Week-4 Assignment

1. Introduction to containerization and Docker fundamentals, Basic Commands.

    - Containerization is a technology used to package and run applications along with all their dependencies (like libraries, configuration files, etc) in an isolated environment called a container.
      Because of containerization:
    - Avoids "It works on my machine" problem.
    - Makes applications **portable** across systems.
    - Ensures **consistency** from development to production.
    - Lightweight compared to traditional virtual machines.

    - **Docker** is a popular platform for containerization. It allows developers to automate the deployment of applications in lightweight, portable containers.

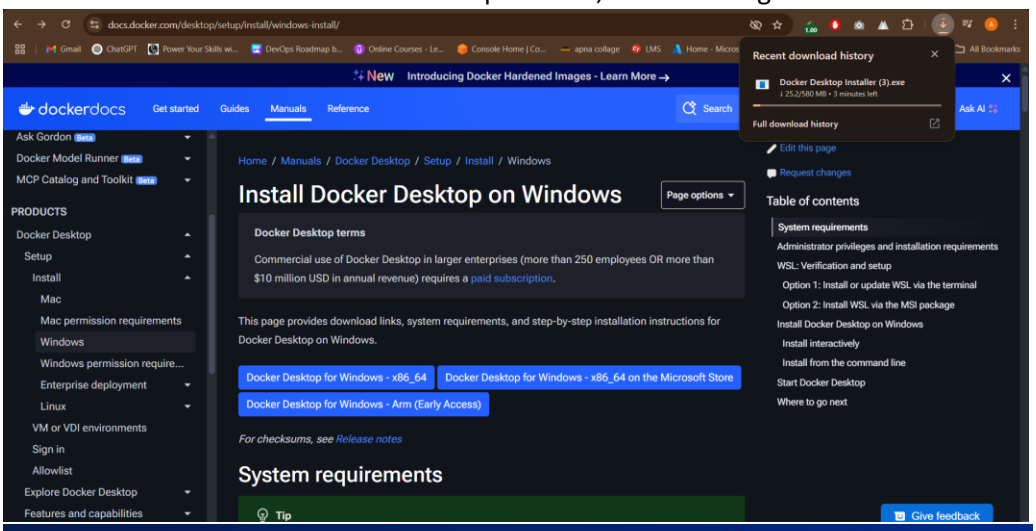
    - **Key Terms in Docker:**

      Image- Blueprint or snapshot used to create containers.
      Container- Running instance of an image.
      Dockerfile- A script with instructions to build a Docker image.
      Docker Hub- Public registry for storing and sharing Docker images.
      Volume- Used to persist data from a container.

- **Basic Docker Commands**
- docker --version
  → Check the installed Docker version.
- docker info
  → Displays system-wide information about Docker (containers, images, storage driver, etc.).
- docker pull <image-name>
  → Downloads an image from Docker Hub.
- docker images
  → Lists all downloaded images on your system.
- docker rmi <image-id or image-name>
  → Removes an image from your system.
- docker run <image-name>
  → Runs a container from an image.
- docker run -d <image-name>
  → Runs the container in **detached mode** (in the background).
- docker run -it ubuntu
  → Runs a container interactively with a terminal session.
- docker ps
  → Shows all **currently running containers**.

- docker ps -a
  → Shows **all containers**, including stopped ones.
- docker stop <container-id>
  → Stops a running container.
- docker rm <container-id>
  → Removes a container from your system.
- docker build -t <tag-name> .
  → Builds a Docker image from a Dockerfile in the current directory and tags it.
  docker run -p <host-port>:<container-port> <image-name>
  → Maps a container port to a port on the host.
- docker run -v <host-path>:<container-path> <image-name>
  → Mounts a host directory as a volume inside the container.

2. Docker installation and basic container operations, Build an image from Dockerfile.



Basic Container Operation:

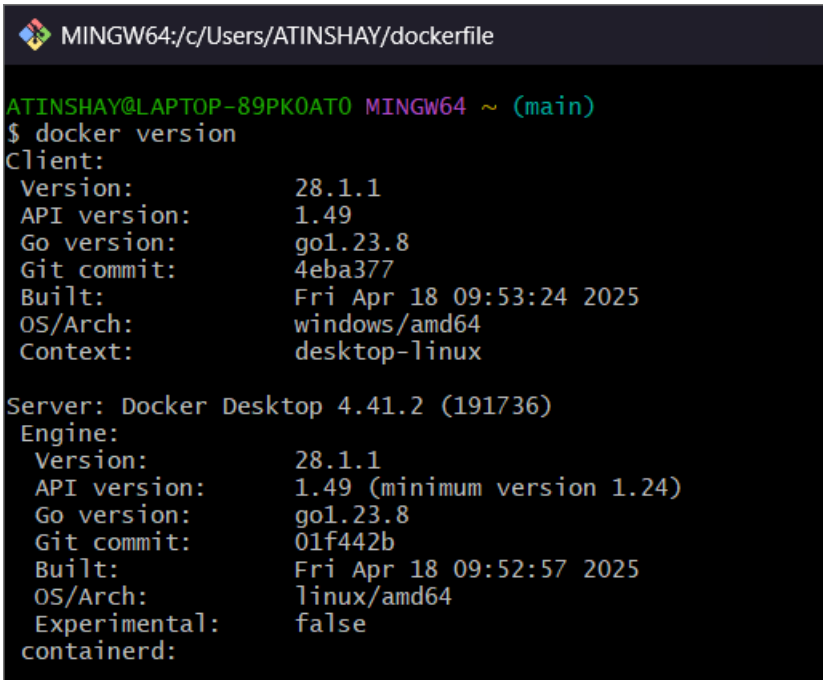
```
C:\Users\ATINSHAY>docker version
Client:
 Version:           28.1.1
 API version:       1.49
 Go version:        go1.23.8
 Git commit:        4eba377
 Built:             Fri Apr 18 09:53:24 2025
 OS/Arch:           windows/amd64
 Context:           desktop-linux

Server: Docker Desktop 4.41.2 (191736)
 Engine:
  Version:          28.1.1
  API version:      1.49 (minimum version 1.24)
  Go version:       go1.23.8
  Git commit:       01f442b
  Built:            Fri Apr 18 09:52:57 2025
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.7.27
  GitCommit:        05044ec0a9a75232cad458027ca83437aae3f4da
 runc:
  Version:          1.2.5
  GitCommit:        v1.2.5-0-g59923ef
 docker-init:
  Version:          0.19.0
  GitCommit:        de40ad0

C:\Users\ATINSHAY>docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
hello-world   latest    dd01f97f2521   5 months ago   20.4kB
```

```
C:\Users\ATINSHAY>docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
d9d352c11bbd: Pull complete
Digest: sha256:b59d21599a2b151e23eea5f6602f4af4d7d31c4e236d22bf0b62b86d2e386b8f
Status: Downloaded newer image for ubuntu:latest
docker.io/library/ubuntu:latest
```

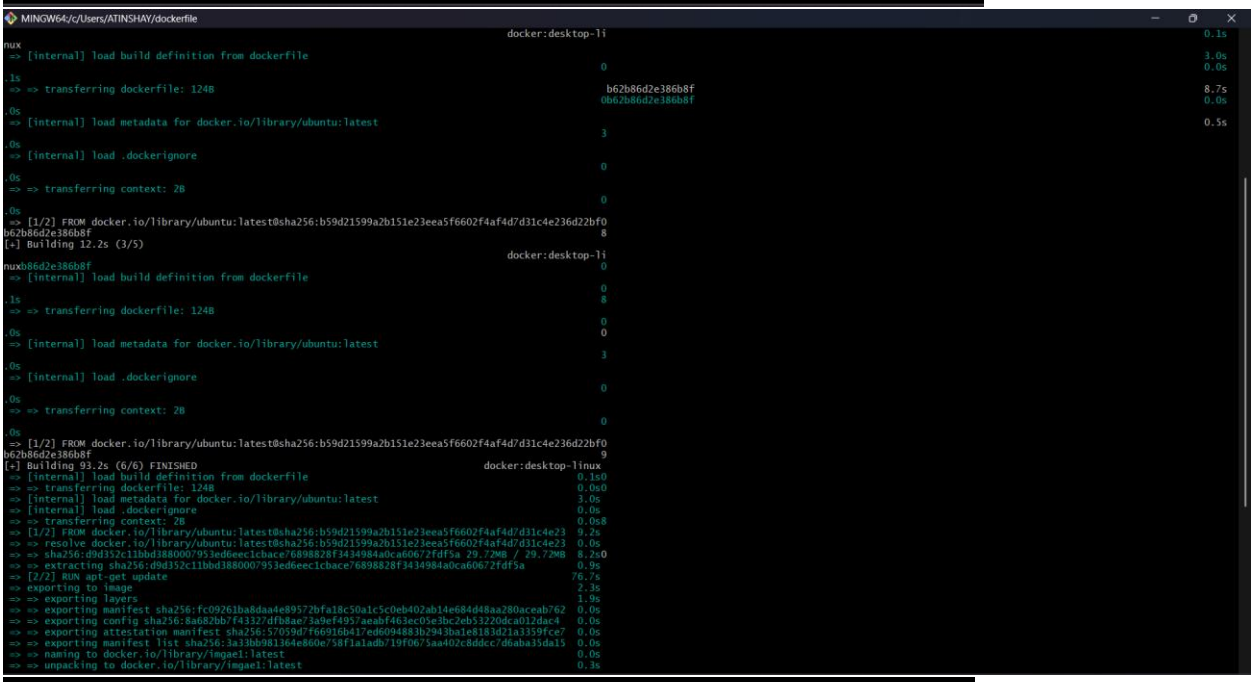
```
C:\Users\ATINSHAY>docker run -it ubuntu
root@0ee1fb832ecd:/#
root@0ee1fb832ecd:/#
root@0ee1fb832ecd:/# exit
exit

C:\Users\ATINSHAY>docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES

C:\Users\ATINSHAY>docker ps -a
CONTAINER ID   IMAGE     COMMAND       CREATED              STATUS                          PORTS     NAMES
0ee1fb832ecd   ubuntu    "/bin/bash"   21 seconds ago       Exited (0) 16 seconds ago                 optimistic_snyder
92819d2589ae   ubuntu    "/bin/bash"   About a minute ago   Exited (127) 34 seconds ago               condescending_wu
```

```
C:\Users\ATINSHAY>docker stop 0ee1fb832ecd
0ee1fb832ecd

C:\Users\ATINSHAY>docker ps -a
CONTAINER ID   IMAGE     COMMAND       CREATED              STATUS                          PORTS     NAMES
0ee1fb832ecd   ubuntu    "/bin/bash"   48 seconds ago       Exited (0) 44 seconds ago                 optimistic_snyder
92819d2589ae   ubuntu    "/bin/bash"   About a minute ago   Exited (127) About a minute ago           condescending_wu

C:\Users\ATINSHAY>docker rm 0ee1fb832ecd
0ee1fb832ecd

C:\Users\ATINSHAY>docker rm 92819d2589ae
92819d2589ae

C:\Users\ATINSHAY>docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES

C:\Users\ATINSHAY>docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
ubuntu        latest    b59d21599a2b   4 weeks ago    117MB
hello-world   latest    dd01f97f2521   5 months ago   20.4kB

C:\Users\ATINSHAY>docker rmi b59d21599a2b dd01f97f2521
Untagged: ubuntu:latest
Deleted: sha256:b59d21599a2b151e23eea5f6602f4af4d7d31c4e236d22bf0b62b86d2e386b8f
Untagged: hello-world:latest
Deleted: sha256:dd01f97f252193ae3210da231b1dca0cffab4aadb3566692d6730bf93f123a48

C:\Users\ATINSHAY>docker images
REPOSITORY   TAG       IMAGE ID   CREATED   SIZE

C:\Users\ATINSHAY>
```

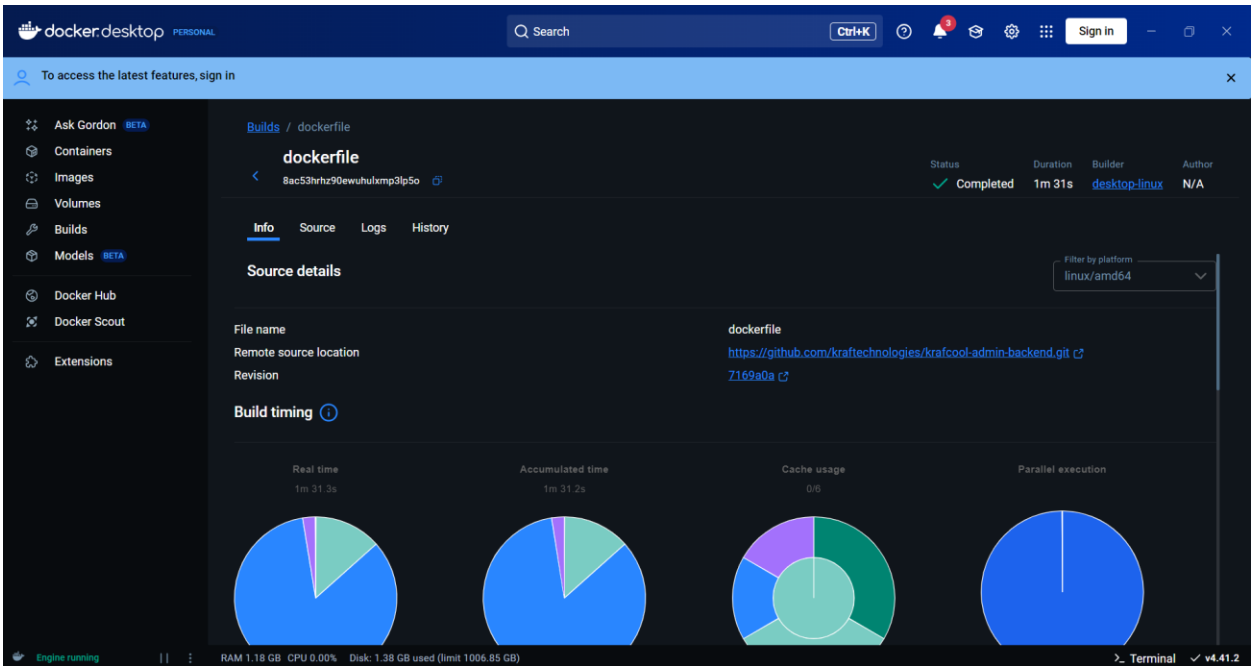
Build an image from dockerfile.



```
ATINSHAY@LAPTOP-89PKOAT0 MINGW64 ~/dockerfile (main)
$ ls
dockerfile

ATINSHAY@LAPTOP-89PKOAT0 MINGW64 ~/dockerfile (main)
$ cat dockerfile
FROM ubuntu
RUN apt-get update
CMD ["echo","hello world from my first image......"]

ATINSHAY@LAPTOP-89PKOAT0 MINGW64 ~/dockerfile (main)
$ ^C

ATINSHAY@LAPTOP-89PKOAT0 MINGW64 ~/dockerfile (main)
$ docker build -t imgae1:latest .
```




```
ATINSHAY@LAPTOP-89PKOAT0 MINGW64 ~/dockerfile (main)
$ docker images
REPOSITORY   TAG       IMAGE ID       CREATED          SIZE
imgae1       latest    3a33bb981364   4 minutes ago    202MB

ATINSHAY@LAPTOP-89PKOAT0 MINGW64 ~/dockerfile (main)
$
```




```
ATINSHAY@LAPTOP-89PKOAT0 MINGW64 ~/dockerfile (main)
$ docker run 3a33bb981364
hello world from my first image......
```

3. Docker Registry, DockerHub, Create a Multi-Stage Build.

- A **Docker Registry** is a storage and distribution system for Docker images.
- It allows users to **push** (upload) and **pull** (download) container images.

Types of Registries:

- **Public Registry:** Anyone can access it (e.g., Docker Hub).
- **Private Registry:** Hosted within your organization for secure access.

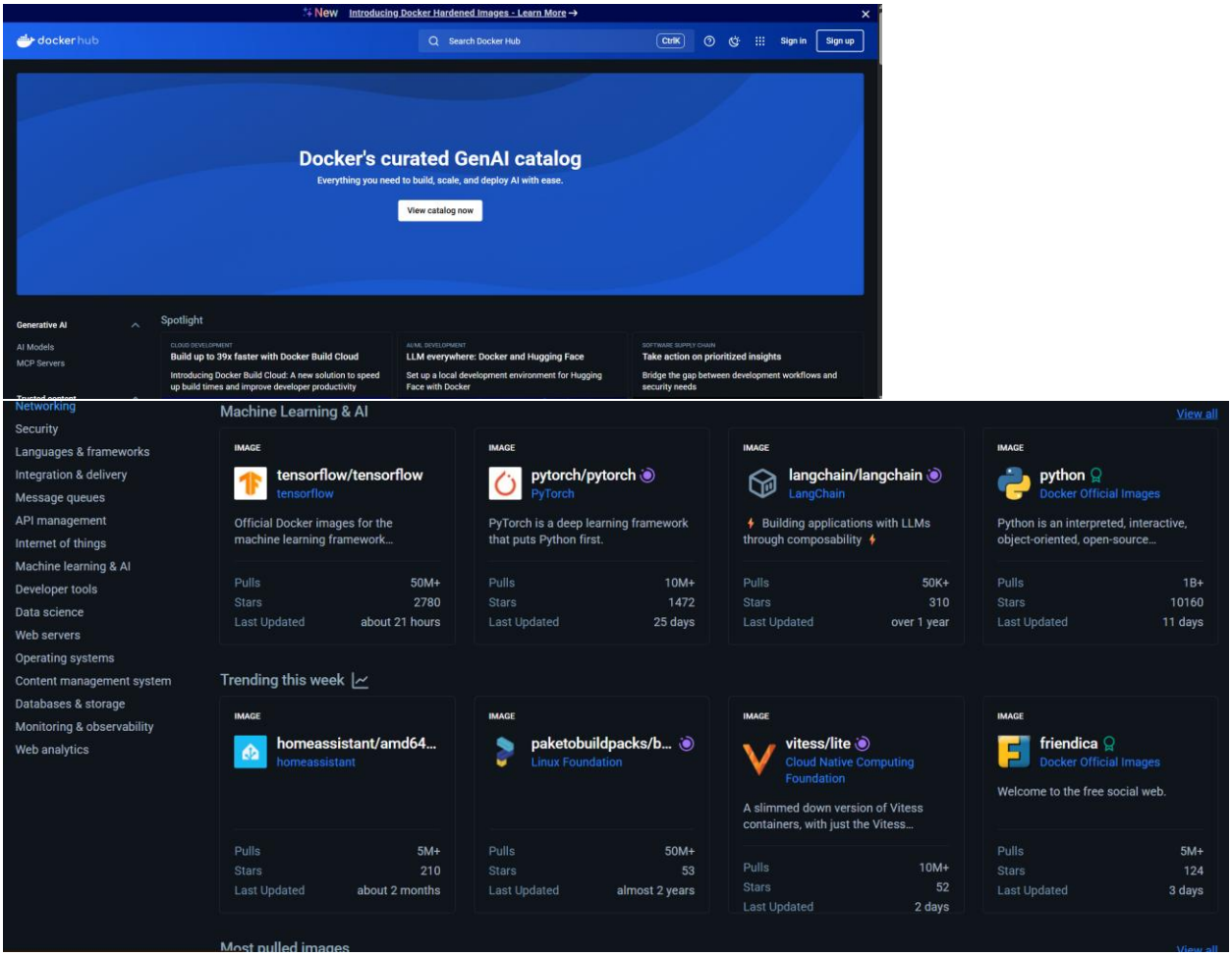
DockerHub

- It is the **default public Docker registry**.
- It hosts millions of public images.
- You can also create **private repositories**.

Features:

- Image hosting
- Versioning with tags

- User authentication
- Web UI to browse and manage images



Create a Multi-Stage Build:

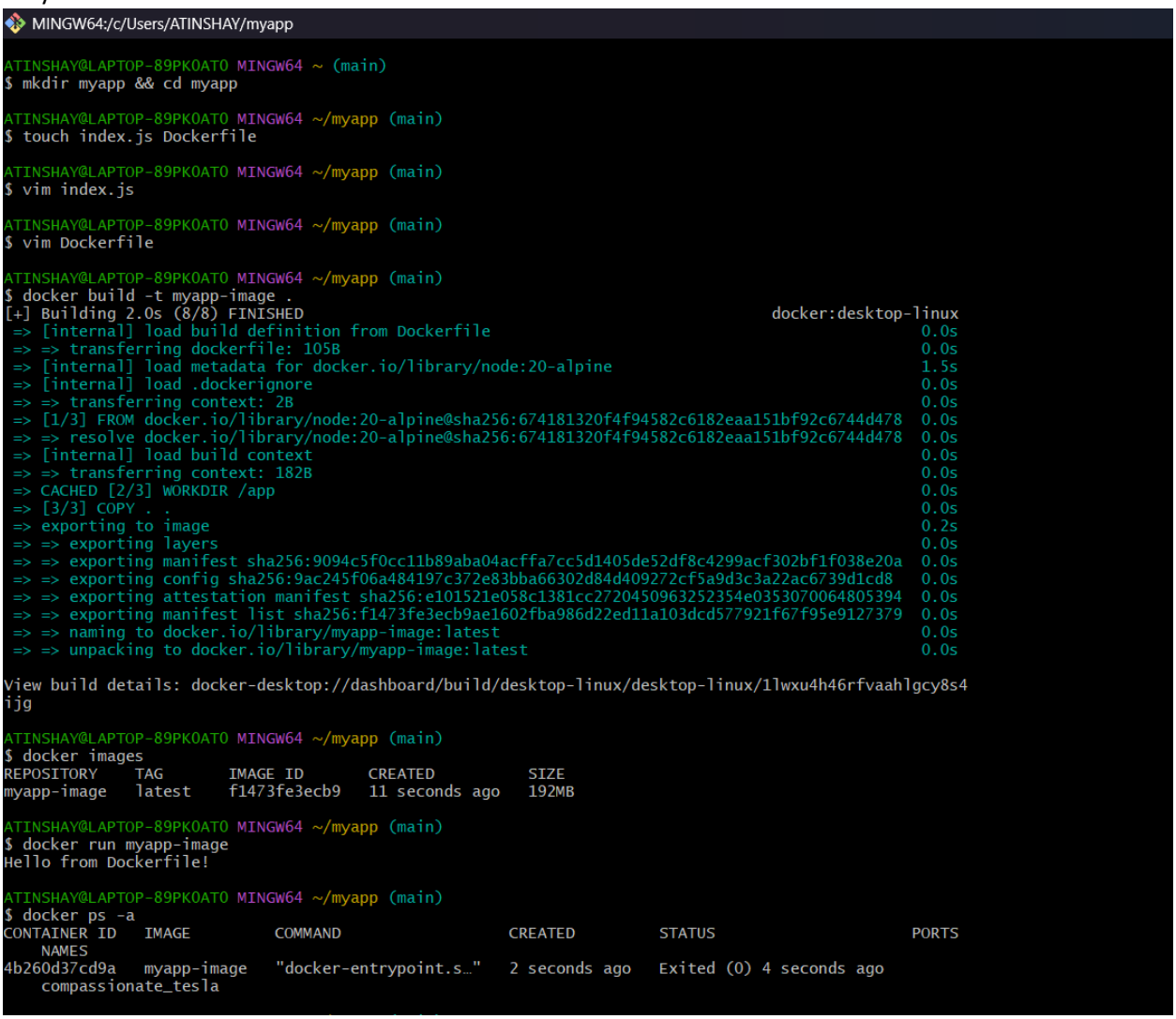
With multi-stage builds, you use multiple FROM statements in your Dockerfile.
Each FROM instruction can use a different base, and each of them begins a new stage of the build. You can selectively copy artifacts from one stage to another, leaving behind everything you don't want in the final image.

```
MINGW64:/c/Users/ATINSHAY/multi-stage-app

ATINSHAY@LAPTOP-89PKOAT0 MINGW64 ~ (main)
$ mkdir multi-stage-app

ATINSHAY@LAPTOP-89PKOAT0 MINGW64 ~ (main)
$ cd multi-stage-app

ATINSHAY@LAPTOP-89PKOAT0 MINGW64 ~/multi-stage-app (main)
$ touch Dockerfile package.json index.js

ATINSHAY@LAPTOP-89PKOAT0 MINGW64 ~/multi-stage-app (main)
$ vim index.js

ATINSHAY@LAPTOP-89PKOAT0 MINGW64 ~/multi-stage-app (main)
$ vim package.json

ATINSHAY@LAPTOP-89PKOAT0 MINGW64 ~/multi-stage-app (main)
$ vim Dockerfile

ATINSHAY@LAPTOP-89PKOAT0 MINGW64 ~/multi-stage-app (main)
$ cat index.js
console.log("Hello from a multi-stage Docker build!");
```

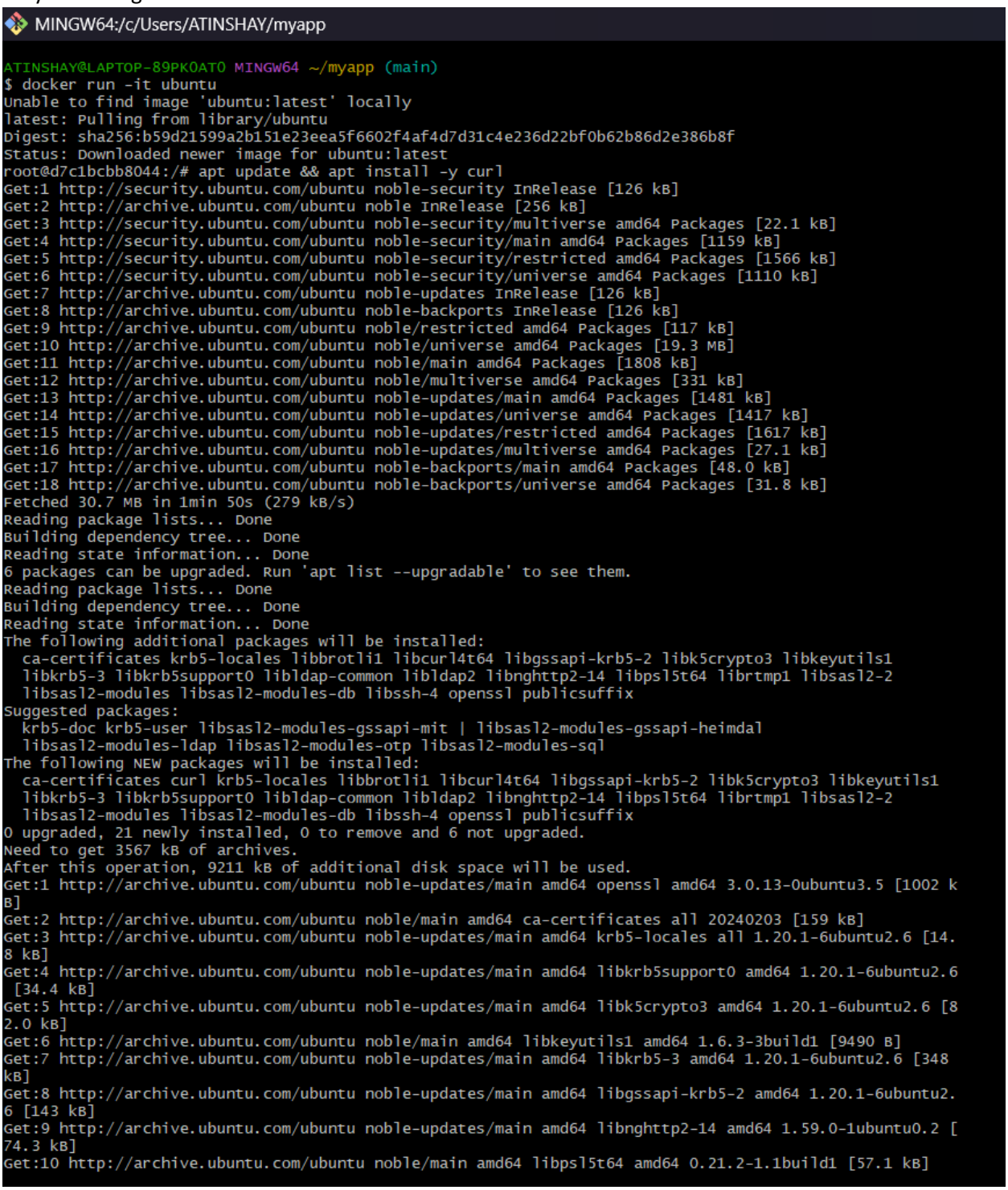
```
ATINSHAY@LAPTOP-89PKOAT0 MINGW64 ~/multi-stage-app (main)
$ docker build -t multi-stage-node .
[+] Building 21.2s (11/11) FINISHED                                    docker:desktop-linux
 => [internal] load build definition from Dockerfile                                   0.0s
 => => transferring dockerfile: 443B                                                   0.0s
 => WARN: FromAsCasing: 'as' and 'FROM' keywords' casing do not match (line 2)         0.0s
 => [internal] load metadata for docker.io/library/node:20-alpine                      3.1s
 => [internal] load .dockerignore                                                      0.0s
 => => transferring context: 2B                                                        0.0s
 => [builder 1/5] FROM docker.io/library/node:20-alpine@sha256:674181320f4f94582c6182  16.5s
 => => resolve docker.io/library/node:20-alpine@sha256:674181320f4f94582c6182eaa151bf  0.0s
 => => sha256:2506673f55362e86b6c8a2ab9c01541ae636887386c92d06e01286d3ddd83871 1.26MB  0.7s
 => => sha256:98c4889b578e94078411d6c14fe8f5daa0303d43e82bbf84d5787ab657c42428 445B /  0.8s
 => => sha256:fe07684b16b82247c3539ed86a65ff37a76138ec25d380bd80c869a1a4c73236 3.80MB  4.6s
 => => sha256:5432aa916e0868c8c9385ef60226d5ef530f13fe7c28fc13c054de1df6d006c 42.99MB  14.3s
 => => extracting sha256:fe07684b16b82247c3539ed86a65ff37a76138ec25d380bd80c869a1a4c73236  0.2s
 => => extracting sha256:5432aa916e0868c8c9385ef60226d5ef530f13fe7c28fc13c054de1df6d006cd  2.0s
 => => extracting sha256:2506673f55362e86b6c8a2ab9c01541ae636887386c92d06e01286d3ddd83871  0.0s
 => => extracting sha256:98c4889b578e94078411d6c14fe8f5daa0303d43e82bbf84d5787ab657c42428  0.0s
 => [internal] load build context                                                      0.0s
 => => transferring context: 726B                                                      0.0s
 => [builder 2/5] WORKDIR /app                                                         0.1s
 => [builder 3/5] COPY package.json .                                                  0.0s
 => [builder 4/5] RUN npm install                                                      0.8s
 => [builder 5/5] COPY . .                                                             0.0s
 => [stage-1 3/3] COPY --from=builder /app .                                           0.0s
 => exporting to image                                                                 0.2s
 => => exporting layers                                                                0.1s
 => => exporting manifest sha256:cee64fd4c7790d2e2bfb4e63f8ea4f279a6358419755877f54df4779ee5b3a7  0.0s
 => => exporting config sha256:821b740132922a47f6ee17864269ed4412b9973df1554c79087d2313af2f706e  0.0s
 => => exporting attestation manifest sha256:f60700cbb9c05e0d0aecda221b5fc27041566998b58d6490749  0.0s
 => => exporting manifest list sha256:c06e2d2c176bf716f9e7037a092f1585e729b7eba6b0771e1592e4c1a0  0.0s
 => => naming to docker.io/library/multi-stage-node:latest                             0.0s
 => => unpacking to docker.io/library/multi-stage-node:latest                          0.0s

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/nzpw86e8egsvqg7uuhzb3t
nan

 1 warning found (use docker --debug to expand):
 - FromAsCasing: 'as' and 'FROM' keywords' casing do not match (line 2)

ATINSHAY@LAPTOP-89PKOAT0 MINGW64 ~/multi-stage-app (main)
$ docker run multi-stage-node

> multi-stage-app@1.0.0 start
> node index.js

Hello from a multi-stage Docker build!

ATINSHAY@LAPTOP-89PKOAT0 MINGW64 ~/multi-stage-app (main)
$
```

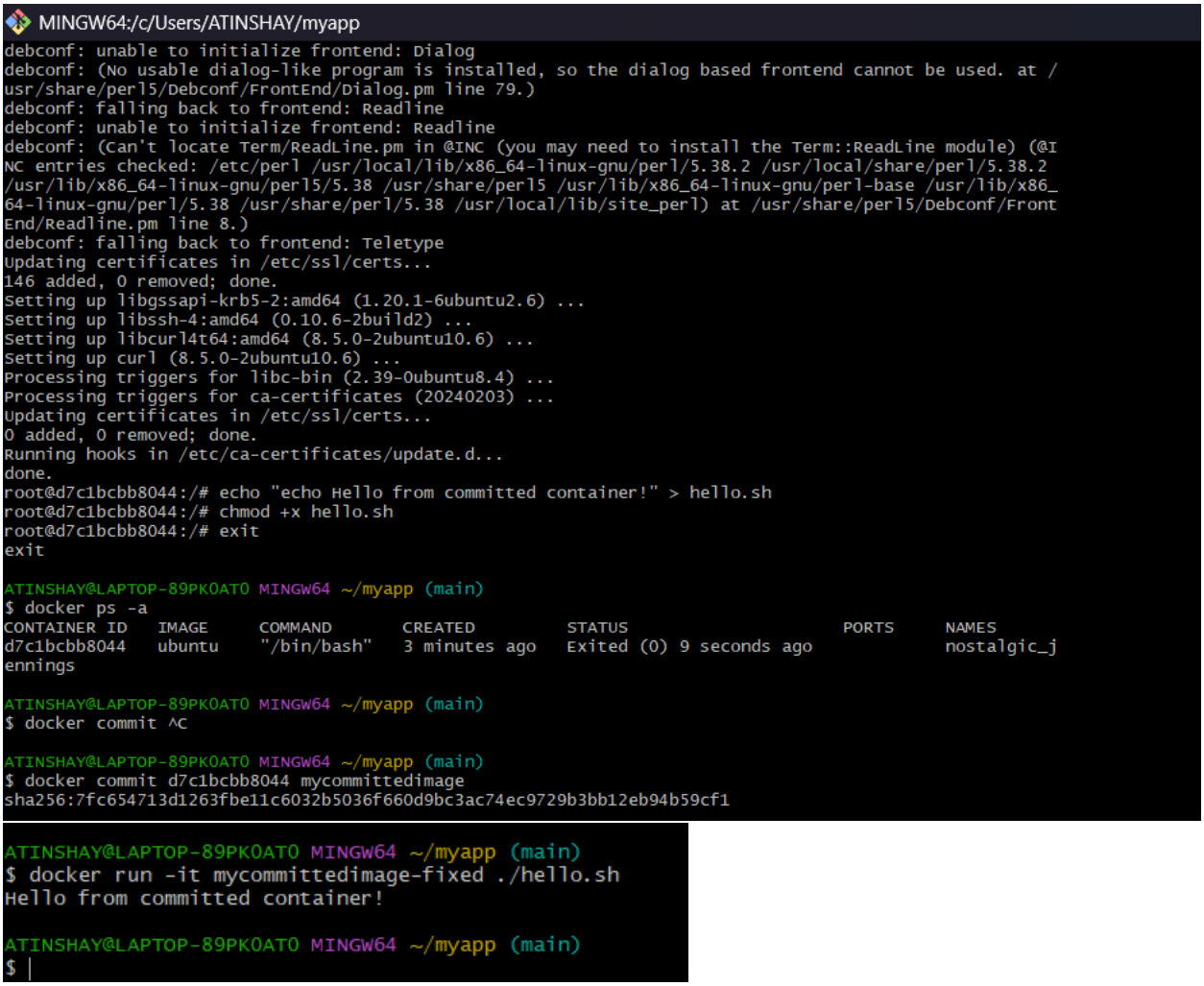
4. Create a docker image from multiple methods likes Dockerfile, running containers..

Way1: Dockerfile



Way2: running containers

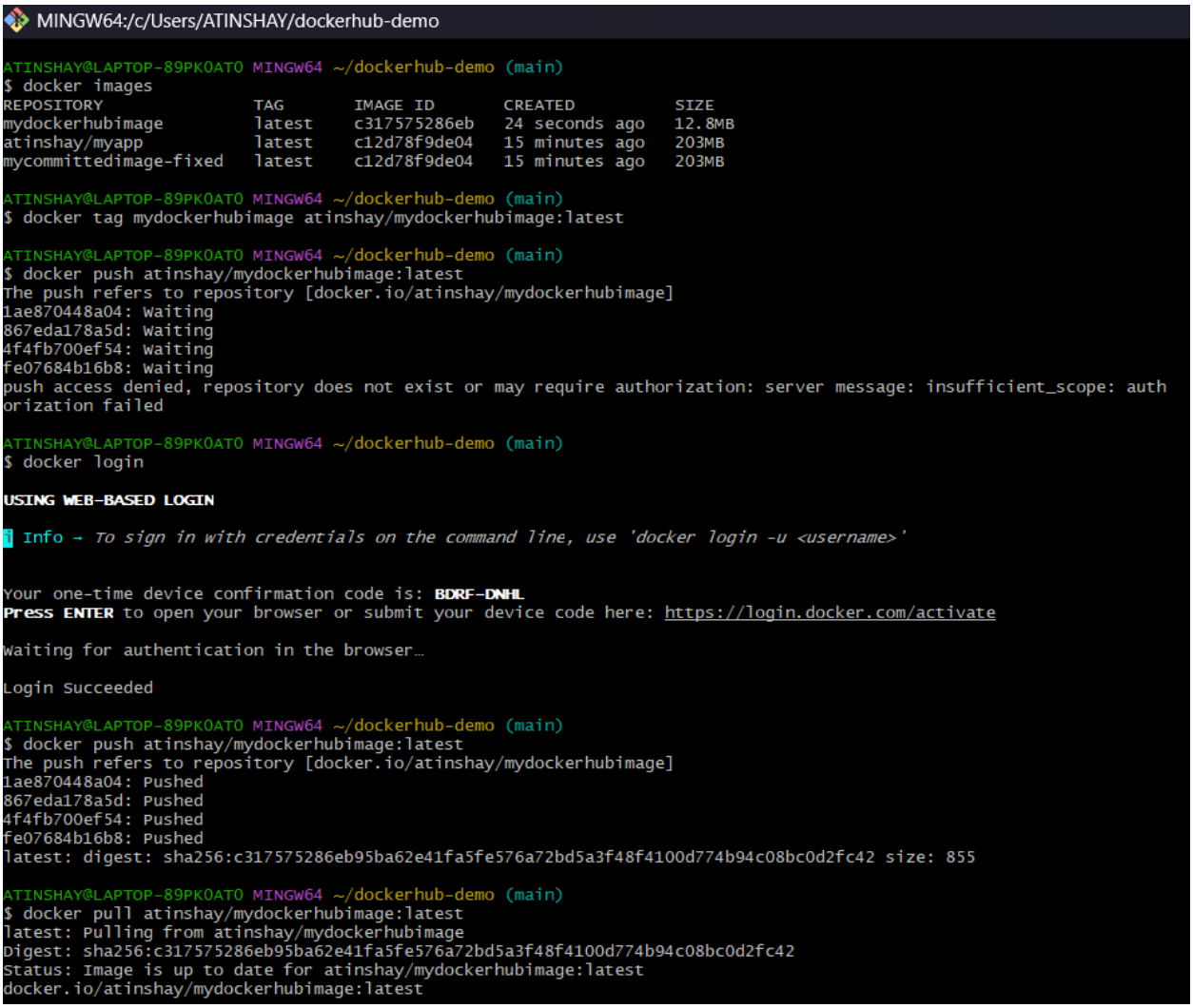


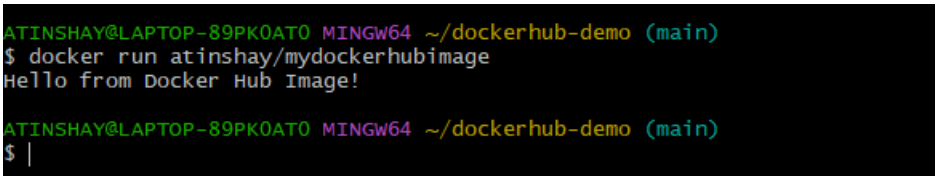

MINGW64:/c/Users/ATINSHAY/myapp
debconf: unable to initialize frontend: Dialog
debconf: (No usable dialog-like program is installed, so the dialog based frontend cannot be used. at /usr/share/perl5/Debconf/FrontEnd/Dialog.pm line 79.)
debconf: falling back to frontend: Readline
debconf: unable to initialize frontend: Readline
debconf: (Can't locate Term/ReadLine.pm in @INC (you may need to install the Term::ReadLine module) (@INC entries checked: /etc/perl /usr/local/lib/x86_64-linux-gnu/perl/5.38.2 /usr/local/share/perl/5.38.2 /usr/lib/x86_64-linux-gnu/perl5/5.38 /usr/share/perl5 /usr/lib/x86_64-linux-gnu/perl-base /usr/lib/x86_64-linux-gnu/perl/5.38 /usr/share/perl/5.38 /usr/local/lib/site_perl) at /usr/share/perl5/Debconf/FrontEnd/Readline.pm line 8.)
debconf: falling back to frontend: Teletype
Updating certificates in /etc/ssl/certs...
146 added, 0 removed; done.
Setting up libgssapi-krb5-2:amd64 (1.20.1-6ubuntu2.6) ...
Setting up libssh-4:amd64 (0.10.6-2build2) ...
Setting up libcurl4t64:amd64 (8.5.0-2ubuntu10.6) ...
Setting up curl (8.5.0-2ubuntu10.6) ...
Processing triggers for libc-bin (2.39-0ubuntu8.4) ...
Processing triggers for ca-certificates (20240203) ...
Updating certificates in /etc/ssl/certs...
0 added, 0 removed; done.
Running hooks in /etc/ca-certificates/update.d...
done.
root@d7c1bcbb8044:/# echo "echo Hello from committed container!" > hello.sh
root@d7c1bcbb8044:/# chmod +x hello.sh
root@d7c1bcbb8044:/# exit
exit
ATINSHAY@LAPTOP-89PK0AT0 MINGW64 ~/myapp (main)
$ docker ps -a
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
d7c1bcbb8044 ubuntu "/bin/bash" 3 minutes ago Exited (0) 9 seconds ago nostalgic_jennings
ATINSHAY@LAPTOP-89PK0AT0 MINGW64 ~/myapp (main)
$ docker commit ^C
ATINSHAY@LAPTOP-89PK0AT0 MINGW64 ~/myapp (main)
$ docker commit d7c1bcbb8044 mycommittedimage
sha256:7fc654713d1263fbe11c6032b5036f660d9bc3ac74ec9729b3bb12eb94b59cf1

ATINSHAY@LAPTOP-89PK0AT0 MINGW64 ~/myapp (main)
$ docker run -it mycommittedimage-fixed ./hello.sh
Hello from committed container!
ATINSHAY@LAPTOP-89PK0AT0 MINGW64 ~/myapp (main)
$

5. Push and pull image to Docker hub and ACR.

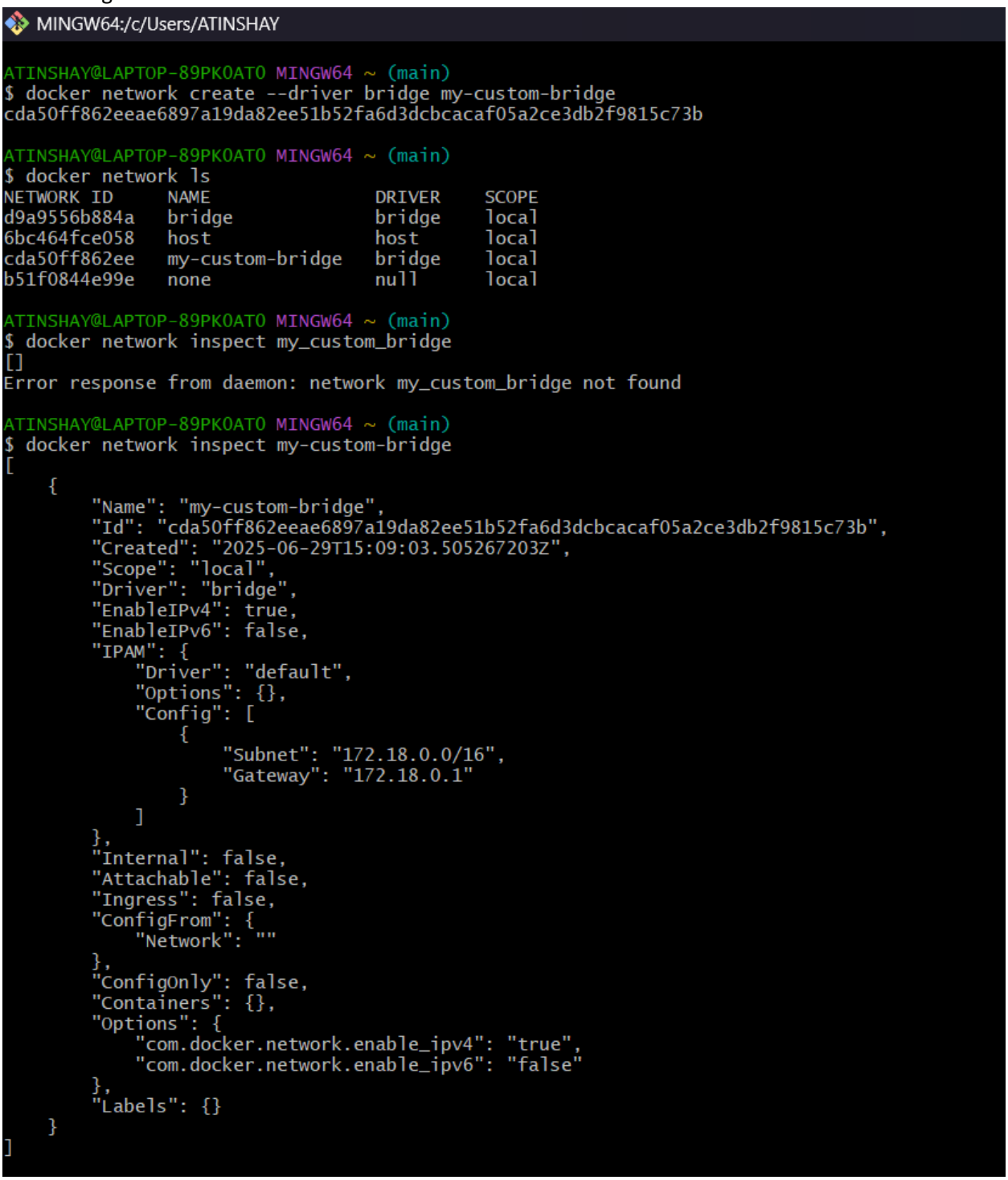
   Part1:





6. Create a Custom Docker Bridge Network.

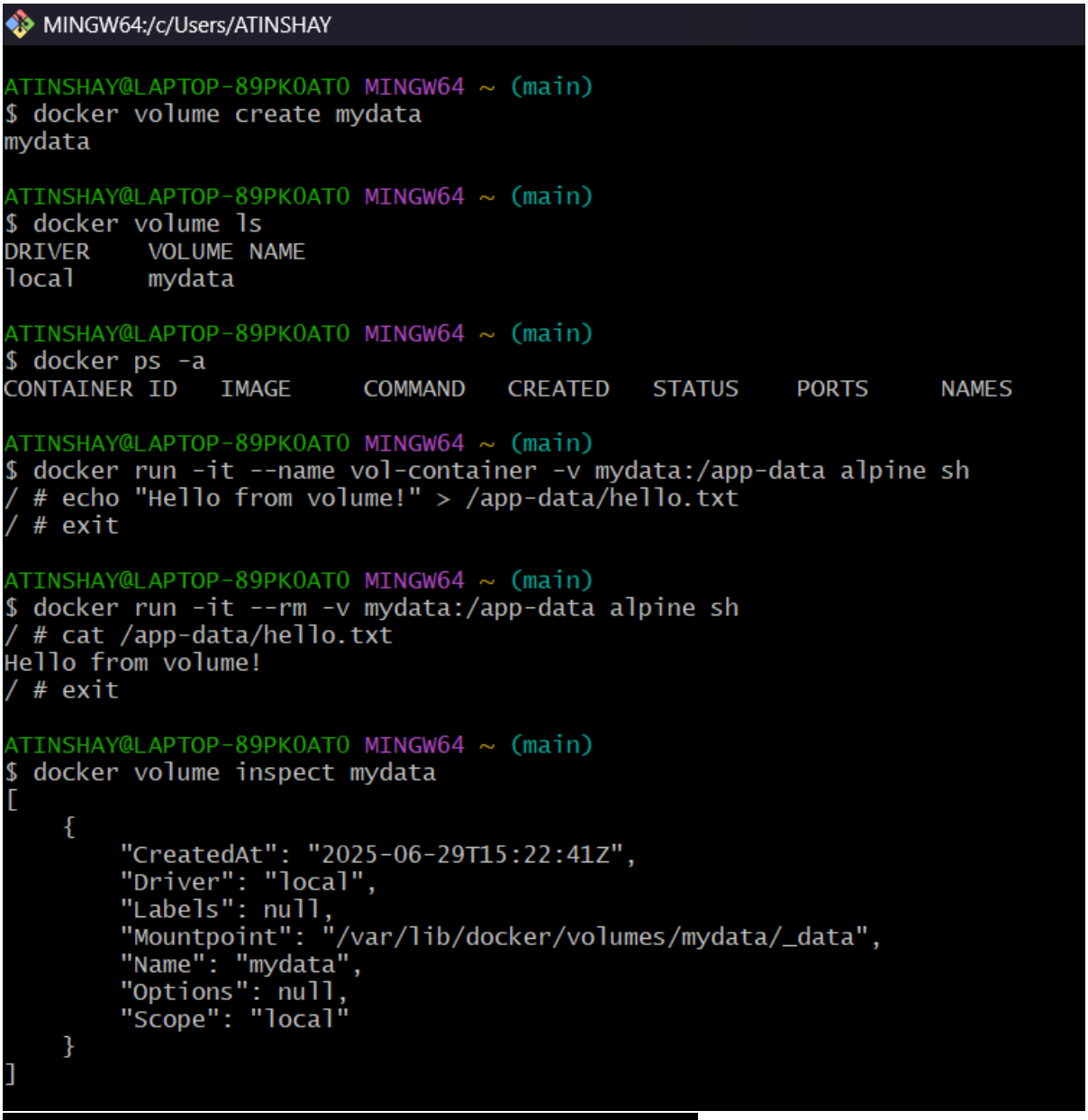
   A **Docker bridge network** allows containers to communicate with each other **by name (DNS)** while being isolated from external networks.

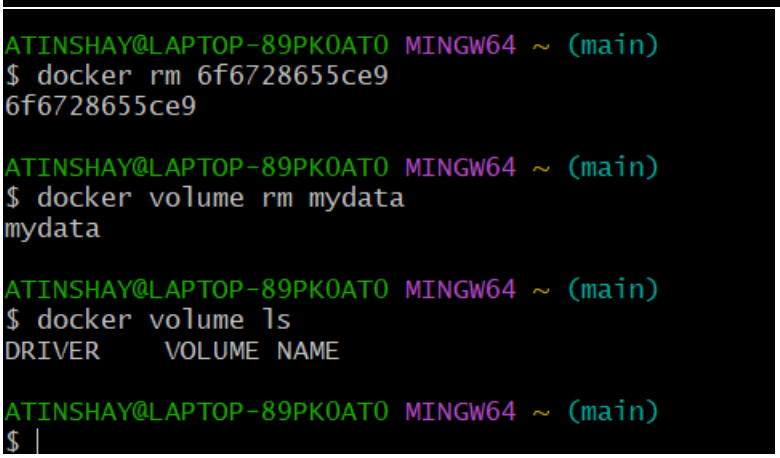


```
ATINSHAY@LAPTOP-89PK0AT0 MINGW64 ~ (main)
$ docker ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES

ATINSHAY@LAPTOP-89PK0AT0 MINGW64 ~ (main)
$ docker run -dit --name container1 --network my-custom-bridge alpine sh
0226fdeaf39e9e26f1f4039b0289d396563ef29e5c33293eb161d93a09baadc1

ATINSHAY@LAPTOP-89PK0AT0 MINGW64 ~ (main)
$ docker run -dit --name container2 --network my-custom-bridge alpine sh
91672ca56004e90f71b2f7e352d480f54abf43b25199dfcc7338710b60254113

ATINSHAY@LAPTOP-89PK0AT0 MINGW64 ~ (main)
$ docker exec -it container1 sh
/ # ping container2
PING container2 (172.18.0.3): 56 data bytes
64 bytes from 172.18.0.3: seq=0 ttl=64 time=0.181 ms
64 bytes from 172.18.0.3: seq=1 ttl=64 time=0.156 ms
64 bytes from 172.18.0.3: seq=2 ttl=64 time=0.101 ms
64 bytes from 172.18.0.3: seq=3 ttl=64 time=0.086 ms
64 bytes from 172.18.0.3: seq=4 ttl=64 time=0.162 ms
64 bytes from 172.18.0.3: seq=5 ttl=64 time=0.167 ms
64 bytes from 172.18.0.3: seq=6 ttl=64 time=0.120 ms
64 bytes from 172.18.0.3: seq=7 ttl=64 time=0.118 ms
64 bytes from 172.18.0.3: seq=8 ttl=64 time=0.118 ms
64 bytes from 172.18.0.3: seq=9 ttl=64 time=0.120 ms
64 bytes from 172.18.0.3: seq=10 ttl=64 time=0.125 ms
64 bytes from 172.18.0.3: seq=11 ttl=64 time=0.130 ms
64 bytes from 172.18.0.3: seq=12 ttl=64 time=0.111 ms
64 bytes from 172.18.0.3: seq=13 ttl=64 time=0.127 ms
64 bytes from 172.18.0.3: seq=14 ttl=64 time=0.154 ms
^C
--- container2 ping statistics ---
15 packets transmitted, 15 packets received, 0% packet loss
round-trip min/avg/max = 0.086/0.131/0.181 ms
/ # exit

ATINSHAY@LAPTOP-89PK0AT0 MINGW64 ~ (main)
$
```

7. Create a Docker volume and mount it to a container.
   A **volume** is a **persistent storage mechanism** used in Docker to:

- Store data **outside** the container's lifecycle.
- **Share data** between containers.
- Avoid losing data when a container is removed or restarted.


MINGW64:/c/Users/ATINSHAY
ATINSHAY@LAPTOP-89PKOAT0 MINGW64 ~ (main)
$ docker volume create mydata
mydata
ATINSHAY@LAPTOP-89PKOAT0 MINGW64 ~ (main)
$ docker volume ls
DRIVER VOLUME NAME
local mydata
ATINSHAY@LAPTOP-89PKOAT0 MINGW64 ~ (main)
$ docker ps -a
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
ATINSHAY@LAPTOP-89PKOAT0 MINGW64 ~ (main)
$ docker run -it --name vol-container -v mydata:/app-data alpine sh
/ # echo "Hello from volume!" > /app-data/hello.txt
/ # exit
ATINSHAY@LAPTOP-89PKOAT0 MINGW64 ~ (main)
$ docker run -it --rm -v mydata:/app-data alpine sh
/ # cat /app-data/hello.txt
Hello from volume!
/ # exit
ATINSHAY@LAPTOP-89PKOAT0 MINGW64 ~ (main)
$ docker volume inspect mydata
[
{
"CreatedAt": "2025-06-29T15:22:41Z",
"Driver": "local",
"Labels": null,
"Mountpoint": "/var/lib/docker/volumes/mydata/_data",
"Name": "mydata",
"Options": null,
"Scope": "local"
}
]


ATINSHAY@LAPTOP-89PKOAT0 MINGW64 ~ (main)
$ docker rm 6f6728655ce9
6f6728655ce9
ATINSHAY@LAPTOP-89PKOAT0 MINGW64 ~ (main)
$ docker volume rm mydata
mydata
ATINSHAY@LAPTOP-89PKOAT0 MINGW64 ~ (main)
$ docker volume ls
DRIVER VOLUME NAME
ATINSHAY@LAPTOP-89PKOAT0 MINGW64 ~ (main)
$

8. Docker Compose for multi-container applications, Docker security best practices.

- Docker Compose lets you **define and run multi-container Docker apps** using a single YAML file.
- Useful when you want to run services like **backend + database**, **frontend + backend**, etc., together.

```
MINGW64:/c/Users/ATINSHAY/my-compose-app

ATINSHAY@LAPTOP-89PKOAT0 MINGW64 ~ (main)
$ mkdir my-compose-app

ATINSHAY@LAPTOP-89PKOAT0 MINGW64 ~ (main)
$ cd my-compose-app

ATINSHAY@LAPTOP-89PKOAT0 MINGW64 ~/my-compose-app (main)
$ touch docker-compose.yml

ATINSHAY@LAPTOP-89PKOAT0 MINGW64 ~/my-compose-app (main)
$ mkdir app

ATINSHAY@LAPTOP-89PKOAT0 MINGW64 ~/my-compose-app (main)
$ cd app

ATINSHAY@LAPTOP-89PKOAT0 MINGW64 ~/my-compose-app/app (main)
$ touch app.py

ATINSHAY@LAPTOP-89PKOAT0 MINGW64 ~/my-compose-app/app (main)
$ cd .

ATINSHAY@LAPTOP-89PKOAT0 MINGW64 ~/my-compose-app/app (main)
$ cd . .
bash: cd: too many arguments

ATINSHAY@LAPTOP-89PKOAT0 MINGW64 ~/my-compose-app/app (main)
$ cd ..

ATINSHAY@LAPTOP-89PKOAT0 MINGW64 ~/my-compose-app (main)
$ mkdir db

ATINSHAY@LAPTOP-89PKOAT0 MINGW64 ~/my-compose-app (main)
$ cd app

ATINSHAY@LAPTOP-89PKOAT0 MINGW64 ~/my-compose-app/app (main)
$ vim app.py

ATINSHAY@LAPTOP-89PKOAT0 MINGW64 ~/my-compose-app/app (main)
$ cat app.py
from flask import Flask
app = Flask(__name__)

@app.route('/')
def home():
    return "Hello from Flask + MySQL using Docker Compose!"

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

```
ATINSHAY@LAPTOP-89PKOAT0 MINGW64 ~/my-compose-app (main)
$ docker-compose down
time="2025-06-29T21:16:10+05:30" level=warning msg="C:\\Users\\ATINSHAY\\my-compose-app\\dock
er-compose.yml: the attribute `version` is obsolete, it will be ignored, please remove it to
avoid potential confusion"

ATINSHAY@LAPTOP-89PKOAT0 MINGW64 ~/my-compose-app (main)
$
```

Docker Security Best Practices:

- Always use images from **verified publishers**.
- Avoid latest tag for critical services
- Scan Images for Vulnerabilities , use tools like **Trivy**, **Clair**, **Anchore**.
- Use smaller base images like alpine or distroless.
- Avoid installing unnecessary packages.
- Never mount /, /var, /etc, or sensitive files from the host.
- Use volumes with strict access if needed.
- Limit Container Capabilities.
- Regularly update Docker to fix vulnerabilities.
- Use Docker Secrets for Sensitive Data (in Swarm Mode)