Assignment 1

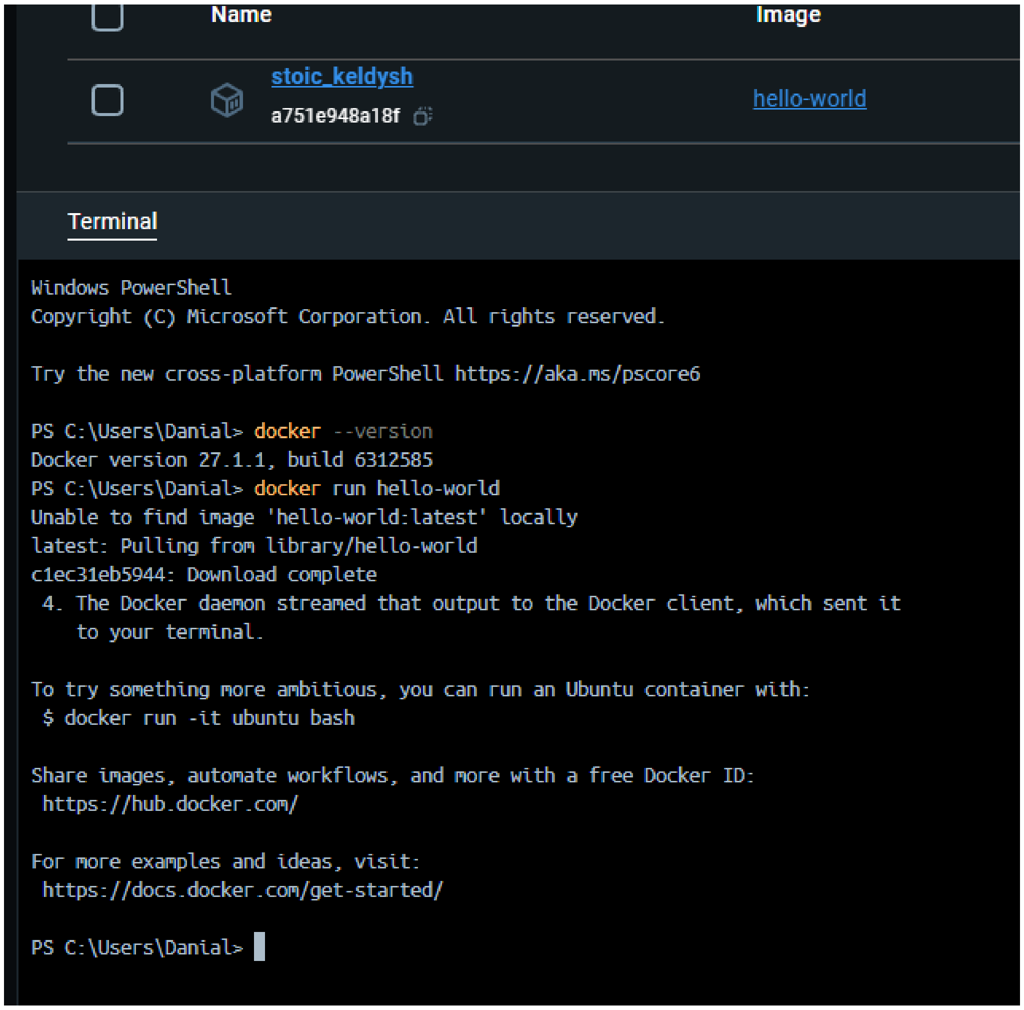
Web Application Development

Student: Danial Serekov

Intro to Containerization: Docker

Exercise 1: Installing Docker

Docker is installed, ***docker --version*** and ***docker run hello-world*** commands are run.



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\Danial> docker --version
Docker version 27.1.1, build 6312585
PS C:\Users\Danial> docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
c1ec31eb5944: Download complete
 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/

PS C:\Users\Danial> 
```

Questions and answers

1. What are the key components of Docker (e.g., Docker Engine, Docker CLI)?

Answer: Docker Engine – the core service that help create containers, and include several key components like:

- **Docker Daemon** – a service that manages Docker objects such as networks, storage, containers. In general, it is responsible for starting, organizing, configuring and managing.
- **Docker CLI** – a command line interface that instructs Docker Daemon to execute a specific command.

Other components of **Docker** are **Docker Image** and **Docker Container**:

- **Docker Image** – is a software package that has the necessary parameters to create a container. This container will store all libraries, code and necessary tools.
- **Docker Container** – containers allow you to package an application and all its dependent components into a single object that can be ported to another system.

Noteworthy components include **Dockerfile** – a text file that contains a specific set of instructions for creating a **Docker Image**; **Docker Hub** – a place where **Docker Images** can be stored, you can make them public or private; **Docker Compose** – a tool that provides management of multiple containers (multi-containers); **Docker Volumes** – a way to store and manage data inside a container; **Docker Network** – helps to provide communication between the host and the local network.

2. How does Docker compare to traditional virtual machines?

Answer: Docker and traditional **Virtual Machines** are used to deploy programs and applications. However, they show differences in their purposes of use, and each is suited to a different configuration and task.

Docker:

- Operates at the operating system level. Each container includes only the application and its parameters. Containers run faster, use fewer resources (CPU, RAM). Less stringent security. High mobility. Mainly used for microservices, CI/CD where deployment and scalability are faster.

Virtual Machine:

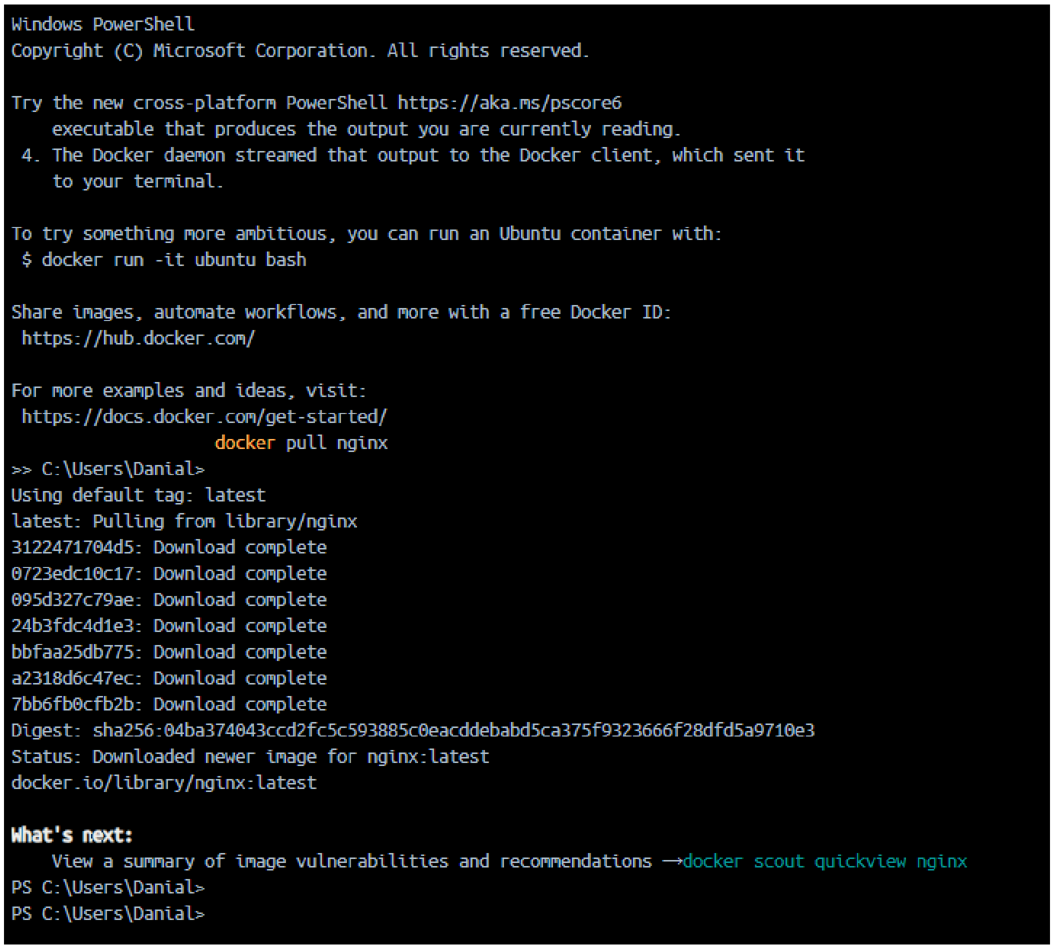
- Runs externally, at the hardware level, on separate operating systems. Requires more time to start, as the startup phase begins with OS startup. Consumes more resources. More secure, but heavy in configuration. Less mobile, too dependent. Demanding OS dependency. More often used for applications with high security requirements and working with different OS.

For example, Netflix video hosting, Spotify audio library, and Airbnb rental search service use Docker because they need speed, light weight, and are well suited for microservice architecture. Amazon Web Services, Microsoft Azure, VMware - resource-intensive services, with strong OS protection and isolation, have complex architecture, bulky.

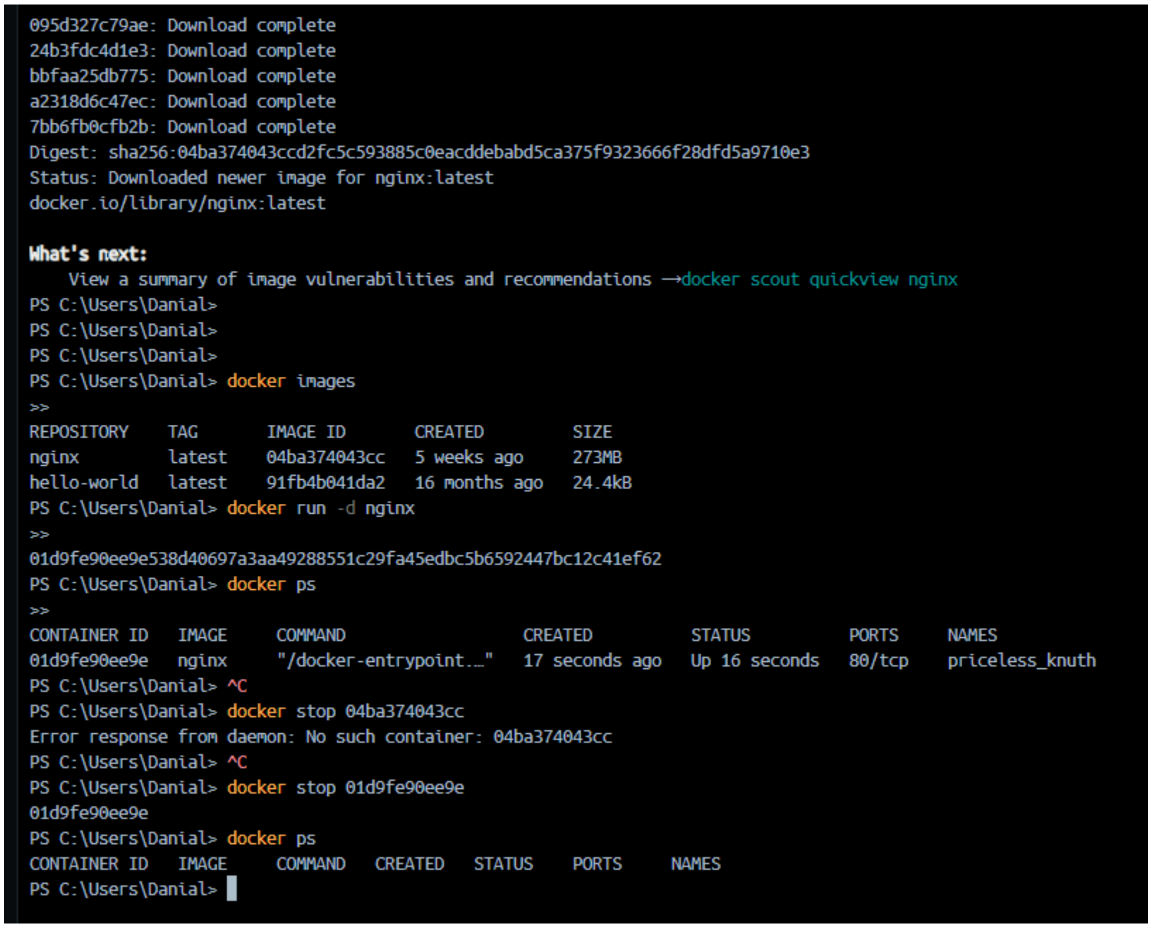
3. What was the output of the docker run hello-world command, and what does it signify?

Answer: This means that the program is installed successfully, and checks for the possibility of communication between the Docker client and daemon, pull images from Docker Hub, and run containers.

Exercise 2: Basic Docker Commands



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6
    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/
                    docker pull nginx
>> C:\Users\Danial>
Using default tag: latest
latest: Pulling from library/nginx
3122471704d5: Download complete
0723edc10c17: Download complete
095d327c79ae: Download complete
24b3fdc4d1e3: Download complete
bbfaa25db775: Download complete
a2318d6c47ec: Download complete
7bb6fb0cfb2b: 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\Danial>
PS C:\Users\Danial>
```



```
095d327c79ae: Download complete
24b3fdc4d1e3: Download complete
bbfaa25db775: Download complete
a2318d6c47ec: Download complete
7bb6fb0cfb2b: 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\Danial>
PS C:\Users\Danial>
PS C:\Users\Danial>
PS C:\Users\Danial> docker images
>>
REPOSITORY    TAG       IMAGE ID       CREATED         SIZE
nginx         latest    04ba374043cc   5 weeks ago     273MB
hello-world   latest    91fb4b041da2   16 months ago   24.4kB
PS C:\Users\Danial> docker run -d nginx
>>
01d9fe90ee9e538d40697a3aa49288551c29fa45edbc5b6592447bc12c41ef62
PS C:\Users\Danial> docker ps
>>
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS     NAMES
01d9fe90ee9e   nginx     "/docker-entrypoint.…"   17 seconds ago   Up 16 seconds   80/tcp    priceless_knuth
PS C:\Users\Danial> ^C
PS C:\Users\Danial> docker stop 04ba374043cc
Error response from daemon: No such container: 04ba374043cc
PS C:\Users\Danial> ^C
PS C:\Users\Danial> docker stop 01d9fe90ee9e
01d9fe90ee9e
PS C:\Users\Danial> docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
PS C:\Users\Danial>
```

Questions and answers

1. What is the difference between ***docker pull*** and ***docker run***?

Answer:

docker pull – this command downloads images from the internet, extracts it, but does not create or run any containers.

docker run – this command extracts an image and creates a container from the image, also, this command can run a container.

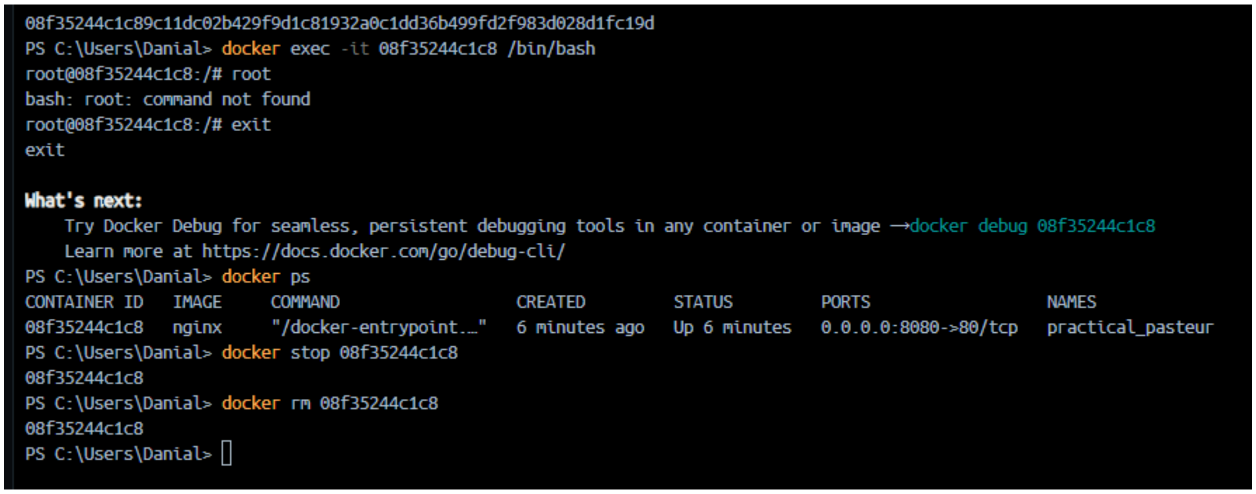
2. How do you find the details of a running container, such as its ID and status?

Answer: To find out the details, you need to enter the docker ps command. This command outputs CONTAINER ID, IMAGE, COMMAND, CREATED, STATUS, PORTS, NAMES.

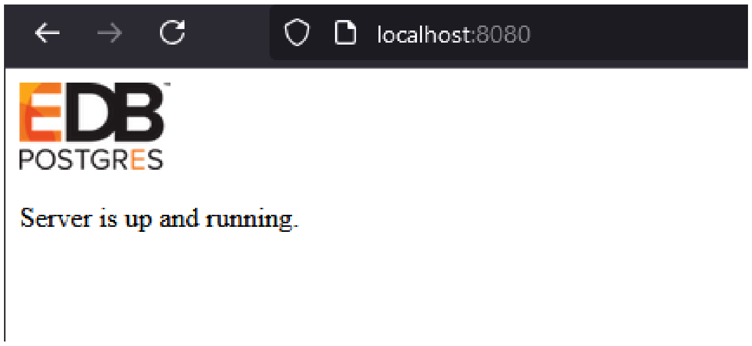
3. What happens to a container after it is stopped? Can it be restarted?

Answer: A container in Docker is stopped using the **'docker stop'** command, this command stops all processes running inside, but the container itself remains inside the system. A stopped container can be restarted using the **docker start <container-id>** command without having to create a new one.

Exercise 3: Working with Docker Containers



```
08f35244c1c89c11dc02b429f9d1c81932a0c1dd36b499fd2f983d028d1fc19d
PS C:\Users\Danial> docker exec -it 08f35244c1c8 /bin/bash
root@08f35244c1c8:/# root
bash: root: command not found
root@08f35244c1c8:/# exit
exit

What's next:
    Try Docker Debug for seamless, persistent debugging tools in any container or image →docker debug 08f35244c1c8
    Learn more at https://docs.docker.com/go/debug-cli/
PS C:\Users\Danial> docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS         PORTS                  NAMES
08f35244c1c8   nginx     "/docker-entrypoint.…"   6 minutes ago   Up 6 minutes   0.0.0.0:8080->80/tcp   practical_pasteur
PS C:\Users\Danial> docker stop 08f35244c1c8
08f35244c1c8
PS C:\Users\Danial> docker rm 08f35244c1c8
08f35244c1c8
PS C:\Users\Danial> 
```



```
localhost:8080
```

Server is up and running.

Questions and answers

1. How does port mapping work in Docker, and why is it important?

Answer: Port mapping in Docker is needed to route all requests passing through the host port to the Docker container. Example: docker run -d -p 5000:5000 my-flask-app command - here -p 5000:5000 maps port 5000 on the host to port 5000 inside the container, which means that invocations will be directed to port 5000 in that application's container.

2. What is the purpose of the ***docker exec*** command?

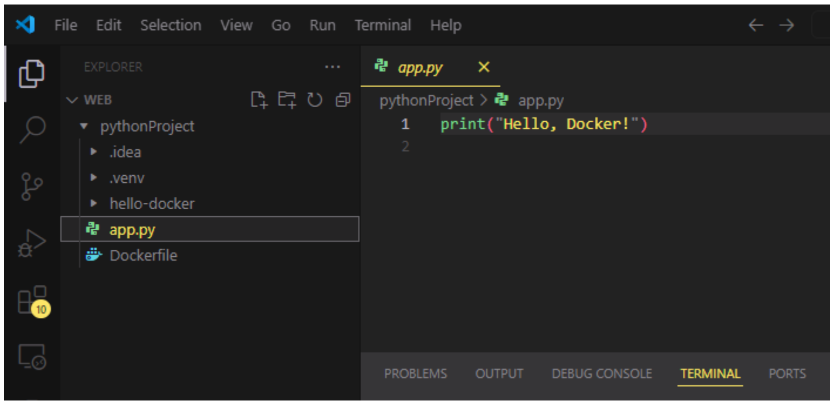
Answer: This command helps to run commands inside the active Docker container and manage all internal processes in real time.

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

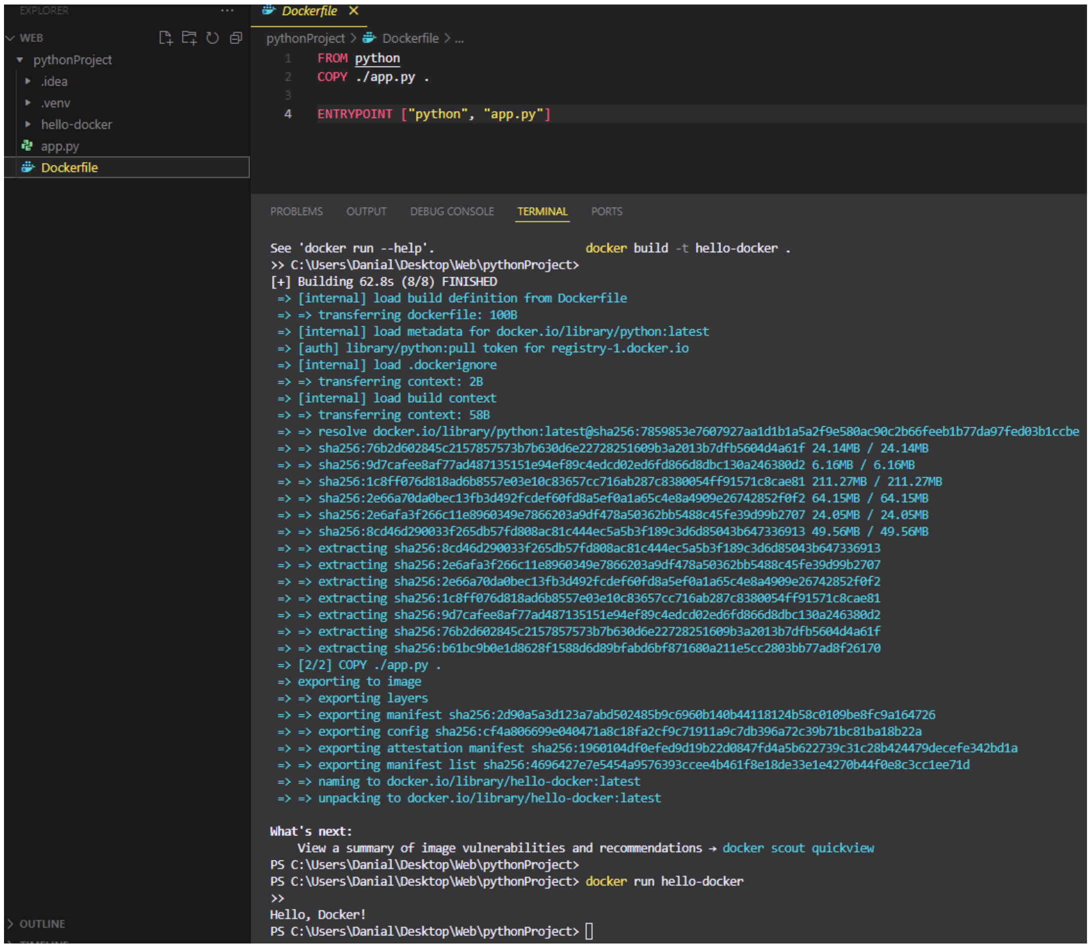
Answer: To check the resource consumption of a stopped container you need to remove it using the **docker rm <container-id>** command.

Dockerfile

Exercise 1: Creating a Simple Dockerfile


File Edit Selection View Go Run Terminal Help
EXPLORER
WEB
pythonProject
.idea
.venv
hello-docker
app.py
Dockerfile
pythonProject > app.py
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

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


EXPLORER
WEB
pythonProject
.idea
.venv
hello-docker
app.py
Dockerfile
pythonProject > Dockerfile > ...
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
OUTLINE

```dockerfile
FROM python
COPY ./app.py .

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

```
See 'docker run --help'.                    docker build -t hello-docker .
>> C:\Users\Danial\Desktop\Web\pythonProject>
[+] Building 62.8s (8/8) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 100B
 => [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
 => [internal] load build context
 => => transferring context: 58B
 => => resolve docker.io/library/python:latest@sha256:7859853e7607927aa1d1b1a5a2f9e580ac90c2b66feeb1b77da97fed03b1ccbe
 => => sha256:76b2d602845c2157857573b7b630d6e22728251609b3a2013b7dfb5604d4a61f 24.14MB / 24.14MB
 => => sha256:9d7cafee8af77ad487135151e94ef89c4edcd02ed6fd866d8dbc130a246380d2 6.16MB / 6.16MB
 => => sha256:1c8ff076d818ad6b8557e03e10c83657cc716ab287c8380054ff91571c8cae81 211.27MB / 211.27MB
 => => sha256:2e66a70da0bec13fb3d492fcdef60fd8a5ef0a1a65c4e8a4909e26742852f0f2 64.15MB / 64.15MB
 => => sha256:2e6afa3f266c11e8960349e7866203a9df478a50362bb5488c45fe39d99b2707 24.05MB / 24.05MB
 => => sha256:8cd46d290033f265db57fd808ac81c444ec5a5b3f189c3d6d85043b647336913 49.56MB / 49.56MB
 => => extracting sha256:8cd46d290033f265db57fd808ac81c444ec5a5b3f189c3d6d85043b647336913
 => => extracting sha256:2e6afa3f266c11e8960349e7866203a9df478a50362bb5488c45fe39d99b2707
 => => extracting sha256:2e66a70da0bec13fb3d492fcdef60fd8a5ef0a1a65c4e8a4909e26742852f0f2
 => => extracting sha256:1c8ff076d818ad6b8557e03e10c83657cc716ab287c8380054ff91571c8cae81
 => => extracting sha256:9d7cafee8af77ad487135151e94ef89c4edcd02ed6fd866d8dbc130a246380d2
 => => extracting sha256:76b2d602845c2157857573b7b630d6e22728251609b3a2013b7dfb5604d4a61f
 => => extracting sha256:b61bc9b0e1d8628f1588d6d89bfabd6bf871680a211e5cc2803bb77ad8f26170
 => [2/2] COPY ./app.py .
 => exporting to image
 => => exporting layers
 => => exporting manifest sha256:2d90a5a3d123a7abd502485b9c6960b140b44118124b58c0109be8fc9a164726
 => => exporting config sha256:cf4a806699e040471a8c18fa2cf9c71911a9c7db396a72c39b71bc81ba18b22a
 => => exporting attestation manifest sha256:1960104df0efed9d19b22d0847fd4a5b622739c31c28b424479decefe342bd1a
 => => exporting manifest list sha256:4696427e7e5454a9576393ccee4b461f8e18de33e1e4270b44f0e8c3cc1ee71d
 => => naming to docker.io/library/hello-docker:latest
 => => unpacking to docker.io/library/hello-docker:latest

What's next:
    View a summary of image vulnerabilities and recommendations → docker scout quickview
PS C:\Users\Danial\Desktop\Web\pythonProject>
PS C:\Users\Danial\Desktop\Web\pythonProject> docker run hello-docker
>>
Hello, Docker!
PS C:\Users\Danial\Desktop\Web\pythonProject> []
```

Questions and answers

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

Answer: the word FROM is written when building the base image in Docker.

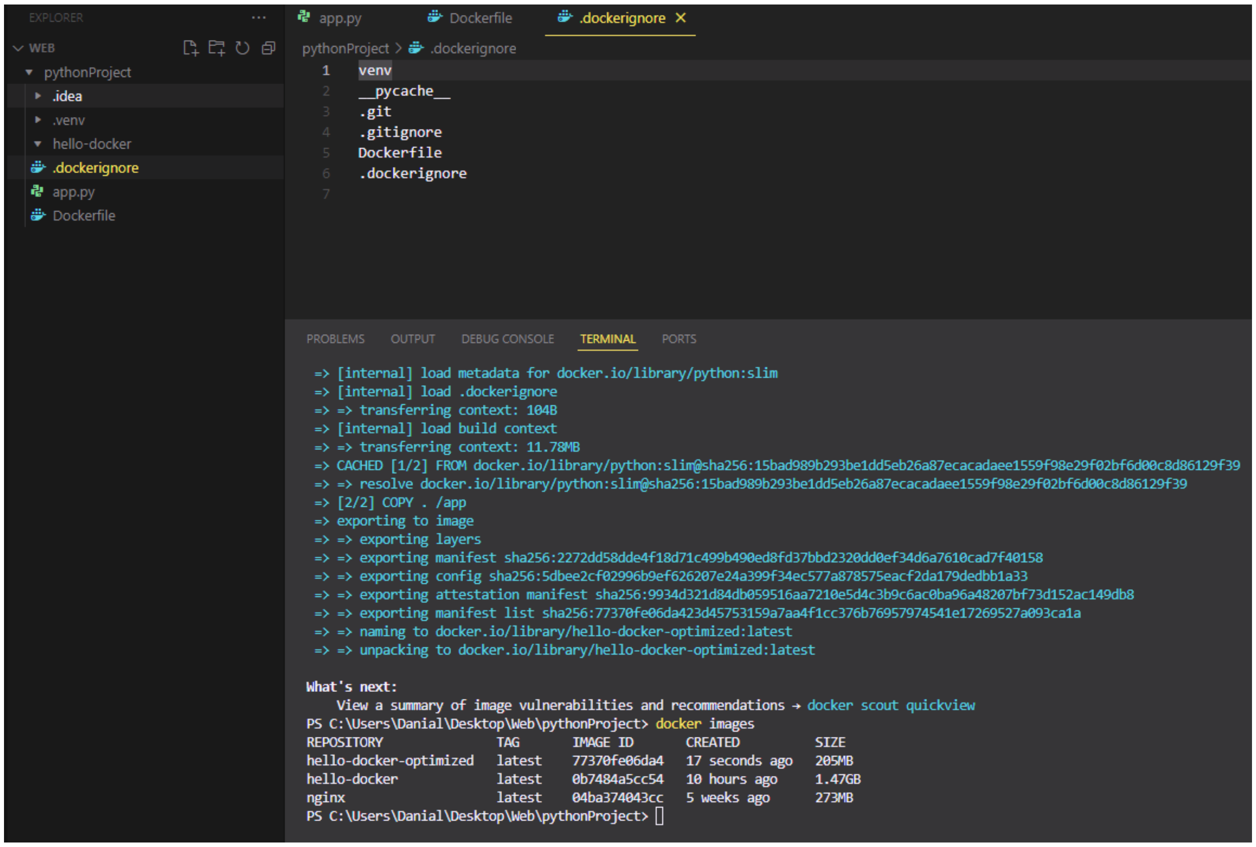
2. How does the COPY instruction work in Dockerfile?

Answer: the word COPY makes it possible to copy files from a local file into it, such as scripts, configs, static files, etc.

3. What is the difference between CMD and ENTRYPOINT in Dockerfile?

Answer: CMD is used to be able to change the command at startup, and ENTRYPOINT is used so that the command is always executed and not changed.

Exercise 2: Optimizing Dockerfile with Layers and Caching



```
venv
__pycache__
.git
.gitignore
Dockerfile
.dockerignore
```

```
 => [internal] load metadata for docker.io/library/python:slim
 => [internal] load .dockerignore
 => => transferring context: 104B
 => [internal] load build context
 => => transferring context: 11.78MB
 => CACHED [1/2] FROM docker.io/library/python:slim@sha256:15bad989b293be1dd5eb26a87ecacadaee1559f98e29f02bf6d00c8d86129f39
 => => resolve docker.io/library/python:slim@sha256:15bad989b293be1dd5eb26a87ecacadaee1559f98e29f02bf6d00c8d86129f39
 => [2/2] COPY . /app
 => exporting to image
 => => exporting layers
 => => exporting manifest sha256:2272dd58dde4f18d71c499b490ed8fd37bbd2320dd0ef34d6a7610cad7f40158
 => => exporting config sha256:5dbee2cf02996b9ef626207e24a399f34ec577a878575eacf2da179dedbb1a33
 => => exporting attestation manifest sha256:9934d321d84db059516aa7210e5d4c3b9c6ac0ba96a48207bf73d152ac149db8
 => => exporting manifest list sha256:77370fe06da423d45753159a7aa4f1cc376b76957974541e17269527a093ca1a
 => => naming to docker.io/library/hello-docker-optimized:latest
 => => unpacking to docker.io/library/hello-docker-optimized:latest

What's next:
    View a summary of image vulnerabilities and recommendations → docker scout quickview
PS C:\Users\Danial\Desktop\Web\pythonProject> docker images
REPOSITORY               TAG       IMAGE ID       CREATED          SIZE
hello-docker-optimized   latest    77370fe06da4   17 seconds ago   205MB
hello-docker             latest    0b7484a5cc54   10 hours ago     1.47GB
nginx                    latest    04ba374043cc   5 weeks ago      273MB
PS C:\Users\Danial\Desktop\Web\pythonProject>
```

Questions and answers

1. What are Docker layers, and how do they affect image size and build times?

Answer: Docker layers are the pieces from which images are built in Docker. Docker stores the same layers for different images in the same place, which helps save memory. Build speed is increased by reuse because it uses existing layers.

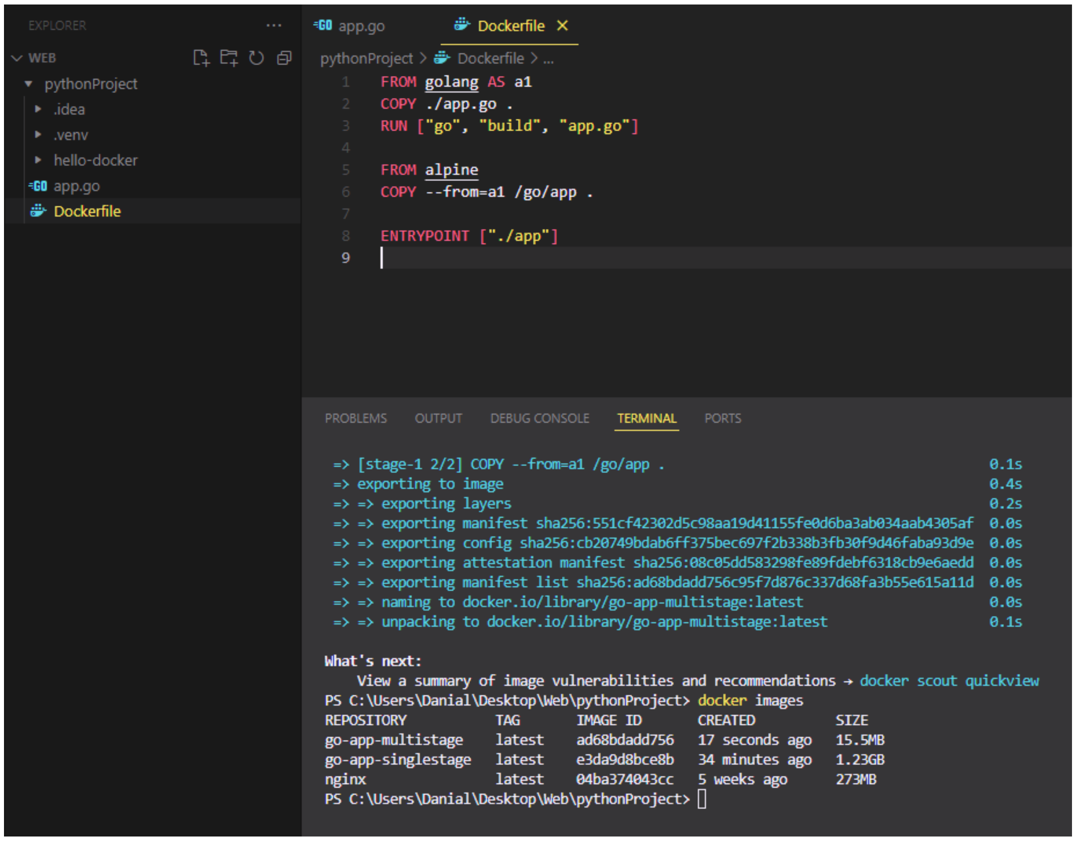
2. How does Docker's build cache work, and how can it speed up the build process?

Answer: The cache in Docker is a mechanism that remembers items that have been used before. The build speed will be increased, files will be rebuilt if they have been changed, and the rest will be taken from the cache.

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

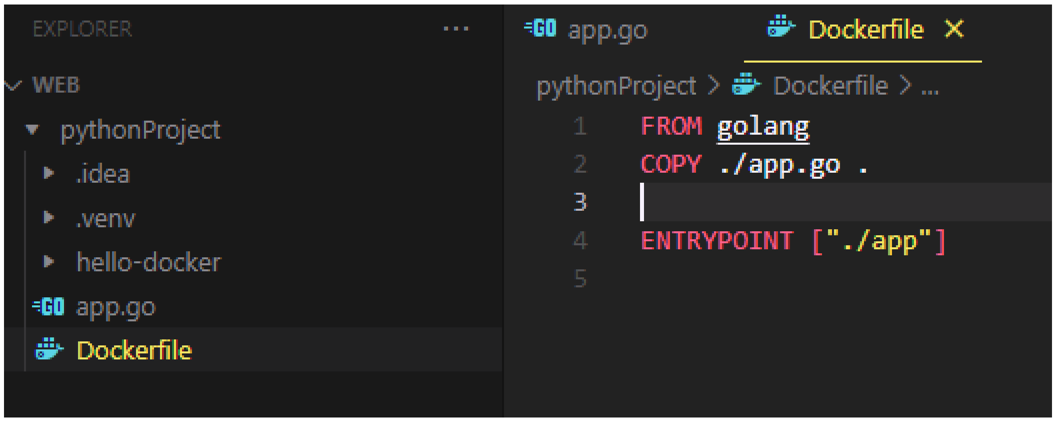
Answer: The .dockerignore file is needed to exclude unnecessary files and folders from the Docker build process. With .dockerignore, you can reduce the size of the build data.

Exercise 3: Multi-Stage Builds

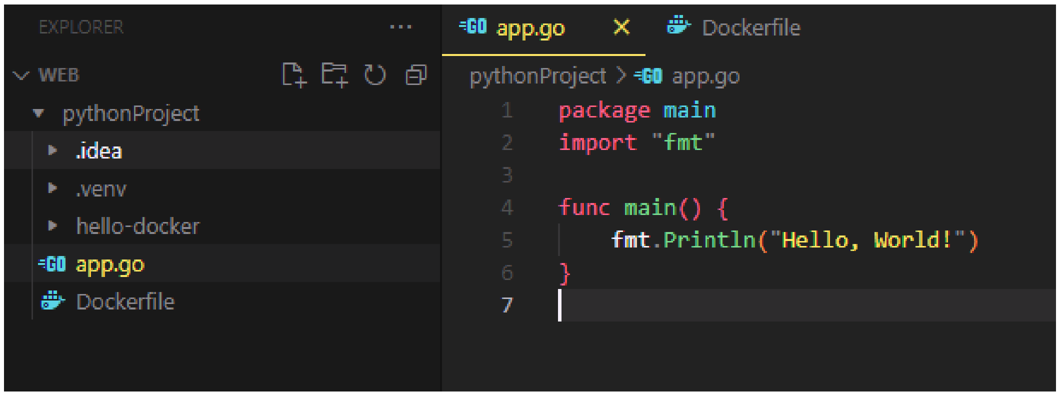


```dockerfile
FROM golang AS a1
COPY ./app.go .
RUN ["go", "build", "app.go"]

FROM alpine
COPY --from=a1 /go/app .

ENTRYPOINT ["./app"]
```

```
 => [stage-1 2/2] COPY --from=a1 /go/app .                                                    0.1s
 => exporting to image                                                                        0.4s
 => => exporting layers                                                                       0.2s
 => => exporting manifest sha256:551cf42302d5c98aa19d41155fe0d6ba3ab034aab4305af  0.0s
 => => exporting config sha256:cb20749bdab6ff375bec697f2b338b3fb30f9d46faba93d9e  0.0s
 => => exporting attestation manifest sha256:08c05dd583298fe89fdebf6318cb9e6aedd  0.0s
 => => exporting manifest list sha256:ad68bdadd756c95f7d876c337d68fa3b55e615a11d  0.0s
 => => naming to docker.io/library/go-app-multistage:latest                                   0.0s
 => => unpacking to docker.io/library/go-app-multistage:latest                                0.1s

What's next:
    View a summary of image vulnerabilities and recommendations → docker scout quickview
PS C:\Users\Danial\Desktop\Web\pythonProject> docker images
REPOSITORY            TAG       IMAGE ID       CREATED          SIZE
go-app-multistage     latest    ad68bdadd756   17 seconds ago   15.5MB
go-app-singlestage    latest    e3da9d8bce8b   34 minutes ago   1.23GB
nginx                 latest    04ba374043cc   5 weeks ago      273MB
PS C:\Users\Danial\Desktop\Web\pythonProject>
```



```dockerfile
FROM golang
COPY ./app.go .

ENTRYPOINT ["./app"]
```



```go
package main
import "fmt"

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

```

Questions and answers

1. What is the benefit of using multi-stage builds in Docker?

Answer: With multi-stage build you can separate the different stages of build and isolate the final image, this makes images easier and cleaner.

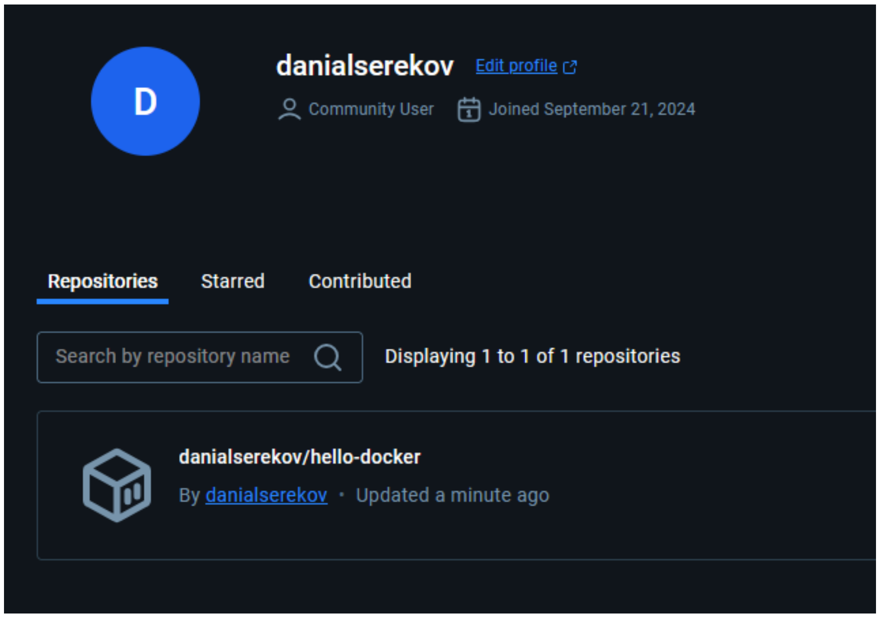
2. How can multi-stage build help reduce the size of Docker images?

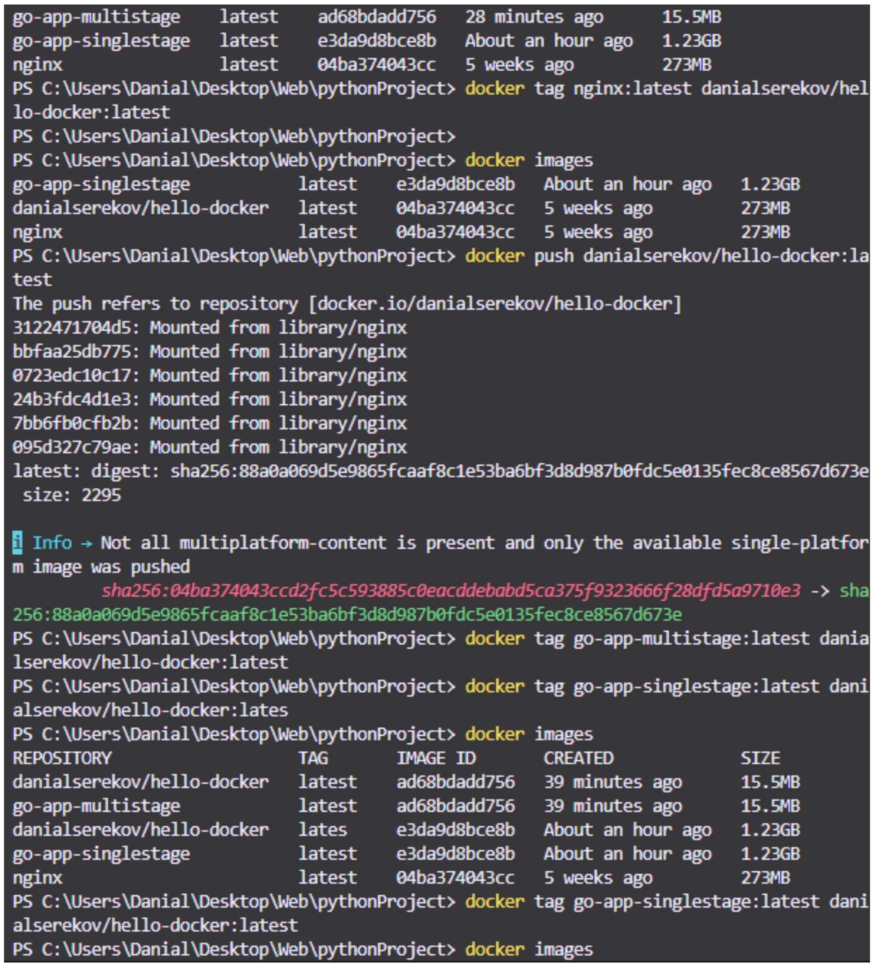
Answer: Docker's multi-stage build eliminates unnecessary libraries, tools that were used to build, but are not needed to run.

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

Answer: Most often multi-stage builds are used in compiled languages such as C++ and Golang, in such applications minimum size, performance and security are important.

Exercise 4: Pushing Docker Images to Docker Hub


D
danialserekov
Edit profile
Community User
Joined September 21, 2024
Repositories
Starred
Contributed
Search by repository name
Displaying 1 to 1 of 1 repositories
danialserekov/hello-docker
By danialserekov · Updated a minute ago



```
go-app-multistage    latest    ad68bdadd756   28 minutes ago      15.5MB
go-app-singlestage   latest    e3da9d8bce8b   About an hour ago   1.23GB
nginx                latest    04ba374043cc   5 weeks ago         273MB
PS C:\Users\Danial\Desktop\Web\pythonProject> docker tag nginx:latest danialserekov/hel
lo-docker:latest
PS C:\Users\Danial\Desktop\Web\pythonProject>
PS C:\Users\Danial\Desktop\Web\pythonProject> docker images
go-app-singlestage            latest    e3da9d8bce8b   About an hour ago   1.23GB
danialserekov/hello-docker    latest    04ba374043cc   5 weeks ago         273MB
nginx                         latest    04ba374043cc   5 weeks ago         273MB
PS C:\Users\Danial\Desktop\Web\pythonProject> docker push danialserekov/hello-docker:la
test
The push refers to repository [docker.io/danialserekov/hello-docker]
3122471704d5: Mounted from library/nginx
bbfaa25db775: Mounted from library/nginx
0723edc10c17: Mounted from library/nginx
24b3fdc4d1e3: Mounted from library/nginx
7bb6fb0cfb2b: Mounted from library/nginx
095d327c79ae: Mounted from library/nginx
latest: digest: sha256:88a0a069d5e9865fcaaf8c1e53ba6bf3d8d987b0fdc5e0135fec8ce8567d673e
 size: 2295

i Info → Not all multiplatform-content is present and only the available single-platfor
m image was pushed
         sha256:04ba374043ccd2fc5c593885c0eacddebabd5ca375f9323666f28dfd5a9710e3 -> sha
256:88a0a069d5e9865fcaaf8c1e53ba6bf3d8d987b0fdc5e0135fec8ce8567d673e
PS C:\Users\Danial\Desktop\Web\pythonProject> docker tag go-app-multistage:latest dania
lserekov/hello-docker:latest
PS C:\Users\Danial\Desktop\Web\pythonProject> docker tag go-app-singlestage:latest dani
alserekov/hello-docker:lates
PS C:\Users\Danial\Desktop\Web\pythonProject> docker images
REPOSITORY                    TAG       IMAGE ID       CREATED             SIZE
danialserekov/hello-docker    latest    ad68bdadd756   39 minutes ago      15.5MB
go-app-multistage             latest    ad68bdadd756   39 minutes ago      15.5MB
danialserekov/hello-docker    lates     e3da9d8bce8b   About an hour ago   1.23GB
go-app-singlestage            latest    e3da9d8bce8b   About an hour ago   1.23GB
nginx                         latest    04ba374043cc   5 weeks ago         273MB
PS C:\Users\Danial\Desktop\Web\pythonProject> docker tag go-app-singlestage:latest dani
alserekov/hello-docker:latest
PS C:\Users\Danial\Desktop\Web\pythonProject> docker images
```

Questions and answers

1. What is the purpose of Docker Hub in containerization?

Answer: Docker Hub is a platform for storing and sharing Docker images where you can upload and download containers, they can be public and private.–

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

Answer: Tagging an image can be done with the docker tag command, this is done to name the image with a username in Docker Hub.

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

Answer: Log in to Docker Hub (docker login) → Tag image (docker tag) → Send image (docker push <username>/<repository-name>).