Web Development
Report for Assignment-1

Software Engineering, 2nd year
Nurzhan Momynkul
ID: 23MD0414

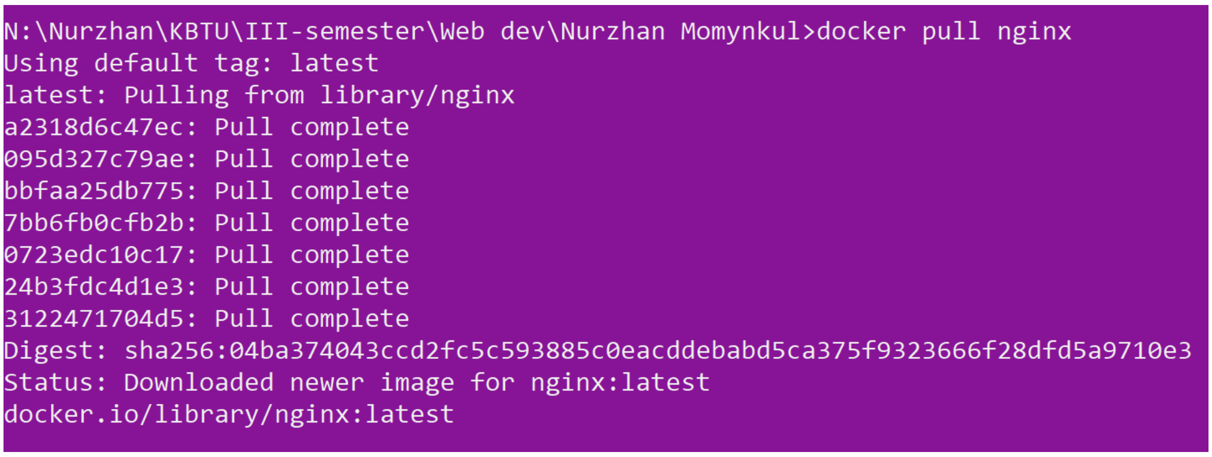
Exercise 1: Installing Docker

```
N:\Nurzhan\KBTU\III-semester\Web dev\Nurzhan Momynkul>docker --version
Docker version 27.1.1, build 6312585
```

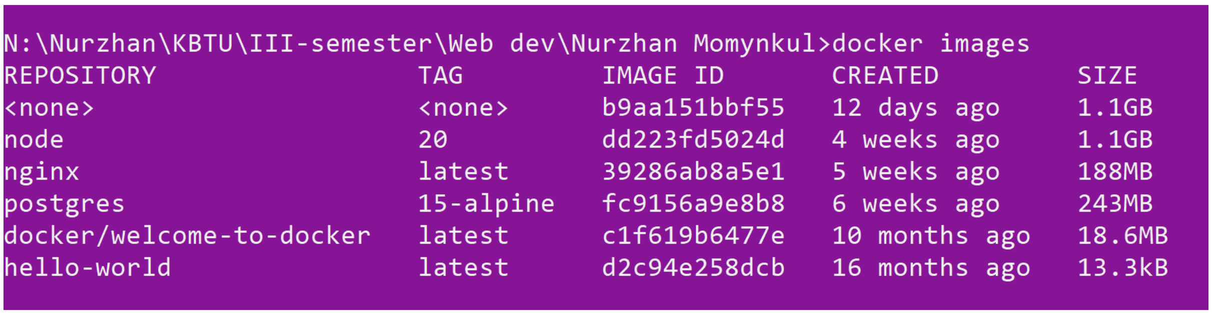
```
N:\Nurzhan\KBTU\III-semester\Web dev\Nurzhan Momynkul>docker run hello-world

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.
    (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/
```

- What are the key components of Docker (e.g., Docker Engine, Docker CLI)?
    - Docker Desktop, Docker Engine, Docker Images, Dockerfile, DockerHub, Docker Volumes, Docker Compose, Docker Containers
- How does Docker compare to traditional virtual machines?
    - With the help of Docker we can run, deploy and scale services on any machine and ensure that our application runs constantly. Conversely a virtual machine is a digital copy of our computer. We can run multiple virtual machines with their own operating systems in the same host directory. Usually, virtual machines are used to create the service's environment.
- What was the output of the `docker run hello-world` command, and what does it signify?
    - The output of this command was a Hello message from Docker itself. It then explains how the message was generated. It signifies that the Docker client contacted the Docker daemon, which then pulled the "hello-world" image from Docker Hub. After that, the daemon created a new container from that image, which provided the output. Finally, the daemon streamed the output to the client.

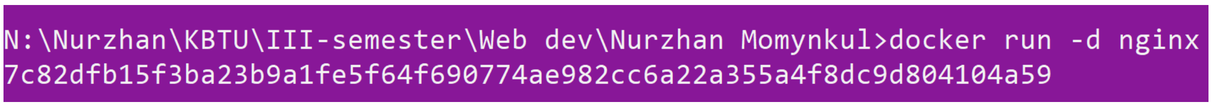
Exercise 2: Basic Docker Commands\



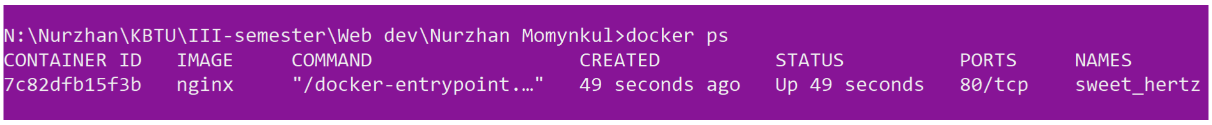
```
N:\Nurzhan\KBTU\III-semester\Web dev\Nurzhan Momynkul>docker pull nginx
Using default tag: latest
latest: Pulling from library/nginx
a2318d6c47ec: Pull complete
095d327c79ae: Pull complete
bbfaa25db775: Pull complete
7bb6fb0cfb2b: Pull complete
0723edc10c17: Pull complete
24b3fdc4d1e3: Pull complete
3122471704d5: Pull complete
Digest: sha256:04ba374043ccd2fc5c593885c0eacddebabd5ca375f9323666f28dfd5a9710e3
Status: Downloaded newer image for nginx:latest
docker.io/library/nginx:latest
```



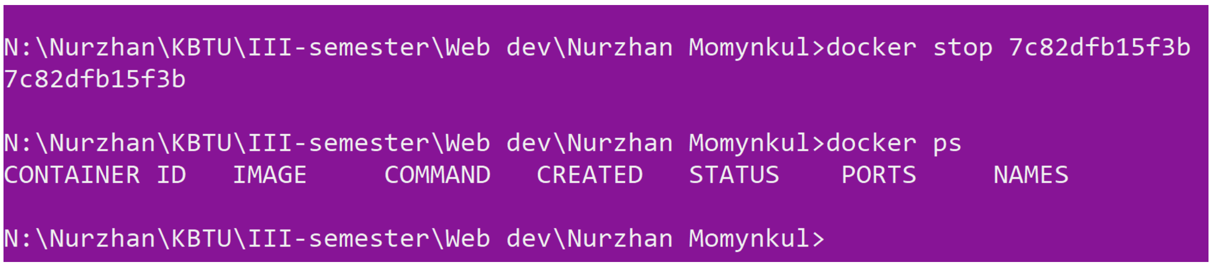
```
N:\Nurzhan\KBTU\III-semester\Web dev\Nurzhan Momynkul>docker images
REPOSITORY                 TAG         IMAGE ID       CREATED         SIZE
<none>                     <none>      b9aa151bbf55   12 days ago     1.1GB
node                       20          dd223fd5024d   4 weeks ago     1.1GB
nginx                      latest      39286ab8a5e1   5 weeks ago     188MB
postgres                   15-alpine   fc9156a9e8b8   6 weeks ago     243MB
docker/welcome-to-docker   latest      c1f619b6477e   10 months ago   18.6MB
hello-world                latest      d2c94e258dcb   16 months ago   13.3kB
```



```
N:\Nurzhan\KBTU\III-semester\Web dev\Nurzhan Momynkul>docker run -d nginx
7c82dfb15f3ba23b9a1fe5f64f690774ae982cc6a22a355a4f8dc9d804104a59
```



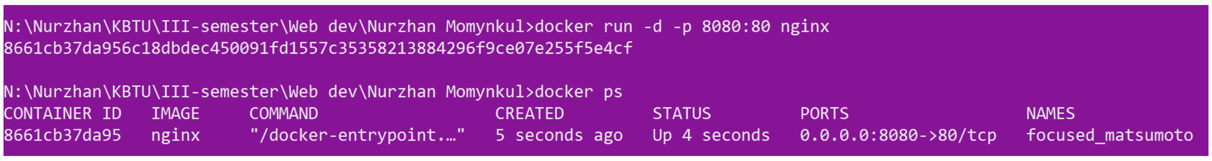
```
N:\Nurzhan\KBTU\III-semester\Web dev\Nurzhan Momynkul>docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS     NAMES
7c82dfb15f3b   nginx     "/docker-entrypoint.…"   49 seconds ago   Up 49 seconds   80/tcp    sweet_hertz
```



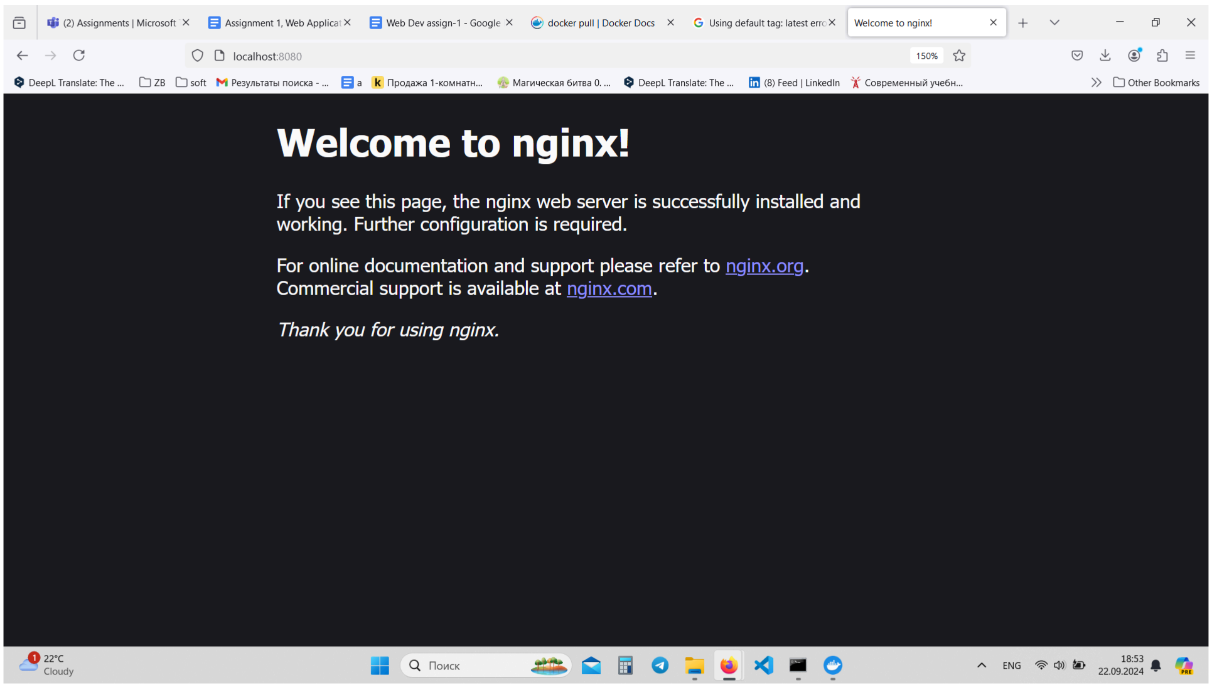
```
N:\Nurzhan\KBTU\III-semester\Web dev\Nurzhan Momynkul>docker stop 7c82dfb15f3b
7c82dfb15f3b

N:\Nurzhan\KBTU\III-semester\Web dev\Nurzhan Momynkul>docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES

N:\Nurzhan\KBTU\III-semester\Web dev\Nurzhan Momynkul>
```

1. What is the difference between `docker pull` and `docker run`?
   - The commands are quite self-explanatory. When we run docker pull, it simply pulls (or downloads) the image to our computer for later use. Meanwhile, docker run starts a container from the image. If the image does not exist locally, it will first pull the image and then create and run the container.
2. How do you find the details of a running container, such as its ID and status?
   - Docker ps
3. What happens to a container after it is stopped? Can it be restarted?
   - After a container is stopped, it remains exited but still exists in out system. And of course, it can be restarted.

Exercise 3: Working with Docker Containers



```
N:\Nurzhan\KBTU\III-semester\Web dev\Nurzhan Momynkul>docker run -d -p 8080:80 nginx
8661cb37da956c18dbdec450091fd1557c35358213884296f9ce07e255f5e4cf

N:\Nurzhan\KBTU\III-semester\Web dev\Nurzhan Momynkul>docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS         PORTS                  NAMES
8661cb37da95   nginx     "/docker-entrypoint.…"   5 seconds ago   Up 4 seconds   0.0.0.0:8080->80/tcp   focused_matsumoto
```

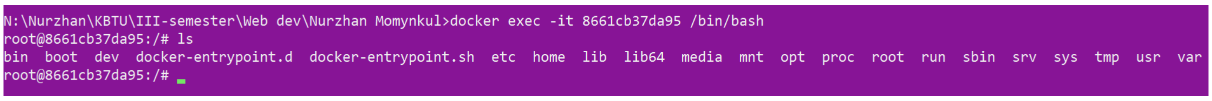


Welcome to nginx!

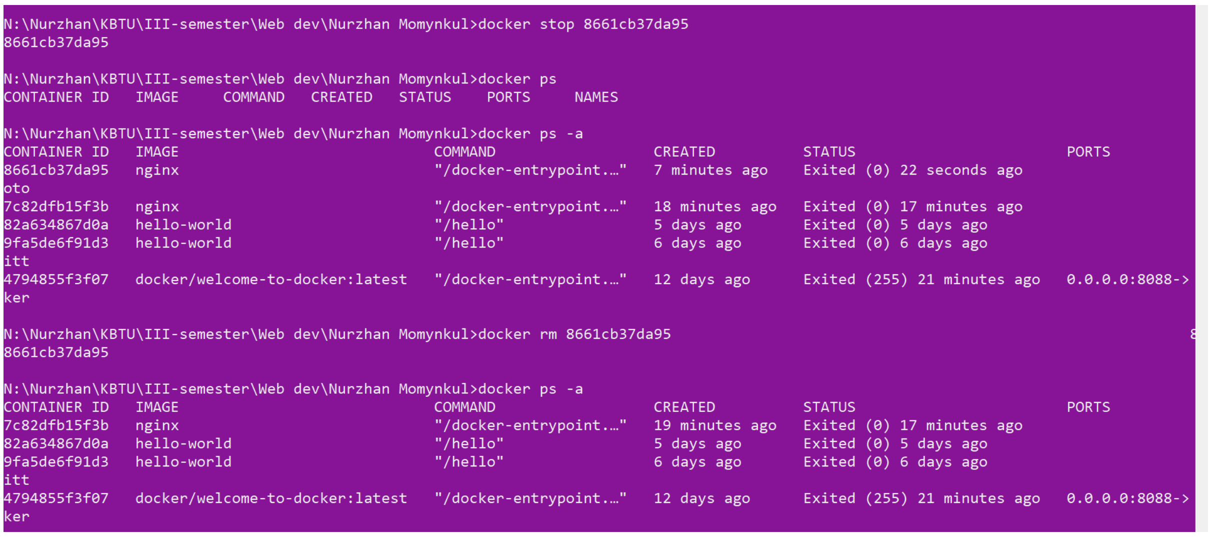
If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

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

Thank you for using nginx.



```
N:\Nurzhan\KBTU\III-semester\Web dev\Nurzhan Momynkul>docker exec -it 8661cb37da95 /bin/bash
root@8661cb37da95:/# ls
bin  boot  dev  docker-entrypoint.d  docker-entrypoint.sh  etc  home  lib  lib64  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
root@8661cb37da95:/#
```



```
N:\Nurzhan\KBTU\III-semester\Web dev\Nurzhan Momynkul>docker stop 8661cb37da95
8661cb37da95

N:\Nurzhan\KBTU\III-semester\Web dev\Nurzhan Momynkul>docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES

N:\Nurzhan\KBTU\III-semester\Web dev\Nurzhan Momynkul>docker ps -a
CONTAINER ID   IMAGE                              COMMAND                  CREATED          STATUS                        PORTS
8661cb37da95   nginx                              "/docker-entrypoint.…"   7 minutes ago    Exited (0) 22 seconds ago
oto
7c82dfb15f3b   nginx                              "/docker-entrypoint.…"   18 minutes ago   Exited (0) 17 minutes ago
82a634867d0a   hello-world                        "/hello"                 5 days ago       Exited (0) 5 days ago
9fa5de6f91d3   hello-world                        "/hello"                 6 days ago       Exited (0) 6 days ago
itt
4794855f3f07   docker/welcome-to-docker:latest    "/docker-entrypoint.…"   12 days ago      Exited (255) 21 minutes ago   0.0.0.0:8088->
ker

N:\Nurzhan\KBTU\III-semester\Web dev\Nurzhan Momynkul>docker rm 8661cb37da95                                                                     8
8661cb37da95

N:\Nurzhan\KBTU\III-semester\Web dev\Nurzhan Momynkul>docker ps -a
CONTAINER ID   IMAGE                              COMMAND                  CREATED          STATUS                        PORTS
7c82dfb15f3b   nginx                              "/docker-entrypoint.…"   19 minutes ago   Exited (0) 17 minutes ago
82a634867d0a   hello-world                        "/hello"                 5 days ago       Exited (0) 5 days ago
9fa5de6f91d3   hello-world                        "/hello"                 6 days ago       Exited (0) 6 days ago
itt
4794855f3f07   docker/welcome-to-docker:latest    "/docker-entrypoint.…"   12 days ago      Exited (255) 21 minutes ago   0.0.0.0:8088->
ker
```

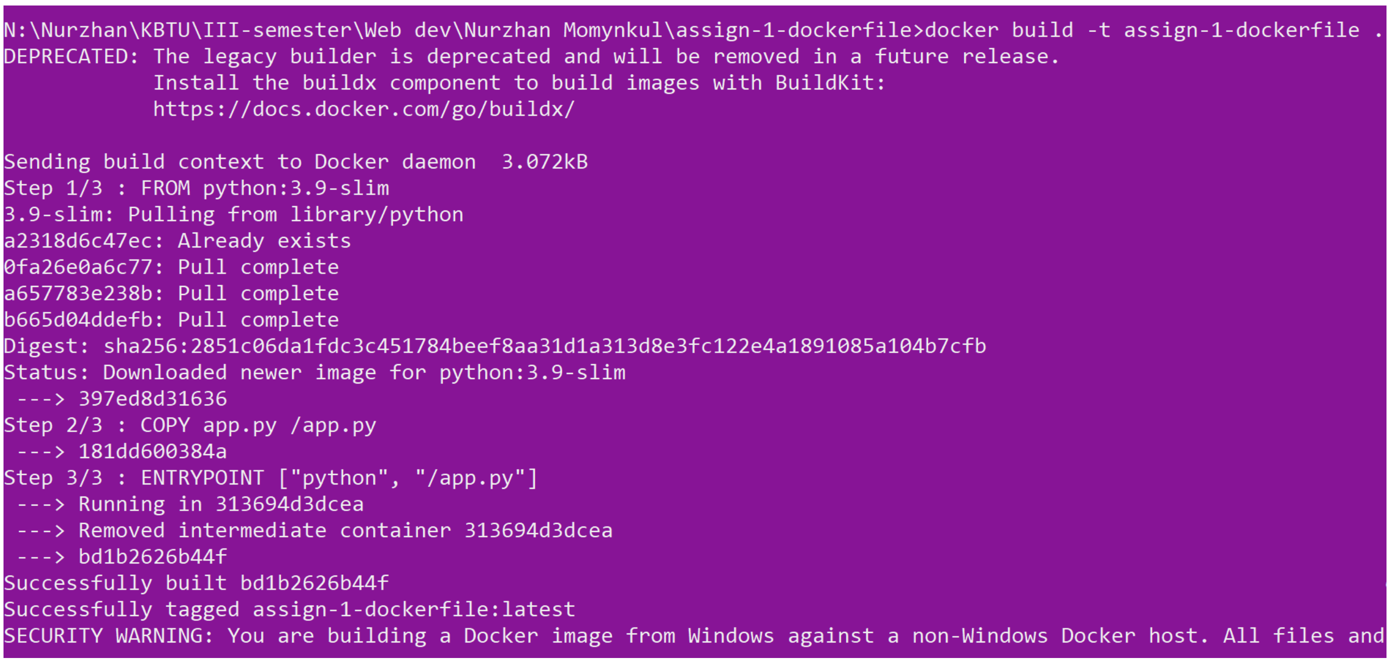
1. How does port mapping work in Docker, and why is it important?
   - In Docker port mapping works with the help of d`ocker run -d -p 8080:80 nginx` command. And after that , it makes accessible Docker container's internal port from the host machine. Port mapping is important, because containers run in isolated environments with their own networking.

Without port mapping, services running inside the container cannot be accessible from the host machine.

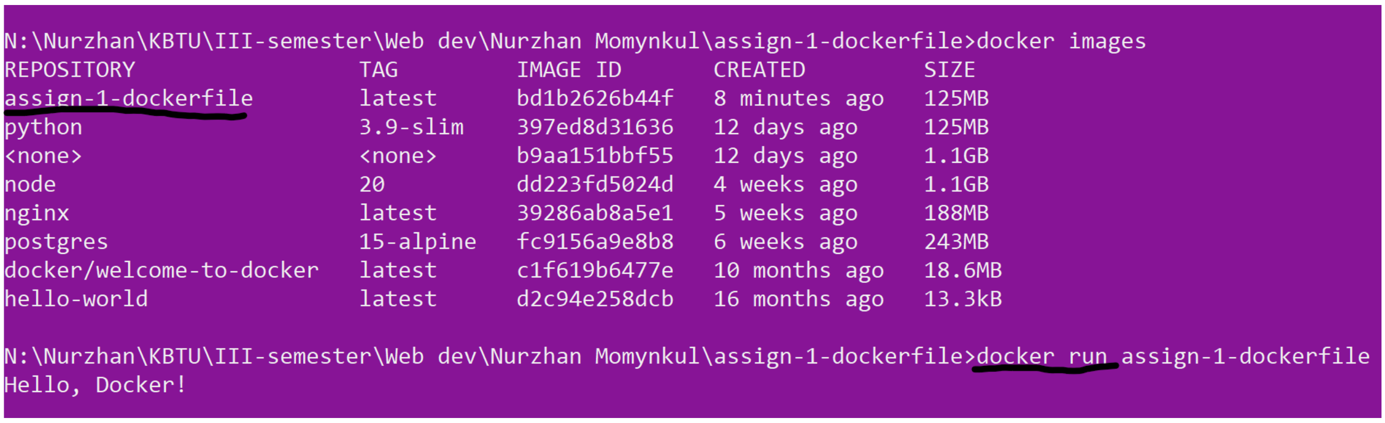
2. What is the purpose of the `docker exec` command?
    - `docker exec` is a command that can be used inside an already running (up) container. It allows us to work with the container's file system as we did. Also, we can execute commands, or start a command line session.
3. How do you ensure that a stopped container does not consume system resources?
    - When we stop a container, it will not take a CPU or memory resources, but it still takes some disk space. And to prevent this, we used `docker rm <container-id>` command.

Dockerfile

Exercise 1: Creating a Simple Dockerfile



```
N:\Nurzhan\KBTU\III-semester\Web dev\Nurzhan Momynkul\assign-1-dockerfile>docker build -t assign-1-dockerfile .
DEPRECATED: The legacy builder is deprecated and will be removed in a future release.
            Install the buildx component to build images with BuildKit:
            https://docs.docker.com/go/buildx/

Sending build context to Docker daemon  3.072kB
Step 1/3 : FROM python:3.9-slim
3.9-slim: Pulling from library/python
a2318d6c47ec: Already exists
0fa26e0a6c77: Pull complete
a657783e238b: Pull complete
b665d04ddefb: Pull complete
Digest: sha256:2851c06da1fdc3c451784beef8aa31d1a313d8e3fc122e4a1891085a104b7cfb
Status: Downloaded newer image for python:3.9-slim
 ---> 397ed8d31636
Step 2/3 : COPY app.py /app.py
 ---> 181dd600384a
Step 3/3 : ENTRYPOINT ["python", "/app.py"]
 ---> Running in 313694d3dcea
 ---> Removed intermediate container 313694d3dcea
 ---> bd1b2626b44f
Successfully built bd1b2626b44f
Successfully tagged assign-1-dockerfile:latest
SECURITY WARNING: You are building a Docker image from Windows against a non-Windows Docker host. All files and
```



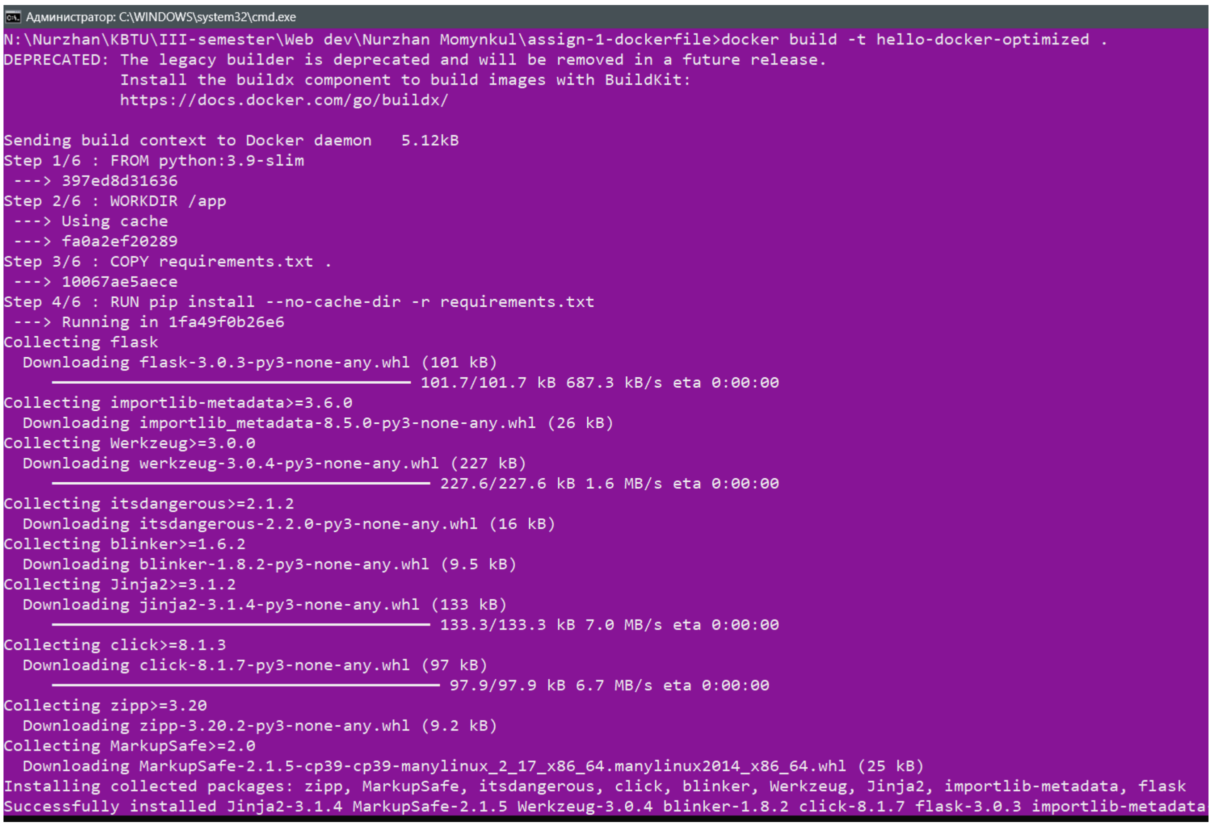
```
N:\Nurzhan\KBTU\III-semester\Web dev\Nurzhan Momynkul\assign-1-dockerfile>docker images
REPOSITORY                 TAG         IMAGE ID       CREATED          SIZE
assign-1-dockerfile        latest      bd1b2626b44f   8 minutes ago    125MB
python                     3.9-slim    397ed8d31636   12 days ago      125MB
<none>                     <none>      b9aa151bbf55   12 days ago      1.1GB
node                       20          dd223fd5024d   4 weeks ago      1.1GB
nginx                      latest      39286ab8a5e1   5 weeks ago      188MB
postgres                   15-alpine   fc9156a9e8b8   6 weeks ago      243MB
docker/welcome-to-docker   latest      c1f619b6477e   10 months ago    18.6MB
hello-world                latest      d2c94e258dcb   16 months ago    13.3kB

N:\Nurzhan\KBTU\III-semester\Web dev\Nurzhan Momynkul\assign-1-dockerfile>docker run assign-1-dockerfile
Hello, Docker!
```

1. What is the purpose of the `FROM` instruction in a Dockerfile?

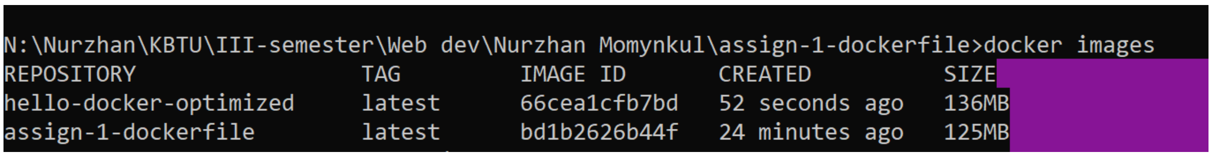
- FROM instruction specfiies the base image for our Docker image. It sets up the environment, software, operating system. Basically, FROM is the first instruction in every Dockerfile and it means the starting point for our building image.

2. How does the `COPY` instruction work in Dockerfile?
   - The COPY instruction is used to copy files and directories from our local filesystem into the Docker image's filesystem during the build process.
3. What is the difference between `CMD` and `ENTRYPOINT` in Dockerfile?
   - ENTRYPOINT is used when we want the container to always run a specific command, whereas CMD allows for more flexibility, acting as default parameters that can be overridden. We can combine both to specify the main command (ENTRYPOINT) and optional parameters (CMD).

Exercise 2: Optimizing Dockerfile with Layers and Caching



```
Администратор: C:\WINDOWS\system32\cmd.exe
N:\Nurzhan\KBTU\III-semester\Web dev\Nurzhan Momynkul\assign-1-dockerfile>docker build -t hello-docker-optimized .
DEPRECATED: The legacy builder is deprecated and will be removed in a future release.
            Install the buildx component to build images with BuildKit:
            https://docs.docker.com/go/buildx/

Sending build context to Docker daemon   5.12kB
Step 1/6 : FROM python:3.9-slim
 ---> 397ed8d31636
Step 2/6 : WORKDIR /app
 ---> Using cache
 ---> fa0a2ef20289
Step 3/6 : COPY requirements.txt .
 ---> 10067ae5aece
Step 4/6 : RUN pip install --no-cache-dir -r requirements.txt
 ---> Running in 1fa49f0b26e6
Collecting flask
  Downloading flask-3.0.3-py3-none-any.whl (101 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 101.7/101.7 kB 687.3 kB/s eta 0:00:00
Collecting importlib-metadata>=3.6.0
  Downloading importlib_metadata-8.5.0-py3-none-any.whl (26 kB)
Collecting Werkzeug>=3.0.0
  Downloading werkzeug-3.0.4-py3-none-any.whl (227 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 227.6/227.6 kB 1.6 MB/s eta 0:00:00
Collecting itsdangerous>=2.1.2
  Downloading itsdangerous-2.2.0-py3-none-any.whl (16 kB)
Collecting blinker>=1.6.2
  Downloading blinker-1.8.2-py3-none-any.whl (9.5 kB)
Collecting Jinja2>=3.1.2
  Downloading jinja2-3.1.4-py3-none-any.whl (133 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 133.3/133.3 kB 7.0 MB/s eta 0:00:00
Collecting click>=8.1.3
  Downloading click-8.1.7-py3-none-any.whl (97 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 97.9/97.9 kB 6.7 MB/s eta 0:00:00
Collecting zipp>=3.20
  Downloading zipp-3.20.2-py3-none-any.whl (9.2 kB)
Collecting MarkupSafe>=2.0
  Downloading MarkupSafe-2.1.5-cp39-cp39-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (25 kB)
Installing collected packages: zipp, MarkupSafe, itsdangerous, click, blinker, Werkzeug, Jinja2, importlib-metadata, flask
Successfully installed Jinja2-3.1.4 MarkupSafe-2.1.5 Werkzeug-3.0.4 blinker-1.8.2 click-8.1.7 flask-3.0.3 importlib-metadata
```



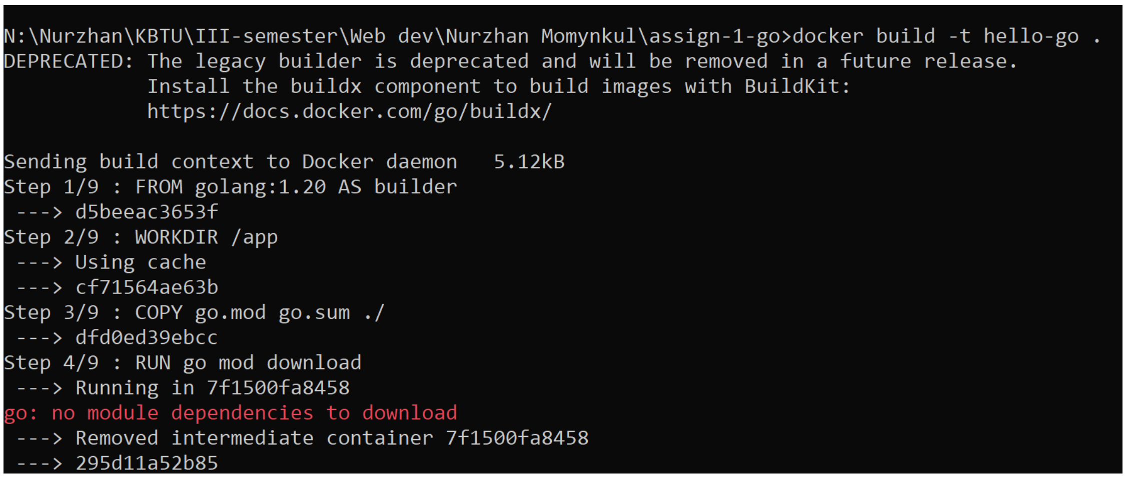
```
N:\Nurzhan\KBTU\III-semester\Web dev\Nurzhan Momynkul\assign-1-dockerfile>docker images
REPOSITORY               TAG       IMAGE ID       CREATED          SIZE
hello-docker-optimized   latest    66cea1cfb7bd   52 seconds ago   136MB
assign-1-dockerfile      latest    bd1b2626b44f   24 minutes ago   125MB
```

1. What are Docker layers, and how do they affect image size and build times?
   - Docker layers are the fundamental building blocks for creating, deploying, and scaling systems. Each layer contains only the differences from the previous

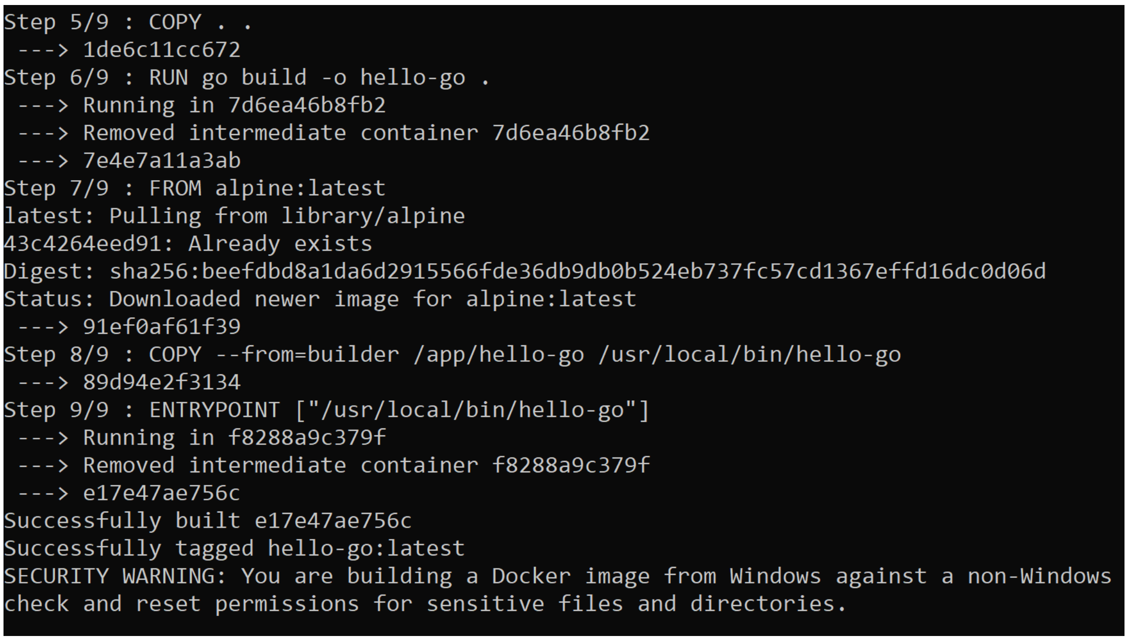
layer. If layers are optimized, the overall image size can be reduced. When we build an image, Docker checks the cache for existing layers. If a layer hasn't changed, Docker reuses it instead of rebuilding it.

2. How does Docker's build cache work, and how can it speed up the build process?
   - Docker caches each layer created during the build process. When we run a build command, Docker checks the instructions in the Dockerfile and compares them to existing layers in the cache. If it finds a match it re-uses the cached layer instead of executing the command again. Then simply, Docker can skip the installation steps and just rebuild the final layers, as a result it will have much faster builds.
3. What is the role of the `.dockerignore` file?
   - This file can ve used to specify files and directories that should be excluded from the build context sent to the Docker daemon. This helps to avoid copying unnecessary files into the image, reducing the build context size.

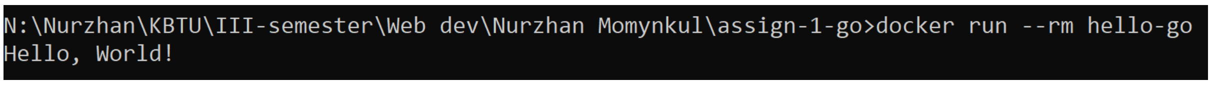
Exercise 3: Multi-Stage Builds



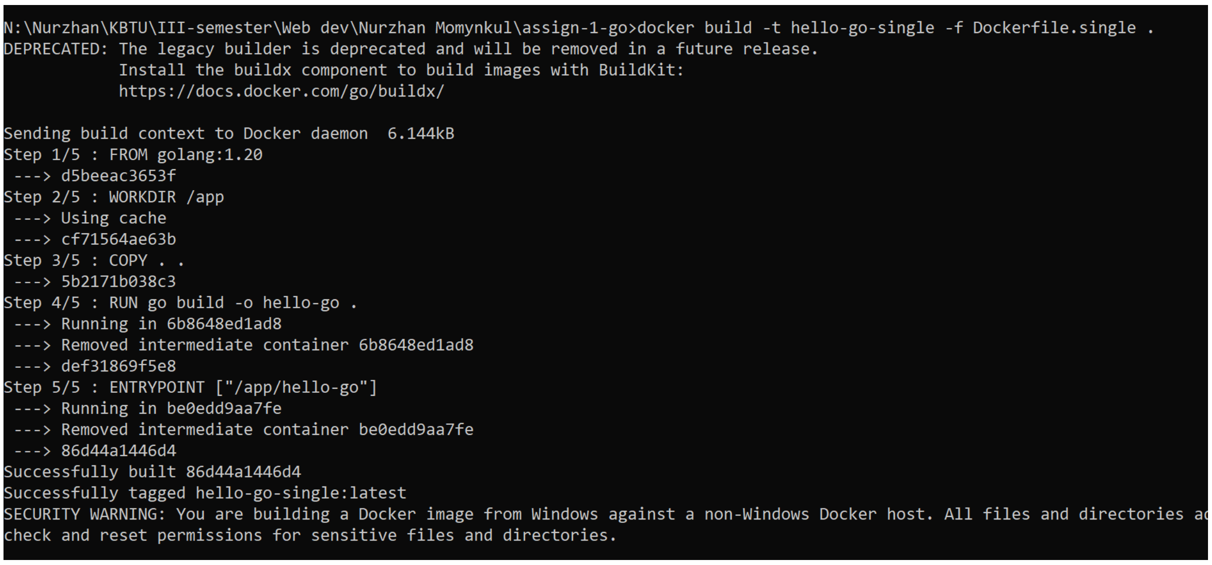
```
N:\Nurzhan\KBTU\III-semester\Web dev\Nurzhan Momynkul\assign-1-go>docker build -t hello-go .
DEPRECATED: The legacy builder is deprecated and will be removed in a future release.
            Install the buildx component to build images with BuildKit:
            https://docs.docker.com/go/buildx/

Sending build context to Docker daemon   5.12kB
Step 1/9 : FROM golang:1.20 AS builder
 ---> d5beeac3653f
Step 2/9 : WORKDIR /app
 ---> Using cache
 ---> cf71564ae63b
Step 3/9 : COPY go.mod go.sum ./
 ---> dfd0ed39ebcc
Step 4/9 : RUN go mod download
 ---> Running in 7f1500fa8458
go: no module dependencies to download
 ---> Removed intermediate container 7f1500fa8458
 ---> 295d11a52b85
```



```
Step 5/9 : COPY . .
 ---> 1de6c11cc672
Step 6/9 : RUN go build -o hello-go .
 ---> Running in 7d6ea46b8fb2
 ---> Removed intermediate container 7d6ea46b8fb2
 ---> 7e4e7a11a3ab
Step 7/9 : FROM alpine:latest
latest: Pulling from library/alpine
43c4264eed91: Already exists
Digest: sha256:beefdbd8a1da6d2915566fde36db9db0b524eb737fc57cd1367effd16dc0d06d
Status: Downloaded newer image for alpine:latest
 ---> 91ef0af61f39
Step 8/9 : COPY --from=builder /app/hello-go /usr/local/bin/hello-go
 ---> 89d94e2f3134
Step 9/9 : ENTRYPOINT ["/usr/local/bin/hello-go"]
 ---> Running in f8288a9c379f
 ---> Removed intermediate container f8288a9c379f
 ---> e17e47ae756c
Successfully built e17e47ae756c
Successfully tagged hello-go:latest
SECURITY WARNING: You are building a Docker image from Windows against a non-Windows
check and reset permissions for sensitive files and directories.
```

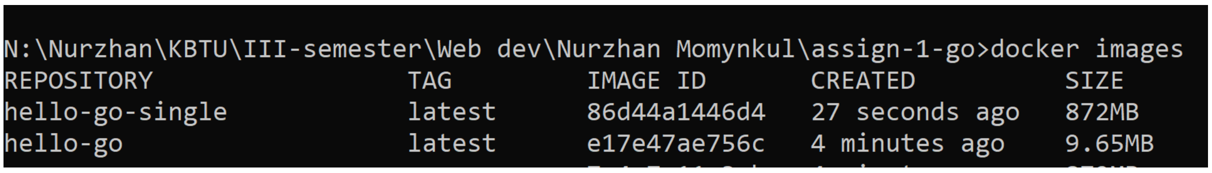


```
N:\Nurzhan\KBTU\III-semester\Web dev\Nurzhan Momynkul\assign-1-go>docker run --rm hello-go
Hello, World!
```



```
N:\Nurzhan\KBTU\III-semester\Web dev\Nurzhan Momynkul\assign-1-go>docker build -t hello-go-single -f Dockerfile.single .
DEPRECATED: The legacy builder is deprecated and will be removed in a future release.
            Install the buildx component to build images with BuildKit:
            https://docs.docker.com/go/buildx/

Sending build context to Docker daemon  6.144kB
Step 1/5 : FROM golang:1.20
 ---> d5beeac3653f
Step 2/5 : WORKDIR /app
 ---> Using cache
 ---> cf71564ae63b
Step 3/5 : COPY . .
 ---> 5b2171b038c3
Step 4/5 : RUN go build -o hello-go .
 ---> Running in 6b8648ed1ad8
 ---> Removed intermediate container 6b8648ed1ad8
 ---> def31869f5e8
Step 5/5 : ENTRYPOINT ["/app/hello-go"]
 ---> Running in be0edd9aa7fe
 ---> Removed intermediate container be0edd9aa7fe
 ---> 86d44a1446d4
Successfully built 86d44a1446d4
Successfully tagged hello-go-single:latest
SECURITY WARNING: You are building a Docker image from Windows against a non-Windows Docker host. All files and directories ac
check and reset permissions for sensitive files and directories.
```

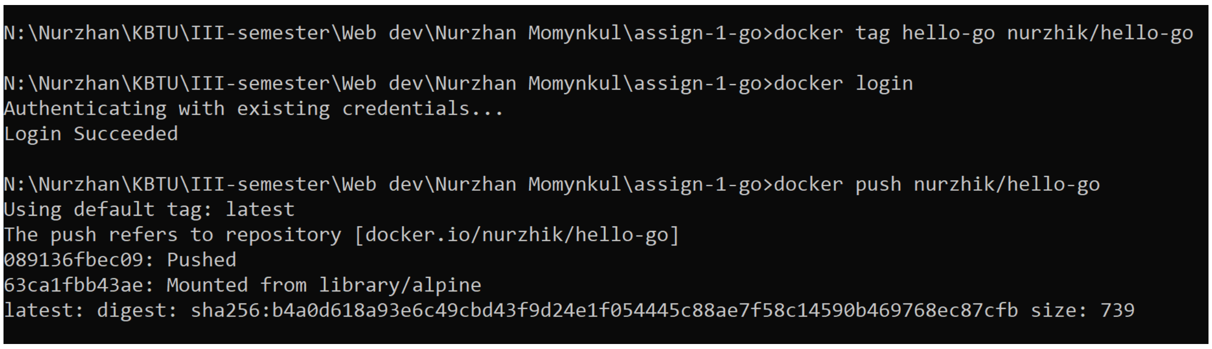
Sizes comparison:



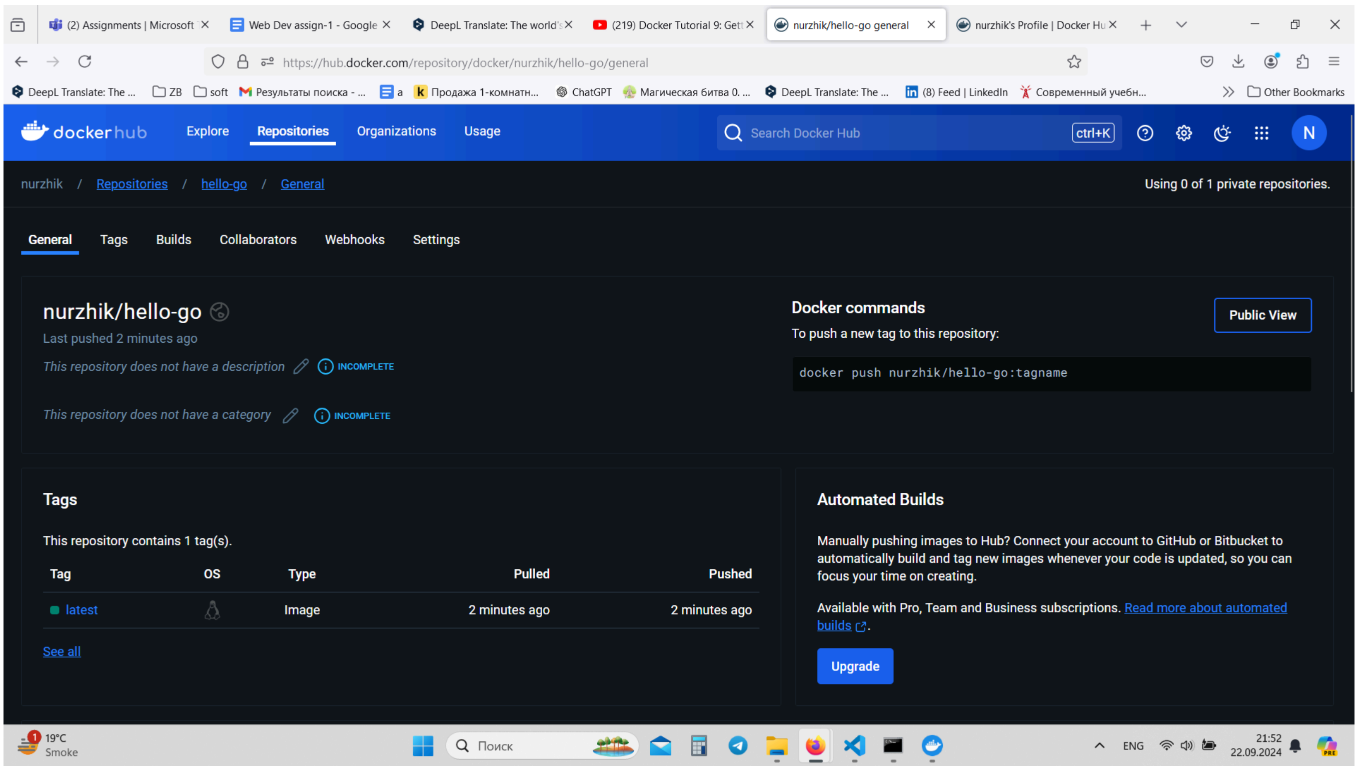
```
N:\Nurzhan\KBTU\III-semester\Web dev\Nurzhan Momynkul\assign-1-go>docker images
REPOSITORY                  TAG         IMAGE ID       CREATED           SIZE
hello-go-single             latest      86d44a1446d4   27 seconds ago    872MB
hello-go                    latest      e17e47ae756c   4 minutes ago     9.65MB
```

1. What are the benefits of using multi-stage builds in Docker?
    - By separating the build and runtime environments, multi-stage builds allow us to copy only the necessary files into the final image.
    - By excluding build tools and source code from the final image, we minimize the attack surface, reducing the risk of vulnerabilities.
    - Multi-stage builds help to organize Dockerfiles more effectively, making them easier to read and maintain.
    - The size
2. How can multi-stage builds help reduce the size of Docker images?
    - Each stage of a multi-stage build can have its own base image, allowing us to choose lightweight images for the final runtime environment. This approach avoids unnecessary dependencies that are only required during the build process.
3. What are some scenarios where multi-stage builds are particularly useful?
    - Frontend Applications: For frontend applications built with frameworks like React or Vue, we can build the static assets in one stage and serve them from a lightweight web server like Nginx in the final stage.
    - In microservices, where each service might have different build requirements, multi-stage builds provide a clean way to handle various environments and dependencies efficiently.

Exercise 4: Pushing Docker Images to Docker Hub



```
N:\Nurzhan\KBTU\III-semester\Web dev\Nurzhan Momynkul\assign-1-go>docker tag hello-go nurzhik/hello-go

N:\Nurzhan\KBTU\III-semester\Web dev\Nurzhan Momynkul\assign-1-go>docker login
Authenticating with existing credentials...
Login Succeeded

N:\Nurzhan\KBTU\III-semester\Web dev\Nurzhan Momynkul\assign-1-go>docker push nurzhik/hello-go
Using default tag: latest
The push refers to repository [docker.io/nurzhik/hello-go]
089136fbec09: Pushed
63ca1fbb43ae: Mounted from library/alpine
latest: digest: sha256:b4a0d618a93e6c49cbd43f9d24e1f054445c88ae7f58c14590b469768ec87cfb size: 739
```



https://hub.docker.com/repository/docker/nurzhik/hello-go/general

nurzhik / Repositories / hello-go / General

Using 0 of 1 private repositories.

General Tags Builds Collaborators Webhooks Settings

nurzhik/hello-go

Last pushed 2 minutes ago

This repository does not have a description INCOMPLETE

This repository does not have a category INCOMPLETE

Docker commands

To push a new tag to this repository:

```
docker push nurzhik/hello-go:tagname
```

Tags

This repository contains 1 tag(s).

Tag	OS	Type	Pulled	Pushed
latest		Image	2 minutes ago	2 minutes ago

See all

Automated Builds

Manually pushing images to Hub? Connect your account to GitHub or Bitbucket to automatically build and tag new images whenever your code is updated, so you can focus your time on creating.

Available with Pro, Team and Business subscriptions. Read more about automated builds.

1. What is the purpose of Docker Hub in containerization?
   - Docker Hub serves as a cloud-based repository where users can push/pull, share, and manage their Docker images. Purpose is to work together with your team and contribute to the repo (as a version control).
2. How do you tag a Docker image for pushing to a remote repository?
   - We used “docker tag” command to give a new tag to our image, which included my Docker Hub username and the aimed repository name.
3. What steps are involved in pushing an image to Docker Hub?
   - Login
   - Tag
   - Push