Assignment 1

Done by: Moldir Polat

Intro to Containerization: Docker

Exercise 1: Installing Docker

1. **Objective**: Install Docker on your local machine.
2. **Steps**:

```
C:\Users\User\Desktop\KBTU-Moldir-Polat\3-semester\Web-app-dev>docker --version
Docker version 27.1.1, build 6312585
```

```
C:\Users\User\Desktop\KBTU-Moldir-Polat\3-semester\Web-app-dev>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 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/
```

3. **Questions**:
   - What are the key components of Docker (e.g., Docker Engine, Docker CLI)?
     Key components of Docker are
     Docker Engine – main part;
     Docker CLI – for interaction;
     Docker Daemon – service to manage containers, images etc;
     Images – light executable packages used to create Containers;
     Containers – running intances that run applications in isolated environment;
     Dockerfile – file with instructions how to build Image;
     Docker hub – public cloud-based repository with Docker images;
     Docker compose – tool for multi-container applications;
     Docker Volumes – mechanisms for storing data;
     Docker Network – for communication between containers and exrernal networks;
     Docker Registry – private service to store Images.

   - How does Docker compare to traditional virtual machines?

The main difference between traditional virtual machines (VMs) and Docker is how they handle virtualization. Docker containers are lighter than virtual machines (VMs) and use less resources because they share the host operating system kernel. Microservices and cloud-native apps benefit greatly from containers' high portability and process-level application isolation. VMs, on the contrary, provide stronger isolation by running entire operating systems on top of a hypervisor, but at the cost of poorer performance, and higher resource usage. Virtual machines are a better choice when there are several running OSes or when you need total isolation of application.

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

  The output is in the screenshot above. It shows that Docker was unable to find the "hello-world" image locally, so it downloaded the most recent version from Docker Hub, confirmed it, and built a container. The "Hello from Docker!" message verifies that Docker is set up and operating properly. The output describes the stages by which the Docker client interacts with the Docker daemon, retrieves the image from Docker Hub, builds a container, runs the image, and also puts the outcome to the terminal. Generally it shows that Docker has been configured correctly and is prepared to execute containers.

Exercise 2: Basic Docker Commands

1. **Objective**: Familiarize yourself with basic Docker commands.
2. **Steps**:

```
C:\Users\User\Desktop\KBTU-Moldir-Polat\3-semester\Web-app-dev>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
```

```
C:\Users\User\Desktop\KBTU-Moldir-Polat\3-semester\Web-app-dev>docker images
REPOSITORY    TAG       IMAGE ID       CREATED         SIZE
nginx         latest    39286ab8a5e1   5 weeks ago     188MB
hello-world   latest    d2c94e258dcb   16 months ago   13.3kB
```

```
C:\Users\User\Desktop\KBTU-Moldir-Polat\3-semester\Web-app-dev>docker run -d nginx
2a03e48eaebe3c62b24953b3dd1f18f875ed1081426c9e4d0de5a90876c8bb63
```

```
C:\Users\User\Desktop\KBTU-Moldir-Polat\3-semester\Web-app-dev>docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS     NAMES
2a03e48eaebe   nginx     "/docker-entrypoint.…"   36 seconds ago   Up 36 seconds   80/tcp    boring_maxwell
```

```
C:\Users\User\Desktop\KBTU-Moldir-Polat\3-semester\Web-app-dev>docker stop boring_maxwell
boring_maxwell

C:\Users\User\Desktop\KBTU-Moldir-Polat\3-semester\Web-app-dev>docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES

C:\Users\User\Desktop\KBTU-Moldir-Polat\3-semester\Web-app-dev>
```

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

     `docker pull` just fetches the needed image from registry to local machine, whereas `docker run` fetches the image if it is absent and also starts the container based on it.

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

     We can use command `docker ps`

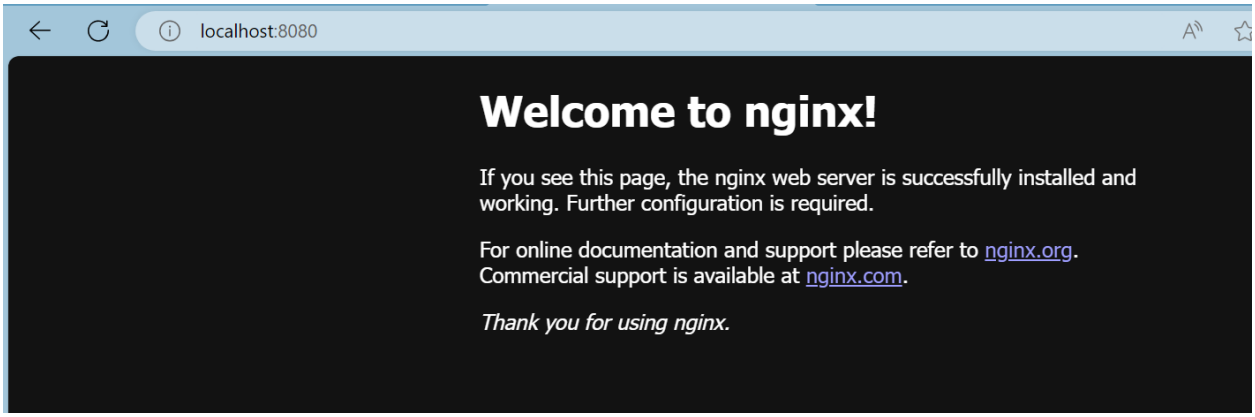
   - What happens to a container after it is stopped? Can it be restarted?

     When container is stopped, its processes are stopped, but the container is not deleted, a state becomes inactive. It can be restarted by using the command `docker start <container_name>.`

Exercise 3: Working with Docker Containers

1. **Objective**: Learn how to manage Docker containers.
2. **Steps**:

```
C:\Users\User\Desktop\KBTU-Moldir-Polat\3-semester\Web-app-dev>docker run -d -p 8080:80 nginx
d1bfdaebf6289ee087081939fd198d01f264b219b42e54f3a9f04031cdbf470f
```




```
C:\Users\User\Desktop\KBTU-Moldir-Polat\3-semester\Web-app-dev>docker exec -it vigorous_snyder /bin/bash
root@d1bfdaebf628:/#
```

```
C:\Users\User\Desktop\KBTU-Moldir-Polat\3-semester\Web-app-dev>docker stop vigorous_snyder
vigorous_snyder

C:\Users\User\Desktop\KBTU-Moldir-Polat\3-semester\Web-app-dev>docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES

C:\Users\User\Desktop\KBTU-Moldir-Polat\3-semester\Web-app-dev>
```

```
C:\Users\User\Desktop\KBTU-Moldir-Polat\3-semester\Web-app-dev>docker rm vigorous_snyder
vigorous_snyder

C:\Users\User\Desktop\KBTU-Moldir-Polat\3-semester\Web-app-dev>docker ps -a
CONTAINER ID   IMAGE         COMMAND                  CREATED        STATUS                     PORTS     NAMES
2a03e48eaebe   nginx         "/docker-entrypoint.…"   23 hours ago   Exited (0) 23 hours ago            boring_maxwell
add2a426182e   hello-world   "/hello"                 25 hours ago   Exited (0) 25 hours ago            elastic_hertz
cdd8346606e4   hello-world   "/hello"                 25 hours ago   Exited (0) 25 hours ago            gallant_jemison

C:\Users\User\Desktop\KBTU-Moldir-Polat\3-semester\Web-app-dev>
```

3. **Questions**:
   - How does port mapping work in Docker, and why is it important?
     `docker run -d -p 8080:80 <image>` maps port 80 inside the container (where the app runs) to port 8080 on the host machine, making the app accessible via localhost:8080. The importance of it can be concluded in the usage of solution by external networks, then if there are multiple containers running on the same internal port, conflicts can be repvented by mapping them to different ports on the host. And last but not least, we are able to give acccess only to specified ports providing security and flexibility to our services.
   - What is the purpose of the `docker exec` command?
     It allows us to run commands inside of already running container. For example we can execute some tasks or interact with processes, debug, inspect the environment.

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

     We use `docker rm <container-id>` to free up all resources consumed by container including storage space, configurations, metatdata.

Dockerfile

Exercise 1: Creating a Simple Dockerfile

1. **Objective**: Write a Dockerfile to containerize a basic application.
2. **Steps**:

```
app.py   ×   Dockerfile
C: > Users > User > Desktop > KBTU-Moldir-Polat > 3-semester > Web-app-dev > Assignment-1 > app.py
1   print("Hello, Docker!")
```

```
app.py        Dockerfile ×
C: > Users > User > Desktop > KBTU-Moldir-Polat > 3-semester > Web-app-dev > Assignment-1 > Dockerfile
1  FROM python:3.9-slim
2
3  WORKDIR /app
4
5  COPY app.py .
6
7  ENTRYPOINT ["python", "app.py"]
8
```

```
C:\Users\User\Desktop\KBTU-Moldir-Polat\3-semester\Web-app-dev\Assignment-1>docker build -t hello-docker .
[+] Building 12.6s (9/9) FINISHED                                                                    docker:desktop-linux
 => [internal] load build definition from Dockerfile                                                                  0.1s
 => => transferring dockerfile: 127B                                                                                  0.0s
 => [internal] load metadata for docker.io/library/python:3.9-slim                                                    4.6s
 => [auth] library/python:pull token for registry-1.docker.io                                                         0.0s
 => [internal] load .dockerignore                                                                                     0.1s
 => => transferring context: 2B                                                                                       0.0s
 => [1/3] FROM docker.io/library/python:3.9-slim@sha256:2851c06da1fdc3c451784beef8aa31d1a313d8e3fc122e4a1891085a104b7cfb  5.2s
 => => resolve docker.io/library/python:3.9-slim@sha256:2851c06da1fdc3c451784beef8aa31d1a313d8e3fc122e4a1891085a104b7cfb  0.0s
 => => sha256:0fa26e0a6c779a41b265beaf4e11ac2899b82fc487fde96e28dec51f08fd831e 3.51MB / 3.51MB                        1.2s
 => => sha256:a657783e238bbb29a690f2f314f9eb40a7bf9ab06f795e95590adf1f86413919 14.74MB / 14.74MB                      2.6s
 => => sha256:b665d04ddefb24a5af0c944a98df2ebfb1e3a26e0a546573041f84e0a4a2150e 251B / 251B                            1.3s
 => => sha256:2851c06da1fdc3c451784beef8aa31d1a313d8e3fc122e4a1891085a104b7cfb 10.41kB / 10.41kB                      0.0s
 => => sha256:d465e807ab2e72c74ec6fa81d1d2751108c7861a9c041f072e3d24b5aaf91fcb 1.75kB / 1.75kB                        0.0s
 => => sha256:397ed8d3163622f16a7ad7f8d235cb365b893a589ce31d79f9d6e61d2a5ae31a 5.22kB / 5.22kB                        0.0s
 => => extracting sha256:0fa26e0a6c779a41b265beaf4e11ac2899b82fc487fde96e28dec51f08fd831e                             0.7s
 => => extracting sha256:a657783e238bbb29a690f2f314f9eb40a7bf9ab06f795e95590adf1f86413919                             2.2s
 => => extracting sha256:b665d04ddefb24a5af0c944a98df2ebfb1e3a26e0a546573041f84e0a4a2150e                             0.0s
 => [internal] load build context                                                                                     0.1s
 => => transferring context: 56B                                                                                      0.0s
 => [2/3] WORKDIR /app                                                                                                2.1s
 => [3/3] COPY app.py .                                                                                               0.1s
 => exporting to image                                                                                                0.2s
 => => exporting layers                                                                                               0.1s
 => => writing image sha256:cf9d94feea9656910d5b220e72dff570c379af7160ac100e7d8233f586db06ab                          0.0s
 => => naming to docker.io/library/hello-docker                                                                       0.0s

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

```
C:\Users\User\Desktop\KBTU-Moldir-Polat\3-semester\Web-app-dev\Assignment-1>docker run hello-docker
Hello, Docker!
```

3. **Questions**:
   - What is the purpose of the `FROM` instruction in a Dockerfile?
     It is a starting point for the Docker image build process. It is like base image from which we inherit environment, libraries, and configurations.
   - How does the `COPY` instruction work in Dockerfile?
     It copies files and resources from our local machine to Docker image. Syntax: COPY <source> <destination>
   - What is the difference between `CMD` and `ENTRYPOINT` in Dockerfile?

     When we use CMD we can override default command by another. However, using ENTRYPOINT default command surely will be executed and other arguments are appended to it.

     Examples:

     CMD ["python", "app.py"] → docker run <image> → python app.py

     docker run <image> python other_script.py → python other_script.py

     ENTRYPOINT ["python", "app.py"] → docker run <image> → (always) python app.py

docker run <image> --version (with arg-s) → python app.py –version

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**:

C: > Users > User > Desktop > KBTU-Moldir-Polat > 3-semester > Web-app-dev > Assignment-1 > exercise-2-with-dockerignore > Dockerfile

```
FROM python:3.9-alpine

WORKDIR /app

COPY requirements.txt /app/

RUN pip install --no-cache-dir -r requirements.txt

COPY . /app

EXPOSE 80

ENV NAME World

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

.dockerignore × | app.py C:\...\exercise-2 2 | Dockerfile C:\...\exercise-2 | app.py C:\...\exercise-2-with-dockerignore 2 | Dockerf

C: > Users > User > Desktop > KBTU-Moldir-Polat > 3-semester > Web-app-dev > Assignment-1 > exercise-2-with-dockerignore > .dockerignore

```
*.pyc
__pycache__/
node_modules
*.log
.git

ignore.txt
ignore1.txt
ignore2.txt
```

```
C:\Users\User\Desktop\KBTU-Moldir-Polat\3-semester\Web-app-dev\Assignment-1\exercise-2-with-dockerignore>docker images
REPOSITORY                TAG         IMAGE ID       CREATED          SIZE
my-app-optimized          latest      50e03f082b56   8 seconds ago    61MB
my-app-original           latest      eec6c5009241   2 minutes ago    455MB
```

3. **Questions**:
   - What are Docker layers, and how do they affect image size and build times?
     Docker layers are instructions that create layers one by one. Executing each layer takes time, affecting to overall build time. Layers can be cached and if there is no change on the layer, it will be reused in the next build. They also affect the image size, because all layers are part of the image. Separating dependency installation from code copying is good practice for optimizing build speed and image size.
   - How does Docker's build cache work, and how can it speed up the build process?
     Docker’s build cache’s main idea is that it checks whether there are any changes between already previous build stepsand new instructions, and if there are no it just reuses the existing results. Docker executes only those steps with changes, in this way build process time can be reduced.
   - What is the role of the `.dockerignore` file?

     It is a file where we write all files and content that should be ignored during build steps and they should not be copied to the image. It helps us to optimize image size and build time.

Exercise 3: Multi-Stage Builds

1. **Objective**: Use multi-stage builds to create leaner Docker images.
2. **Steps**:
   Single stage

main.go | Dockerfile C:\...\ex3-single-stage | Dockerfile C:\...\ex3-multi-stage

C: > Users > User > Desktop > KBTU-Moldir-Polat > 3-semester > Web-app-dev > Assignment-1 > ex3-single-stage > Dockerfile

```
FROM golang:1.20

WORKDIR /app

COPY . .

RUN go mod init hello && go build -o hello

CMD ["./hello"]
```

```
C:\Users\User\Desktop\KBTU-Moldir-Polat\3-semester\Web-app-dev\Assignment-1\ex3-single-stage>docker build -t go-hello-single-stage .
[+] Building 9.1s (10/10) FINISHED                                                                                  docker:desktop-linux
 => [internal] load build definition from Dockerfile                                                                                0.0s
 => => transferring dockerfile: 148B                                                                                                0.0s
 => [internal] load metadata for docker.io/library/golang:1.20                                                                      1.9s
 => [auth] library/golang: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/golang:1.20@sha256:8f9af7094d0cb27cc783c697ac5ba25efdc4da35f8526db21f7aebb0b0b4f18a                0.0s
 => [internal] load build context                                                                                                   0.0s
 => => transferring context: 174B                                                                                                   0.0s
 => CACHED [2/4] WORKDIR /app                                                                                                       0.0s
 => [3/4] COPY . .                                                                                                                  0.1s
 => [4/4] RUN go mod init hello && go build -o hello                                                                                6.5s
 => exporting to image                                                                                                              0.3s
 => => exporting layers                                                                                                             0.2s
 => => writing image sha256:a48e387d7abfcb59bf875a958569f974fa1af16b12f4c9e413918ae46cd0c854                                        0.0s
 => => naming to docker.io/library/go-hello-single-stage                                                                            0.0s

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

C:\Users\User\Desktop\KBTU-Moldir-Polat\3-semester\Web-app-dev\Assignment-1\ex3-single-stage>docker run go-hello-single-stage
Hello, World!
```

Multi-stage:

main.go | Dockerfile C:\...\ex3-single-stage | Dockerfile C:\...\ex3-multi-stage

C: > Users > User > Desktop > KBTU-Moldir-Polat > 3-semester > Web-app-dev > Assignment-1 > ex3-multi-stage > Dockerfile

```
FROM golang:1.20 as builder

WORKDIR /app

COPY . .

RUN go mod init hello && go build -o hello

FROM alpine:latest

WORKDIR /app

COPY --from=builder /app/hello .

CMD ["./hello"]
```

```
C:\Users\User\Desktop\KBTU-Moldir-Polat\3-semester\Web-app-dev\Assignment-1\ex3-multi-stage>docker build -t go-hello-multi-stage .
[+] Building 11.7s (15/15) FINISHED                                                                                  docker:desktop-linux
 => [internal] load build definition from Dockerfile                                                                                  0.1s
 => => transferring dockerfile: 235B                                                                                                  0.0s
 => WARN: FromAsCasing: 'as' and 'FROM' keywords' casing do not match (line 1)                                                        0.1s
 => [internal] load metadata for docker.io/library/golang:1.20                                                                        1.9s
 => [internal] load metadata for docker.io/library/alpine:latest                                                                      3.5s
 => [auth] library/golang:pull token for registry-1.docker.io                                                                         0.0s
 => [auth] library/alpine:pull token for registry-1.docker.io                                                                         0.0s
 => [internal] load .dockerignore                                                                                                     0.1s
 => => transferring context: 2B                                                                                                       0.0s
 => [builder 1/4] FROM docker.io/library/golang:1.20@sha256:8f9af7094d0cb27cc783c697ac5ba25efdc4da35f8526db21f7aebb0b0b4f18a    0.0s
 => [internal] load build context                                                                                                     0.2s
 => => transferring context: 353B                                                                                                     0.0s
 => [stage-1 1/3] FROM docker.io/library/alpine:latest@sha256:beefdbd8a1da6d2915566fde36db9db0b524eb737fc57cd1367effd16dc0d06d  0.2s
 => => resolve docker.io/library/alpine:latest@sha256:beefdbd8a1da6d2915566fde36db9db0b524eb737fc57cd1367effd16dc0d06d          0.1s
 => => sha256:beefdbd8a1da6d2915566fde36db9db0b524eb737fc57cd1367effd16dc0d06d 1.85kB / 1.85kB                                        0.0s
 => => sha256:33735bd63cf84d7e388d9f6d297d348c523c044410f553bd878c6d7829612735 528B / 528B                                           0.0s
 => => sha256:91ef0af61f39ece4d6710e465df5ed6ca12112358344fd51ae6a3b886634148b 1.47kB / 1.47kB                                       0.0s
 => CACHED [builder 2/4] WORKDIR /app                                                                                                 0.0s
 => [builder 3/4] COPY . .                                                                                                            0.1s
 => [stage-1 2/3] WORKDIR /app                                                                                                        0.1s
 => [builder 4/4] RUN go mod init hello && go build -o hello                                                                          7.2s
 => [stage-1 3/3] COPY --from=builder /app/hello .                                                                                    0.1s
 => exporting to image                                                                                                                0.2s
 => => exporting layers                                                                                                               0.1s
 => => writing image sha256:0a11a3f51a1f623cfc50ff2d30e3f55df7b8680fd0f726e9370247423bfbc18b                                          0.0s
 => => naming to docker.io/library/go-hello-multi-stage                                                                               0.0s

 1 warning found (use docker --debug to expand):
 - FromAsCasing: 'as' and 'FROM' keywords' casing do not match (line 1)

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

C:\Users\User\Desktop\KBTU-Moldir-Polat\3-semester\Web-app-dev\Assignment-1\ex3-multi-stage>docker run go-hello-multi-stage
Hello, World!
```

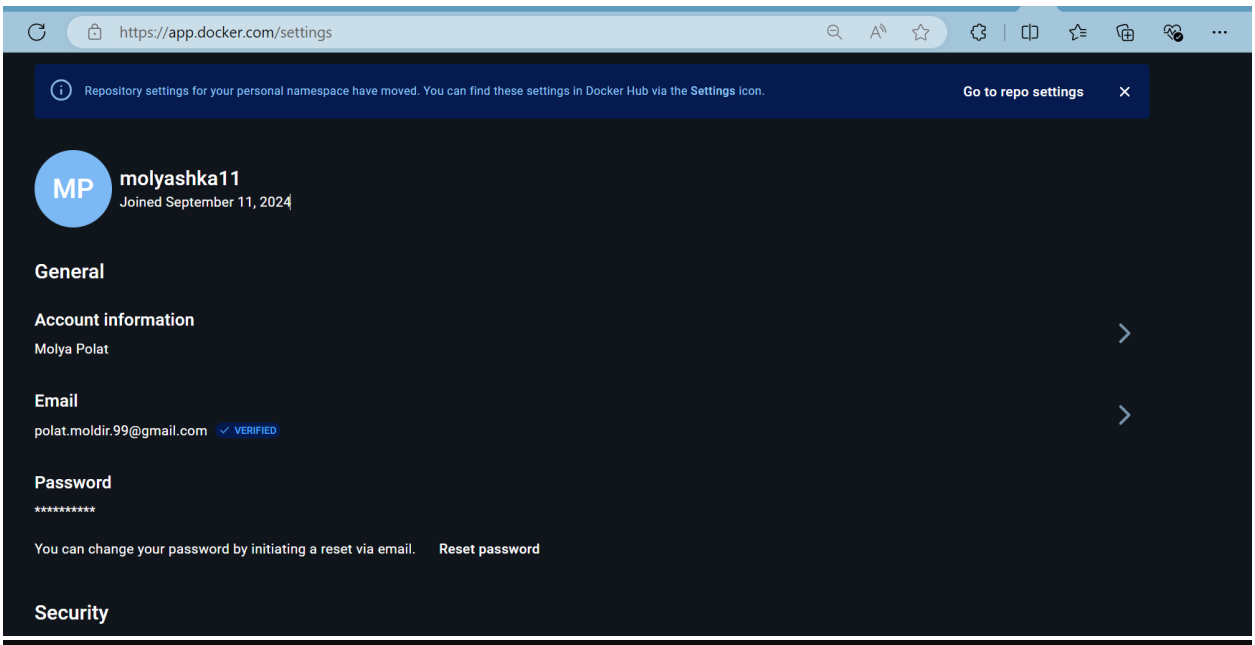
```
C:\Users\User\Desktop\KBTU-Moldir-Polat\3-semester\Web-app-dev\Assignment-1\ex3-multi-stage>docker images
REPOSITORY                  TAG         IMAGE ID       CREATED          SIZE
go-hello-multi-stage        latest      0a11a3f51a1f   11 seconds ago   9.65MB
go-hello-single-stage       latest      a48e387d7abf   5 minutes ago    872MB
```

3. **Questions**:
   - What are the benefits of using multi-stage builds in Docker?
     Multi-stage builds in Docker has some benefits, it can reduce image size by separating the build environment from and the final runtime environment. It compiles the app in one stage and then copies only the final binary to minimal base image, in this way we create more efficient image. Therefore deployment time is reduced as well. It is very useful for optimizing prod environment and continuous deployment process.
   - How can multi-stage builds help reduce the size of Docker images?
     It allows us to use separate build stage that includes all necessary tools and dependencies and then only final output is copied to minimal runtime image. As a result we get small image with the needed components without other tools and dependencies.
   - What are some scenarios where multi-stage builds are particularly useful?

     Multi-stage builds are particularly useful in scenarios where we need efficient Docker image size to improve performance in production environments. It is very usefyl for complex applications where fast build and deployment is important in CI/CD pipelines. Another case is when we have heavy tools to compile code but require simple runtime. It is also beneficial for security, and cross-compilation for different platforms.

Exercise 4: Pushing Docker Images to Docker Hub

1. **Objective**: Learn how to share Docker images by pushing them to Docker Hub.
2. **Steps**:

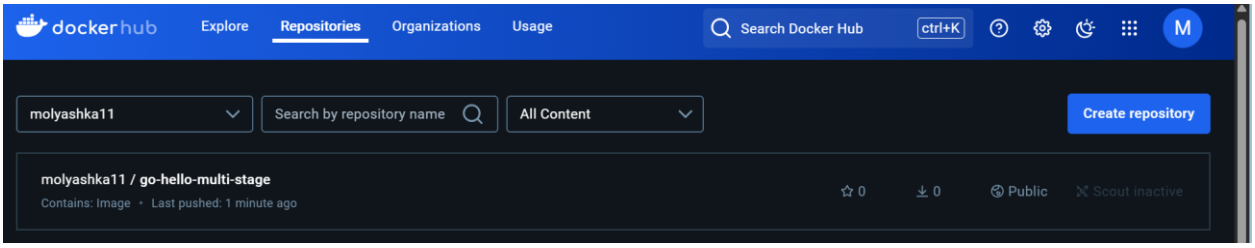



```
C:\Users\User\Desktop\KBTU-Moldir-Polat\3-semester\Web-app-dev\Assignment-1\ex3-multi-stage>docker login
Authenticating with existing credentials...
Login Succeeded
```

```
C:\Users\User\Desktop\KBTU-Moldir-Polat\3-semester\Web-app-dev\Assignment-1\ex3-multi-stage>docker tag go-hello-multi-stage molyashka11/go-hello-multi-stage

C:\Users\User\Desktop\KBTU-Moldir-Polat\3-semester\Web-app-dev\Assignment-1\ex3-multi-stage>docker images
REPOSITORY                      TAG       IMAGE ID       CREATED         SIZE
go-hello-multi-stage            latest    0a11a3f51a1f   3 hours ago     9.65MB
molyashka11/go-hello-multi-stage latest   0a11a3f51a1f   3 hours ago     9.65MB
```

```
C:\Users\User\Desktop\KBTU-Moldir-Polat\3-semester\Web-app-dev\Assignment-1\ex3-multi-stage>docker push molyashka11/go-hello-multi-stage
Using default tag: latest
The push refers to repository [docker.io/molyashka11/go-hello-multi-stage]
4119e8079fdb: Pushed
2c21065e199c: Pushed
63ca1fbb43ae: Mounted from library/alpine
latest: digest: sha256:e1295c18ca8867013bae27a64ec37e0de970242fa8a3a5d38417d964887c4821 size: 945
```



3. **Questions**:
   - What is the purpose of Docker Hub in containerization?
     The purpose of Docker Hub in containerization is that it is a centralized repository where we can store images, manage them and share with others. There we can find pre-built images, pull them for deployment. Overall it enhances development workflow for containerized applications.
   - How do you tag a Docker image for pushing to a remote repository?
     We use docker tag command
     For example my image name is go-hello-multi-stage, my username in Docker Hub is molyashka11, then I should use this command:
     docker tag go-hello-multi-stage molyashka11/go-hello-multi-stage
   - What steps are involved in pushing an image to Docker Hub?

     At first we need to login our Docker Hub account (docker login). Then we should tag our image like in previous answer. After that we push this image (docker push

molyashka11/go-hello-multi-stage) to Docker Hub. To verify we can open Repositories section in web browser.