uickview ubuntu
(base) asan@MacBook-Air-Asanali ~ % docker images
REPOSITORY               TAG       IMAGE ID       CREATED         SIZE
ubuntu                   latest    2b1b17d5e5a2   3 weeks ago     101MB
hello-world              latest    ee301c921b8a   16 months ago   9.14kB
docker/getting-started   latest    289dc403af49   21 months ago   46.5MB
(base) asan@MacBook-Air-Asanali ~ % docker run -d ubuntu
c084af9d2ea716470bb5042f1ca7052e243974e6915696380afb83d07ace9840
(base) asan@MacBook-Air-Asanali ~ % docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
(base) asan@MacBook-Air-Asanali ~ % docker stop c084a
c084a
(base) asan@MacBook-Air-Asanali ~ % docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
(base) asan@MacBook-Air-Asanali ~ %
```

Fig 2. Basic Docker Commands

Questions:

What is the difference between docker pull and docker run?

Docker pull downloads images, while docker run launches them. And if we don't have an image, docker run would download it first and then launch it.

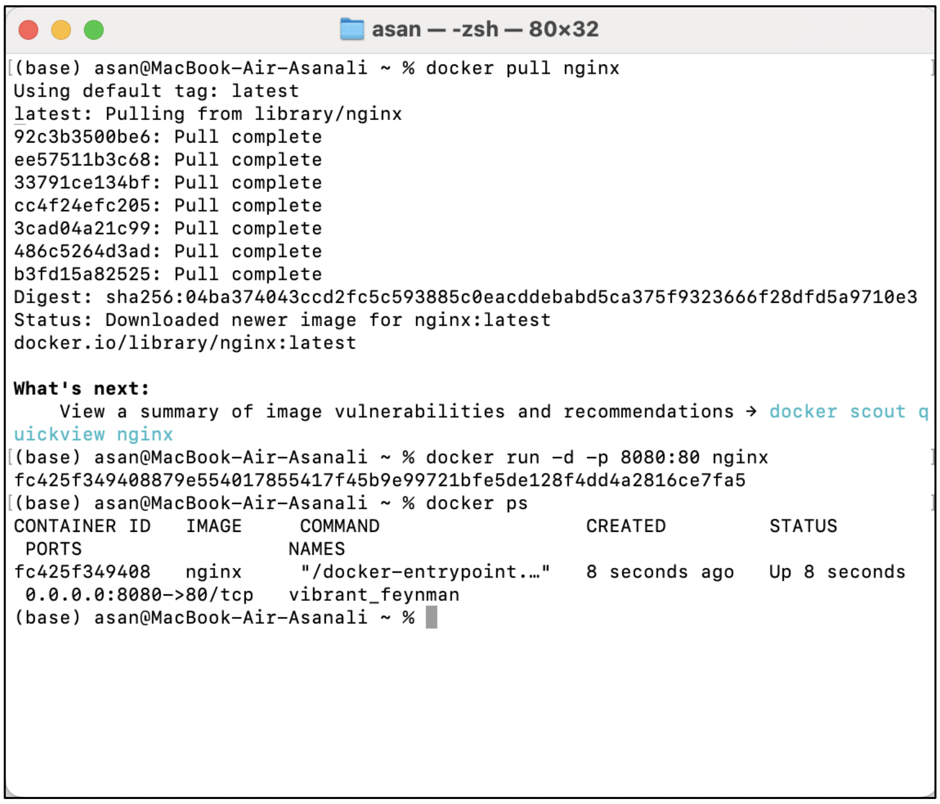
How do you find the details of a running container, such as its ID and status?

We use the command - docker ps.

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

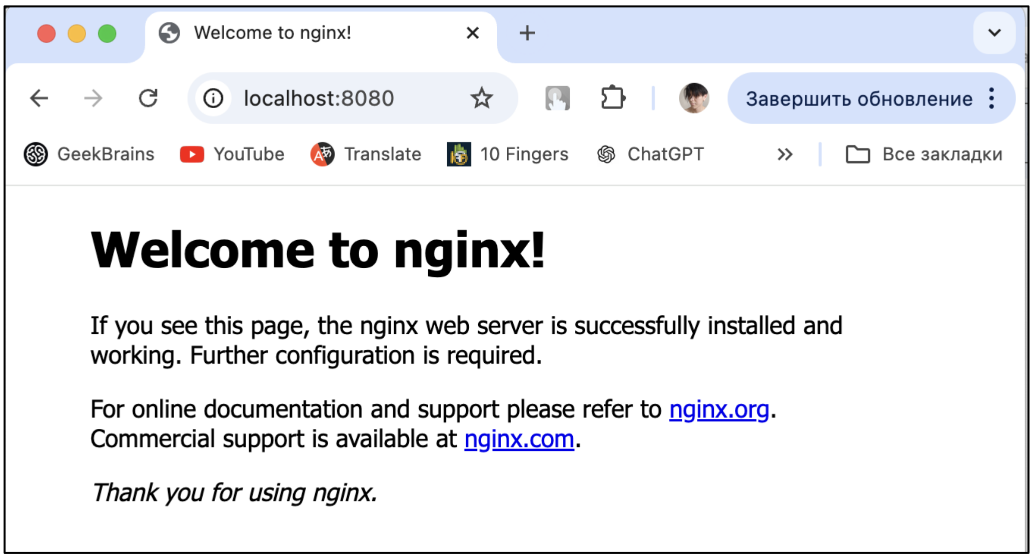
If the changes are saved, they will be stored correctly on a local machine. If not, it would be stored without any changes. We restart a container by using docker start <container name or id>

Exercise 3. Working with Docker Containers.



```
(base) asan@MacBook-Air-Asanali ~ % docker pull nginx
Using default tag: latest
latest: Pulling from library/nginx
92c3b3500be6: Pull complete
ee57511b3c68: Pull complete
33791ce134bf: Pull complete
cc4f24efc205: Pull complete
3cad04a21c99: Pull complete
486c5264d3ad: Pull complete
b3fd15a82525: 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 q
uickview nginx
(base) asan@MacBook-Air-Asanali ~ % docker run -d -p 8080:80 nginx
fc425f349408879e554017855417f45b9e99721bfe5de128f4dd4a2816ce7fa5
(base) asan@MacBook-Air-Asanali ~ % docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS
 PORTS                  NAMES
fc425f349408   nginx     "/docker-entrypoint.…"   8 seconds ago   Up 8 seconds
 0.0.0.0:8080->80/tcp   vibrant_feynman
(base) asan@MacBook-Air-Asanali ~ %
```

Fig 3. Running nginx



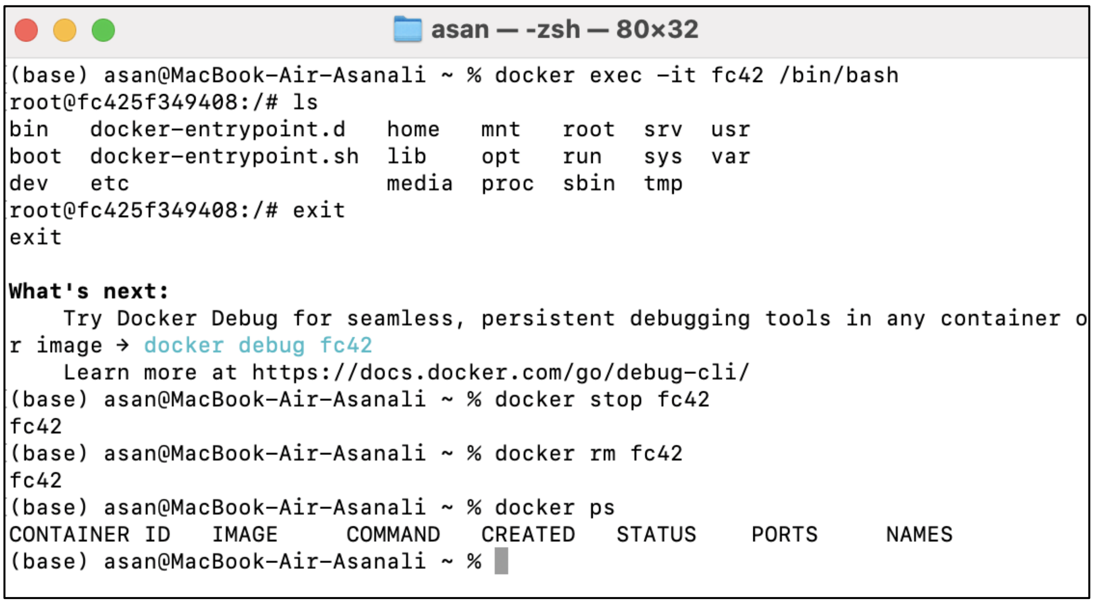
Welcome to nginx!

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.

Fig 4. Localhost page



```
(base) asan@MacBook-Air-Asanali ~ % docker exec -it fc42 /bin/bash
root@fc425f349408:/# ls
bin   docker-entrypoint.d   home  mnt   root  srv  usr
boot  docker-entrypoint.sh  lib   opt   run   sys  var
dev   etc                   media proc  sbin  tmp
root@fc425f349408:/# exit
exit

What's next:
    Try Docker Debug for seamless, persistent debugging tools in any container o
r image → docker debug fc42
    Learn more at https://docs.docker.com/go/debug-cli/
(base) asan@MacBook-Air-Asanali ~ % docker stop fc42
fc42
(base) asan@MacBook-Air-Asanali ~ % docker rm fc42
fc42
(base) asan@MacBook-Air-Asanali ~ % docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
(base) asan@MacBook-Air-Asanali ~ % 
```

Fig 5. Exploring the container and stopping it

Questions:

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

We use parameter -p to set up a port. It allows us to access applications running inside the container from an external environment, providing isolation and flexibility.

What is the purpose of the docker exec command?

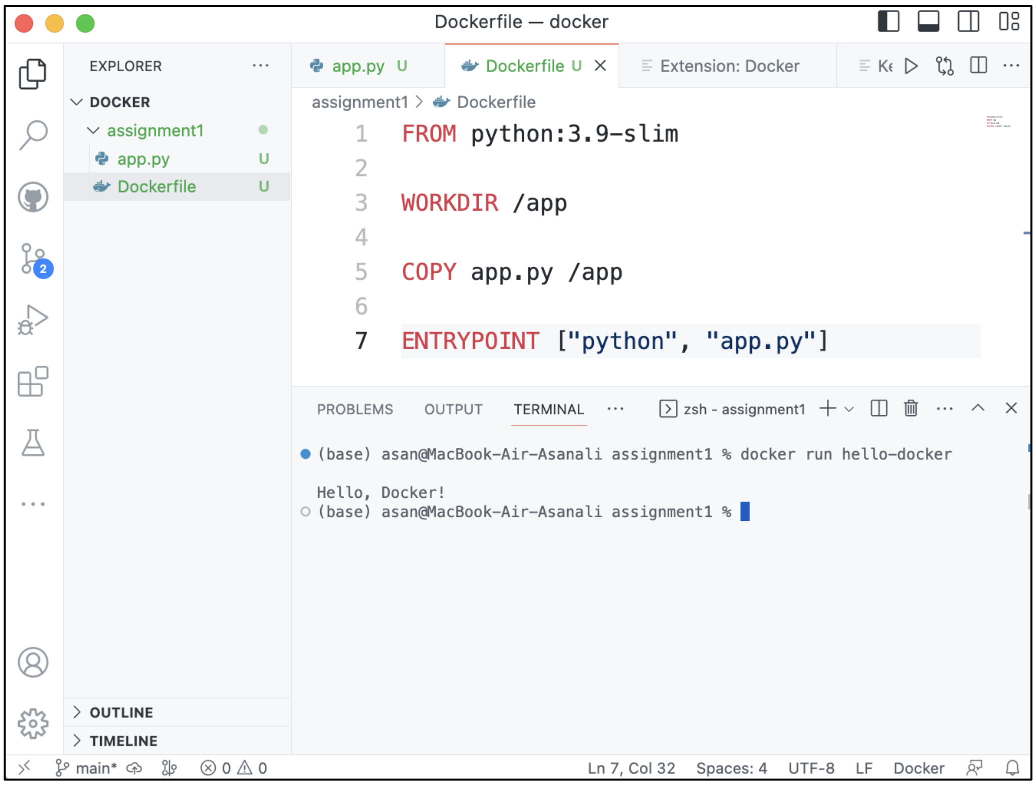
It allows to run a command inside a running container. Thus we can explore its file system.

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

We can ensure that by deleting a container. For that, we use docker rm <container>.

Dockerfile

Exercise 1. Creating a Simple Dockerfile.



```
FROM python:3.9-slim

WORKDIR /app

COPY app.py /app

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

```
(base) asan@MacBook-Air-Asanali assignment1 % docker run hello-docker

Hello, Docker!
(base) asan@MacBook-Air-Asanali assignment1 %
```

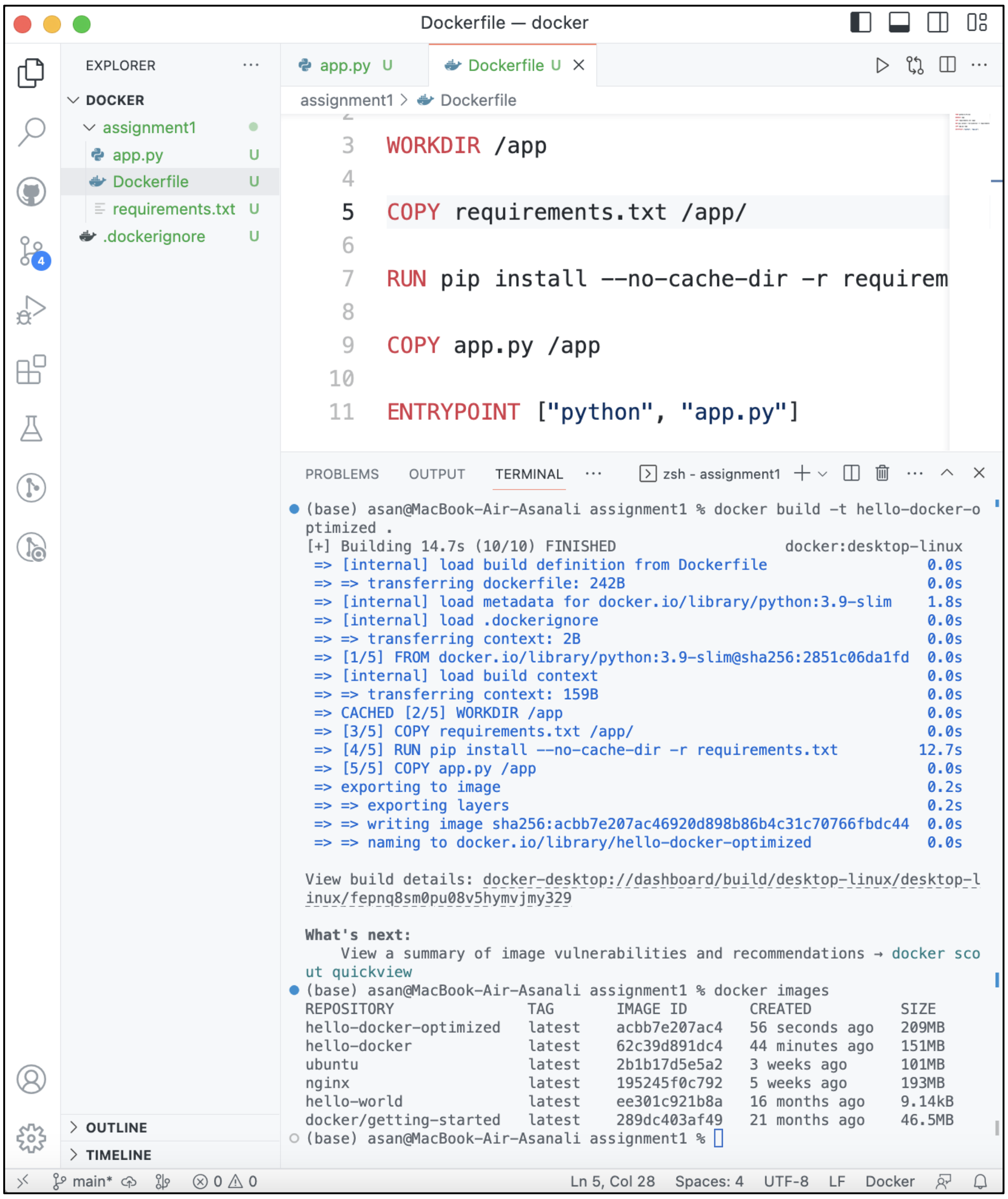
Fig 6. Dockerfile

Questions:
What is the purpose of the FROM instruction in a Dockerfile?
We use FROM to set up the base image. The base image provides all the necessary stuff like OS, libraries, and tools.

How does the COPY instruction work in Dockerfile?
We use COPY to copy files and directories from the host machine into the Docker image.

What is the difference between CMD and ENTRYPOINT in Dockerfile?
Both are used to provide commands that will be executed with the docker run command. The difference is that docker run will override the CMD, while with ENTRYPOINT it won’t.

Exercise 2. Optimizing Dockerfile with Layers and Caching.



```
WORKDIR /app

COPY requirements.txt /app/

RUN pip install --no-cache-dir -r requirem

COPY app.py /app

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

```
(base) asan@MacBook-Air-Asanali assignment1 % docker build -t hello-docker-o
ptimized .
[+] Building 14.7s (10/10) FINISHED                           docker:desktop-linux
 => [internal] load build definition from Dockerfile                          0.0s
 => => transferring dockerfile: 242B                                          0.0s
 => [internal] load metadata for docker.io/library/python:3.9-slim            1.8s
 => [internal] load .dockerignore                                             0.0s
 => => transferring context: 2B                                               0.0s
 => [1/5] FROM docker.io/library/python:3.9-slim@sha256:2851c06da1fd          0.0s
 => [internal] load build context                                             0.0s
 => => transferring context: 159B                                             0.0s
 => CACHED [2/5] WORKDIR /app                                                 0.0s
 => [3/5] COPY requirements.txt /app/                                         0.0s
 => [4/5] RUN pip install --no-cache-dir -r requirements.txt                 12.7s
 => [5/5] COPY app.py /app                                                    0.0s
 => exporting to image                                                        0.2s
 => => exporting layers                                                       0.2s
 => => writing image sha256:acbb7e207ac46920d898b86b4c31c70766fbdc44          0.0s
 => => naming to docker.io/library/hello-docker-optimized                     0.0s

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-l
inux/fepnq8sm0pu08v5hymvjmy329

What's next:
    View a summary of image vulnerabilities and recommendations → docker sco
ut quickview
(base) asan@MacBook-Air-Asanali assignment1 % docker images
REPOSITORY               TAG       IMAGE ID       CREATED          SIZE
hello-docker-optimized   latest    acbb7e207ac4   56 seconds ago   209MB
hello-docker             latest    62c39d891dc4   44 minutes ago   151MB
ubuntu                   latest    2b1b17d5e5a2   3 weeks ago      101MB
nginx                    latest    195245f0c792   5 weeks ago      193MB
hello-world              latest    ee301c921b8a   16 months ago    9.14kB
docker/getting-started   latest    289dc403af49   21 months ago    46.5MB
(base) asan@MacBook-Air-Asanali assignment1 %
```

Fig 7. Optimizing Dockerfile

Questions:

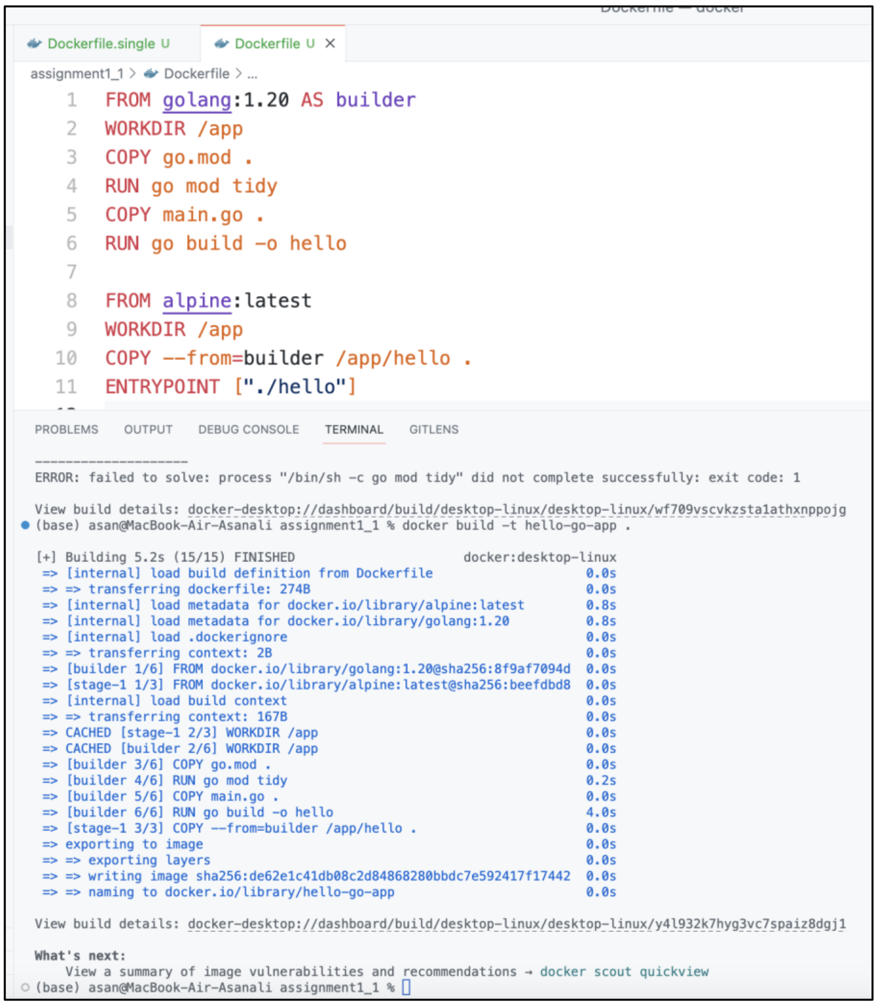
What are Docker layers, and how do they affect image size and build times?

Docker layers contain the changes that we've made to an image. Each layer contains only the changes that have been made compared with the previous layer. Thus, it requires less size as all layers use one base image. It also reduces building time because of using a cache.

How does Docker's build cache work, and how can it speed up the build process?
Build cache saves the results of the previous layers so that we don't need to execute some commands again and again. We reduce the building time because we don't waste time on repetitive commands.

What is the role of the .dockerignore file?
In the .dockerignore file, we list the unnecessary files and directories that should be excluded from the Docker build context.

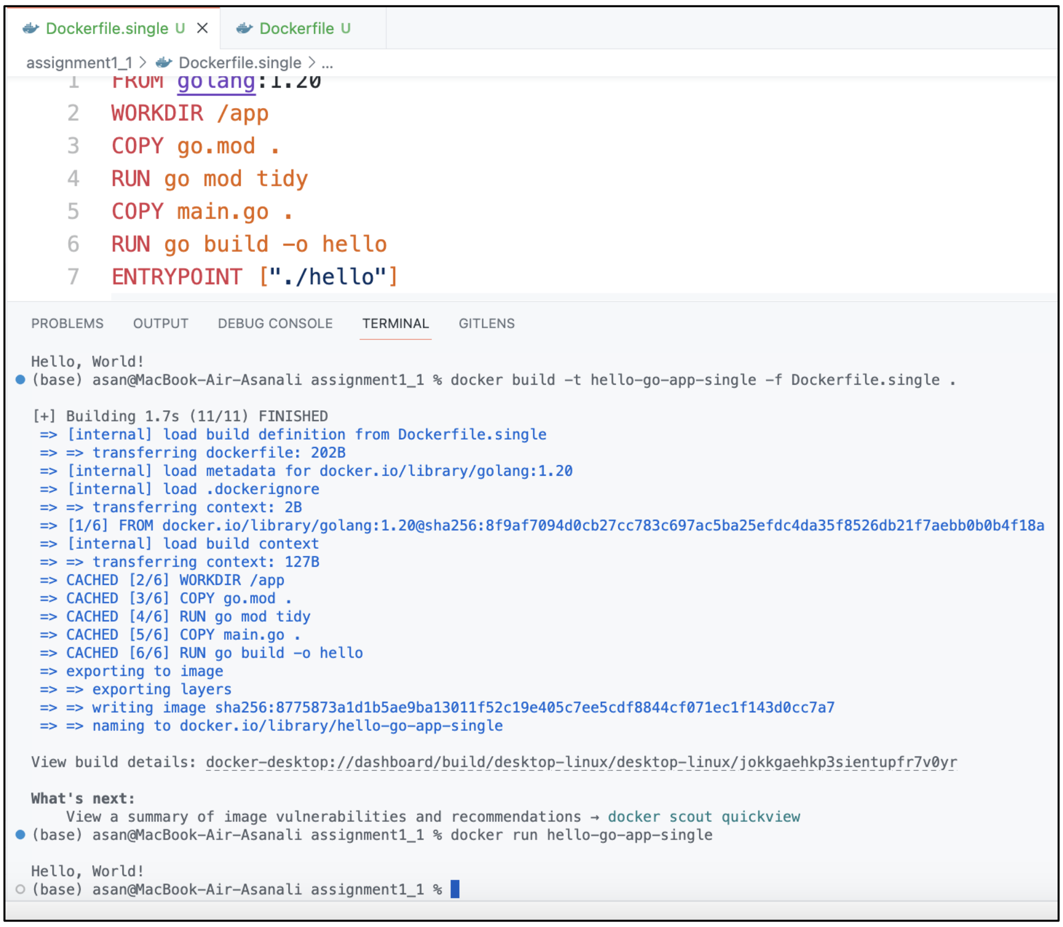
Exercise 3. Multi-Stage Builds.



```
FROM golang:1.20 AS builder
WORKDIR /app
COPY go.mod .
RUN go mod tidy
COPY main.go .
RUN go build -o hello

FROM alpine:latest
WORKDIR /app
COPY --from=builder /app/hello .
ENTRYPOINT ["./hello"]
```

```
--------------------
ERROR: failed to solve: process "/bin/sh -c go mod tidy" did not complete successfully: exit code: 1

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/wf709vscvkzsta1athxnppojg
(base) asan@MacBook-Air-Asanali assignment1_1 % docker build -t hello-go-app .

[+] Building 5.2s (15/15) FINISHED                               docker:desktop-linux
 => [internal] load build definition from Dockerfile                             0.0s
 => => transferring dockerfile: 274B                                             0.0s
 => [internal] load metadata for docker.io/library/alpine:latest                 0.8s
 => [internal] load metadata for docker.io/library/golang:1.20                   0.8s
 => [internal] load .dockerignore                                                0.0s
 => => transferring context: 2B                                                  0.0s
 => [builder 1/6] FROM docker.io/library/golang:1.20@sha256:8f9af7094d           0.0s
 => [stage-1 1/3] FROM docker.io/library/alpine:latest@sha256:beefdbd8           0.0s
 => [internal] load build context                                                0.0s
 => => transferring context: 167B                                                0.0s
 => CACHED [stage-1 2/3] WORKDIR /app                                            0.0s
 => CACHED [builder 2/6] WORKDIR /app                                            0.0s
 => [builder 3/6] COPY go.mod .                                                  0.0s
 => [builder 4/6] RUN go mod tidy                                                0.2s
 => [builder 5/6] COPY main.go .                                                 0.0s
 => [builder 6/6] RUN go build -o hello                                          4.0s
 => [stage-1 3/3] COPY --from=builder /app/hello .                               0.0s
 => exporting to image                                                           0.0s
 => => exporting layers                                                          0.0s
 => => writing image sha256:de62e1c41db08c2d84868280bbdc7e592417f17442           0.0s
 => => naming to docker.io/library/hello-go-app                                  0.0s

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/y4l932k7hyg3vc7spaiz8dgj1

What's next:
    View a summary of image vulnerabilities and recommendations → docker scout quickview
(base) asan@MacBook-Air-Asanali assignment1_1 %
```

Fig 8. Hello, World on Go with Dockerfile


Dockerfile.single U ×   Dockerfile U

assignment1_1 > Dockerfile.single > ...

```
FROM golang:1.20
WORKDIR /app
COPY go.mod .
RUN go mod tidy
COPY main.go .
RUN go build -o hello
ENTRYPOINT ["./hello"]
```

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   GITLENS

```
Hello, World!
● (base) asan@MacBook-Air-Asanali assignment1_1 % docker build -t hello-go-app-single -f Dockerfile.single .

[+] Building 1.7s (11/11) FINISHED
 => [internal] load build definition from Dockerfile.single
 => => transferring dockerfile: 202B
 => [internal] load metadata for docker.io/library/golang:1.20
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [1/6] FROM docker.io/library/golang:1.20@sha256:8f9af7094d0cb27cc783c697ac5ba25efdc4da35f8526db21f7aebb0b0b4f18a
 => [internal] load build context
 => => transferring context: 127B
 => CACHED [2/6] WORKDIR /app
 => CACHED [3/6] COPY go.mod .
 => CACHED [4/6] RUN go mod tidy
 => CACHED [5/6] COPY main.go .
 => CACHED [6/6] RUN go build -o hello
 => exporting to image
 => => exporting layers
 => => writing image sha256:8775873a1d1b5ae9ba13011f52c19e405c7ee5cdf8844cf071ec1f143d0cc7a7
 => => naming to docker.io/library/hello-go-app-single

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/jokkgaehkp3sientupfr7v0yr

What's next:
    View a summary of image vulnerabilities and recommendations → docker scout quickview
● (base) asan@MacBook-Air-Asanali assignment1_1 % docker run hello-go-app-single

Hello, World!
○ (base) asan@MacBook-Air-Asanali assignment1_1 % 
```

Fig 9. Hello, World on Go with Dockerfile.single

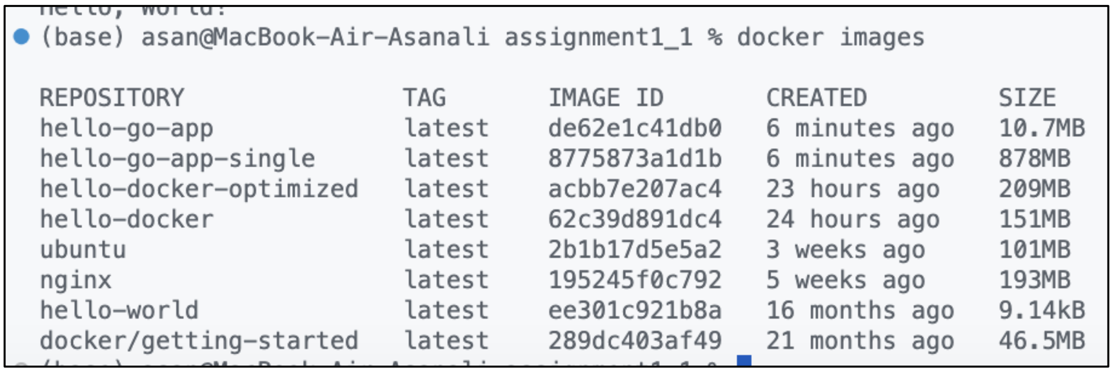

```
● (base) asan@MacBook-Air-Asanali assignment1_1 % docker images

  REPOSITORY               TAG      IMAGE ID       CREATED         SIZE
  hello-go-app             latest   de62e1c41db0   6 minutes ago   10.7MB
  hello-go-app-single      latest   8775873a1d1b   6 minutes ago   878MB
  hello-docker-optimized   latest   acbb7e207ac4   23 hours ago    209MB
  hello-docker             latest   62c39d891dc4   24 hours ago    151MB
  ubuntu                   latest   2b1b17d5e5a2   3 weeks ago     101MB
  nginx                    latest   195245f0c792   5 weeks ago     193MB
  hello-world              latest   ee301c921b8a   16 months ago   9.14kB
  docker/getting-started   latest   289dc403af49   21 months ago   46.5MB
```

Fig 10. Docker images

Questions:

What are the benefits of using multi-stage builds in Docker?

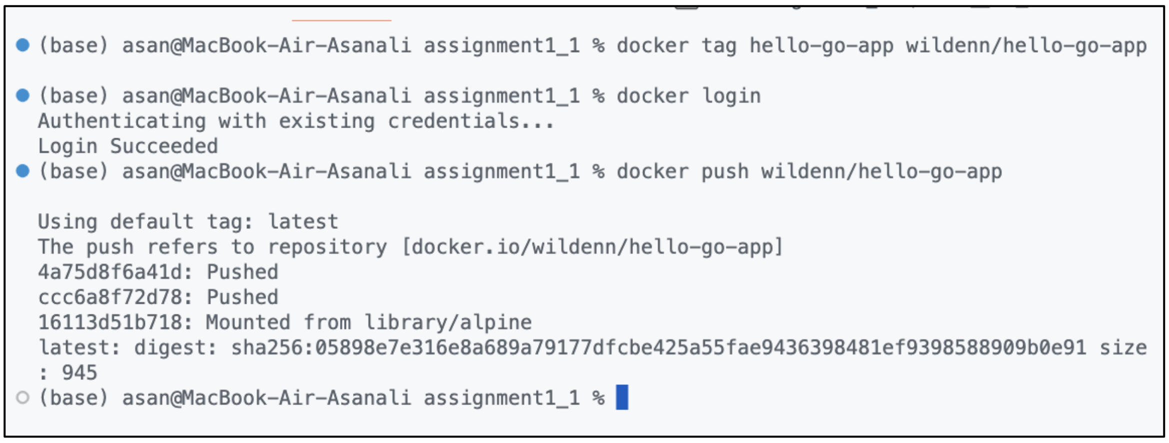
We reduce image size and complexity by using multi-stage builds as we only use the necessary components in the final image.

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

We separate the building process and the runtime environment so that the final image contains only the necessary components to run the app correctly.

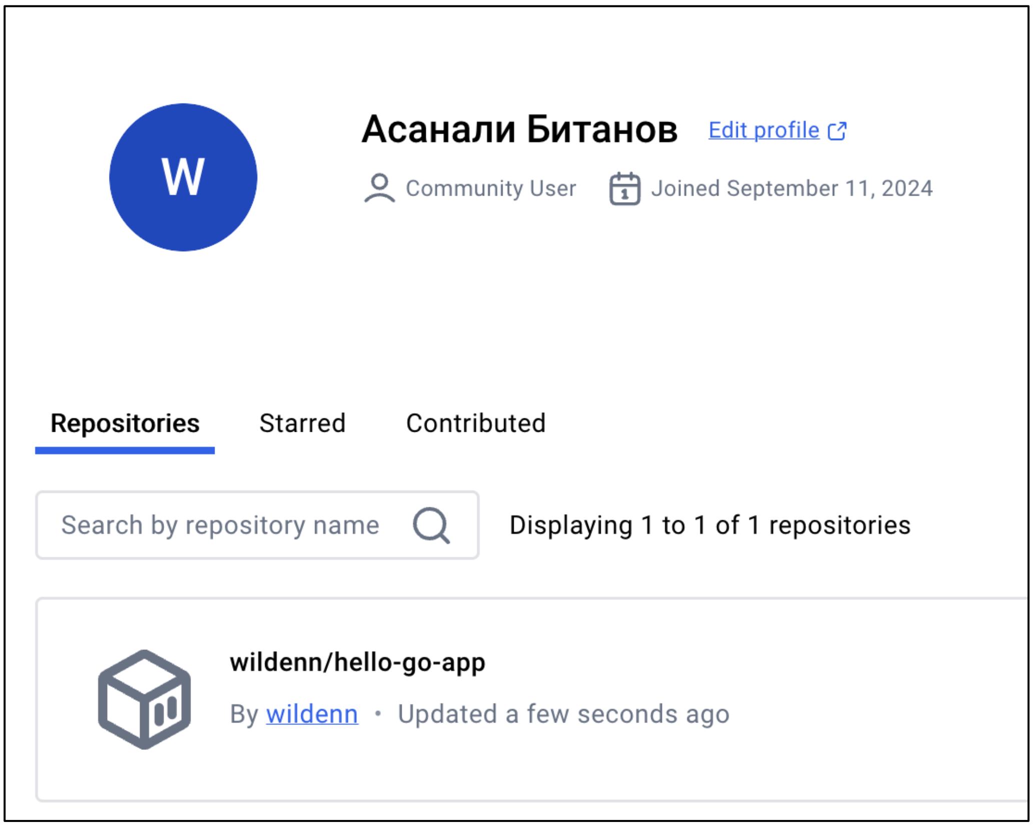
What are some scenarios where multi-stage builds are particularly useful?
It's useful when we build an app with comipled languages like GO or Java. We also can add a test layer, so that the final image will be a correctly working app.

Exercise 4. Pushing Docker Image to Docker Hub



```
● (base) asan@MacBook-Air-Asanali assignment1_1 % docker tag hello-go-app wildenn/hello-go-app

● (base) asan@MacBook-Air-Asanali assignment1_1 % docker login
  Authenticating with existing credentials...
  Login Succeeded
● (base) asan@MacBook-Air-Asanali assignment1_1 % docker push wildenn/hello-go-app

  Using default tag: latest
  The push refers to repository [docker.io/wildenn/hello-go-app]
  4a75d8f6a41d: Pushed
  ccc6a8f72d78: Pushed
  16113d51b718: Mounted from library/alpine
  latest: digest: sha256:05898e7e316e8a689a79177dfcbe425a55fae9436398481ef9398588909b0e91 size
  : 945
○ (base) asan@MacBook-Air-Asanali assignment1_1 %
```

Fig 11. Tagging and uploading the image



Асанали Битанов Edit profile

Community User · Joined September 11, 2024

Repositories · Starred · Contributed

Search by repository name

Displaying 1 to 1 of 1 repositories

wildenn/hello-go-app

By wildenn · Updated a few seconds ago

Fig 12. Uploaded image

Questions:

What is the purpose of Docker Hub in containerization?

Docker Hub is a store where we can share and manage Docker images. It allows us to make development easier and more stable.

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

To tag a Docker image we use the docker tag <image_name> <username>/<repository-name> command.

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

Firstly, we use docker tag command. Then we log into the account by using docker login command. Lastly, We push the image by using docker push <username>/<repository_name> command.