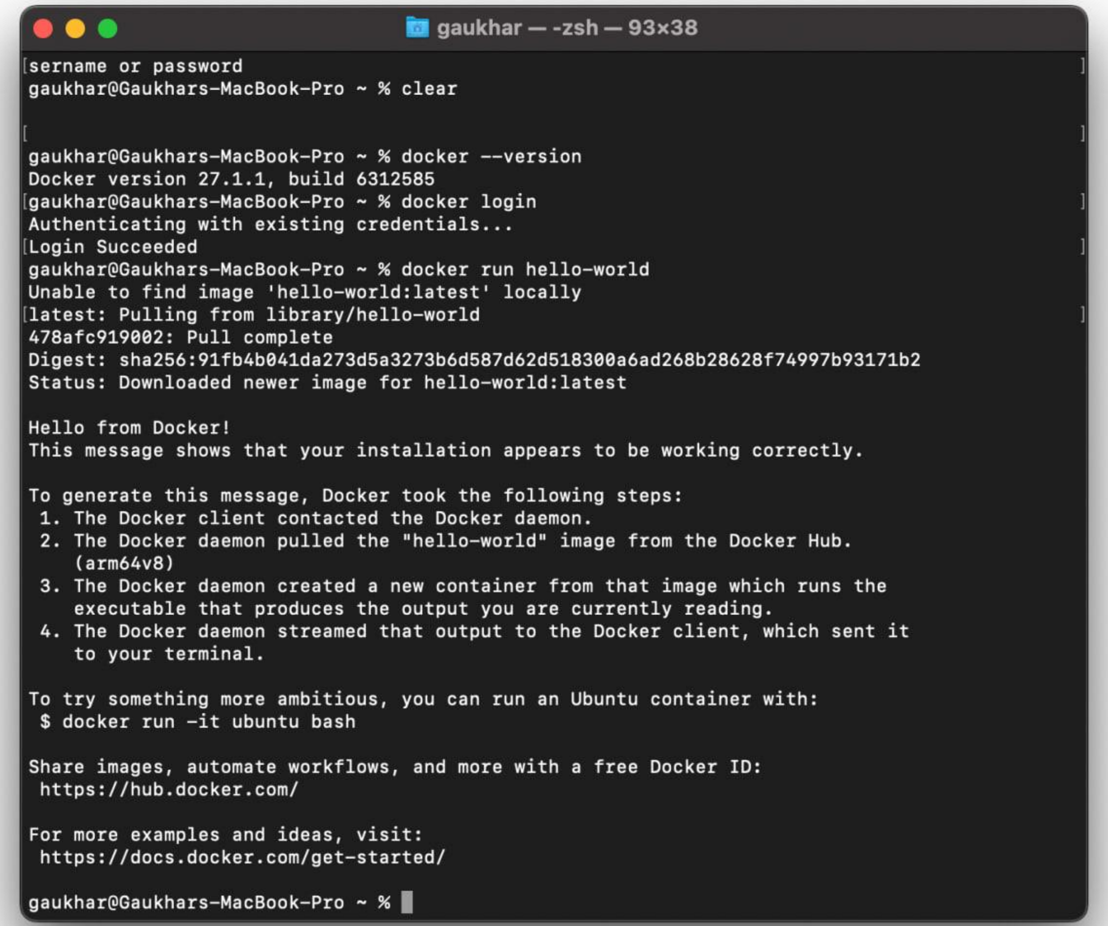
GAUKHAR UZAKBAY 23MD0449

Exercise 1: Installing Docker



```
[sername or password                                                                           ]
gaukhar@Gaukhars-MacBook-Pro ~ % clear

[                                                                                              ]
gaukhar@Gaukhars-MacBook-Pro ~ % docker --version
Docker version 27.1.1, build 6312585
[gaukhar@Gaukhars-MacBook-Pro ~ % docker login                                                 ]
Authenticating with existing credentials...
[Login Succeeded                                                                               ]
gaukhar@Gaukhars-MacBook-Pro ~ % docker run hello-world
Unable to find image 'hello-world:latest' locally
[latest: Pulling from library/hello-world                                                      ]
478afc919002: Pull complete
Digest: sha256:91fb4b041da273d5a3273b6d587d62d518300a6ad268b28628f74997b93171b2
Status: Downloaded newer image for hello-world:latest

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

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (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/

gaukhar@Gaukhars-MacBook-Pro ~ % 
```

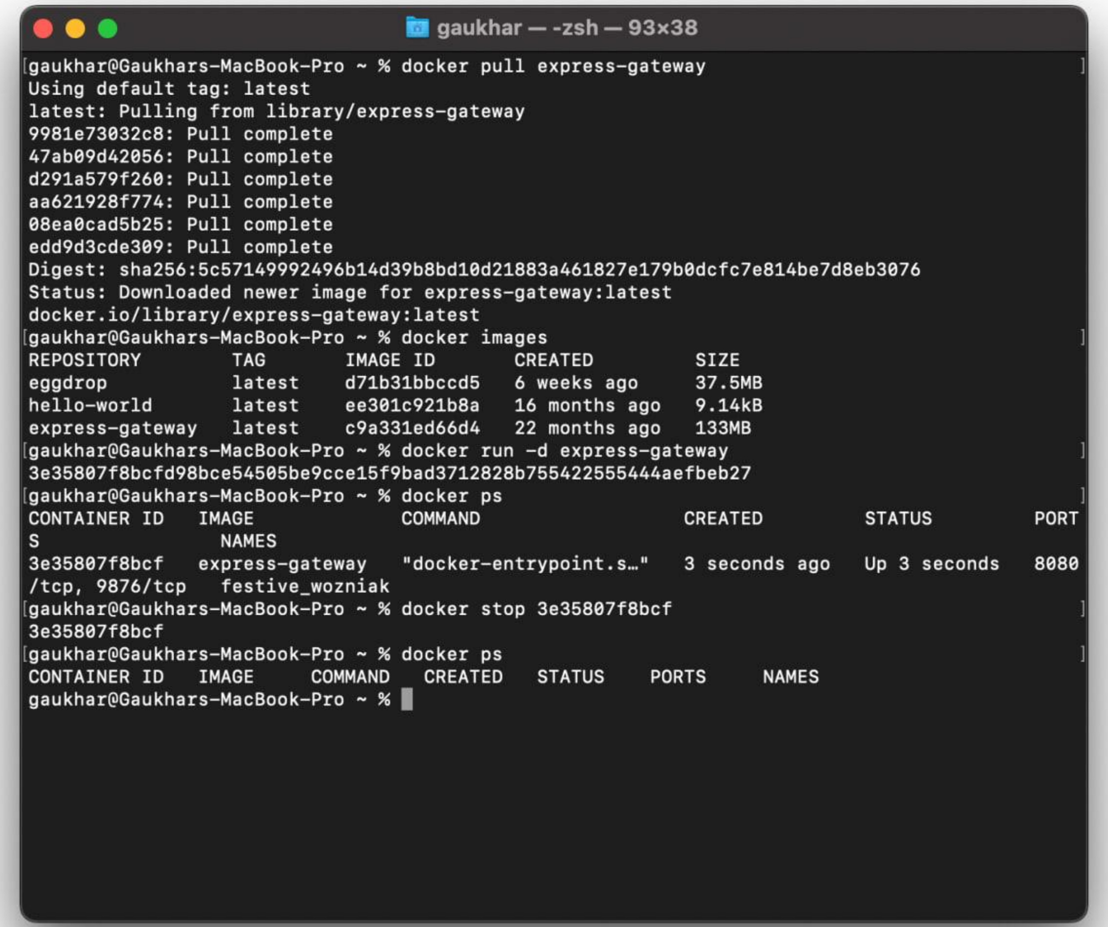
Answers:

1) Docker Engine: The core service that runs containers.
   Docker CLI: The command-line interface to interact with Docker.
   Docker Daemon: Manages Docker objects like containers, images, and networks.
   Docker Hub: A public registry to share and store Docker images.

2) Docker uses containers which are lightweight, sharing the host OS kernel, making them faster and more resource-efficient. VMs use hypervisors and each VM has its own OS, making them heavier and slower to start than containers.

3) This signifies that Docker is installed correctly and the container was successfully run.

Exercise 2: Basic Docker Commands



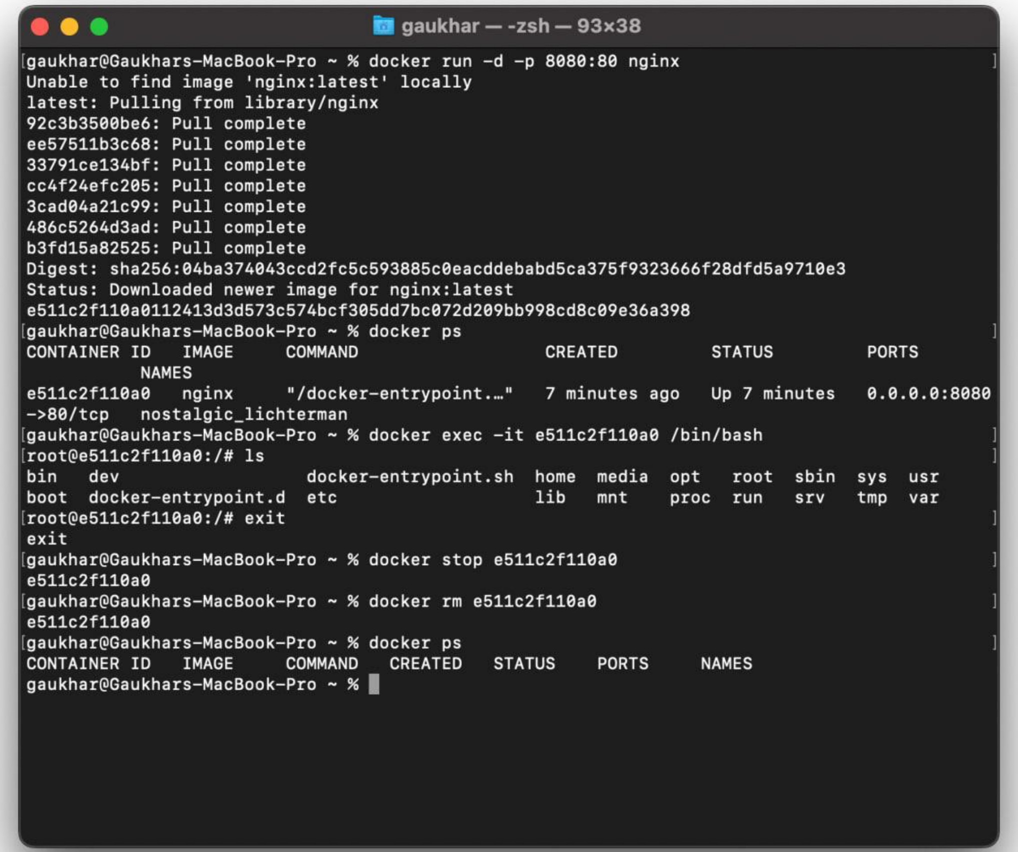
```
gaukhar@Gaukhars-MacBook-Pro ~ % docker pull express-gateway
Using default tag: latest
latest: Pulling from library/express-gateway
9981e73032c8: Pull complete
47ab09d42056: Pull complete
d291a579f260: Pull complete
aa621928f774: Pull complete
08ea0cad5b25: Pull complete
edd9d3cde309: Pull complete
Digest: sha256:5c57149992496b14d39b8bd10d21883a461827e179b0dcfc7e814be7d8eb3076
Status: Downloaded newer image for express-gateway:latest
docker.io/library/express-gateway:latest
gaukhar@Gaukhars-MacBook-Pro ~ % docker images
REPOSITORY        TAG       IMAGE ID       CREATED         SIZE
eggdrop           latest    d71b31bbccd5   6 weeks ago     37.5MB
hello-world       latest    ee301c921b8a   16 months ago   9.14kB
express-gateway   latest    c9a331ed66d4   22 months ago   133MB
gaukhar@Gaukhars-MacBook-Pro ~ % docker run -d express-gateway
3e35807f8bcfd98bce54505be9cce15f9bad3712828b755422555444aefbeb27
gaukhar@Gaukhars-MacBook-Pro ~ % docker ps
CONTAINER ID   IMAGE             COMMAND                  CREATED         STATUS         PORT
S                NAMES
3e35807f8bcf   express-gateway   "docker-entrypoint.s…"   3 seconds ago   Up 3 seconds   8080
/tcp, 9876/tcp   festive_wozniak
gaukhar@Gaukhars-MacBook-Pro ~ % docker stop 3e35807f8bcf
3e35807f8bcf
gaukhar@Gaukhars-MacBook-Pro ~ % docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
gaukhar@Gaukhars-MacBook-Pro ~ %
```

Answers:

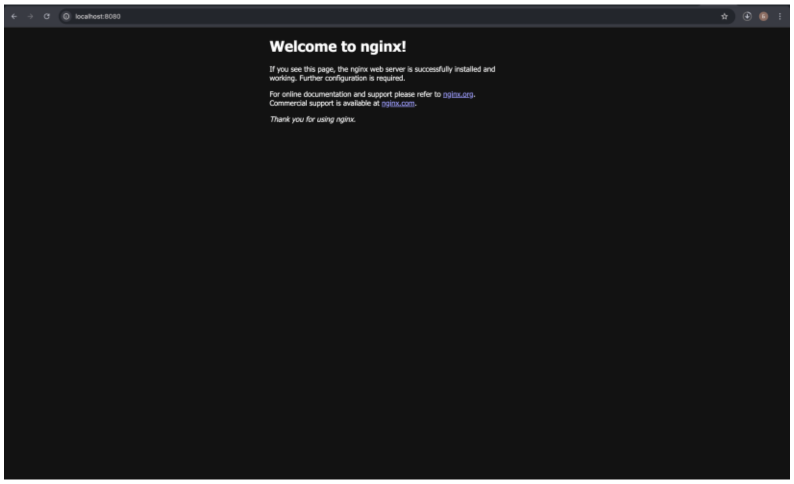
1) **docker pull** downloads an image from a registry (e.g., Docker Hub) to your local machine.
   docker run pulls the image if it's not already downloaded and then creates and starts a container from the image.
2) Use **docker ps** to see details like container ID, name, status, and other information about running containers. For more details about any container (running or stopped), you can use **docker inspect <container-id>**.

3) The container's state is saved, and it can be restarted. To restart it, use docker start <container-id> without needing to create it again.

Exercise 3: Working with Docker Containers



```
[gaukhar@Gaukhars-MacBook-Pro ~ % docker run -d -p 8080:80 nginx                               ]
Unable to find image 'nginx:latest' locally
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
e511c2f110a0112413d3d573c574bcf305dd7bc072d209bb998cd8c09e36a398
[gaukhar@Gaukhars-MacBook-Pro ~ % docker ps                                                    ]
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS         PORTS
           NAMES
e511c2f110a0   nginx     "/docker-entrypoint.…"   7 minutes ago   Up 7 minutes   0.0.0.0:8080
->80/tcp   nostalgic_lichterman
[gaukhar@Gaukhars-MacBook-Pro ~ % docker exec -it e511c2f110a0 /bin/bash                       ]
[root@e511c2f110a0:/# ls                                                                       ]
bin   dev                  docker-entrypoint.sh  home  media  opt   root  sbin  sys  usr
boot  docker-entrypoint.d  etc                   lib   mnt    proc  run   srv   tmp  var
[root@e511c2f110a0:/# exit                                                                     ]
exit
[gaukhar@Gaukhars-MacBook-Pro ~ % docker stop e511c2f110a0                                     ]
e511c2f110a0
[gaukhar@Gaukhars-MacBook-Pro ~ % docker rm e511c2f110a0                                       ]
e511c2f110a0
[gaukhar@Gaukhars-MacBook-Pro ~ % docker ps                                                    ]
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
gaukhar@Gaukhars-MacBook-Pro ~ % 
```



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.

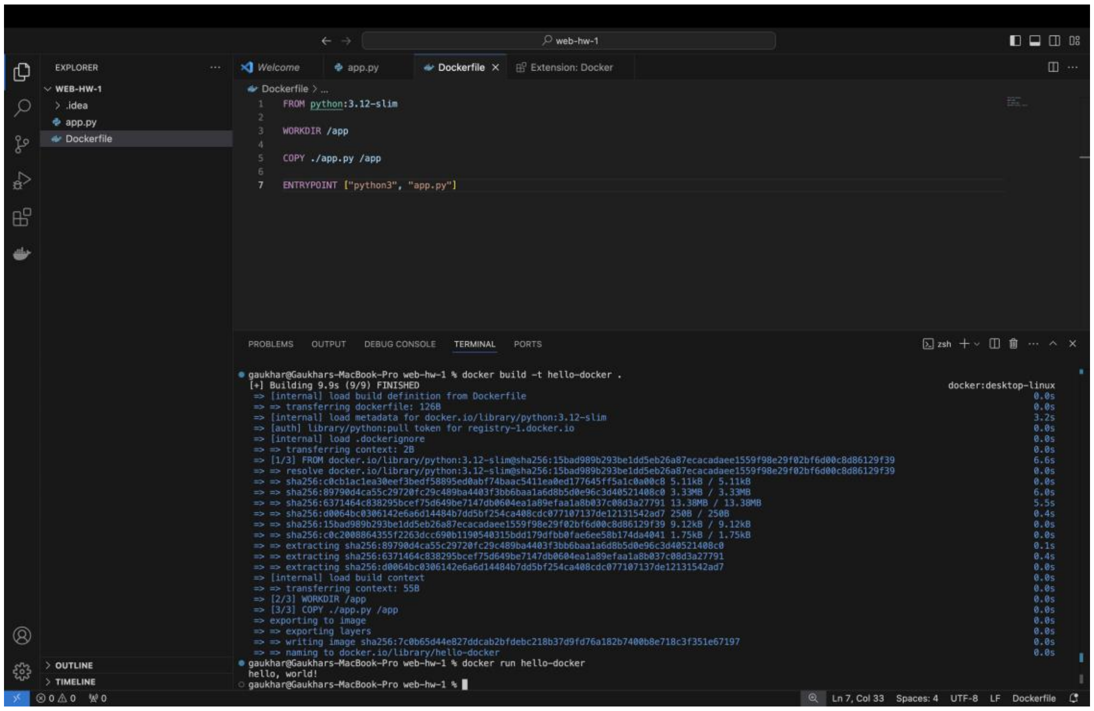
Thank you for using nginx.

Answers:

1) Port mapping links a port on the host machine to a port inside the container (e.g., -p 8080:80 maps host's port 8080 to container's port 80). It's important because it allows external access to services running inside the container.
2) **docker exec** runs a command inside an already running container. It's useful for tasks like opening a shell (**docker exec -it <container> bash**) to inspect or interact with a container.
3) A stopped container doesn't use CPU or memory, but it still consumes disk space. To fully remove it, use **docker rm <container-id>** after stopping it.

Dockerfile

Exercise 1: Creating a Simple Dockerfile



```
FROM python:3.12-slim

WORKDIR /app

COPY ./app.py /app

ENTRYPOINT ["python3", "app.py"]
```

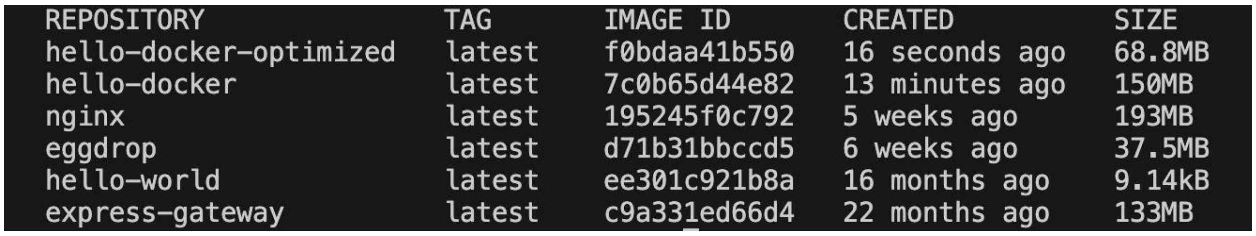
```
gaukhar@Gaukhars-MacBook-Pro web-hw-1 % docker build -t hello-docker .
[+] Building 9.9s (9/9) FINISHED                                                              docker:desktop-linux
 => [internal] load build definition from Dockerfile                                                          0.0s
 => => transferring dockerfile: 126B                                                                          0.0s
 => [internal] load metadata for docker.io/library/python:3.12-slim                                           3.2s
 => [auth] library/python:pull token for registry-1.docker.io                                                 0.0s
 => [internal] load .dockerignore                                                                             0.0s
 => => transferring context: 2B                                                                               0.0s
 => [1/3] FROM docker.io/library/python:3.12-slim@sha256:15bad989b293be1dd5eb26a87ecacadaee1559f98e29f02bf6d00c8d86129f39  6.6s
 => => resolve docker.io/library/python:3.12-slim@sha256:15bad989b293be1dd5eb26a87ecacadaee1559f98e29f02bf6d00c8d86129f39  0.0s
 => => sha256:c0cb1ac1ea30eef3bedf58895ed0abf74baac5411ea0ed177645ff5a1c0a00c8 5.11kB / 5.11kB                 0.0s
 => => sha256:89790d4ca55c29720fc29c489ba4403f3bb6baa1a6d8b5d0e96c3d40521408c0 3.33MB / 3.33MB                 6.0s
 => => sha256:6371464c838295bcef75d649be7147db0604ea1a89efaa1a8b037c08d3a27791 13.38MB / 13.38MB               5.5s
 => => sha256:d0064bc0306142e6a6d14484b7dd5bf254ca408cdc077107137de12131542ad7 250B / 250B                     0.4s
 => => sha256:15bad989b293be1dd5eb26a87ecacadaee1559f98e29f02bf6d00c8d86129f39 9.12kB / 9.12kB                 0.0s
 => => sha256:c0c2008864355f2263dcc690b1190540315bdd179dfbb0fae6ee58b174da4041 1.75kB / 1.75kB                 0.0s
 => => extracting sha256:89790d4ca55c29720fc29c489ba4403f3bb6baa1a6d8b5d0e96c3d40521408c0                     0.1s
 => => extracting sha256:6371464c838295bcef75d649be7147db0604ea1a89efaa1a8b037c08d3a27791                     0.4s
 => => extracting sha256:d0064bc0306142e6a6d14484b7dd5bf254ca408cdc077107137de12131542ad7                     0.0s
 => [internal] load build context                                                                             0.0s
 => => transferring context: 55B                                                                              0.0s
 => [2/3] WORKDIR /app                                                                                        0.0s
 => [3/3] COPY ./app.py /app                                                                                  0.0s
 => exporting to image                                                                                        0.0s
 => => exporting layers                                                                                       0.0s
 => => writing image sha256:7c0b65d44e827ddcab2bfdebc218b37d9fd76a182b7400b8e718c3f351e67197                  0.0s
 => => naming to docker.io/library/hello-docker                                                               0.0s
gaukhar@Gaukhars-MacBook-Pro web-hw-1 % docker run hello-docker
hello, world!
gaukhar@Gaukhars-MacBook-Pro web-hw-1 %
```

Answers:

1) **FROM** specifies the base image for the Dockerfile, which sets up the environment for the container. It's the starting point for building the image.
2) **COPY** copies files or directories from the host machine into the Docker image. For example, COPY ./app.py /app/ places app.py in the /app/ directory inside the image.

3) **CMD** provides default arguments that can be overridden when running the container (docker run). **ENTRYPOINT** defines the command that always runs when the container starts, and its arguments can be supplemented by CMD or at runtime.

Exercise 2: Optimizing Dockerfile with Layers and Caching



```
REPOSITORY               TAG       IMAGE ID       CREATED          SIZE
hello-docker-optimized   latest    f0bdaa41b550   16 seconds ago   68.8MB
hello-docker             latest    7c0b65d44e82   13 minutes ago   150MB
nginx                    latest    195245f0c792   5 weeks ago      193MB
eggdrop                  latest    d71b31bbccd5   6 weeks ago      37.5MB
hello-world              latest    ee301c921b8a   16 months ago    9.14kB
express-gateway          latest    c9a331ed66d4   22 months ago    133MB
```

Answers:

1) Docker images are built in layers, with each Dockerfile instruction creating a new layer. Layers are cached and reused, reducing image size and speeding up future builds since unchanged layers don't need to be rebuilt.
2) Docker caches layers from previous builds. If a step hasn’t changed, Docker reuses the cached layer rather than re-running the instruction, speeding up the build process significantly.
3) The **.dockerignore** file specifies which files or directories to exclude when building the Docker image. It helps reduce image size and speeds up builds by ignoring unnecessary files (e.g., node_modules, .git).

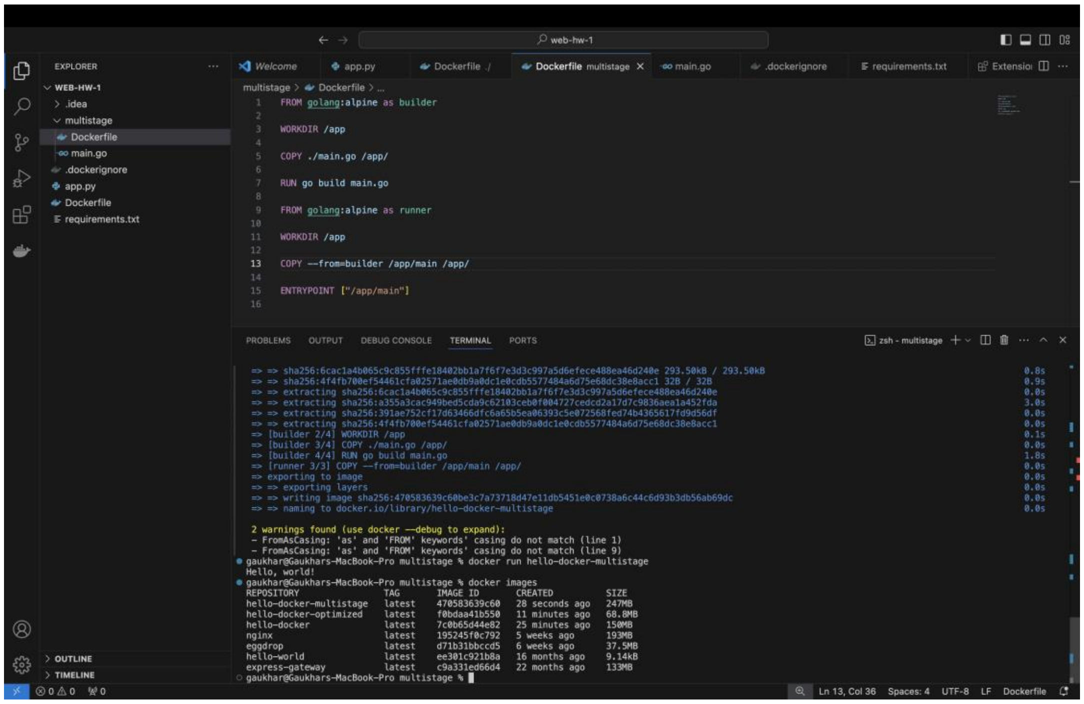
Exercise 3: Multi-Stage Builds

Answers:

1) Multi-stage builds allow you to use multiple FROM instructions in a Dockerfile, each for a different stage. This helps separate the build process from the final runtime environment, making the image cleaner and smaller.
2) In multi-stage builds, you can perform heavy build tasks (e.g., compiling code) in one stage and only copy the necessary artifacts (like compiled binaries) to the final image. This avoids shipping build tools and dependencies in the final image, drastically reducing its size.
3)

- **Compiled languages**: Building Go, Java, or C++ applications where you don’t want to include compilers in the final image.
- **Security-focused builds**: Ensuring that only necessary runtime files are included, minimizing the attack surface.

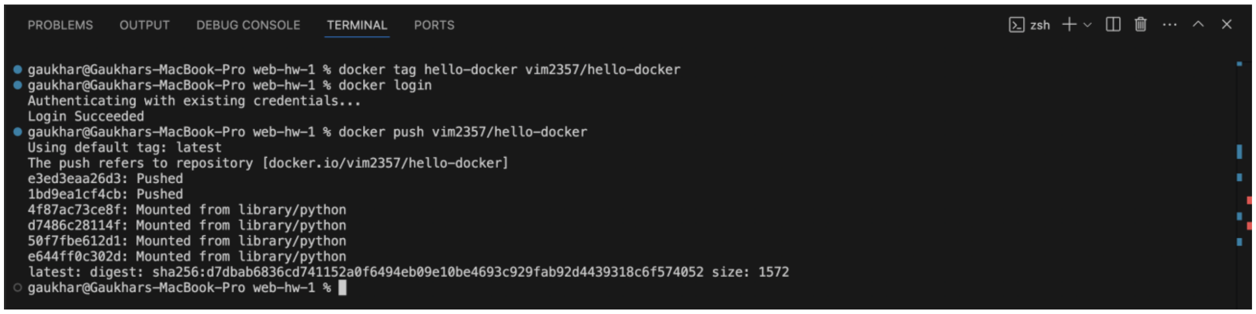
- **Testing**: Running tests in one stage and deploying only the tested, minimal artifacts in the final image.



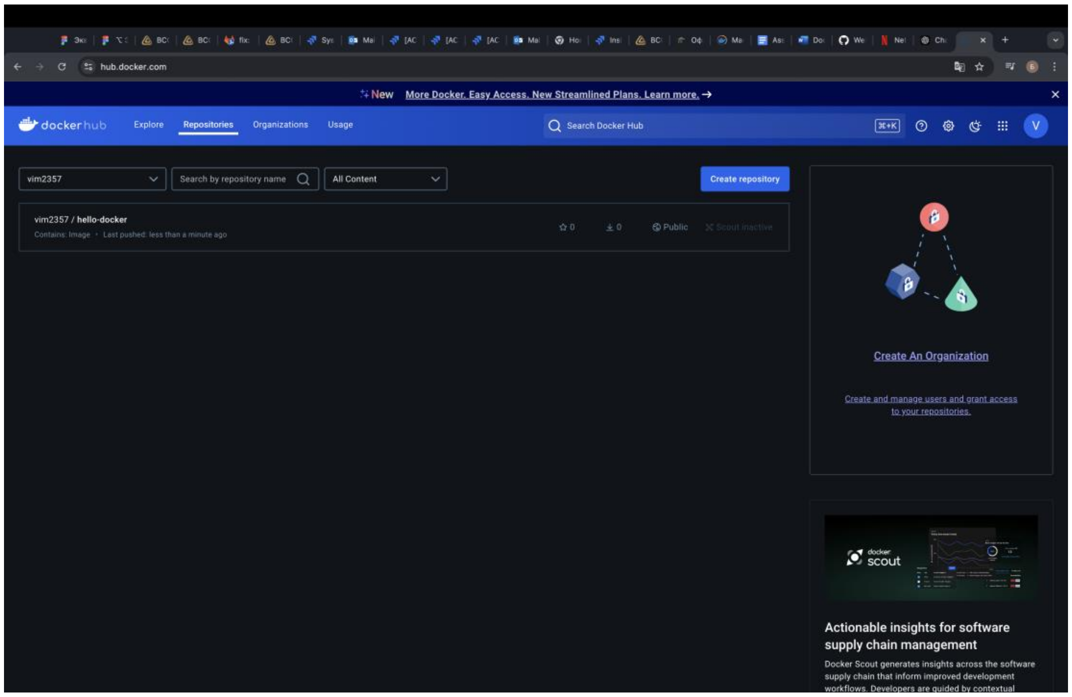
```
FROM golang:alpine as builder

WORKDIR /app

COPY ./main.go /app/

RUN go build main.go

FROM golang:alpine as runner

WORKDIR /app

COPY --from=builder /app/main /app/

ENTRYPOINT ["/app/main"]
```

```
=> => sha256:6cac1a4b065c9c855fffe18402bb1a7f6f7e3d3c997a5d6efece488ea46d240e 293.50kB / 293.50kB    0.8s
=> => sha256:4f4fb700ef54461cfa02571ae0db9a0dc1e0cdb5577484a6d75e68dc38e8acc1 32B / 32B              0.9s
=> => extracting sha256:6cac1a4b065c9c855fffe18402bb1a7f6f7e3d3c997a5d6efece488ea46d240e              0.0s
=> => extracting sha256:a355a3cac949bed5cda9c62103ceb0f004727cedcd2a17d7c9836aea1a452fda              3.0s
=> => extracting sha256:391ae752cf17d63466dfc6a65b5ea06393c5e072568fed74b4365617fd9d56df              0.0s
=> => extracting sha256:4f4fb700ef54461cfa02571ae0db9a0dc1e0cdb5577484a6d75e68dc38e8acc1              0.0s
=> [builder 2/4] WORKDIR /app                                                                          0.1s
=> [builder 3/4] COPY ./main.go /app/                                                                  0.0s
=> [builder 4/4] RUN go build main.go                                                                  1.8s
=> [runner 3/3] COPY --from=builder /app/main /app/                                                    0.0s
=> exporting to image                                                                                  0.0s
=> => exporting layers                                                                                 0.0s
=> => writing image sha256:470583639c60be3c7a73718d47e11db5451e0c0738a6c44c6d93b3db56ab69dc           0.0s
=> => naming to docker.io/library/hello-docker-multistage                                              0.0s

 2 warnings found (use docker --debug to expand):
 - FromAsCasing: 'as' and 'FROM' keywords' casing do not match (line 1)
 - FromAsCasing: 'as' and 'FROM' keywords' casing do not match (line 9)
gaukhar@Gaukhars-MacBook-Pro multistage % docker run hello-docker-multistage
Hello, world!
gaukhar@Gaukhars-MacBook-Pro multistage % docker images
REPOSITORY                TAG       IMAGE ID       CREATED          SIZE
hello-docker-multistage   latest    470583639c60   28 seconds ago   247MB
hello-docker-optimized    latest    f0bdaa41b550   11 minutes ago   68.8MB
hello-docker              latest    7c0b65d44e82   25 minutes ago   150MB
nginx                     latest    195245f0c792   5 weeks ago      193MB
eggdrop                   latest    d71b31bbccd5   6 weeks ago      37.5MB
hello-world               latest    ee301c921b8a   16 months ago    9.14kB
express-gateway           latest    c9a331ed66d4   22 months ago    133MB
gaukhar@Gaukhars-MacBook-Pro multistage %
```

Exercise 4: Pushing Docker Images to Docker Hub



```
gaukhar@Gaukhars-MacBook-Pro web-hw-1 % docker tag hello-docker vim2357/hello-docker
gaukhar@Gaukhars-MacBook-Pro web-hw-1 % docker login
Authenticating with existing credentials...
Login Succeeded
gaukhar@Gaukhars-MacBook-Pro web-hw-1 % docker push vim2357/hello-docker
Using default tag: latest
The push refers to repository [docker.io/vim2357/hello-docker]
e3ed3eaa26d3: Pushed
1bd9ea1cf4cb: Pushed
4f87ac73ce8f: Mounted from library/python
d7486c28114f: Mounted from library/python
50f7fbe612d1: Mounted from library/python
e644ff0c302d: Mounted from library/python
latest: digest: sha256:d7dbab6836cd741152a0f6494eb09e10be4693c929fab92d4439318c6f574052 size: 1572
gaukhar@Gaukhars-MacBook-Pro web-hw-1 %
```


hub.docker.com
New More Docker. Easy Access. New Streamlined Plans. Learn more. →
docker hub
Explore
Repositories
Organizations
Usage
Search Docker Hub
vim2357
Search by repository name
All Content
Create repository
vim2357 / hello-docker
Contains: Image · Last pushed: less than a minute ago
0
0
Public
Scout inactive
Create An Organization
Create and manage users and grant access to your repositories.
docker scout
Actionable insights for software supply chain management
Docker Scout generates insights across the software supply chain that inform improved development workflows. Developers are guided by contextual

Answers:

1) Docker Hub is a cloud-based registry where you can store, share, and manage Docker images. It allows you to pull images for your projects and push your own images for others to use.
2) I tagged a Docker image using the command: **docker tag hello-docker vim2357/hello-docker.**
3) To push an image, I followed these steps:

- Logged in to Docker Hub: **docker login**.
- Tagged.
- Pushed the image: **docker push vim2357/hello-docker**