

Assignment- Week 4

Task-1: Push and Pull Image to Docker Hub and Azure Container Registry (ACR)

Docker images can be pushed to remote registries to enable sharing and deployment. Docker Hub is the default public registry, while Azure Container Registry (ACR) is a private registry hosted in Azure. Pushing involves tagging the image appropriately and authenticating to the registry. Pulling retrieves the image from the registry to a local system.

```
ujjwal@ubuntu:~$ docker login
Username: ujjwalnimbhokar
Password: *****
Login Succeeded

ujjwal@ubuntu:~$ docker tag myapp:latest ujjwalnimbhokar/myapp:latest
ujjwal@ubuntu:~$ docker push ujjwalnimbhokar/myapp:latest
The push refers to repository [docker.io/ujjwalnimbhokar/myapp]
latest: digest: sha256:xyz... size: 1784
Pushed
```

```
ujjwal@ubuntu:~$ docker pull ujjwalnimbhokar/myapp:latest
latest: Pulling from ujjwalnimbhokar/myapp
Digest: sha256:xyz...
Status: Downloaded newer image
```

Azure Container Registry

```
ujjwal@ubuntu:~$ docker pull ujjwalacr.azurecr.io/myapp:latest
latest: Pulling from myapp
Status: Downloaded newer image
```

```
ujjwal@ubuntu:~$ az acr login --name ujjwalacr
Login Succeeded

ujjwal@ubuntu:~$ docker tag myapp:latest ujjwalacr.azurecr.io/myapp:latest
ujjwal@ubuntu:~$ docker push ujjwalacr.azurecr.io/myapp:latest
The push refers to repository [ujjwalacr.azurecr.io/myapp]
latest: digest: sha256:xyz... size: 1846
Pushed
```

Task-2 : Create a Custom Docker Bridge Network

A Docker bridge network allows containers to communicate with each other in isolation from the host network. This custom network provides DNS-based service discovery among containers and facilitates secure communication in microservices-based architecture. It is useful for linking services like web and database containers.

```
ujjwal@ubuntu:~$ docker network create --driver bridge ujjwal_bridge
ujjwal@ubuntu:~$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
d69bc...            ujjwal_bridge       bridge              local

ujjwal@ubuntu:~$ docker run -dit --name c1 --network ujjwal_bridge alpine
ujjwal@ubuntu:~$ docker run -dit --name c2 --network ujjwal_bridge alpine

ujjwal@ubuntu:~$ docker network inspect ujjwal_bridge
```

Task-3: Create a Docker Volume and Mount It to a Container

Docker volumes are persistent storage mechanisms that survive container restarts or removal. Mounting a volume to a container enables data to be stored outside of the container filesystem. This is ideal for databases, logs, and user-generated content. Volumes are managed by Docker and can be reused between containers.

```
ajayi@Ajay MINGW64 ~
$ docker volume create my_volume
my_volume

ajayi@Ajay MINGW64 ~
$ docker volume ls
DRIVER      VOLUME NAME
local       2bf70e36e4a3f226bc8dab31d5449f69f96abcfb89d05a86e874d0d9ec3def8f
local       8b2d153ecd0408bf557d40fe17fd4c81db55ecd6e57bb5ccc578f436da321157
local       8f26f99ebf706ddca92809a0ee250710dcb11247ca0969833ef26d8761b137b8
local       9c1c52afa2dded5a2dd0ca6817f1ee61e80273fb76659817095d852c28ba420a
local       031c2958051af6f03152edd72525475dd755719eae48f0834839756b173eb73
local       054c21400c07601da831faeecd277ad3b8e182fe944ab6953ac6eb5d5a8068bb
local       59cd6087970be38d034994ff45df1622cb8e42968024fce114145431558b2de4
local       570f6124c8a478caab0812b43d91a6d5d64547f7ca1fd5317e696760a3fb03b4
local       5524c0162a55078c618eccddf13d9323c5353f52c8d42c4fb08be2600f84bfe7
local       604533d629594b9fe1585ff8ffa88d64bf2c78b4432c6e74310e147ba34ec1fa
local       9556854fedd5c34616060be2775a65ac22d46827da25f92448f0270822de5bcf
local       b8a78bdb49a29b8924a8326b5b17f51c59a64f030de6cfc065f9d1446ea19e04
local       c01a91f7f239a1a8d50ccd3c22fb9aa8157e99cf49d2af1a7bbaa8600cfd9e96
local       d4af4bbaece922122e11f0f14d8fa63f4bcb118ade2928cea7f345296ccc91eb
local       d6df1c1fef1dc27dd24195f5707c119f4f7f012448c630b6807f8474c98db727
local       d119d3a187ccb797840f7aa99280d0f3fa53910151871a97e1399154aeb10e94
local       db6b4a486c6f158c2e6d21b8d4a47fbb11a208b41215183db3054f400b515af3
local       f684c190dc2ee84a1f9b24ee2efeca7698762dba67bbbe818285f1b346e30200
local       my_volume

ajayi@Ajay MINGW64 ~
$ |
```

Task-4 Docker Compose and Security Best Practices

Docker Compose simplifies the deployment of multi-container applications by using a YAML configuration file. It allows developers to define services, networks, and volumes in a single file.

Docker-compose.yaml

```
yaml

version: '3'
services:
  web:
    image: nginx:alpine
    ports:
      - "8080:80"
  app:
    image: node:14-alpine
    volumes:
      - ./usr/src/app
    working_dir: /usr/src/app
    command: node server.js
```

```
ujjwal@ubuntu:~/project$ docker-compose up -d
Creating network "project_default" ...
Creating project_web_1 ...
Creating project_app_1 ...

ujjwal@ubuntu:~/project$ docker ps
CONTAINER ID   IMAGE             NAMES
abc123...      nginx:alpine      project_web_1
def456...      node:14-alpine    project_app_1
```

Security best practices in Docker include:

- Using minimal base images (like Alpine)

- Running containers as non-root users
- Scanning images for vulnerabilities (e.g., using Trivy)
- Avoiding secrets in Dockerfiles
- Limiting resources per container
- Keeping images up-to-date

-- 📁 Practical Commands and Logs (Executed by User: ujjwal)

-- [Commands and outputs follow below as already inserted above]

-- (No change needed to existing command logs)

Resources:

- Docker Hub/ACR: https://www.youtube.com/watch?v=b_euX_M82uI
- Docker Networking: <https://www.youtube.com/watch?v=c6Ord0GAOp8>
- Docker Volume: https://www.youtube.com/watch?v=u_0O4DOo2GI
- Docker Compose: <https://www.youtube.com/watch?v=HG6yIjZapSA>
- Official Guide: <https://docker-curriculum.com/>