

Assignment-Week 4

Task-3 Docker Registry, DockerHub, and Multi-Stage Build

Introduction

As part of modern software development, Docker plays a crucial role in packaging applications into containers. This document explores three advanced Docker topics: Docker Registry, DockerHub, and Multi-Stage Builds, which are essential for optimizing image management, storage, and build efficiency.

Docker Registry

A Docker Registry is a system for storing and distributing Docker images. It allows developers to share and manage container images securely.

- ❖ Acts as a centralized storage for Docker images
- ❖ Can be public or private
- ❖ Examples: DockerHub, GitHub Container Registry, Google Container Registry

Self-hosted Registry:

You can run your own registry using Docker

```
$ docker run -d -p 5000:5000 --name registry registry:2
```

```
ajayi@Ajay MINGW64 ~  
$ docker run -d -p 5000:5000 --name myregistry registry:2  
Unable to find image 'registry:2' locally  
2: Pulling from library/registry  
44cf07d57ee4: Pull complete  
bbbdd6c6894b: Pull complete  
8e82f80af0de: Pull complete  
3493bf46cdec: Pull complete  
6d464ea18732: Pull complete  
Digest: sha256:a3d8aaa63ed8681a604f1dea0aa03f100d5895b6a58ace528858a7b332415373  
Status: Downloaded newer image for registry:2  
7529f278c8731d40839b033acffb5158deefeb2de2e85fcc83a099692c9b7afe
```

DockerHub

DockerHub is the default and most widely used Docker registry provided by Docker Inc.

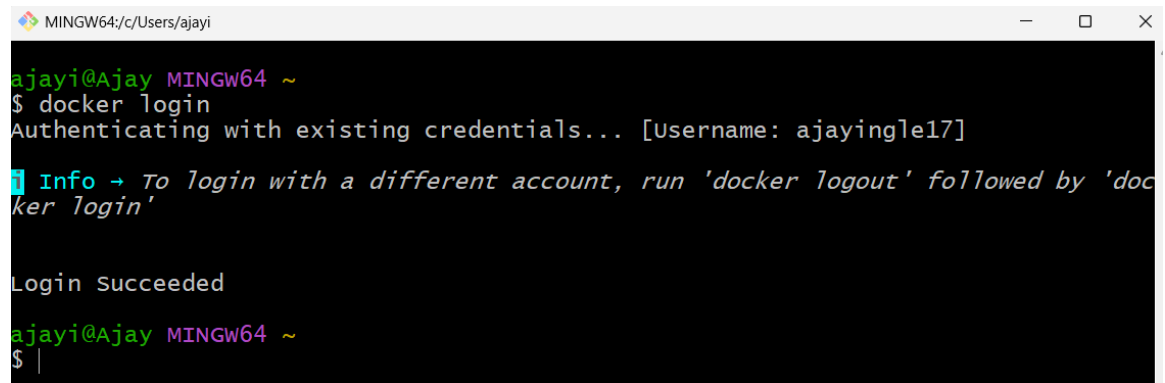
Features of DockerHub:

- ❖ Hosts official images (e.g., ubuntu, nginx)
- ❖ Allows developers to push and pull public/private images
- ❖ Integration with CI/CD tools

Basic Commands:

1. Login:

```
$ docker login
```

A terminal window titled 'MINGW64:/c/Users/ajayi' shows the execution of 'docker login'. The prompt is 'ajayi@Ajay MINGW64 ~'. The command '\$ docker login' is entered, followed by the message 'Authenticating with existing credentials... [Username: ajayingle17]'. A blue information box appears with the text: 'Info → To login with a different account, run 'docker logout' followed by 'docker login''. Below this, it says 'Login Succeeded'. The prompt returns to 'ajayi@Ajay MINGW64 ~' with a '\$' character and a cursor.

```
ajayi@Ajay MINGW64 ~
$ docker login
Authenticating with existing credentials... [Username: ajayingle17]

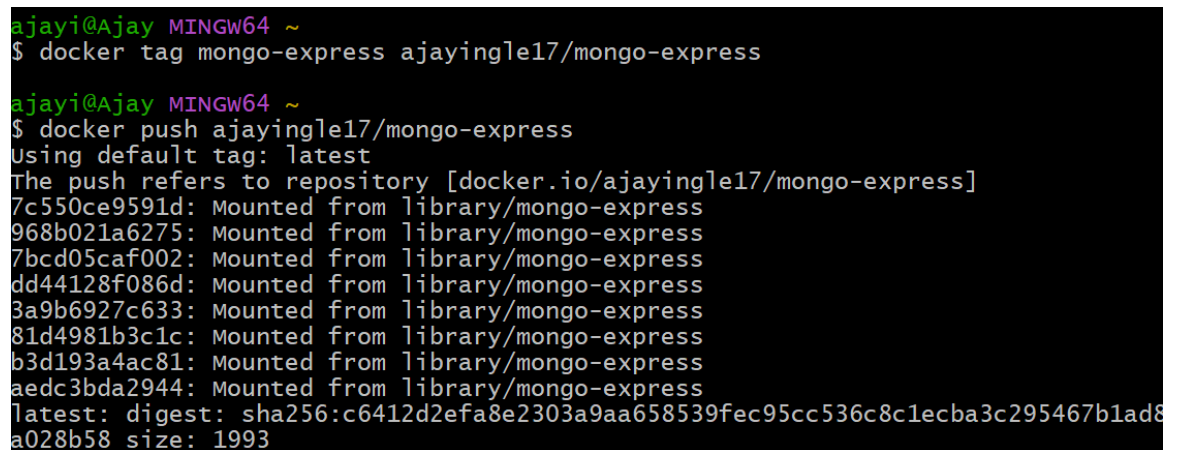
Info → To login with a different account, run 'docker logout' followed by 'docker login'

Login Succeeded
ajayi@Ajay MINGW64 ~
$ |
```

2. Push an image:

```
$ docker tag mongo-express ajayingle17/mongo-express
```

```
$ docker push ajayingle17/mongo-express
```

A terminal window titled 'MINGW64:/c/Users/ajayi' shows the execution of 'docker tag' and 'docker push'. The prompt is 'ajayi@Ajay MINGW64 ~'. The command '\$ docker tag mongo-express ajayingle17/mongo-express' is entered. Then, '\$ docker push ajayingle17/mongo-express' is entered, followed by 'Using default tag: latest' and 'The push refers to repository [docker.io/ajayingle17/mongo-express]'. A list of image IDs is shown, each followed by 'Mounted from library/mongo-express'. The list ends with 'latest: digest: sha256:c6412d2efa8e2303a9aa658539fec95cc536c8c1ecba3c295467b1ad8a028b58 size: 1993'.

```
ajayi@Ajay MINGW64 ~
$ docker tag mongo-express ajayingle17/mongo-express

ajayi@Ajay MINGW64 ~
$ docker push ajayingle17/mongo-express
Using default tag: latest
The push refers to repository [docker.io/ajayingle17/mongo-express]
7c550ce9591d: Mounted from library/mongo-express
968b021a6275: Mounted from library/mongo-express
7bcd05caf002: Mounted from library/mongo-express
dd44128f086d: Mounted from library/mongo-express
3a9b6927c633: Mounted from library/mongo-express
81d4981b3c1c: Mounted from library/mongo-express
b3d193a4ac81: Mounted from library/mongo-express
aedc3bda2944: Mounted from library/mongo-express
latest: digest: sha256:c6412d2efa8e2303a9aa658539fec95cc536c8c1ecba3c295467b1ad8a028b58 size: 1993
```

3. Pull an image:

```
$ docker pull nginx
```

Multi-Stage Build

A Multi-Stage Build allows you to use multiple `FROM` statements in a Dockerfile to optimize the build process by reducing final image size.

Why use Multi-Stage Builds?

- ❖ Keeps final image lightweight
- ❖ Avoids including development tools and intermediate files
- ❖ Improves security and performance

Example Dockerfile:

```
``Dockerfile

# Stage 1: Build

FROM node:18 AS builder

WORKDIR /app

COPY . .

RUN npm install && npm run build


# Stage 2: Serve

FROM nginx:alpine

COPY --from=builder /app/build /usr/share/nginx/html

``
```

```

MINGW64/d/Web Development/react-projects/todo-application
[+] Building 18.0s (6/13)
=> [internal] load build definition from Dockerfile
[+] Building 18.1s (6/13)
                                docker:desktop-linux:18
=> [internal] load build definition from Dockerfile
                                0.1s
=> => transferring dockerfile: 225B
                                0.0s
=> [internal] load metadata for docker.io/library/node:18
                                5.2s6:c6ae79e38498325db67193d3
=> [internal] load metadata for docker.io/library/nginx:alpine
                                5.2s9addb13b7de3c1f11bdee6b9 2
=> [auth] library/nginx:pull token for registry-1.docker.io
                                0.0s0efdbee159f29638783778c0 1
=> [auth] library/node:pull token for registry-1.docker.io
                                0.0sb82108adcf0605a200294964 1
=> [internal] load .dockerignore
                                0.0sf5cd2d014180e4d3684d34ab 1
=> => transferring context: 2B
                                0.0s
=> [builder 1/4] FROM docker.io/library/node:18@sha256:c6ae79e38498325db67193d3
91e6ec1d224d96c693a8a4d943498556716d3783
                                12.9s2e814d28359e77bd0aa5fed193

```

In this example, the Node.js app is built in the first stage and only the final build output is copied to the Nginx container for serving.

Conclusion

Docker Registry and DockerHub are essential for storing and sharing container images. Multi-Stage Builds help optimize image size and security by separating build and runtime environments. Mastering these concepts is critical for efficient DevOps workflows and modern software deployment.