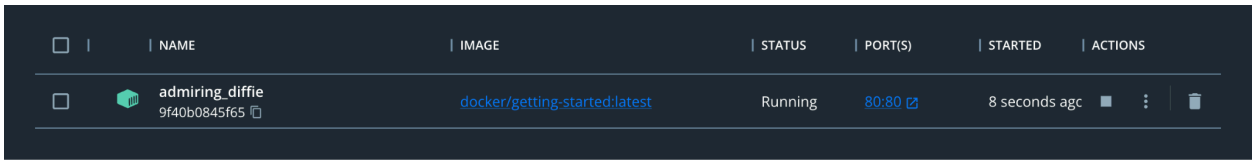
Web - Assignment - 1

Intro to Containerization: Docker

Exercise 1: Installing Docker



Questions:

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

ANS: Docker Engine, Docker CLI, Docker Images, Docker Containers, Docker Registries

2) How does Docker compare to traditional virtual machines?

ANS: Docker containers are lightweight and run on the host's OS, making them faster and more efficient than virtual machines (VMs), which each need a full operating system. Containers are great for quickly starting apps and scaling, but they have less security isolation compared to VMs. VMs provide stronger isolation and can run different operating systems, but they use more resources and take longer to start

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

ANS: Running `docker run hello-world` pulls the "hello-world" image from Docker Hub, creates a container, and outputs a message confirming that Docker is installed and

working correctly. It verifies that Docker can pull images, create containers, and run them successfully.

Exercise 2: Basic Docker Commands

```
~ (14.803s)
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
```

```
~ (0.205s)
docker images

REPOSITORY                    TAG       IMAGE ID       CREATED         SIZE
sail-8.3/app                  latest    0773c9c26605   7 days ago      1.44GB
selenium/standalone-chromium  latest    684b53d3f657   2 weeks ago     1.49GB
axllent/mailpit               latest    72d6bf673c53   2 weeks ago     28.8MB
getmeili/meilisearch          latest    f5f9e6642c06   2 weeks ago     137MB
nginx                         latest    195245f0c792   5 weeks ago     193MB
redis                         alpine    8ff9c3b88372   8 weeks ago     42.3MB
```

```
~ (0.468s)
docker run -d nginx

b799f30af3e23de54232ae65613bccc3429a8c1d06f6ae4f10c3d649c2051765

~ (0.174s)
docker ps

CONTAINER ID   IMAGE                  COMMAND                  CREATED          STATUS          PORTS                NAMES
b799f30af3e2   nginx                  "/docker-entrypoint.…"   12 seconds ago   Up 11 seconds   80/tcp               wizardly_mclaren
9f40b0845f65   docker/getting-started "/docker-entrypoint.…"   18 minutes ago   Up 18 minutes   0.0.0.0:80->80/tcp   admiring_diffie
```

```
~ (0.091s)
docker stop b799f30af3e2

b799f30af3e2

~ (0.093s)
docker ps

CONTAINER ID   IMAGE                  COMMAND                  CREATED          STATUS          PORTS                NAMES
9f40b0845f65   docker/getting-started "/docker-entrypoint.…"   19 minutes ago   Up 19 minutes   0.0.0.0:80->80/tcp   admiring_diffie
```

Questions:

4) What is the difference between `docker pull` and `docker run`?

ANS: `docker pull` fetches, and `docker run` fetches and runs the image.

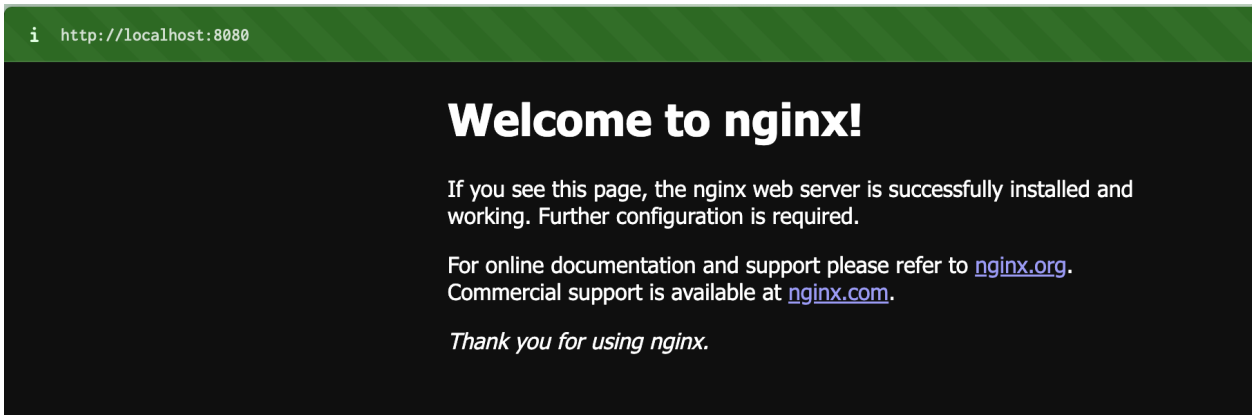
5) How do you find the details of a running container, such as its ID and status?

ANS: `docker ps` will display the container ID, status, name, and other relevant details for all running containers.

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

ANS: After it is stopped, it remains in the system but is in an inactive state. We can restart the stopped container using the `docker start <conatiner_id>` command. The container's data and configuration are preserved unless explicitly removed.

Exercise 3: Working with Docker Containers



```
~ (29.881s)
docker exec -it b50011819bdb bin/bash

root@b50011819bdb:/# exit
exit
```

```
~ (0.298s)
docker stop b50011819bdb

b50011819bdb
```

```
~ (0.106s)
docker rm b50011819bdb

b50011819bdb
```

Questions:

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

ANS: Port mapping in Docker allows a container's internal ports to be exposed to the host machine by mapping the container's port to a host port. It enables external access to services running inside the container, like a web server.

8) What is the purpose of the `docker exec` command?

ANS: The `docker exec` command allows you to run a command inside a running Docker container. It's useful for interacting with a container, such as opening a shell or executing a script.

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

ANS: To ensure a stopped container doesn't consume system resources, you can remove it using the `docker rm <container_id>` command. This deletes the container and frees up disk space

Dockerfile

Exercise 1: Creating a Simple Dockerfile

```
~ (0.03s)
cd Desktop/Projects
```

```
~/Desktop/Projects (0.034s)
mkdir hw-py-app
```

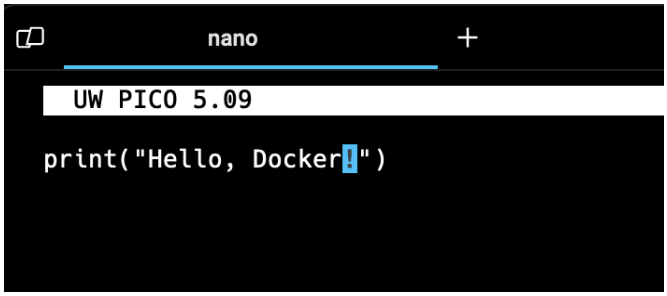
```
~/Desktop/Projects (0.027s)
cd hw-py-app/
```

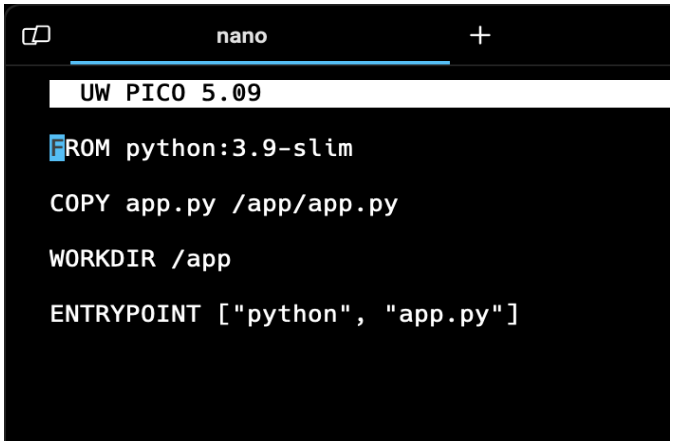
```
~/Desktop/Projects/hw-py-app (0.035s)
touch app.py
```

```
~/Desktop/Projects/hw-py-app (1m 3.12s)
nano app.py
```

```
~/Desktop/Projects/hw-py-app (0.037s)
touch app.py
```

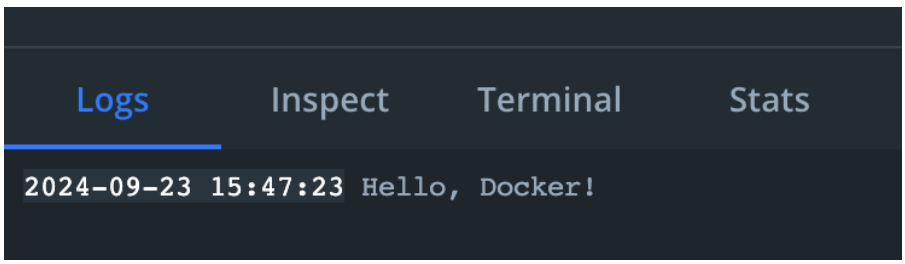
```
~/Desktop/Projects/hw-py-app (0.029s)
ls
app.py
```






```
~/Desktop/Projects/hw-py-app (0.463s)
docker run -d hello-docker

fb3f50743086abcd7d103331d001d53bb9ed81a6fb2120c5da426de1f14e56f3
```

```
~/Desktop/Projects/hw-py-app (10.972s)
docker build -t hello-docker .


[+] Building 8.2s (8/8) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 173B
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load metadata for docker.io/library/python:3.9-slim
 => [internal] load build context
 => => transferring context: 94B
 => [1/3] FROM docker.io/library/python:3.9-slim@sha256:2851c06da1fdc3c451784beef8aa31d1a313d8e3fc122e4a1891085a104b7cfb
 => => resolve docker.io/library/python:3.9-slim@sha256:2851c06da1fdc3c451784beef8aa31d1a313d8e3fc122e4a1891085a104b7cfb
 => => sha256:89790d4ca55c29720fc29c489ba4403f3bb6baa1a6d8b5d0e96c3d40521408c0 3.33MB / 3.33MB
 => => sha256:04acf592cf1a560c87343c39e76e3372e54bed9bcdc59bbc2c5289ff4eeb9e08 14.71MB / 14.71MB
 => => sha256:c21204f5797c735b45941b864c2f521033d92bb12a150b4257afde4f5570a250 250B / 250B
 => => sha256:2851c06da1fdc3c451784beef8aa31d1a313d8e3fc122e4a1891085a104b7cfb 10.41kB / 10.41kB
 => => sha256:17934128833e308455054ef8eb75d87e1760f4470a2ae77a27d488e0e5411d92 1.75kB / 1.75kB
 => => sha256:c0e4ee20d94c77d442f8c8b0153a5e15052211ff874dc08a23e3b955d0689765 5.22kB / 5.22kB
 => => extracting sha256:89790d4ca55c29720fc29c489ba4403f3bb6baa1a6d8b5d0e96c3d40521408c0
 => => extracting sha256:04acf592cf1a560c87343c39e76e3372e54bed9bcdc59bbc2c5289ff4eeb9e08
 => => extracting sha256:c21204f5797c735b45941b864c2f521033d92bb12a150b4257afde4f5570a250
 => [2/3] COPY app.py /app/app.py
 => [3/3] WORKDIR /app
 => exporting to image
 => => exporting layers
 => => writing image sha256:0e1484afbd652efd679bc41697b4727d2b9eaf68d9ed64ee4e24b91ff059972e
 => => naming to docker.io/library/hello-docker

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them
```



Questions:

10) What is the purpose of the `FROM` instruction in a Dockerfile?

ANS: The `FROM` instruction specifies the base image for your Docker image.

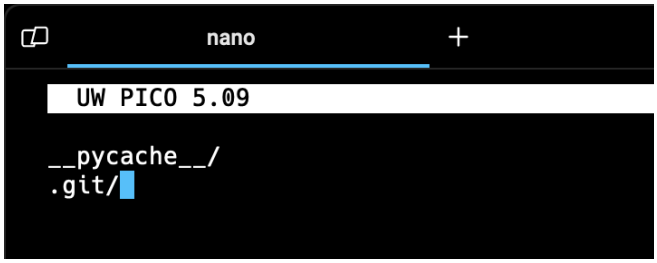
11) How does the `COPY` instruction work in Dockerfile?

ANS: The `COPY` instruction in a Dockerfile copies files or directories from your local filesystem into the Docker image. It takes two arguments: the source path on your host machine and the destination path inside the container.

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

ANS: The main difference is that `CMD` sets the default command to run in a container but can be overridden during `docker run`, while `ENTRYPOINT` defines the main executable that always runs and typically cannot be overridden

Exercise 2: Optimizing Dockerfile with Layers and Caching




```
~/Desktop/Projects/hw-py-app (1.984s)
docker build -t hello-docker-optimized .

[+] Building 1.1s (8/8) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 168B
 => [internal] load .dockerignore
 => => transferring context: 71B
 => [internal] load metadata for docker.io/library/python:3.9-slim
 => [internal] load build context
 => => transferring context: 64B
 => [1/3] FROM docker.io/library/python:3.9-slim@sha256:2851c06da1fdc3c451784beef8aa31d1a313d8e3fc122e4a1891085a104b7cfb
 => CACHED [2/3] WORKDIR /app
 => [3/3] COPY app.py /app/
 => exporting to image
 => => exporting layers
 => => writing image sha256:0a75f2be1b7178e33ac9236d22750cdd8ba4df8ccf56225ffa72d136e1c131d1
 => => naming to docker.io/library/hello-docker-optimized
```

Questions:

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

ANS: Docker layers are individual instructions in a Dockerfile like `FROM`, `COPY`, `RUN`, each creating a new layer in the image. Layers are cached, so if a layer hasn't changed, Docker reuses it in future builds, which speeds up build times.

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

ANS: Docker's build cache stores the results of previously executed layers instructions in a Dockerfile. If a layer hasn't changed, Docker reuses the cached result instead of re-executing it, which significantly speeds up the build process

15) What is the role of the `.dockerignore` file?

ANS: The .dockerignore file specifies files and directories that should be excluded from the Docker build context, preventing them from being copied into the image. This helps reduce the image size, speeds up the build process, and avoids including unnecessary or sensitive files in the final image.

Exercise 3: Multi-Stage Builds

```
~/Desktop/Projects (0.03s)
mkdir go-docker-app

~/Desktop/Projects (0.027s)
cd go-docker-app/

~/Desktop/Projects/go-docker-app (0.034s)
touch main.go

~/Desktop/Projects/go-docker-app (12.258s)
nano main.go

~/Desktop/Projects/go-docker-app (0.03s)
touch Dockerfile

~/Desktop/Projects/go-docker-app (2m 31.10s)
nano Dockerfile
```

```
# Go application builder
FROM golang:1.17 as builder

# Working directory
WORKDIR /app

# Copying Go application into container
COPY main.go .

ENV GO111MODULE=off

# Compiling
RUN go build -o hello-world

# creating a smaller image using Alpine
FROM alpine:latest

# Copying compiled Go binary from the builder
COPY --from=builder /app/hello-world /usr/local/bin/hello-world

# Location to run the Go binary
ENTRYPOINT ["/usr/local/bin/hello-world"]
```

```
~/Desktop/Projects/go-docker-app (2.486s)
docker build -t hello-world-multistage .

[+] Building 1.6s (12/12) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 533B
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load metadata for docker.io/library/alpine:latest
 => [internal] load metadata for docker.io/library/golang:1.17
 => [internal] load build context
 => => transferring context: 124B
 => [builder 1/4] FROM docker.io/library/golang:1.17@sha256:87262e4a4c7db56158a80a18fefdc4fee5accc41b59cde821e691d05541bbb18
 => CACHED [stage-1 1/2] FROM docker.io/library/alpine:latest@sha256:beefdbd8a1da6d2915566fde36db9db0b524eb737fc57cd1367effd16dc0d06d
 => CACHED [builder 2/4] WORKDIR /app
 => CACHED [builder 3/4] COPY main.go .
 => [builder 4/4] RUN go build -o hello-world
 => [stage-1 2/2] COPY --from=builder /app/hello-world /usr/local/bin/hello-world
 => exporting to image
 => => exporting layers
 => => writing image sha256:3b288a389f64d87c17efef530e5d047c937c998818739948634fb5e7a95b0187
 => => naming to docker.io/library/hello-world-multistage

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them

~/Desktop/Projects/go-docker-app (0.525s)
docker run hello-world-multistage

Hello, World!
```

Questions:

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

ANS: Multi-stage builds create smaller, more secure images by separating the build and runtime environments. They reduce image size by excluding unnecessary files and dependencies. This also improves build efficiency and security.

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

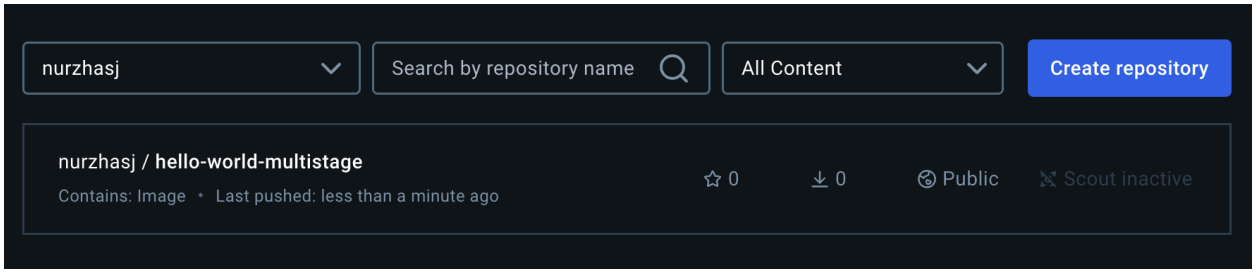
ANS: Multi-stage builds help reduce Docker image size by copying only the necessary files, like the compiled application, into the final image. This removes build tools and unnecessary dependencies, resulting in a leaner image.

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

ANS: Multi-stage builds help make Docker images smaller and more secure by keeping only the important files. They are useful for building code and making lightweight containers, especially in production or when testing.

Exercise 4: Pushing Docker Images to Docker Hub

```
~/Desktop/Projects/go-docker-app (0.111s)
docker ps -a
CONTAINER ID   IMAGE                     COMMAND                  CREATED          STATUS                      PORTS     NAMES
f24be592c356   hello-world-multistage    "/usr/local/bin/hell…"   10 minutes ago   Exited (0) 10 minutes ago             modest_euclid

~/Desktop/Projects/go-docker-app (0.206s)
docker tag hello-world-multistage nurzhasj/hello-world-multistage

~/Desktop/Projects/go-docker-app (15.541s)
docker login

Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head over to https://hub.docker.com to create one.
Username: nurzhasj
Password:
Login Succeeded

Logging in with your password grants your terminal complete access to your account.
For better security, log in with a limited-privilege personal access token. Learn more at https://docs.docker.com/go/access-tokens/

~/Desktop/Projects/go-docker-app (12.785s)
docker push nurzhasj/hello-world-multistage

Using default tag: latest
The push refers to repository [docker.io/nurzhasj/hello-world-multistage]
d8753b4bad53: Pushed
16113d51b718: Mounted from library/alpine
latest: digest: sha256:7701c4d3aeeca591dac8855ce3345fd86b4c3f780bbe74d7fa3c233422556df0 size: 738
```



Questions:

19)What is the purpose of Docker Hub in containerization?

ANS: Docker Hub is a central repository for storing, sharing Docker images. It allows developers to easily push, pull, collaborate on containerized applications.

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

ANS: docker tag local-image <username>/repository:tag

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

ANS: Tagging the image, logging to Docker, pushing the image to Docker HUB

Commands:

1) docker tag image-name <username>/repository:tag
2) docker login
3) docker push <username>/repository:tag