

Assignment 1, Web Application Development

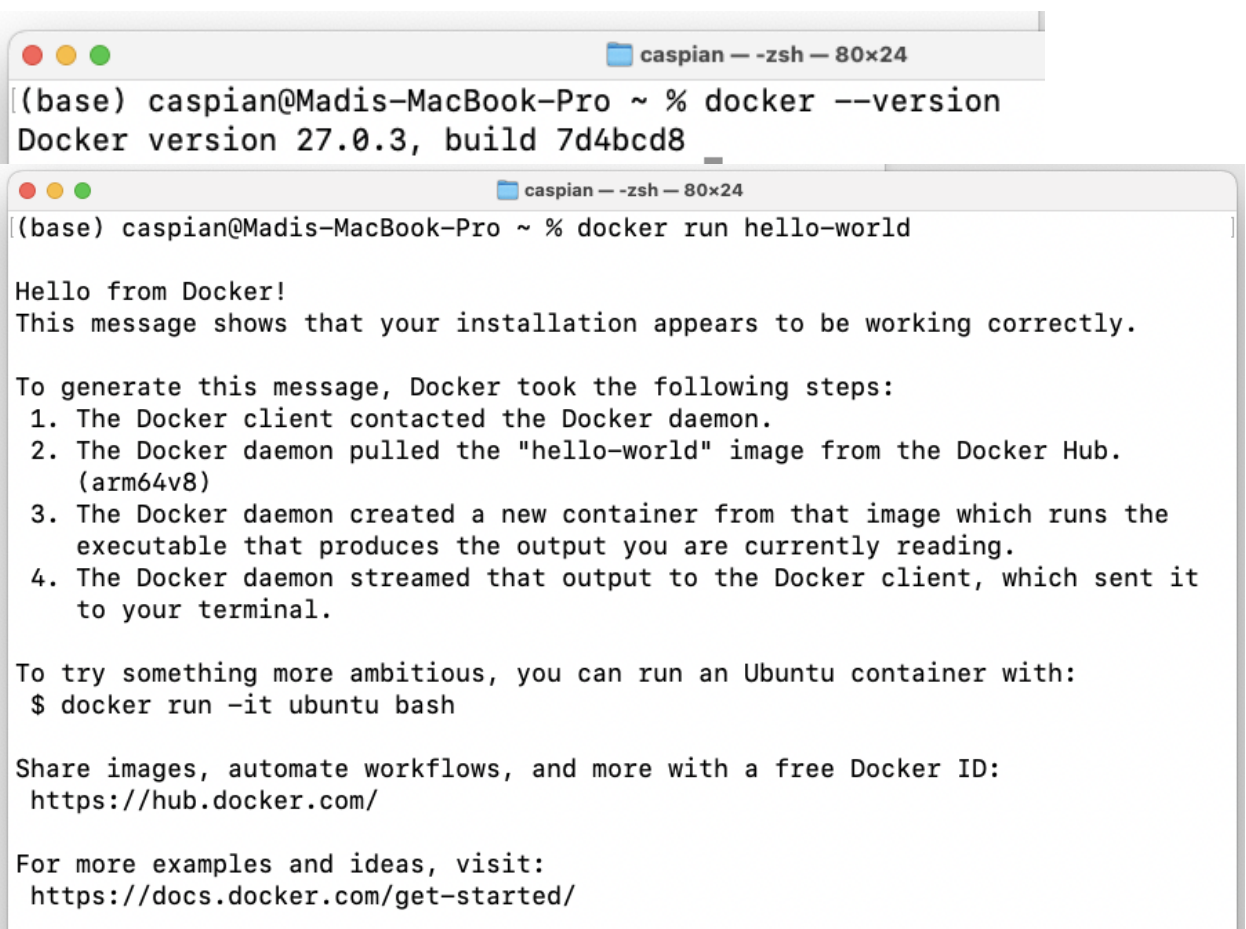
Madi Khassenov, 23MD0450

Intro to Containerization: Docker

Exercise 1: Installing Docker

Steps:

1.

A screenshot of a terminal window titled 'caspian - zsh - 80x24'. The terminal shows the command 'docker --version' being executed, resulting in 'Docker version 27.0.3, build 7d4bcd8'. Below this, the command 'docker run hello-world' is executed, producing a 'Hello from Docker!' message and a list of steps Docker took to generate the message. The steps include contacting the Docker daemon, pulling the 'hello-world' image from Docker Hub, creating a new container, and streaming output to the terminal. The terminal also provides instructions on how to run an Ubuntu container and links to Docker's website and documentation.

```
(base) caspian@Madis-MacBook-Pro ~ % docker --version
Docker version 27.0.3, build 7d4bcd8

(base) caspian@Madis-MacBook-Pro ~ % 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.
    (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/
```

2.

Questions:

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

Key components of Docker are:

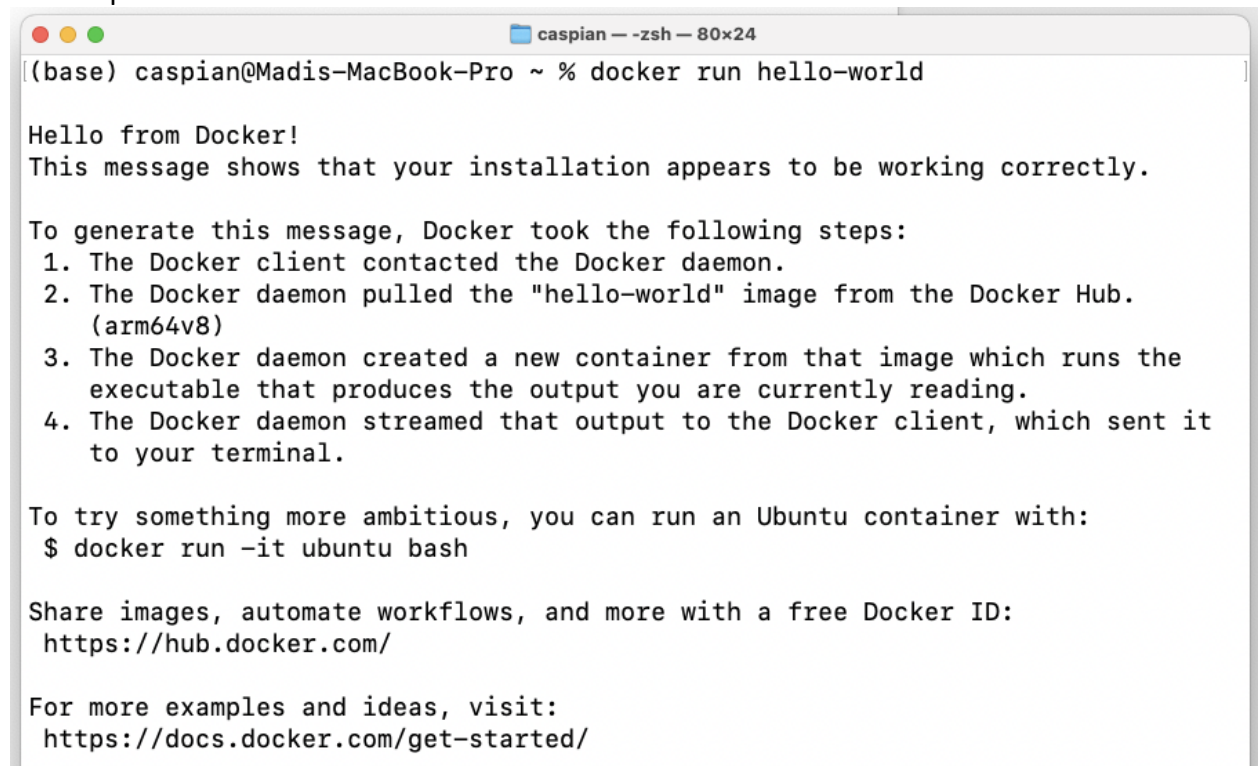
- a. Docker Engine – the core of Docker which includes Server (Docker Daemon), REST API, and CLI.

- b. Docker Images – lightweight read-only files that contain everything needed to run an application.
 - c. Dockerfile – a script with instructions on how to make a Docker Image. Contains information about OS, languages, environment variables, ports etc.
 - d. Docker Hub – a cloud-based archive of Docker images that allows users to push and pull images either publicly or privately.
 - e. Docker Volumes – allows users to preserve and store data across containers and mount it to new ones.
 - f. Docker Compose – allows users to run multi-container Docker applications using a YAML file.
 - g. Docker Desktop – a GUI application used for interacting with Docker.
 - h. Docker Containers – active instances of Docker Images that allow interaction.
2. How does Docker compare to traditional virtual machines?

Docker uses the host OS' kernel where each container runs as an isolated process, whereas each VM uses a full OS and a hypervisor. This makes Docker more lightweight, resource efficient, and portable.

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

The output was:



```
caspián — zsh — 80x24
(base) caspián@Madis-MacBook-Pro ~ % 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.
    (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

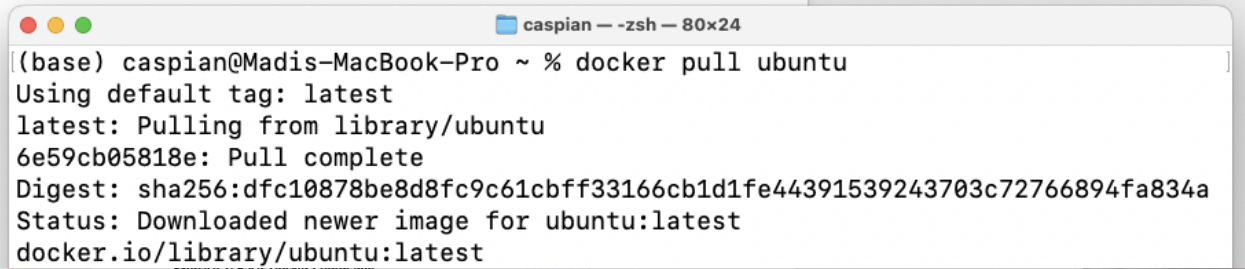
Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

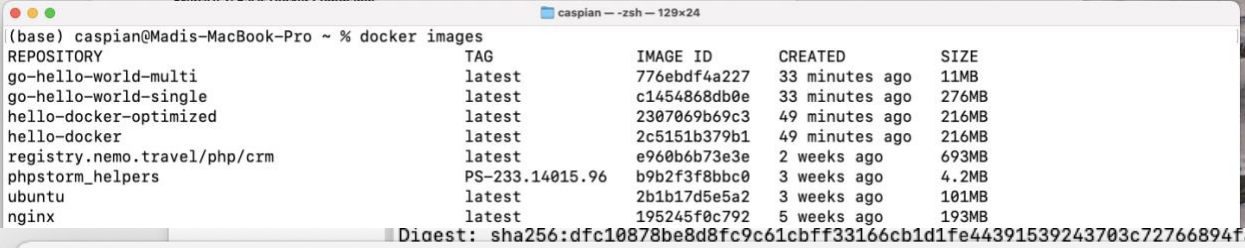
For more examples and ideas, visit:
https://docs.docker.com/get-started/
```

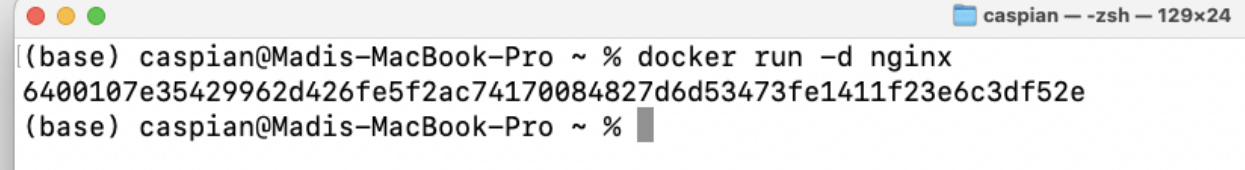
Which means that Docker is correctly installed and functions correctly. The output also described what exactly happened after the command was run.

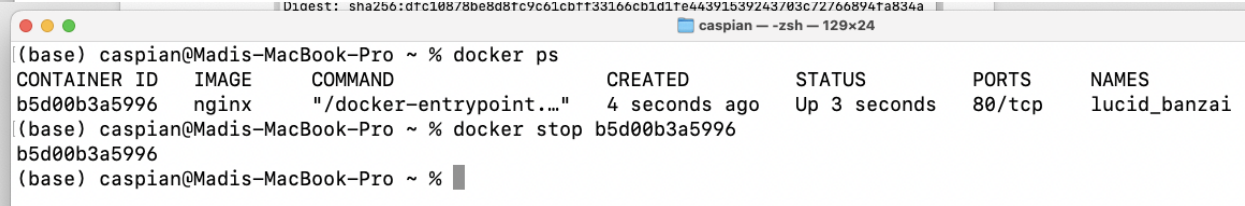
Exercise 2: Basic Docker Commands

Steps:

1. 

```
(base) caspian@Madis-MacBook-Pro ~ % 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
```
2. 

```
(base) caspian@Madis-MacBook-Pro ~ % docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
go-hello-world-multi latest             776ebdf4a227       33 minutes ago     11MB
go-hello-world-single latest             c1454868db0e       33 minutes ago     276MB
hello-docker-optimized latest             2307069b69c3       49 minutes ago     216MB
hello-docker         latest             2c5151b379b1       49 minutes ago     216MB
registry.nemo.travel/php/crm latest             e960b6b73e3e       2 weeks ago        693MB
phpstorm_helpers     PS-233.14015.96   b9b2f3f8bbc0       3 weeks ago        4.2MB
ubuntu               latest             2b1b17d5e5a2       3 weeks ago        101MB
nginx                latest             195245f0c792       5 weeks ago        193MB
```
3. 

```
(base) caspian@Madis-MacBook-Pro ~ % docker run -d nginx
6400107e35429962d426fe5f2ac74170084827d6d53473fe1411f23e6c3df52e
(base) caspian@Madis-MacBook-Pro ~ %
```
4. 

```
(base) caspian@Madis-MacBook-Pro ~ % docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS      NAMES
b5d00b3a5996   nginx    "/docker-entrypoint...." 4 seconds ago  Up 3 seconds  80/tcp     lucid_banzai

(base) caspian@Madis-MacBook-Pro ~ % docker stop b5d00b3a5996
b5d00b3a5996
(base) caspian@Madis-MacBook-Pro ~ %
```

Questions:

1. What is the difference between docker pull and docker run?

docker pull just pulls the image without running the container, while docker run does both.

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

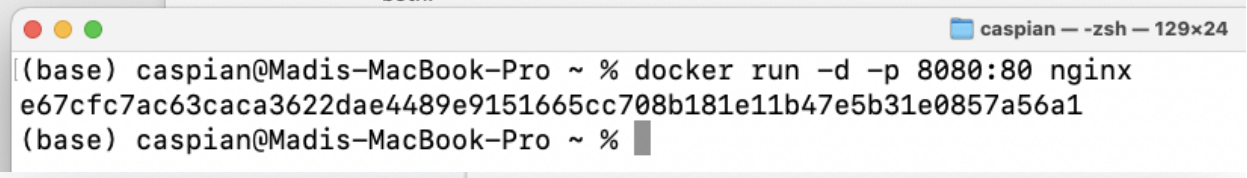
Using the docker ps command.

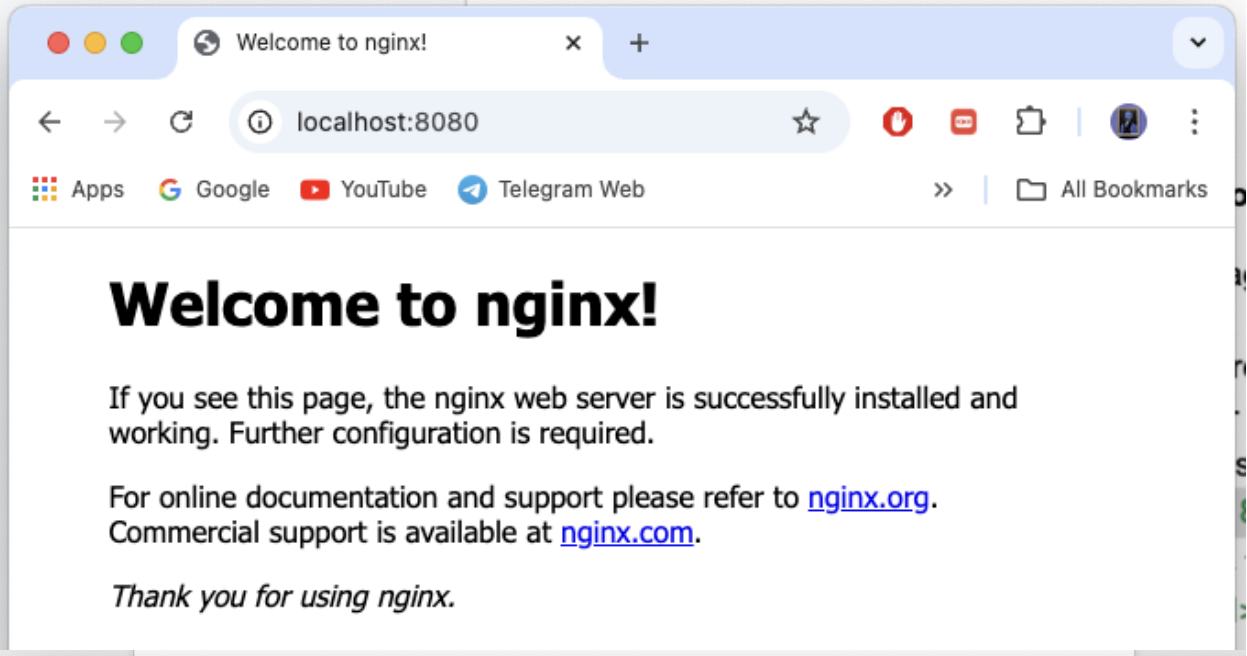
3. What happens to a container after it is stopped? Can it be restarted?

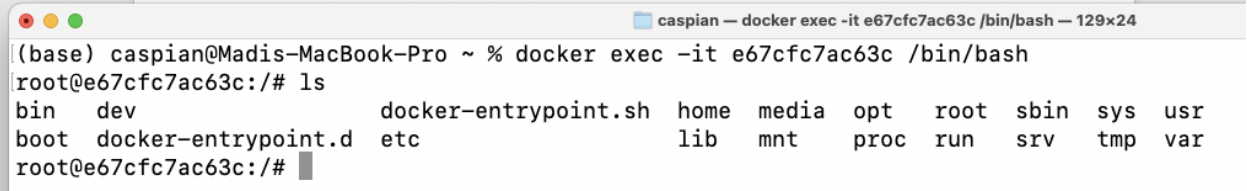
When a container is stopped its processes are terminated and its status is changed to 'stopped', the container's resources are released. Yes, it can be restarted.

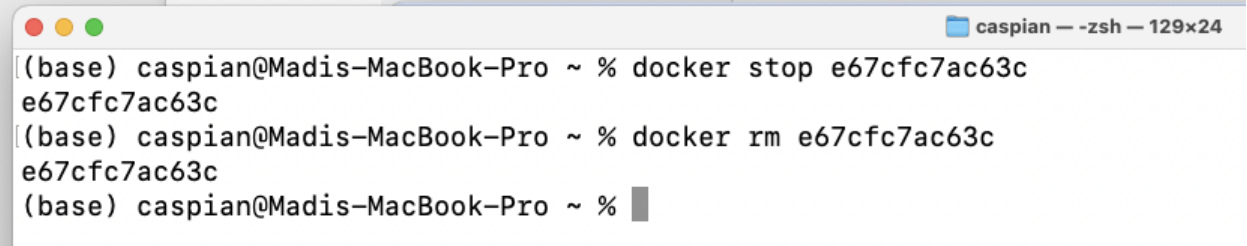
Exercise 3: Working with Docker Containers

Steps:

1. 

```
(base) caspian@Madis-MacBook-Pro ~ % docker run -d -p 8080:80 nginx
e67cfc7ac63caca3622dae4489e9151665cc708b181e11b47e5b31e0857a56a1
(base) caspian@Madis-MacBook-Pro ~ %
```
2. 

The browser window shows the 'Welcome to nginx!' page. The address bar displays 'localhost:8080'. The page content includes the heading 'Welcome to nginx!', a message stating 'If you see this page, the nginx web server is successfully installed and working. Further configuration is required.', links to 'nginx.org' for documentation and 'nginx.com' for commercial support, and a closing message 'Thank you for using nginx.'
3. 

```
(base) caspian@Madis-MacBook-Pro ~ % docker exec -it e67cfc7ac63c /bin/bash
root@e67cfc7ac63c:/# 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@e67cfc7ac63c:/#
```
4. 

```
(base) caspian@Madis-MacBook-Pro ~ % docker stop e67cfc7ac63c
e67cfc7ac63c
(base) caspian@Madis-MacBook-Pro ~ % docker rm e67cfc7ac63c
e67cfc7ac63c
(base) caspian@Madis-MacBook-Pro ~ %
```

Questions:

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

Port mapping works by exposing the container services to the host machine, it is done using the `-p` or `--publish` option in the `docker run` command. It is important to make services accessible from outside the container or to run multiple containers.

2. What is the purpose of the `docker exec` command?

Using the `docker exec` commands we can run commands inside of a running container.

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

We can remove the container.

Dockerfile

Exercise 1: Creating a Simple Dockerfile

Steps:

1.

```
app.py ×  
Ex_1 > app.py  
1 print("Hello, Docker!")
```

2.

```
Dockerfile ×  
Ex_1 > Dockerfile > ...  
1 FROM python:3.12-slim  
2  
3 WORKDIR /app  
4  
5 COPY . /app  
6  
7 RUN pip install --no-cache-dir -r requirements.txt  
8  
9 ENTRYPOINT [ "python", "app.py" ]
```

- ```
(base) caspian@Madis-MacBook-Pro Ex_1 % docker build -t hello-docker .
[+] Building 1.8s (10/10) FINISHED docker:desktop-linux
=> [internal] load build definition from Dockerfile 0.0s
=> => transferring dockerfile: 174B 0.0s
=> [internal] load metadata for docker.io/library/python:3.12-slim 1.8s
=> [auth] library/python:pull token for registry-1.docker.io 0.0s
=> [internal] load .dockerignore 0.0s
=> => transferring context: 2B 0.0s
=> [1/4] FROM docker.io/library/python:3.12-slim@sha256:15bad989b293be1dd5eb26a87ecacadaee1559f98e29f02bf6d00c8d86 0.0s
=> [internal] load build context 0.0s
=> => transferring context: 282B 0.0s
=> CACHED [2/4] WORKDIR /app 0.0s
=> CACHED [3/4] COPY . /app 0.0s
=> CACHED [4/4] RUN pip install --no-cache-dir -r requirements.txt 0.0s
=> exporting to image 0.0s
=> => exporting layers 0.0s
=> => writing image sha256:e25539fb0b83e3cec2439587a42e46172f97e43fc2b4e1e0cc1e86e9c6993994 0.0s
=> => naming to docker.io/library/hello-docker 0.0s
```
3. What's next:  
View a summary of image vulnerabilities and recommendations → [docker scout quickview](#)
- ```
(base) caspian@Madis-MacBook-Pro Ex_1 % docker run hello-docker
Hello, Docker!
```
4. —

Questions:

1. What is the purpose of the FROM instruction in a Dockerfile?

The FROM instruction is used to specify the base image of our own image.

2. How does the COPY instruction work in Dockerfile?

The COPY instruction is used to copy files and directories from our host machine to the image's filesystem during the build.

3. What is the difference between CMD and ENTRYPOINT in Dockerfile?

Both will run when we start our container, however CMD can be overridden.

Exercise 2: Optimizing Dockerfile with Layers and Caching

Steps:


```

Dockerfile X
Ex_2 > Dockerfile > ...
1 FROM python:3.12-slim
2
3 WORKDIR /app
4
5 COPY requirements.txt /app
6
7 RUN pip install --no-cache-dir -r requirements.txt
8
9 COPY app.py /app
10
11 ENTRYPOINT [ "python", "app.py" ]

```

- ```

(base) caspian@Madis-MacBook-Pro Ex_2 % docker build -t hello-docker-optimized .
[+] Building 1.7s (11/11) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 207B
=> [internal] load metadata for docker.io/library/python:3.12-slim
=> [auth] library/python:pull token for registry-1.docker.io
=> [internal] load .dockerignore
=> => transferring context: 79B
=> [1/5] FROM docker.io/library/python:3.12-slim@sha256:15bad989b293be1dd5eb26a87ecacadaee1559f98e29f02bf6d00c8d86
=> [internal] load build context
=> => transferring context: 106B
=> CACHED [2/5] WORKDIR /app
=> CACHED [3/5] COPY requirements.txt /app
=> CACHED [4/5] RUN pip install --no-cache-dir -r requirements.txt
=> CACHED [5/5] COPY app.py /app
=> exporting to image
=> => exporting layers
=> => writing image sha256:fb930f53b7d7b59ae9e931a4c091c2c13997387b970be9efd2f2d9d16d3f824d
=> => naming to docker.io/library/hello-docker-optimized

```
- What's next:  
View a summary of image vulnerabilities and recommendations → [docker scout quickview](#)

```

(base) caspian@Madis-MacBook-Pro Ex 1 % docker images

```

| REPOSITORY             | TAG    | IMAGE ID     | CREATED           | SIZE  |
|------------------------|--------|--------------|-------------------|-------|
| go-hello-world-multi   | latest | 776ebdf4a227 | 53 minutes ago    | 11MB  |
| go-hello-world-single  | latest | c1454868db0e | 53 minutes ago    | 276MB |
| hello-docker-optimized | latest | fb930f53b7d7 | About an hour ago | 216MB |
| hello-docker           | latest | e25539fb0b83 | About an hour ago | 216MB |

## Questions:

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

Docker Layers represent a set of system changes and instructions, they are stacked on top of each other. Each layers adds to the size of the image. Layers can be cached if they haven't been changed, which helps in reducing the built time of the image.

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

Basically if layers haven't been changed then Docker just reuses them.

3. What is the role of the .dockerignore file?

The .dockerignore file specifies the files that must be excluded from the build context, thus reducing the image size.


### Exercise 3: Multi-Stage Builds

Steps:

```
Ex_3 > multi > -go main.go > main
1 package main
2
3 import "fmt"
4
5 func main() {
6 fmt.Println("Hello, World!")
7 }
8
```

1.



 Dockerfile ×

Ex\_3 > multi >  Dockerfile > ...

```
1 FROM golang:1.23-alpine AS builder
2
3 WORKDIR /app
4
5 COPY main.go .
6
7 RUN go build -o hello-world main.go
8
9 FROM alpine:latest
10
11 WORKDIR /app
12
13 COPY --from=builder /app/hello-world .
14
15 CMD ["/hello-world"]
```

2.

```
Dockerfile X
Ex_3 > single > Dockerfile > ...
1 FROM golang:1.23-alpine
2
3 WORKDIR /app
4
5 COPY main.go .
6
7 RUN go build -o hello-world main.go
8
9 CMD ["/hello-world"]
```

3.

| REPOSITORY            | TAG    | IMAGE ID     | CREATED        | SIZE  |
|-----------------------|--------|--------------|----------------|-------|
| go-hello-world-multi  | latest | 776ebdf4a227 | 53 minutes ago | 11MB  |
| go-hello-world-single | latest | c1454868db0e | 53 minutes ago | 276MB |

Questions:

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

The benefits of using multi-staged builds in Docker are smaller final image size, cleaner builds, separation of concerns, optimized build times.

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

We can avoid using heavy dependencies by using separate base images for building our application and running it.

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

When our application is compiled, when we need to reduce the final image size, when we are building for different environments.

#### Exercise 4: Pushing Docker Images to Docker Hub

Steps:



docker

EARLY ACCESS



**caspiank**

Joined July 16, 2024

1.

```
When our application is compiled, when we need to reduce the final image size, when
caspiank - zsh - 129x24
[(base) caspiank@Madis-MacBook-Pro ~ % docker tag hello-docker caspiank/hello-docker
(base) caspiank@Madis-MacBook-Pro ~ %
```

2.

```
[(base) caspiank@Madis-MacBook-Pro ~ % docker login
Authenticating with existing credentials...
Login Succeeded
```

3.

```
(base) caspiank@Madis-MacBook-Pro ~ % docker push caspiank/hello-docker
Using default tag: latest
The push refers to repository [docker.io/caspiank/hello-docker]
8a9e27154a41: Pushed
b02119b415ef: Pushed
66f37d6585c7: Pushed
4f87ac73ce8f: Mounted from library/python
d7486c28114f: Mounted from library/python
50f7fbe612d1: Mounted from library/python
e644ff0c302d: Mounted from library/python
```

4.

```
latest: digest: sha256:5f2b823593b8ea1cd5347920eadd291708dc0c6e439cd7554cd1174c4b14ea37 size: 1784
```

The screenshot shows the Docker Hub interface for the repository 'caspiank/hello-docker'. The top navigation bar includes 'dockerhub', 'Explore', 'Repositories' (active), 'Organizations', and 'Usage'. A search bar is on the right. The breadcrumb trail is 'caspiank / Repositories / hello-docker / General'. Below this are tabs for 'General', 'Tags', 'Builds', 'Collaborators', 'Webhooks', and 'Settings'. The 'General' tab is active, showing the repository name 'caspiank/hello-docker' with a Docker icon, 'Last pushed 1 minute ago', and two incomplete fields: 'This repository does not have a description' and 'This repository does not have a category', both with 'INCOMPLETE' status. Below this is a 'Tags' section stating 'This repository contains 1 tag(s)'. A table lists the tag 'latest' as an 'Image' type, pulled and pushed 'a few seconds ago'. A 'See all' link is at the bottom.

| Tag    | OS | Type  | Pulled            | Pushed            |
|--------|----|-------|-------------------|-------------------|
| latest |    | Image | a few seconds ago | a few seconds ago |

5.

## Questions:

1. What is the purpose of Docker Hub in containerization?

Docker Hub allows image sharing and collaboration, provides access to official images, simplifies cloud deployments, has CI/CD integration.

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

Using the docker tag command.

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

Tagging the image and pushing it.

```
docker tag SOURCE_IMAGE TARGET_IMAGE
```

```
docker push TARGET_IMAGE
```