Sarsengaliyev Zhaisan, ID: 23MD0430

Assignment 1, Web Application Development

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

```
(base) zajsan@MacBook-Air-Zhaisan-2 ~ % docker --version
Docker version 26.0.0, build 2ae903e86c
(base) zajsan@MacBook-Air-Zhaisan-2 ~ %
```

```
(base) zajsan@MacBook-Air-Zhaisan-2 ~ % docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (arm64v8)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

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

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

3. **Questions**:
   - What are the key components of Docker (e.g., Docker Engine, Docker CLI)?
     Answer:
     - Docker Engine: Main component which is responsible for running and

managing containers, it has:
- Docker Daemon: Manages Docker objects (images, containers, networks and etc).
- Docker CLI: Tool which allows to work with Docker.
- Docker Compose: Tool for declaring and executing multi container Docker apps.

- How does Docker compare to traditional virtual machines?
  Answer:
  - Docker containers share host OS kernel, VMs need full OS for each instance, that's why Docker is lighter.
  - Docker does isolation at application level, VMs do isolation at hardware level.
  - Docker is easy to configure across different environments, VMs need more configurations.

- What was the output of the `docker run hello-world` command, and what does it signify?
  Answer:
  This command printed "Hello from Docker!", it shows that Docker pulls image and created container correctly.

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>`.

```
(base) zajsan@MacBook-Air-Zhaisan-2 ~ % 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
```

```
(base) zajsan@MacBook-Air-Zhaisan-2 ~ % docker images
REPOSITORY       TAG               IMAGE ID       CREATED         SIZE
nginx            latest            195245f0c792   5 weeks ago     193MB
scada-base-app   latest            b55736f1fbfc   3 months ago    118MB
redis            latest            fb60dcb2df98   5 months ago    139MB
postgres         12                51257c6d6416   7 months ago    440MB
postgres         15.6              b608c067b765   7 months ago    447MB
postgres         16.2-alpine3.19   9107b150d04f   7 months ago    249MB
redis            alpine            0bdbdda94acb   8 months ago    41.6MB
hello-world      latest            ee301c921b8a   16 months ago   9.14kB
(base) zajsan@MacBook-Air-Zhaisan-2 ~ % docker run -d nginx
0c7f71e7621ca4b501d006f8d931f77a195b3fe1ca338371cee94fbe689785f7
(base) zajsan@MacBook-Air-Zhaisan-2 ~ % docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED               STATUS               PORTS     NAMES
0c7f71e7621c   nginx     "/docker-entrypoint.…"   About a minute ago    Up About a minute    80/tcp    trusting_benz
(base) zajsan@MacBook-Air-Zhaisan-2 ~ % docker stop 0c7f71e7621c
0c7f71e7621c
(base) zajsan@MacBook-Air-Zhaisan-2 ~ % docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS   PORTS     NAMES
(base) zajsan@MacBook-Air-Zhaisan-2 ~ %
```

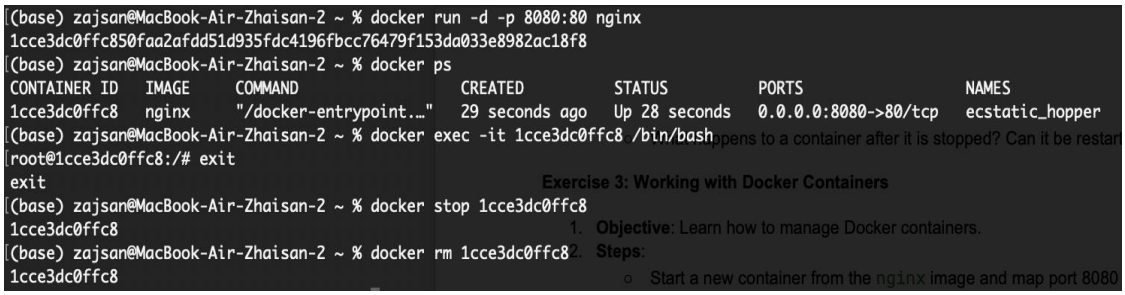
3. **Questions**:
   - What is the difference between `docker pull` and `docker run`?
     Answer:
     Pull downloads images from registry to local machine, Run creates and starts container from image.

   - How do you find the details of a running container, such as its ID and status?
     Answer:
     Command ‘docker ps’ shows the list of containers with id, name, status, image.

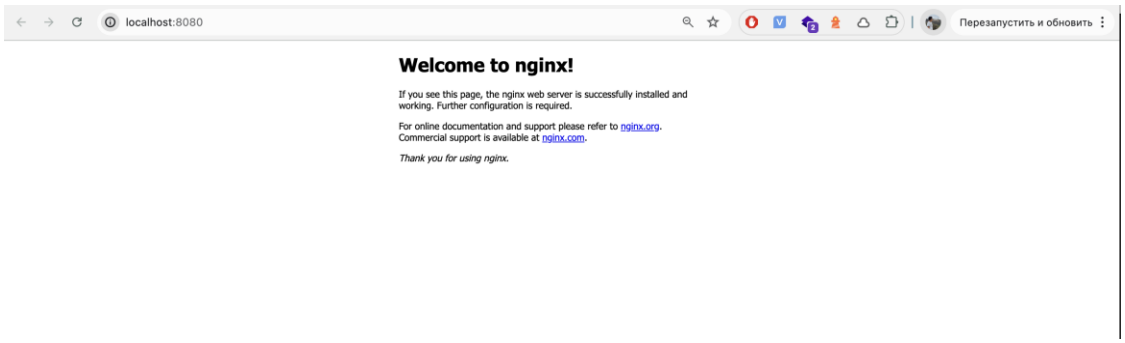
   - What happens to a container after it is stopped? Can it be restarted?
     Answer:
     It will be just stopped, it can be restarted just using command ‘docker start <cointainerID>’.

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>`.



```
(base) zajsan@MacBook-Air-Zhaisan-2 ~ % docker run -d -p 8080:80 nginx
1cce3dc0ffc850faa2afdd51d935fdc4196fbcc76479f153da033e8982ac18f8
(base) zajsan@MacBook-Air-Zhaisan-2 ~ % docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS                   NAMES
1cce3dc0ffc8   nginx     "/docker-entrypoint.…"   29 seconds ago   Up 28 seconds   0.0.0.0:8080->80/tcp    ecstatic_hopper
(base) zajsan@MacBook-Air-Zhaisan-2 ~ % docker exec -it 1cce3dc0ffc8 /bin/bash
root@1cce3dc0ffc8:/# exit
exit
(base) zajsan@MacBook-Air-Zhaisan-2 ~ % docker stop 1cce3dc0ffc8
1cce3dc0ffc8
(base) zajsan@MacBook-Air-Zhaisan-2 ~ % docker rm 1cce3dc0ffc8
1cce3dc0ffc8
```



3. **Questions**:
   - How does port mapping work in Docker, and why is it important?
     Answer:
     Port mapping connects container's internal port to host's port. We need it to make visible the service inside container to the outside. In our command 8080 is the host port, 80 is container port where the web server is running.

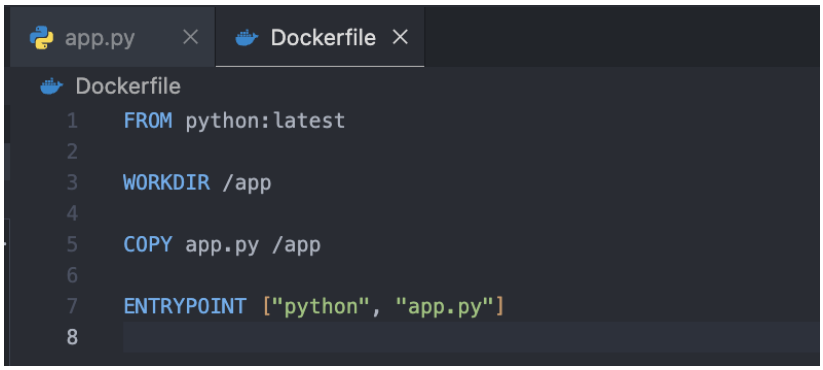
   - What is the purpose of the `docker exec` command?
     Answer:
     This command we use to run commands inside running container. For example, we can connect to database container like 'docker exec -it api-db sh'.

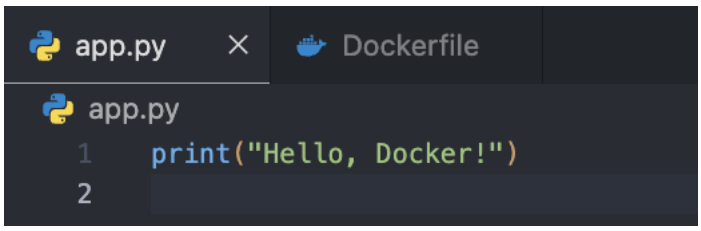
- How do you ensure that a stopped container does not consume system resources?
  Answer:
  To ensure that stopped container doesn't consume resources, we can just remove it.

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`.





```
(base) zajsan@MacBook-Air-Zhaisan-2 asg1 % docker build -t hello-docker .

[+] Building 379.8s (9/9) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 158B
 => [internal] load metadata for docker.io/library/python:latest
 => [auth] library/python:pull token for registry-1.docker.io
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [1/3] FROM docker.io/library/python:latest@sha256:7859853e7607927aa1d1b1a5a2f9e580ac90c2b66feeb1b77da97fed03b1ccbe
 => => resolve docker.io/library/python:latest@sha256:7859853e7607927aa1d1b1a5a2f9e580ac90c2b66feeb1b77da97fed03b1ccbe
 => => sha256:56c9b9253ff98351db158cb6789848656b8d54f411c0037347bf2358efb18f39 49.59MB / 49.59MB
 => => sha256:7859853e7607927aa1d1b1a5a2f9e580ac90c2b66feeb1b77da97fed03b1ccbe 9.72kB / 9.72kB
 => => sha256:dbb5b79c2668bb64927f07f3dfcb3d0c506f2ab8c300a9cf40718bda1c06600b 5.86kB / 5.86kB
 => => sha256:364d19f59f69474a80c53fc78da91f85553e16e8ba6a28063cbebf259821119e 23.59MB / 23.59MB
 => => sha256:843b1d8321825bc8302752ae003026f13bd15c6eef2efe032f3ca1520c5bbc07 64.00MB / 64.00MB
 => => sha256:9c1a7e7e41337f687fa7e5196aada121fd4897cb32f28139970933ed9a37cee1 2.33kB / 2.33kB
 => => sha256:a348c2a8d94613f34ce7d0ac4fd4e51800d8f4aa8a0bc9347fe5c8436b4c3bd5 202.65MB / 202.65MB
 => => extracting sha256:56c9b9253ff98351db158cb6789848656b8d54f411c0037347bf2358efb18f39
 => => sha256:353de11681b2d36db971d2402ebdeb137a2651bc3fcf5545329be637b6b5c6a4 6.24MB / 6.24MB
 => => extracting sha256:364d19f59f69474a80c53fc78da91f85553e16e8ba6a28063cbebf259821119e
 => => sha256:799b63efcab5e4f11215dd6a8caf59d117ae572e0589bc0c4ba40dcb61136f48 23.65MB / 23.65MB
 => => sha256:238a68c3d761a41b503e61aed92cb67f43c456139d427d47ef96dfa2caca87c4 251B / 251B
 => => extracting sha256:843b1d8321825bc8302752ae003026f13bd15c6eef2efe032f3ca1520c5bbc07
 => => extracting sha256:a348c2a8d94613f34ce7d0ac4fd4e51800d8f4aa8a0bc9347fe5c8436b4c3bd5
 => => extracting sha256:353de11681b2d36db971d2402ebdeb137a2651bc3fcf5545329be637b6b5c6a4
 => => extracting sha256:799b63efcab5e4f11215dd6a8caf59d117ae572e0589bc0c4ba40dcb61136f48
 => => extracting sha256:238a68c3d761a41b503e61aed92cb67f43c456139d427d47ef96dfa2caca87c4
 => [internal] load build context
 => => transferring context: 94B
 => [2/3] WORKDIR /app
 => [3/3] COPY app.py /app
 => exporting to image
 => => exporting layers
 => => writing image sha256:802109f5c1b17862142ddea78aa1ba1eefda93e003a46381a52323eeb48ff1ad
 => => naming to docker.io/library/hello-docker

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/hhet2iyc3hq7ekaurlen43nbf
(base) zajsan@MacBook-Air-Zhaisan-2 asg1 % docker run hello-docker

Hello, Docker!
(base) zajsan@MacBook-Air-Zhaisan-2 asg1 %
```

3. **Questions**:
   - What is the purpose of the `FROM` instruction in a Dockerfile?
     Answer:
     Dockerfile starts from ‘FROM’ which indicates environment like FROM golang:1.22.2 on which my app will be executed.

   - How does the `COPY` instruction work in Dockerfile?
     Answer:
     COPY copies files or directories from my host machine to the Docker image.

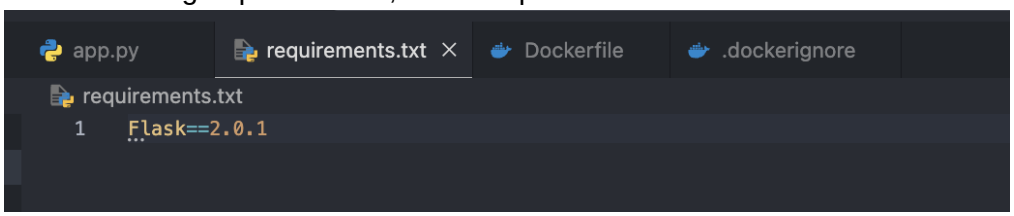
   - What is the difference between `CMD` and `ENTRYPOINT` in Dockerfile?
     Answer:
     CMD sets default command or argument for container, but we can override(change) it, for example if we set in default like ‘CMD ["echo", "Hello, World!"]’, we can change when run container to this ‘docker run <image> echo "Hi!"’.
     ENTRYPOINT sets fixed command which will be run always in the container.

Exercise 2: Optimizing Dockerfile with Layers and Caching

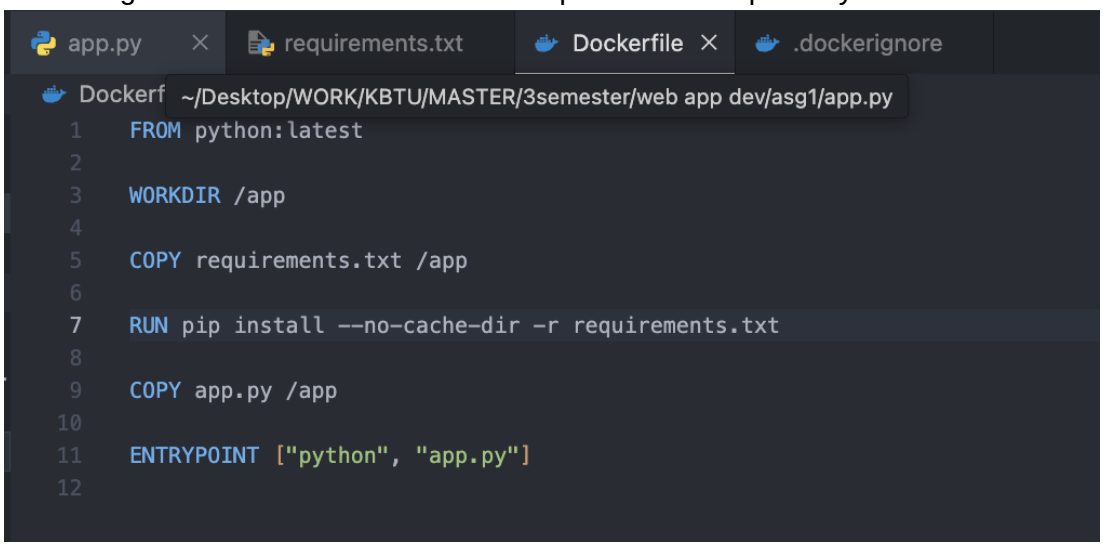
1. **Objective**: Learn how to optimize a Dockerfile for smaller image sizes and faster builds.

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.

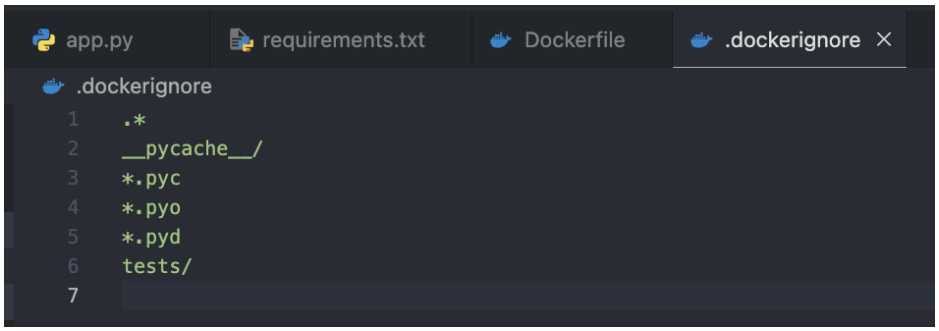
This for storing dependencies, for example Flask

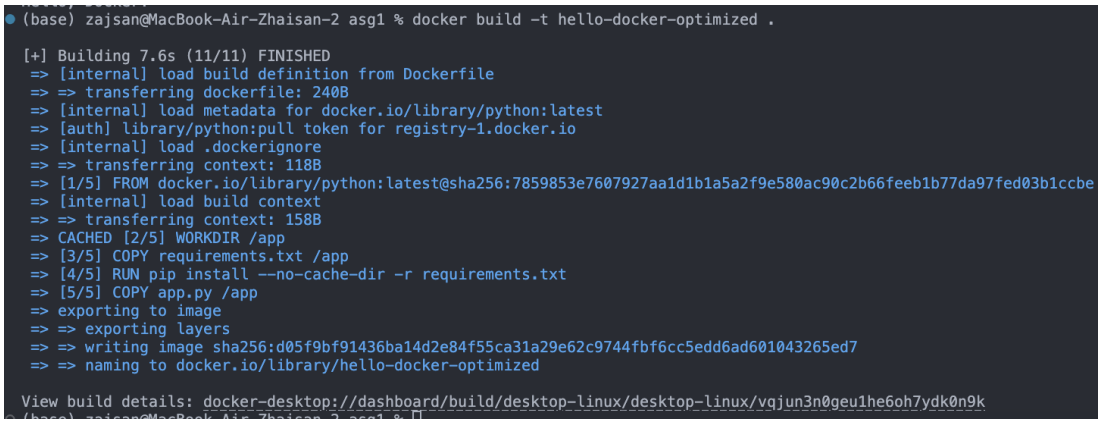


We changed Dockerfile so that to install dependencies separately



So here we added files which should not be included in the image.
* means all hidden files which starts from (.) as .git, gitignore which are not needed in container.
__pycache__/ is auto generated folder by python for storing compiled versions of files, these are also not needed in container.
*.pyc, *.pyo, *.pyd these are also auto generated files that are not needed in container.
Tests/ is folder which includes tests, they are also not needed in container.





3. **Questions**:
   - What are Docker layers, and how do they affect image size and build times?
     Answer:
     Docker builds images in layers. If some layer changes, Docker rebuilds that layer, but it can make image bigger and the build time can be increased.

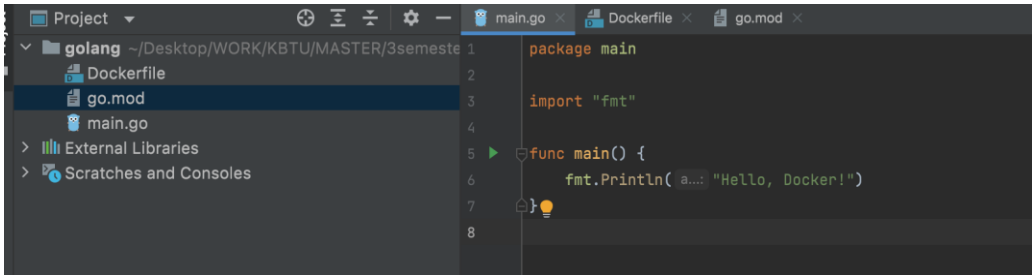
   - How does Docker's build cache work, and how can it speed up the build process?
     Answer:
     Docker stores layers in cache, if the layer did not change, docker just reuses it. And the next build will be fast.

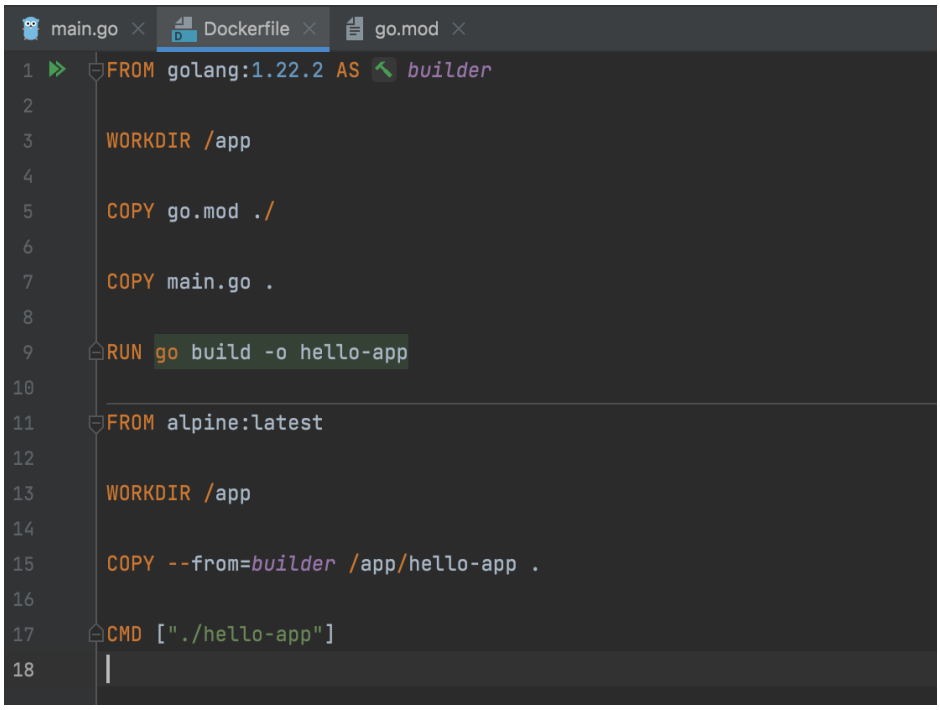
   - What is the role of the `.dockerignore` file?
     Answer:
     This file just say to Docker which files need to be ignored during building image. It useful not to include some unnecessary files to image.

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.





```
(base) zajsan@MacBook-Air-Zhaisan-2 golang % docker build -t go-docker-multi-stage .
[+] Building 3.4s (14/14) FINISHED                                                                                  docker:desktop-linux
 => [internal] load build definition from Dockerfile                                                                                0.0s
 => => transferring dockerfile: 235B                                                                                                0.0s
 => [internal] load metadata for docker.io/library/alpine:latest                                                                    1.3s
 => [internal] load metadata for docker.io/library/golang:1.22.2                                                                    1.2s
 => [internal] load .dockerignore                                                                                                   0.0s
 => => transferring context: 2B                                                                                                     0.0s
 => [builder 1/5] FROM docker.io/library/golang:1.22.2@sha256:d5302d40dc5fbbf38ec472d1848a9d2391a13f93293a6a5b0b87c99dc0eaa6ae      0.0s
 => [stage-1 1/3] FROM docker.io/library/alpine:latest@sha256:beefdbd8a1da6d2915566fde36db9db0b524eb737fc57cd1367effd16dc0d06d       0.0s
 => [internal] load build context                                                                                                   0.0s
 => => transferring context: 87B                                                                                                    0.0s
 => CACHED [builder 2/5] WORKDIR /app                                                                                               0.0s
 => CACHED [stage-1 2/3] WORKDIR /app                                                                                               0.0s
 => [builder 3/5] COPY go.mod ./                                                                                                    0.0s
 => [builder 4/5] COPY main.go .                                                                                                    0.0s
 => [builder 5/5] RUN go build -o hello-app                                                                                         2.0s
 => [stage-1 3/3] COPY --from=builder /app/hello-app .                                                                              0.0s
 => exporting to image                                                                                                              0.0s
 => => exporting layers                                                                                                             0.0s
 => => writing image sha256:9e0ffe4a285937a0449548122 6b64452b63189c5c00311831b68705fcda24b60                                       0.0s
 => => naming to docker.io/library/go-docker-multi-stage                                                                            0.0s

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/fz42b4ey0vu8r2nsajswqbtf0
```

```
(base) zajsan@MacBook-Air-Zhaisan-2 golang % docker run go-docker-multi-stage
Hello, Docker!
(base) zajsan@MacBook-Air-Zhaisan-2 golang % docker images
REPOSITORY                TAG               IMAGE ID       CREATED          SIZE
go-docker-multi-stage     latest            9e0ffe4a2859   21 seconds ago   10.8MB
hello-docker-optimized    latest            d05f9bf91436   9 hours ago      1.03GB
hello-docker              latest            802109f5c1b1   31 hours ago     1.02GB
nginx                     latest            195245f0c792   5 weeks ago      193MB
scada-base-app            latest            b55736f1fbfc   3 months ago     118MB
redis                     latest            fb60dcb2df98   5 months ago     139MB
postgres                  12                51257c6d6416   7 months ago     440MB
postgres                  15.6              b608c067b765   7 months ago     447MB
postgres                  16.2-alpine3.19   9107b150d04f   7 months ago     249MB
redis                     alpine            0bdbdda94acb   8 months ago     41.6MB
hello-world               latest            ee301c921b8a   16 months ago    9.14kB
```

3. **Questions**:
   - What are the benefits of using multi-stage builds in Docker?
     Answer:
     Multi stage builds allows to split the building to stages, this makes the image smaller, efficient, because in final image will not be included some heavy tools like compilers and etc.

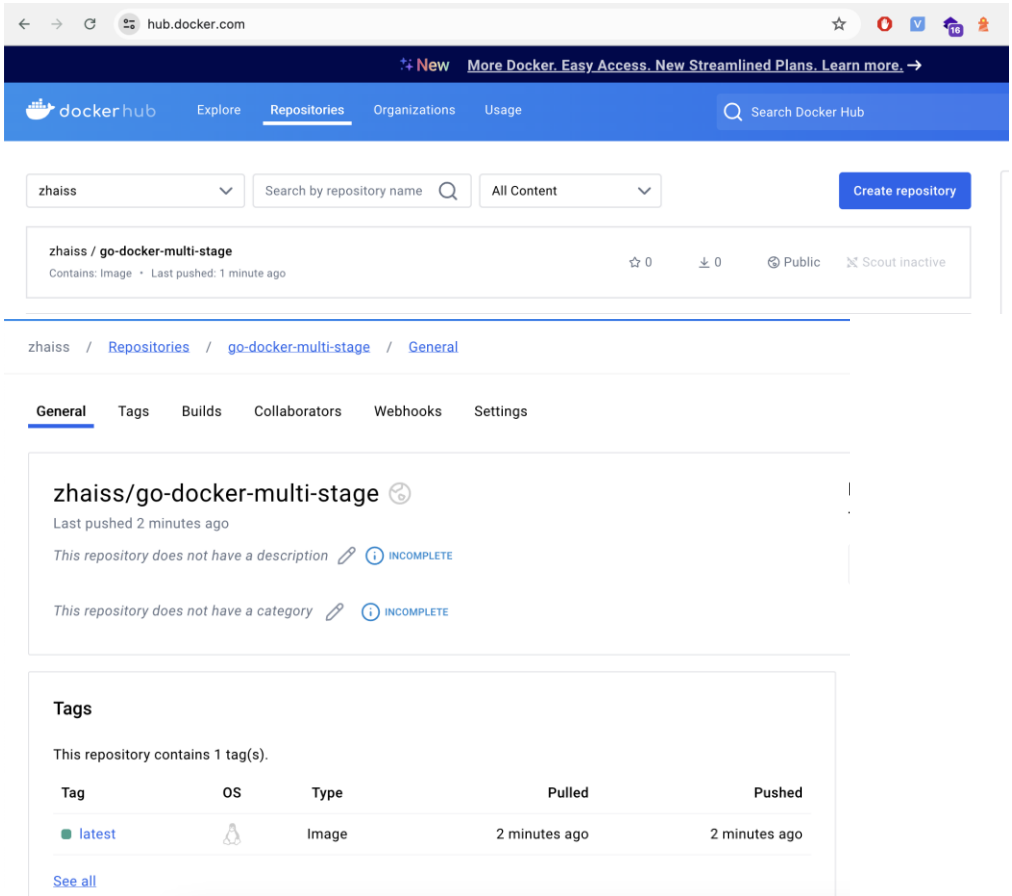
   - How can multi-stage builds help reduce the size of Docker images?
     Answer:
     So it can reduce the size when one stage builds the app, second stage copies the final output. We don't need to add some extra libraries to image.

   - What are some scenarios where multi-stage builds are particularly useful?
     Answer:
     - It is useful when we need to be image small and run fast.
     - It is useful when our app has complex build but we need to be final image lightweight.

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.

```
(base) zajsan@MacBook-Air-Zhaisan-2 golang % docker tag go-docker-multi-stage zhaiss/go-docker-multi-stage
(base) zajsan@MacBook-Air-Zhaisan-2 golang % docker push zhaiss/go-docker-multi-stage
Using default tag: latest
The push refers to repository [docker.io/zhaiss/go-docker-multi-stage]
cca98d20b76b: Pushed
c87b8bd56dc3: Pushed
16113d51b718: Mounted from library/alpine
latest: digest: sha256:aa753207ce03e6ca3bd55960896c0be0a68c573a14e1808624cc57c9f85782e2 size: 945
```



3. **Questions**:

- What is the purpose of Docker Hub in containerization?
  Answer:
  Docker hub is repository where we can store, share Docker images. We can pull images from public, private registries.

- How do you tag a Docker image for pushing to a remote repository?
  Answer:
  ‘docker tag <image-id> <username>/<repository-name>:<tag>’

- What steps are involved in pushing an image to Docker Hub?
  Answer:
  - docker login
  - tag image
  - push image