KAZAKH KBTU BRITISH
TECHNICAL
UNIVERSITY

Assignment №1

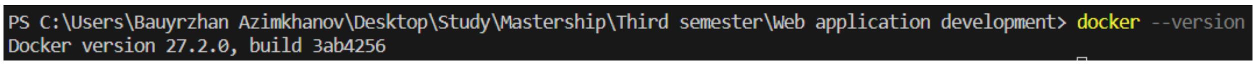
Prepared by Azimkhanov Bauyrzhan

Almaty, 2024

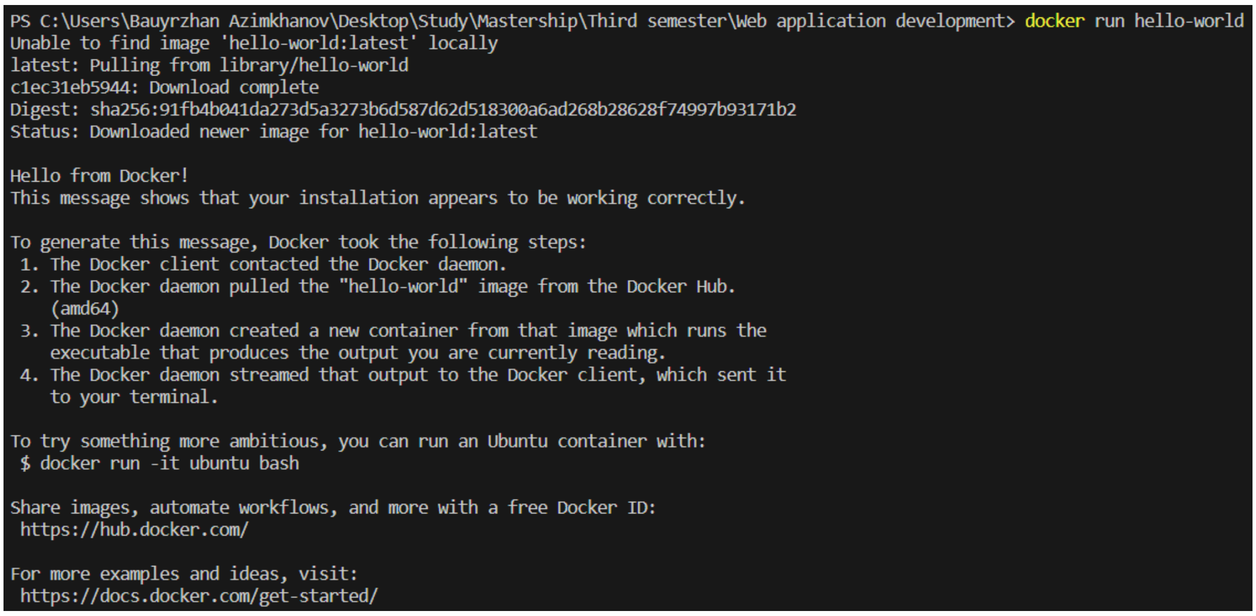
Part 1 - Installing Docker

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.
   - Run the command `docker run hello-world` to verify that Docker is set up correctly.
3. **Questions**:
   - What are the key components of Docker (e.g., Docker Engine, Docker CLI)?
   - How does Docker compare to traditional virtual machines?
   - What was the output of the `docker run hello-world` command, and what does it signify?



```
PS C:\Users\Bauyrzhan Azimkhanov\Desktop\Study\Mastership\Third semester\Web application development> docker --version
Docker version 27.2.0, build 3ab4256
```



```
PS C:\Users\Bauyrzhan Azimkhanov\Desktop\Study\Mastership\Third semester\Web application development> docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
c1ec31eb5944: Download 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/
```

1. Docker consists of several key components:
   - Docker Engine: This is the core part of Docker, which includes:
   - Docker Daemon (dockerd): Manages Docker objects like images, containers, networks, and volumes.
   - REST API: Allows programs to interact with the Docker daemon and instruct it on what to do.
   - Docker CLI (Command Line Interface): A client that allows users to interact with Docker using commands.
   - Docker Client: The primary way users interact with Docker. It sends commands to the Docker daemon.
   - Docker Registries: Locations where Docker images are stored. Docker Hub is the default public registry, but private registries can also be used.

- Docker Objects: These include images, containers, networks, and volumes.

2. Docker and traditional virtual machines (VMs) both provide isolated environments for running applications, but they do so in different ways:

   Virtual Machines:

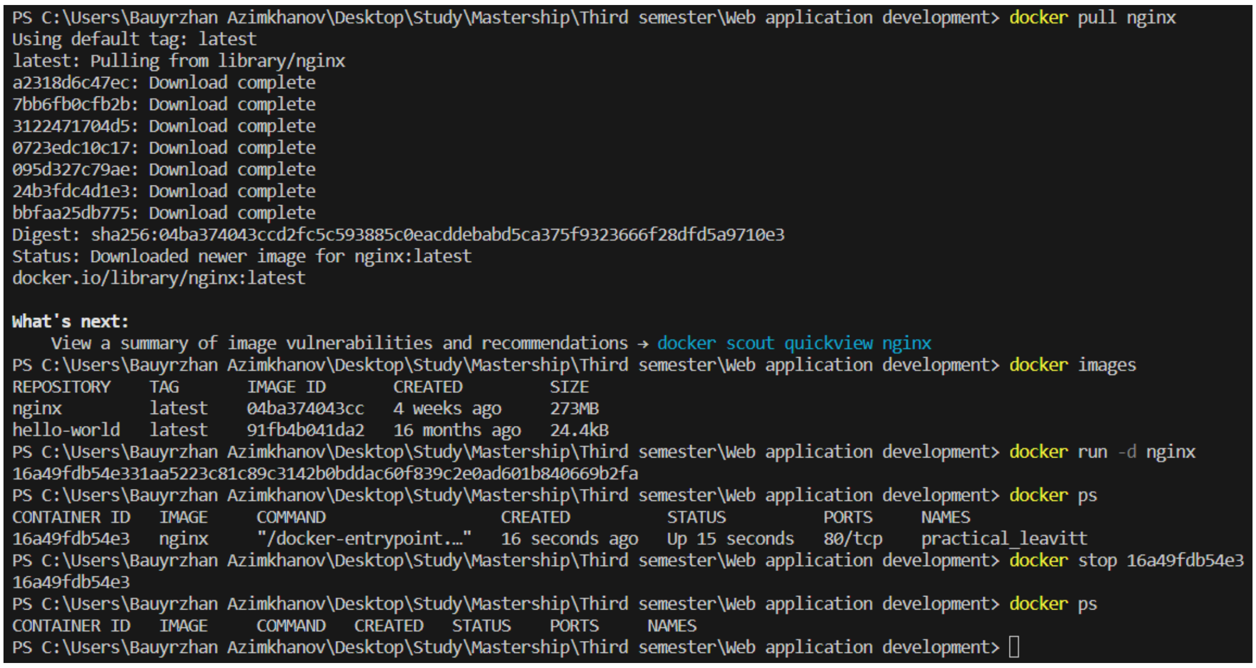
   - Each VM runs a full operating system, including its own kernel, on top of a hypervisor.
   - VMs are heavier and take longer to start because they need to boot an entire OS.
   - They provide strong isolation as each VM is completely separate from others.

   Docker Containers:

   - Containers share the host OS kernel and run as isolated processes in user space.
   - They are lightweight and start almost instantly because they don't need to boot an OS.
   - Containers are more efficient in terms of resource usage and can run more instances on the same hardware compared to VMs.
3. This signifies that Docker is installed correctly and is able to pull images and run containers.

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>`.
   - List all Docker images on your system using `docker images`.
   - Run a container from the pulled image using `docker run -d <image-name>`.
   - List all running containers using `docker ps` and stop a container using `docker stop <container-id>`.
3. **Questions**:
   - What is the difference between `docker pull` and `docker run`?
   - How do you find the details of a running container, such as its ID and status?
   - What happens to a container after it is stopped? Can it be restarted?



```
PS C:\Users\Bauyrzhan Azimkhanov\Desktop\Study\Mastership\Third semester\Web application development> docker pull nginx
Using default tag: latest
latest: Pulling from library/nginx
a2318d6c47ec: Download complete
7bb6fb0cfb2b: Download complete
3122471704d5: Download complete
0723edc10c17: Download complete
095d327c79ae: Download complete
24b3fdc4d1e3: Download complete
bbfaa25db775: Download 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
PS C:\Users\Bauyrzhan Azimkhanov\Desktop\Study\Mastership\Third semester\Web application development> docker images
REPOSITORY    TAG       IMAGE ID       CREATED          SIZE
nginx         latest    04ba374043cc   4 weeks ago      273MB
hello-world   latest    91fb4b041da2   16 months ago    24.4kB
PS C:\Users\Bauyrzhan Azimkhanov\Desktop\Study\Mastership\Third semester\Web application development> docker run -d nginx
16a49fdb54e331aa5223c81c89c3142b0bddac60f839c2e0ad601b840669b2fa
PS C:\Users\Bauyrzhan Azimkhanov\Desktop\Study\Mastership\Third semester\Web application development> docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS     NAMES
16a49fdb54e3   nginx     "/docker-entrypoint.…"   16 seconds ago   Up 15 seconds   80/tcp    practical_leavitt
PS C:\Users\Bauyrzhan Azimkhanov\Desktop\Study\Mastership\Third semester\Web application development> docker stop 16a49fdb54e3
16a49fdb54e3
PS C:\Users\Bauyrzhan Azimkhanov\Desktop\Study\Mastership\Third semester\Web application development> docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
PS C:\Users\Bauyrzhan Azimkhanov\Desktop\Study\Mastership\Third semester\Web application development> 
```

1. Difference between docker pull and docker run:

   - docker pull: Downloads a Docker image from a registry (like Docker Hub) to your local machine.
   - docker run: Creates and starts a container from a specified image. If the image is not already downloaded, it will pull the image first.

2. Finding details of a running container:

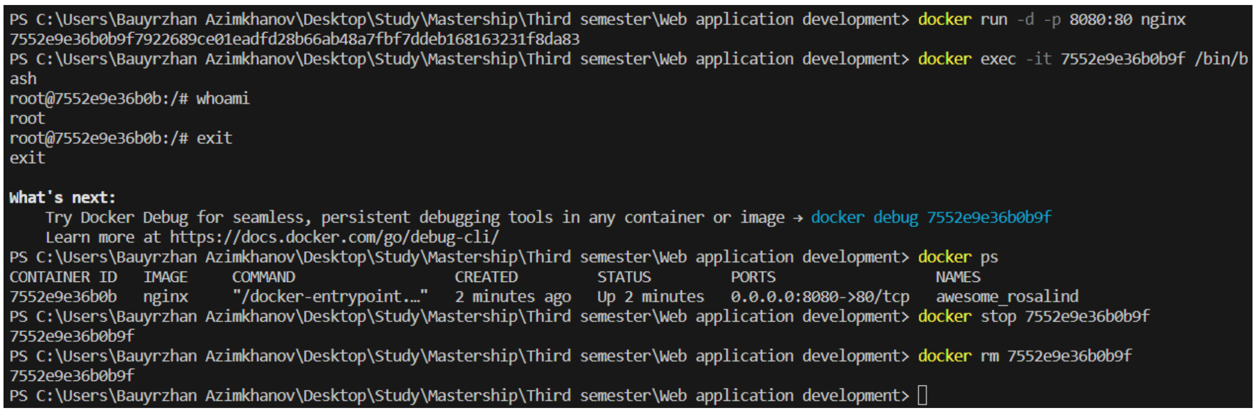
   - You can use the docker ps command to find details of running containers. This command lists container IDs, names, statuses, and other information.

3. What happens to a stopped container:

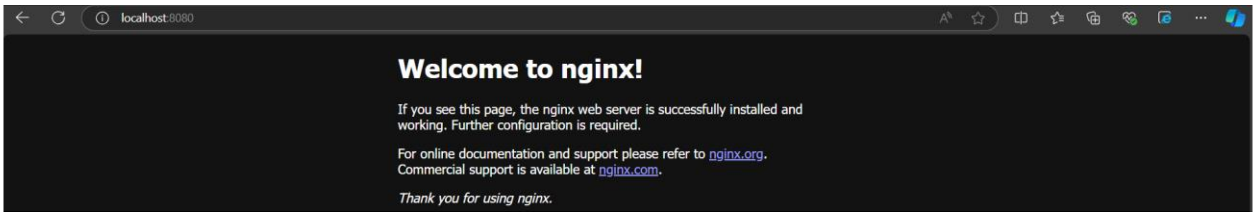
   - When a container is stopped, it remains on your system in a stopped state.

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`.
   - 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`.
   - Stop and remove the container using `docker stop <container-id>` and `docker rm <container-id>`.
3. **Questions**:
   - How does port mapping work in Docker, and why is it important?
   - What is the purpose of the `docker exec` command?
   - How do you ensure that a stopped container does not consume system resources?



```
PS C:\Users\Bauyrzhan Azimkhanov\Desktop\Study\Mastership\Third semester\Web application development> docker run -d -p 8080:80 nginx
7552e9e36b0b9f7922689ce01eadfd28b66ab48a7fbf7ddeb168163231f8da83
PS C:\Users\Bauyrzhan Azimkhanov\Desktop\Study\Mastership\Third semester\Web application development> docker exec -it 7552e9e36b0b9f /bin/b
ash
root@7552e9e36b0b:/# whoami
root
root@7552e9e36b0b:/# exit
exit

What's next:
    Try Docker Debug for seamless, persistent debugging tools in any container or image → docker debug 7552e9e36b0b9f
    Learn more at https://docs.docker.com/go/debug-cli/
PS C:\Users\Bauyrzhan Azimkhanov\Desktop\Study\Mastership\Third semester\Web application development> docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS         PORTS                  NAMES
7552e9e36b0b   nginx     "/docker-entrypoint.…"   2 minutes ago   Up 2 minutes   0.0.0.0:8080->80/tcp   awesome_rosalind
PS C:\Users\Bauyrzhan Azimkhanov\Desktop\Study\Mastership\Third semester\Web application development> docker stop 7552e9e36b0b9f
7552e9e36b0b9f
PS C:\Users\Bauyrzhan Azimkhanov\Desktop\Study\Mastership\Third semester\Web application development> docker rm 7552e9e36b0b9f
7552e9e36b0b9f
PS C:\Users\Bauyrzhan Azimkhanov\Desktop\Study\Mastership\Third semester\Web application development>
```



localhost:8080

Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

1. How port mapping works in Docker:

    - Port mapping in Docker allows you to connect the ports of a Docker container to ports on the host machine. This is important because it enables external access to the services running inside the container. For example, using -p 8080:80 in the docker run command maps port 80 in the container to port 8080 on the host.

2. Purpose of the docker exec command:

    - The docker exec command is used to run commands inside an already running container. This is useful for tasks like debugging, running additional processes, or interacting with the container's environment.

3. Ensuring a stopped container does not consume system resources:

    - To ensure a stopped container does not consume system resources, you can remove it using the docker rm <container_id> command. This deletes the container and frees up the resources it was using.

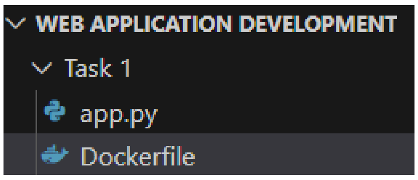
Part 2 - Dockerfile

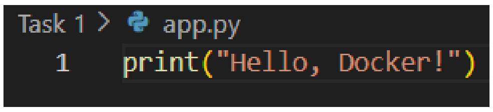
Exercise 1: Creating a Simple Dockerfile

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 .`.
    - Run the container using `docker run hello-docker`.

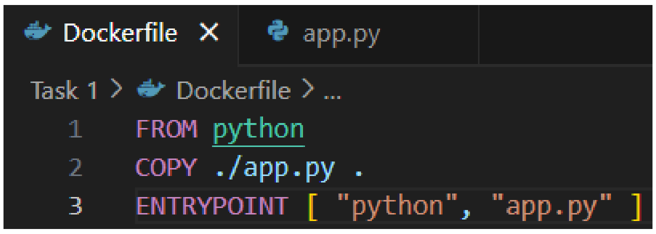
3. **Questions**:
   - What is the purpose of the `FROM` instruction in a Dockerfile?
   - How does the `COPY` instruction work in Dockerfile?

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

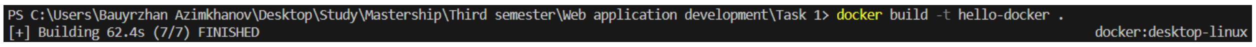




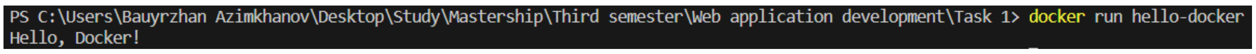
```
print("Hello, Docker!")
```



```
FROM python
COPY ./app.py .
ENTRYPOINT [ "python", "app.py" ]
```



```
PS C:\Users\Bauyrzhan Azimkhanov\Desktop\Study\Mastership\Third semester\Web application development\Task 1> docker build -t hello-docker .
[+] Building 62.4s (7/7) FINISHED                                                                                     docker:desktop-linux
```



```
PS C:\Users\Bauyrzhan Azimkhanov\Desktop\Study\Mastership\Third semester\Web application development\Task 1> docker run hello-docker
Hello, Docker!
```

1. Purpose of the FROM instruction:

   - The FROM instruction specifies the base image for building a new Docker image. It must be the first instruction in a Dockerfile.

2. How the COPY instruction works:

   - The COPY instruction copies files or directories from the host machine into the Docker image at a specified path.

3. Difference between CMD and ENTRYPOINT:

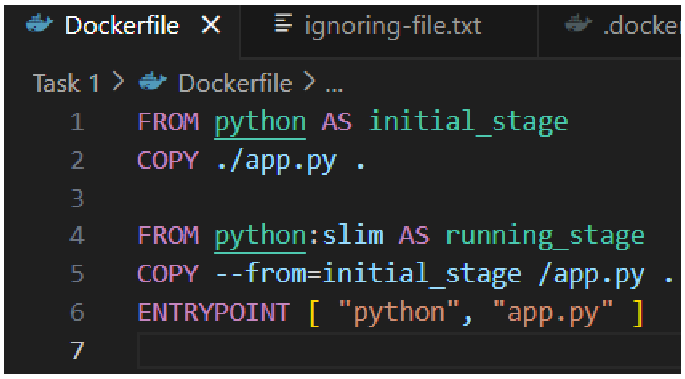
   - CMD: Sets default commands and/or parameters for a container, which can be overridden when starting the container.
   - ENTRYPOINT: Defines a fixed command that will always run when the container starts. It is not easily overridden.

Exercise 2: Optimizing Dockerfile with Layers and Caching

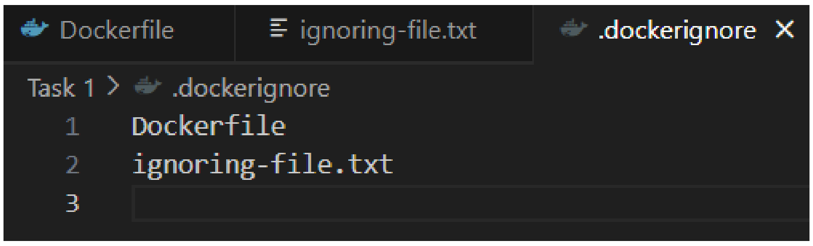
1. **Objective**: Learn how to optimize a Dockerfile for smaller image sizes and faster builds.
2. **Steps**:
   - Modify the Dockerfile created in the previous exercise to:
     - Separate the installation of Python dependencies (if any) from the copying of application code.
     - Use a `.dockerignore` file to exclude unnecessary files from the image.
   - Rebuild the Docker image and observe the build process to understand how caching works.
   - Compare the size of the optimized image with the original.

3. **Questions**:
   - What are Docker layers, and how do they affect image size and build times?
   - How does Docker's build cache work, and how can it speed up the build process?

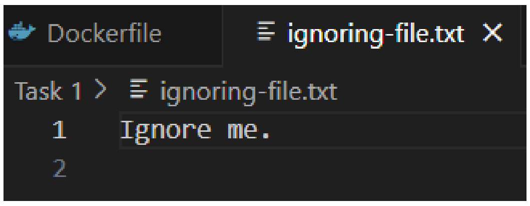
What is the role of the `.dockerignore` file?



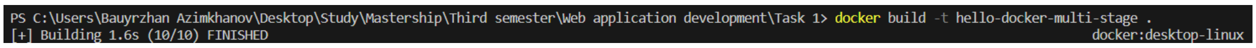
```
Task 1 > Dockerfile > ...
FROM python AS initial_stage
COPY ./app.py .

FROM python:slim AS running_stage
COPY --from=initial_stage /app.py .
ENTRYPOINT [ "python", "app.py" ]
```



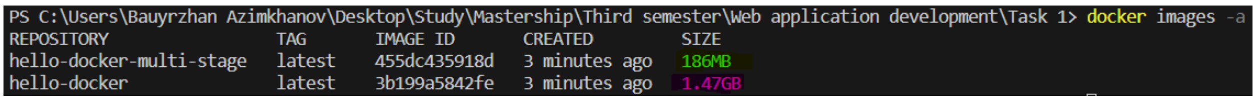
```
Task 1 > .dockerignore
Dockerfile
ignoring-file.txt
```



```
Task 1 > ignoring-file.txt
Ignore me.
```



```
PS C:\Users\Bauyrzhan Azimkhanov\Desktop\Study\Mastership\Third semester\Web application development\Task 1> docker build -t hello-docker-multi-stage .
[+] Building 1.6s (10/10) FINISHED                                                                                              docker:desktop-linux
```



```
PS C:\Users\Bauyrzhan Azimkhanov\Desktop\Study\Mastership\Third semester\Web application development\Task 1> docker images -a
REPOSITORY                TAG       IMAGE ID       CREATED         SIZE
hello-docker-multi-stage  latest    455dc435918d   3 minutes ago   186MB
hello-docker              latest    3b199a5842fe   3 minutes ago   1.47GB
```

1. Docker layers:

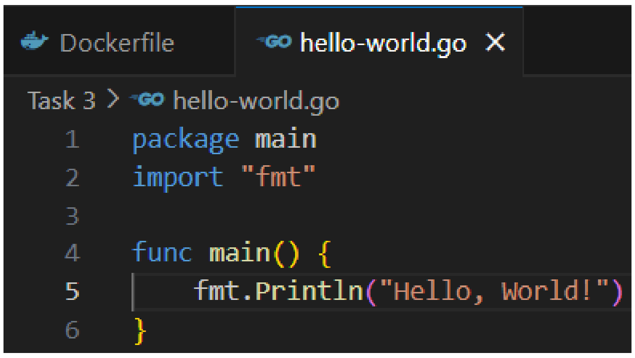
- Docker images are made up of multiple layers, each representing a set of filesystem changes. Layers are created by instructions in the Dockerfile (e.g., RUN, COPY). Fewer layers generally result in smaller image sizes and faster builds.

2. Docker’s build cache:

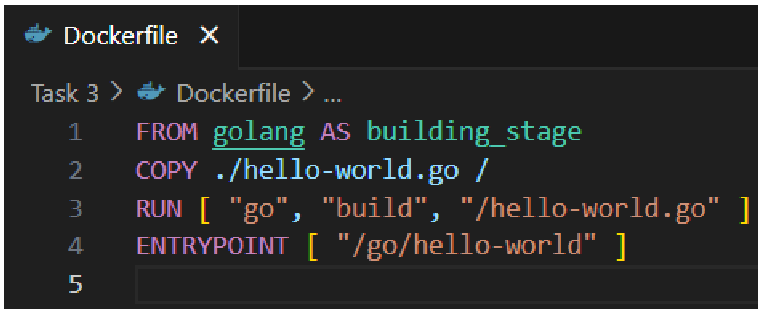
- Docker’s build cache stores intermediate layers created during the build process. If a layer hasn’t changed, Docker reuses it from the cache, speeding up subsequent builds by avoiding redundant steps.

Exercise 3: Multi-Stage Builds

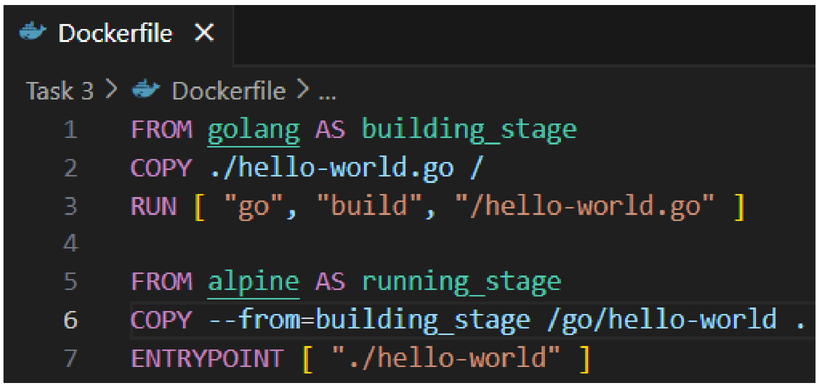
1. **Objective**: Use multi-stage builds to create leaner Docker images.
2. **Steps**:
   - Create a new project that involves compiling a simple Go application (e.g., a "Hello, World!" program).
   - Write a Dockerfile that uses multi-stage builds:
     - The first stage should use a Golang image to 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.
3. **Questions**:
   - What are the benefits of using multi-stage builds in Docker?
   - How can multi-stage builds help reduce the size of Docker images?
   - What are some scenarios where multi-stage builds are particularly useful?



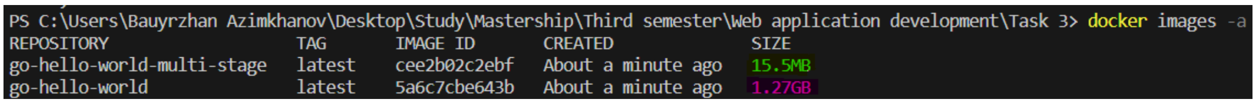
```go
package main
import "fmt"

func main() {
    fmt.Println("Hello, World!")
}
```



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

```



```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" ]
```

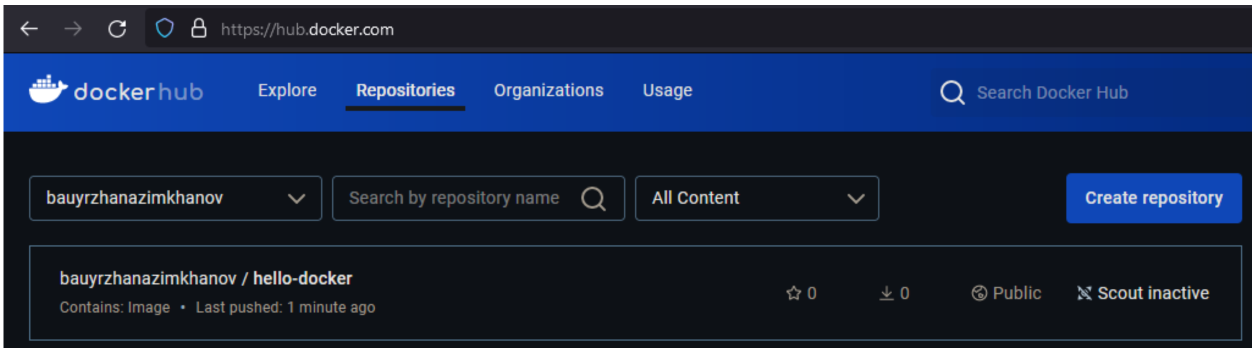


```
PS C:\Users\Bauyrzhan Azimkhanov\Desktop\Study\Mastership\Third semester\Web application development\Task 3> docker images -a
REPOSITORY                TAG       IMAGE ID       CREATED              SIZE
go-hello-world-multi-stage   latest    cee2b02c2ebf   About a minute ago   15.5MB
go-hello-world            latest    5a6c7cbe643b   About a minute ago   1.27GB
```

1. Benefits of using multi-stage builds:
   - Multi-stage builds allow you to use multiple FROM statements in a Dockerfile to create intermediate images. This helps in separating build dependencies from runtime dependencies, leading to cleaner and more efficient builds.
2. Reducing Docker image size:
   - By using multi-stage builds, you can copy only the necessary artifacts from the intermediate stages to the final image, excluding unnecessary build tools and dependencies. This significantly reduces the final image size.
3. Useful scenarios for multi-stage builds:
   - You need to compile code (e.g., Go, Java) and only want to include the compiled binaries in the final image.
   - You want to run tests in one stage and only proceed to the next stage if tests pass.
   - You need to build complex applications with multiple dependencies.

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`.
   - 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?
   - How do you tag a Docker image for pushing to a remote repository?
   - What steps are involved in pushing an image to Docker Hub?



bauyrzhanazimkhanov / hello-docker
Contains: Image • Last pushed: 1 minute ago
☆ 0 ⤓ 0 Public Scout inactive

1. Purpose of Docker hub:

- Docker Hub is a cloud-based repository where you can find, share, and store Docker images. It serves as a central hub for container images, making it easier for developers to distribute and collaborate on their containerized applications.

2. Tagging a Docker image:

- To tag a Docker image for pushing to a remote repository, use the docker tag command:

```
docker tag <image_id> <repository_name>:<tag>
```

For example:

```
docker tag my-image my-repo/my-image:latest
```

3. Steps to push an image to Docker hub:

- Log in to Docker hub:
  ```
  docker login
  ```
- Tag the image (if not already tagged):
  ```
  docker tag my-image my-repo/my-image:latest
  ```
- Push the image:
  ```
  docker push my-repo/my-image:latest
  ```