Web Application Development - Assignment 1

First and Last Name: Tamiris Abildayeva

Link to GitHub:

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.

```
[tamirisabildayeva@MBP-Tamiris ~ % docker --version
Docker version 27.1.1, build 6312585
[tamirisabildayeva@MBP-Tamiris ~ %
```

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

```
tamirisabildayeva@MBP-Tamiris ~ % 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/

tamirisabildayeva@MBP-Tamiris ~ %
```

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

        Docker Engine is necessary for facilitating communication between the main Docker components and for running and managing containers. **The server** runs a background program called a daemon to manage containers, images, and volumes. **The REST API** organizes communication between the Docker client and the Docker daemon. The client allows users to interact with the server using commands in the interface (**CLI**).

    - How does Docker compare to traditional virtual machines?

        Docker and traditional **virtual machines** (VMs) are used to run applications but work differently. **Docker** containers are smaller, start up faster, use fewer resources, and are more portable. **VMs** are bigger, provide stronger isolation, and can be more complex to move.

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

        When you run the command docker run hello-world, the output will include "Hello, Docker!" This message confirms that Docker installation is working correctly and signifies that Docker is installed and running, then we can pull images from Docker Hub and run containers, and we can start using Docker to run our applications.

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>.

```
tamirisabildayeva@MBP-Tamiris ~ % 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 quickview nginx
tamirisabildayeva@MBP-Tamiris ~ %
```

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

```
tamirisabildayeva@MBP-Tamiris ~ % docker images
REPOSITORY    TAG       IMAGE ID       CREATED         SIZE
nginx         latest    195245f0c792   5 weeks ago     193MB
hello-world   latest    ee301c921b8a   16 months ago   9.14kB
tamirisabildayeva@MBP-Tamiris ~ %
```

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

```
tamirisabildayeva@MBP-Tamiris ~ % docker run -d nginx
83eedf413476520a3850ef1908bc07dfb53fc85e5b8e53ce82b382e679f92531
tamirisabildayeva@MBP-Tamiris ~ %
```

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

```
tamirisabildayeva@MBP-Tamiris ~ % docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED              STATUS              PORTS     NAMES
83eedf413476   nginx     "/docker-entrypoint.…"   About a minute ago   Up About a minute   80/tcp    flamboyant_kirch
tamirisabildayeva@MBP-Tamiris ~ % docker stop 83eedf413476
83eedf413476
tamirisabildayeva@MBP-Tamiris ~ % docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
tamirisabildayeva@MBP-Tamiris ~ %
```

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

     Docker pull downloads an image. Docker run creates and starts a container from the downloaded image.

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

     With the command docker ps, we can immediately view running containers along with their IDs and statuses.

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

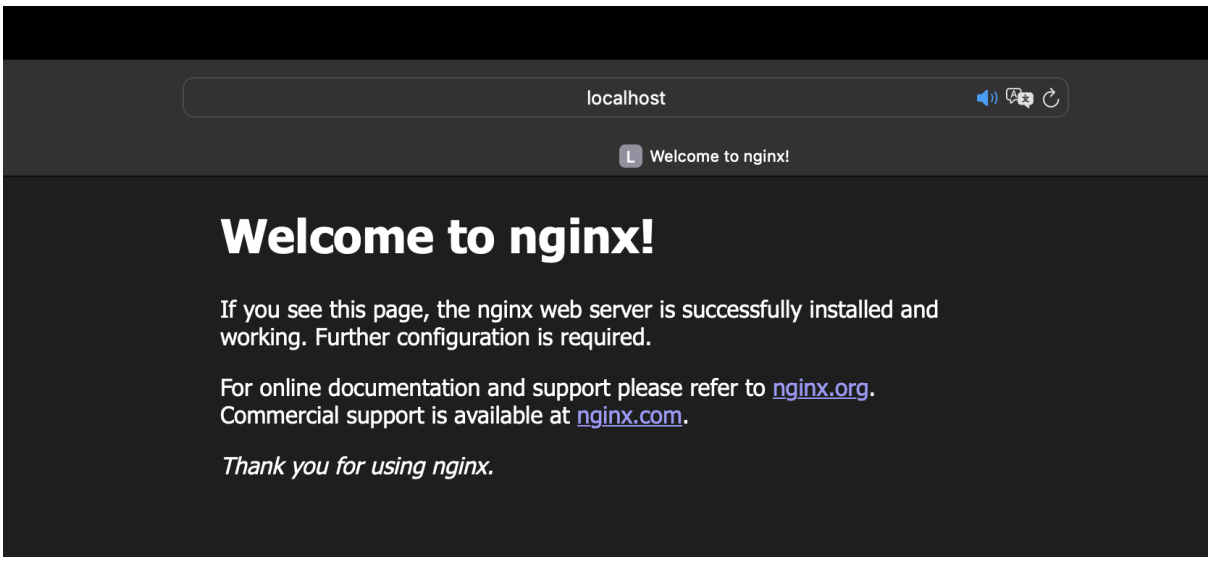
     Once a container is stopped, it stays on our system but is not running. Restart it using the command docker start <container-id>.

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.

```
tamirisabildayeva@MBP-Tamiris ~ % docker run -d -p 8080:80 nginx
4416faee1040e500ab59775388182f6172107acddcd374a527b46408dbc87b98
```

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



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

```
[tamirisabildayeva@MBP-Tamiris ~ % docker exec -it 5c117518d4fa /bin/bash

What's next:
    Try Docker Debug for seamless, persistent debugging tools in any container or image → docker debug 5c117518d4fa
    Learn more at https://docs.docker.com/go/debug-cli/
Error response from daemon: container 5c117518d4fa0ad4cae139b58dc1d0ccd4e780cd7976f122986270a6b304efd4 is not running
[tamirisabildayeva@MBP-Tamiris ~ %
tamirisabildayeva@MBP-Tamiris ~ %
```

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

```
[tamirisabildayeva@MBP-Tamiris ~ % docker stop 5c117518d4fa
5c117518d4fa
[tamirisabildayeva@MBP-Tamiris ~ % docker rm  5c117518d4fa
5c117518d4fa
tamirisabildayeva@MBP-Tamiris ~ %
```

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

     Port mapping is to connect a port on the host to a port on a container. For example, in our case, a container runs a web server on port 80, and we can confidently map it to port 8080 on our host. This is important because it provides external access to services running inside the container.

   - What is the purpose of the docker exec command?

     The docker exec command is used to run a new command in a running container without stopping it. This is important for tasks like debugging, installing software, or checking logs.

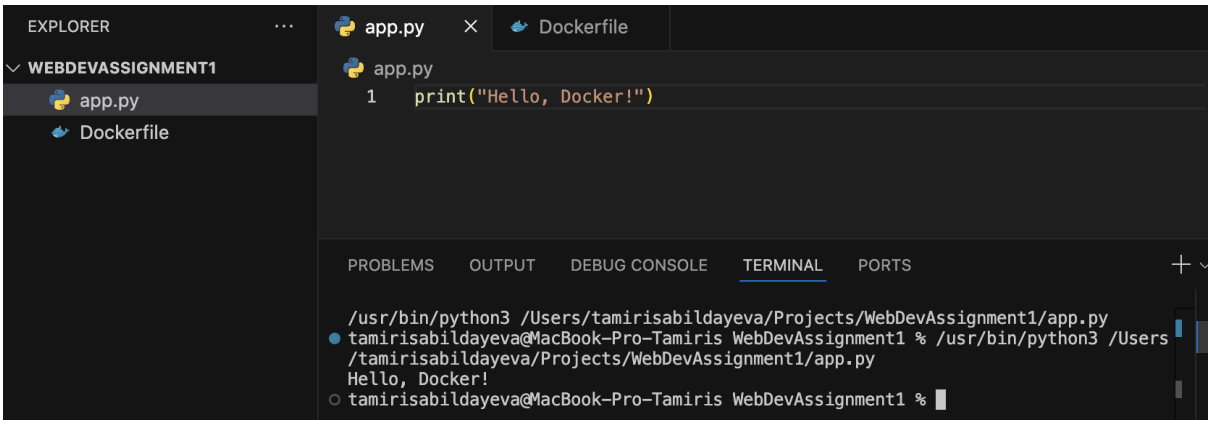
- How do you ensure that a stopped container does not consume system resources?

  To free up system resources, need to use the command docker rm <container-id> to delete a stopped container.

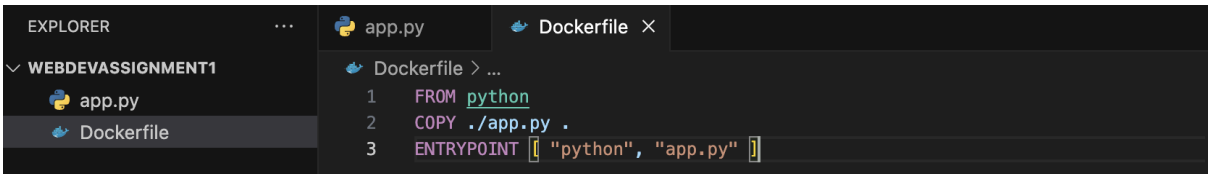
Dockerfile

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 ..

```
tamirisabildayeva@MacBook-Pro-Tamiris WebDevAssignment1 % docker build -t hello-docker .
[+] Building 60.3s (7/7) FINISHED                                                                                  docker:desktop-linux
 => [internal] load build definition from Dockerfile                                                                                  0.0s
 => => transferring dockerfile: 135B                                                                                                  0.0s
 => [internal] load metadata for docker.io/library/python:latest                                                                      2.8s
 => [internal] load .dockerignore                                                                                                     0.0s
 => => transferring context: 2B                                                                                                       0.0s
 => [internal] load build context                                                                                                     0.0s
 => => transferring context: 93B                                                                                                      0.0s
 => [1/2] FROM docker.io/library/python:latest@sha256:7859853e7607927aa1d1b1a5a2f9e580ac90c2b66feeb1b77da97fed03b1  57.4s
 => => resolve docker.io/library/python:latest@sha256:7859853e7607927aa1d1b1a5a2f9e580ac90c2b66feeb1b77da97fed03b1c  0.0s
 => => sha256:9c1a7e7e41337f687fa7e5196aada121fd4897cb32f28139970933ed9a37cee1 2.33kB / 2.33kB                                        0.0s
 => => sha256:56c9b9253ff98351db158cb6789848656b8d54f411c0037347bf2358efb18f39 49.59MB / 49.59MB                                     29.6s
 => => sha256:364d19f59f69474a80c53fc78da91f85553e16e8ba6a28063cbebf259821119e 23.59MB / 23.59MB                                     19.4s
 => => sha256:843b1d8321825bc8302752ae003026f13bd15c6eef2efe032f3ca1520c5bbc07 64.00MB / 64.00MB                                     22.8s
 => => sha256:7859853e7607927aa1d1b1a5a2f9e580ac90c2b66feeb1b77da97fed03b1ccbe 9.72kB / 9.72kB                                         0.0s
 => => sha256:dbb5b79c2668bb64927f07f3dfcb3d0c506f2ab8c300a9cf40718bda1c06600b 5.86kB / 5.86kB                                         0.0s
 => => sha256:a348c2a8d94613f34ce7d0ac4fd4e51800d8f4aa8a0bc9347fe5c8436b4c3bd5 202.65MB / 202.65MB                                    53.4s
 => => sha256:353de11681b2d36db971d2402ebdeb137a2651bc3fcf5545329be637b6b5c6a4 6.24MB / 6.24MB                                        25.0s
 => => sha256:799b63efcab5e4f11215dd6a8caf59d117ae572e0589bc0c4ba40dcb61136f48 23.65MB / 23.65MB                                      35.6s
 => => sha256:238a68c3d761a41b503e61aed92cb67f43c456139d427d47ef96dfa2caca87c4 251B / 251B                                            29.8s
 => => extracting sha256:56c9b9253ff98351db158cb6789848656b8d54f411c0037347bf2358efb18f39                                             1.1s
 => => extracting sha256:364d19f59f69474a80c53fc78da91f85553e16e8ba6a28063cbebf259821119e                                             0.3s
 => => extracting sha256:843b1d8321825bc8302752ae003026f13bd15c6eef2efe032f3ca1520c5bbc07                                             1.3s
 => => extracting sha256:a348c2a8d94613f34ce7d0ac4fd4e51800d8f4aa8a0bc9347fe5c8436b4c3bd5                                             3.2s
 => => extracting sha256:353de11681b2d36db971d2402ebdeb137a2651bc3fcf5545329be637b6b5c6a4                                             0.1s
 => => extracting sha256:799b63efcab5e4f11215dd6a8caf59d117ae572e0589bc0c4ba40dcb61136f48                                             0.4s
 => => extracting sha256:238a68c3d761a41b503e61aed92cb67f43c456139d427d47ef96dfa2caca87c4                                             0.0s
 => [2/2] COPY ./app.py .                                                                                                             0.1s
 => exporting to image                                                                                                                0.0s
 => => exporting layers                                                                                                               0.0s
 => => writing image sha256:e988581edc1b87b539708ab6bb75a0ec3951342e6aee5fb5afab76e84ef9006b                                          0.0s
 => => naming to docker.io/library/hello-docker                                                                                       0.0s

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/kx6utoo7stw6fpjewpz13mpg4

What's next:
    View a summary of image vulnerabilities and recommendations → docker scout quickview
tamirisabildayeva@MacBook-Pro-Tamiris WebDevAssignment1 %
```

- Run the container using docker run hello-docker.

```
tamirisabildayeva@MacBook-Pro-Tamiris WebDevAssignment1 % docker run hello-docker
Hello, Docker!
tamirisabildayeva@MacBook-Pro-Tamiris WebDevAssignment1 %
```

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

     FROM is used to specify the base image for the container. It determines the starting point for building a custom image.

   - How does the COPY instruction work in Dockerfile?

     COPY is used to transfer files or directories from the host machine to the container. Only need to specify the source path on the host and the destination path in the container.

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

     CMD for default options and ENTRYPOINT for basic commands.

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.

```
EXPLORER
WEBDEVASSIGNMENT1
  .dockerignore
  app.py
  Dockerfile
  ignore-file.txt

app.py | Dockerfile | .dockerignore | ignore-file.txt
Dockerfile > ENTRYPOINT
1  # FROM python
2  FROM python AS initial_stage
3  COPY ./app.py .
4  # ENTRYPOINT [ "python", "app.py" ]
5
6  FROM python:slim AS running_stage
7  COPY --from=initial_stage /app.py .
8  ENTRYPOINT [ "python", "app.py" ]
```

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

```
EXPLORER
WEBDEVASSIGNMENT1
  .dockerignore
  app.py
  Dockerfile
  ignore-file.txt

app.py | Dockerfile | .dockerignore | ignore-file.txt
.dockerignore
1  Dockerfile
2  ignore-file.txt
```

```
EXPLORER
WEBDEVASSIGNMENT1
  .dockerignore
  app.py
  Dockerfile
  ignore-file.txt

app.py | Dockerfile | .dockerignore | ignore-file.txt
ignore-file.txt
1  ignore text
```

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

```
tamirisabildayeva@MacBook-Pro-Tamiris WebDevAssignment1 % docker build -t hello-docker-multi-stage .
[+] Building 0.9s (10/10) FINISHED                                                          docker:desktop-linux
 => [internal] load build definition from Dockerfile                                                       0.0s
 => => transferring dockerfile: 275B                                                                       0.0s
 => [internal] load metadata for docker.io/library/python:slim                                             0.8s
 => [internal] load metadata for docker.io/library/python:latest                                           0.8s
 => [internal] load .dockerignore                                                                          0.0s
 => => transferring context: 103B                                                                          0.0s
 => [internal] load build context                                                                          0.0s
 => => transferring context: 64B                                                                           0.0s
 => [initial_stage 1/2] FROM docker.io/library/python:latest@sha256:7859853e7607927aa1d1b1a5a2f9e580ac90c2b66feeb1b77  0.0s
 => [running_stage 1/2] FROM docker.io/library/python:slim@sha256:15bad989b293be1dd5eb26a87ecacadaee1559f98e29f02bf6d  0.0s
 => CACHED [initial_stage 2/2] COPY ./app.py .                                                             0.0s
 => CACHED [running_stage 2/2] COPY --from=initial_stage /app.py .                                         0.0s
 => exporting to image                                                                                     0.0s
 => => exporting layers                                                                                    0.0s
 => => writing image sha256:4d6c44d84e51f36b60fb85904af0fda0d537c663b3ce21f4818064ad61f8a5c5               0.0s
 => => naming to docker.io/library/hello-docker-multi-stage                                                0.0s

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/mjvgeuhveaj37ljn1rr1d0543

What's next:
    View a summary of image vulnerabilities and recommendations → docker scout quickview
tamirisabildayeva@MacBook-Pro-Tamiris WebDevAssignment1 %
```

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

```
tamirisabildayeva@MacBook-Pro-Tamiris WebDevAssignment1 % docker images -a
REPOSITORY                  TAG       IMAGE ID       CREATED          SIZE
hello-docker-multi-stage    latest    4d6c44d84e51   3 minutes ago    150MB
hello-docker                latest    e988581edc1b   15 minutes ago   1.02GB
```

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

        Docker uses layers to track changes such as adding files or installing software. This helps to keep image sizes small and speeds up the build process by allowing Docker to reuse unchanged layers from its cache.

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

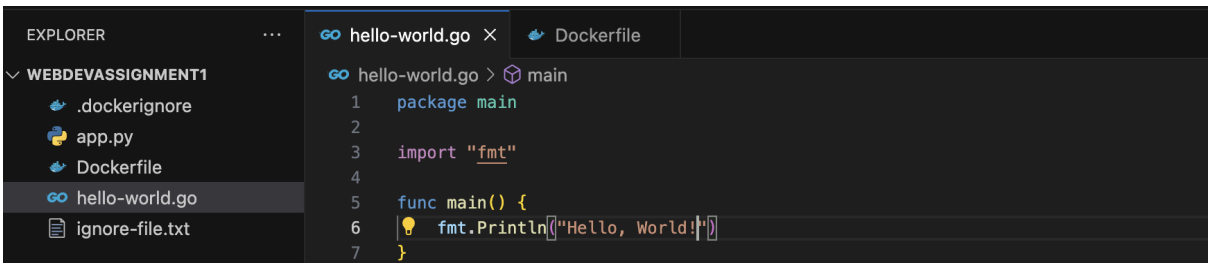
        The Docker build cache stores the layers created during the image build. When rebuilding, Docker checks for changes and reuses the cached layers instead of rebuilding, speeding up the process, especially for large or complex builds.

    - What is the role of the .dockerignore file?

        To ignore unnecessary files and directories from the Docker image build process, reducing image size and speeding up the 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.

```
tamirisabildayeva@MacBook-Pro-Tamiris WebDevAssignment1 % docker build -t hello-docker-go .
[+] Building 28.7s (8/8) FINISHED                                                                    docker:desktop-linux
 => [internal] load build definition from Dockerfile                                                                  0.0s
 => => transferring dockerfile: 199B                                                                                  0.0s
 => [internal] load metadata for docker.io/library/golang:latest                                                      3.2s
 => [internal] load .dockerignore                                                                                     0.0s
 => => transferring context: 103B                                                                                     0.0s
 => [internal] load build context                                                                                     0.0s
 => => transferring context: 152B                                                                                     0.0s
 => [1/3] FROM docker.io/library/golang:latest@sha256:2fe82a3f3e006b4f2a316c6a21f62b66e1330ae211d039bb8d1128e12ed57b  23.7s
 => => resolve docker.io/library/golang:latest@sha256:2fe82a3f3e006b4f2a316c6a21f62b66e1330ae211d039bb8d1128e12ed57bf  0.0s
 => => sha256:2737b855c4589a4564aba00d16fc897fa4b8489ed918ec317ff872dc988fbbfe 2.86kB / 2.86kB                        0.0s
 => => sha256:ecb27c98d5b9e78892d876693427ae0a01e3113b36989718360a5aa9e319fd80 86.29MB / 86.29MB                     16.3s
 => => sha256:a355a3cac949bed5cda9c62103ceb0f004727cedcd2a17d7c9836aea1a452fda 70.62MB / 70.62MB                     21.1s
 => => sha256:83f1399aa9166438efeea4696812a3fa3f3397ff114492577a919f9b09f3c1ea 126B / 126B                            0.9s
 => => sha256:2fe82a3f3e006b4f2a316c6a21f62b66e1330ae211d039bb8d1128e12ed57bf1 9.74kB / 9.74kB                        0.0s
 => => sha256:c6b19df41c8e5073a0a91d530741ff8768020b8de487b3f6902955322eab9d0e 2.32kB / 2.32kB                        0.0s
 => => sha256:4f4fb700ef54461cfa02571ae0db9a0dc1e0cdb5577484a6d75e68dc38e8acc1 32B / 32B                              1.3s
 => => extracting sha256:ecb27c98d5b9e78892d876693427ae0a01e3113b36989718360a5aa9e319fd80                             1.2s
 => => extracting sha256:a355a3cac949bed5cda9c62103ceb0f004727cedcd2a17d7c9836aea1a452fda                             2.4s
 => => extracting sha256:83f1399aa9166438efeea4696812a3fa3f3397ff114492577a919f9b09f3c1ea                             0.0s
 => => extracting sha256:4f4fb700ef54461cfa02571ae0db9a0dc1e0cdb5577484a6d75e68dc38e8acc1                             0.0s
 => [2/3] COPY ./hello-world.go /                                                                                     0.2s
 => [3/3] RUN ["go", "build", "/hello-world.go"]                                                                      1.6s
 => exporting to image                                                                                                0.1s
 => => exporting layers                                                                                               0.1s
 => => writing image sha256:99ad4f5b5fe58b54d8024bf40aca61426b5f50947481db187ad545ba15a2ef12                          0.0s
 => => naming to docker.io/library/hello-docker-go                                                                    0.0s

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/kr6n8ialy6nrncqjpaf5bf5m2

What's next:
    View a summary of image vulnerabilities and recommendations → docker scout quickview
tamirisabildayeva@MacBook-Pro-Tamiris WebDevAssignment1 %
```

EXPLORER

WEBDEVASSIGNMENT1

- .dockerignore
- app.py
- Dockerfile
- hello-world.go
- ignore-file.txt

hello-world.go | Dockerfile

Dockerfile > ...

```
FROM golang AS building_stage
COPY ./hello-world.go /
RUN ["go", "build", "/hello-world.go"]

FROM alpine AS running_stage
COPY --from=building_stage /go/hello-world .
ENTRYPOINT [ "./hello-world" ]
```

```
tamirisabildayeva@MacBook-Pro-Tamiris WebDevAssignment1 % docker build -t hello-docker-go-multi-stage .
[+] Building 3.9s (11/11) 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                                                      2.8s
 => [internal] load metadata for docker.io/library/golang:latest                                                      0.8s
 => [internal] load .dockerignore                                                                                     0.0s
 => => transferring context: 103B                                                                                     0.0s
 => [internal] load build context                                                                                     0.0s
 => => transferring context: 72B                                                                                      0.0s
 => [running_stage 1/2] FROM docker.io/library/alpine:latest@sha256:beefdbd8a1da6d2915566fde36db9db0b524eb737fc57cd13  1.1s
 => => resolve docker.io/library/alpine:latest@sha256:beefdbd8a1da6d2915566fde36db9db0b524eb737fc57cd1367effd16dc0d06  0.0s
 => => sha256:beefdbd8a1da6d2915566fde36db9db0b524eb737fc57cd1367effd16dc0d06d 1.85kB / 1.85kB                        0.0s
 => => sha256:9cee2b382fe2412cd77d5d437d15a93da8de373813621f2e4d406e3df0cf0e7c 528B / 528B                            0.0s
 => => sha256:c157a85ed455142fd79bff5dce951fd5f5b0d0c6e45e6f54cfd0c4e2bdec587b 1.49kB / 1.49kB                        0.0s
 => => sha256:cf04c63912e16506c4413937c7f4579018e4bb25c272d989789cfba77b12f951 4.09MB / 4.09MB                        0.9s
 => => extracting sha256:cf04c63912e16506c4413937c7f4579018e4bb25c272d989789cfba77b12f951                             0.1s
 => [building_stage 1/3] FROM docker.io/library/golang:latest@sha256:2fe82a3f3e006b4f2a316c6a21f62b66e1330ae211d039bb  0.0s
 => CACHED [building_stage 2/3] COPY ./hello-world.go /                                                               0.0s
 => CACHED [building_stage 3/3] RUN ["go", "build", "/hello-world.go"]                                                0.0s
 => [running_stage 2/2] COPY --from=building_stage /go/hello-world .                                                  0.0s
 => exporting to image                                                                                                0.0s
 => => exporting layers                                                                                               0.0s
 => => writing image sha256:b2f82f2fa21c4964f3bec82e8e107eaf215bf4d030f0afe3618f83cfdadf4d41                          0.0s
 => => naming to docker.io/library/hello-docker-go-multi-stage                                                        0.0s

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/ys5xp4itmb00ywrcgiteeoomo

What's next:
    View a summary of image vulnerabilities and recommendations → docker scout quickview
tamirisabildayeva@MacBook-Pro-Tamiris WebDevAssignment1 %
```

```
tamirisabildayeva@MacBook-Pro-Tamiris WebDevAssignment1 % docker images -a
REPOSITORY                  TAG       IMAGE ID       CREATED          SIZE
hello-docker-go-multi-stage latest    b2f82f2fa21c   23 seconds ago   11MB
hello-docker-go             latest    99ad4f5b5fe5   2 minutes ago    876MB
hello-docker-multi-stage    latest    4d6c44d84e51   17 minutes ago   150MB
hello-docker                latest    e988581edc1b   30 minutes ago   1.02GB
```

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

        By copying only the essential files to the final image, its size is reduced. Smaller images have a smaller attack surface and reusing layers can significantly reduce build times.

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

        Multi-stage builds can help make the final product smaller and faster. This is done by including tools and extra stuff needed to build the software early on, but not including them in the finished product. This makes the final software cleaner and more efficient.

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

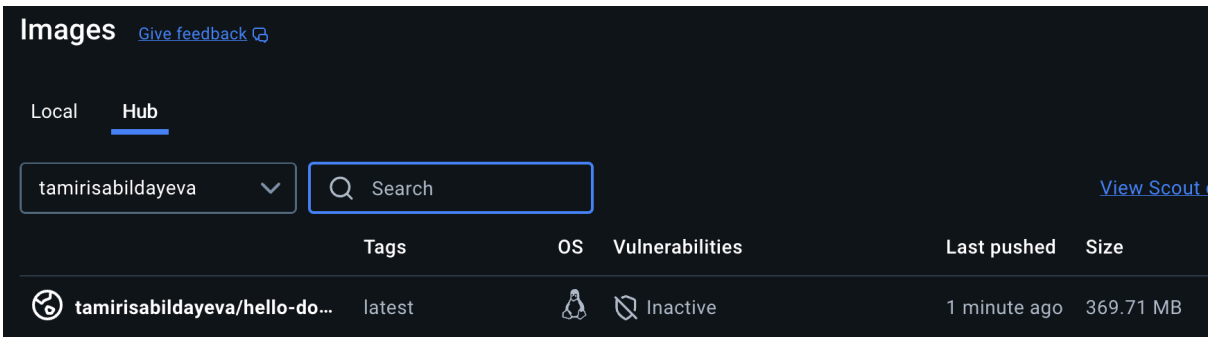
        When creating applications from source code, make the file size smaller by removing development tools, and enhance security by getting rid of any sensitive files or credentials.

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.

```
[tamirisabildayeva@MacBook-Pro-Tamiris WebDevAssignment1 % docker tag hello-docker tamirisabildayeva/hello-docker
[tamirisabildayeva@MacBook-Pro-Tamiris WebDevAssignment1 % docker login

USING WEB BASED LOGIN
To sign in with credentials on the command line, use 'docker login -u <username>'

Your one-time device confirmation code is: JDDZ-NCWV
Press ENTER to open your browser or submit your device code here: https://login.docker.com/activate

Waiting for authentication in the browser…

Login Succeeded
[tamirisabildayeva@MacBook-Pro-Tamiris WebDevAssignment1 % docker push tamirisabildayeva/hello-docker
Using default tag: latest
The push refers to repository [docker.io/tamirisabildayeva/hello-docker]
3d9f92c7f3c2: Pushed
075c5d4a5750: Mounted from library/python
b2a9d25bd8bf: Mounted from library/python
90eec9d96301: Mounted from library/python
cc322244ab6f: Mounted from library/python
61f03db12f5b: Mounted from library/golang
391e69d2f4b6: Mounted from library/golang
61f1e6ae67cb: Mounted from library/golang
latest: digest: sha256:79e5c9a9acfedbb23f87e90c845037312ef1d33882ebb4994d40990406bf146a size: 2003
tamirisabildayeva@MacBook-Pro-Tamiris WebDevAssignment1 %
```

    - Verify that the image is available on Docker Hub and share it with others.



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

     To simplify the storage, distribution, and management of different versions of images and provide official images for popular applications. In simple words, it makes easier the process of working with container technology.

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

     Via command docker tag hello-docker <your-username>/hello-docker. After that, need to log in if this part was missed previously and push it to docker.

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

     After logging in (docker login), set the local name (in profile settings), tag the image (docker tag hello-docker <your-username>/hello-docker), and push the tagged image (docker push <your-username>/hello-docker).