3. Containerization with Docker:

Tool: Docker Desktop, VS code, Eclipse

Program:

- Write a `Dockerfile` for a Python or Node.js application.
- Build and run a Docker image.
- Push the image to Docker Hub.
- Use Docker Compose to manage multi-container applications

Docker basic commands:

1. Docker Version

- docker --version

  Docker version 24.0.2, build 12345abc

2. List Docker Images

- docker images

3. Download Image from Docker Hub

- docker pull <image_name>:<tag>

  Example: docker pull nginx:latest

4. Build Docker Image

- docker build -t <image_name>:<tag> .

  Example:docker build -t myapp:1.0 .

5. List Running Containers

- docker ps

6. List All Containers (Including Stopped)

- docker ps -a

7. Start a Container

- docker start <container_name>/<container_id>

8. Stop a Container

- docker stop <container_name>/ container_id>

9. Restart a Container

docker restart <container_name>/ container_id>

10. Run a Container (Interactive Mode)

- docker run -it -d <image_name> / <container_id>

11. Run with Port Mapping

- docker run –p <host_port>:<container_port> <image_name>

Example : docker run -p 8080:80 nginx

12. Tag an Image for Pushing to Docker Hub

If you want to push an image to Docker Hub, it must be tagged with your Docker Hub username.

docker tag myapp:v1.0 myusername/myapp:v1.0

- myusername is your Docker Hub username.
- myapp:v1.0 is the image you're tagging.

Push the Image to Docker Hub:

docker push myusername/myapp:v1.

<u>Write a Dockerfile for an application.</u>

Step-1: Create maven project with .war format

Step-2: Create one html/jsp file in "src/main/webapp/index.html" and add

HTML code to it.

Step-3: Create "src/main/webapp/WEB-INF" folder. In that

"src/main/webapp/WEB-INF/web.xml" file.

Step-4: Include below plugin in "web.xml"

```
<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-war-plugin</artifactId>
<configuration>
<webXml>src\main\webapp\WEB-INF\index.jsp</webXml> (mention which page to start)
</configuration>
</plugin>
```

Step-5: Update project – Maven clean install compile test (make sure build

Successful)

Prerequisites:

Make sure you have the following installed:

1. **Docker Desktop** (running)
2. **VS Code**
3. **Docker extension for VS Code** (optional but helpful)
4. A .war file ready to use (e.g., sample.war)

Step 1: Place your .war File on VS code Terminal

- Right-click the project folder in Eclipse →select properties→ Copy the project path
- Open VS code terminal →type cd\ command --> now you are in local disk c:\> type cd pate the project path
- Go to File > Open Folder
- Select the eclipse project folder you copied
- VS Code will load the project

Step 2: Create a new file with name Dockerfile inside ur project folder→include below lines of code

```
FROM tomcat:9.0
RUN rm -rf /usr/local/tomcat/webapps/*
COPY /target/sample.war /usr/local/tomcat/webapps/ROOT.war
EXPOSE 8080
```

Step 3: Build the Docker Image by running the following command

```
docker build -t sample-app .
```

Step 4: Run the Container

```
docker run -d -p 8095(give unused port number ):8080 sample-app;
```

Step 5: Access the App (Goto web browser)

http://localhost:8095

- **To Stop and Clean Up**

```
docker ps # get container ID
docker stop <container_id>
docker rm <container_id>
```

- **Use Docker Compose to manage multi-container applications**

To containerize two simple applications, enable communication between them, and deploy them on a local server using Docker

Multi-container-app (**create one main folder**)

```
├── app1/    (subfolder)
│   ├── app.py
│   ├── requirements.txt
│   └── Dockerfile
├── app2/   (subfolder)
│   ├── app.py
│   ├── requirements.txt
│   └── Dockerfile
└── docker-compose.yml
```

Step 1: Create App 1 (Service Provider Folder in VS-CODE)

app1/app.py(File)

```
from flask import Flask
app = Flask(__name__)
@app.route('/')
def hello():
    return "Hello from App 1!"
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

app1/requirements.txt (file)

```
flask==3.0.0
```

app1/Dockerfile (file)

```
FROM python:3.12-slim
WORKDIR /app

COPY requirements.txt   •
RUN pip install --no-cache-dir -r requirements.txt
COPY app.py  .
EXPOSE 5000
CMD ["python", "app.py"]
```

Step 2: Create App 2 (Service Consumer Folder in VS-CODE)

app2/app.py (file)

```
import requests
response = requests.get("http://app1:5000/")
print("Response from App 1:", response.text)
```

app2/requirements.txt (file)

```
requests==2.31.0
```

app2/Dockerfile(file)

```
FROM python:3.12-slim
WORKDIR /app

COPY  requirements.txt    •
RUN pip install --no-cache-dir -r requirements.txt
COPY app.py .
CMD ["python", "app.py"]
```

Step 3: Create Docker Compose File

docker-compose.yml

```
version: '3.9'
services:
  app1:
    build: ./app1
    networks:
      - app-network
    ports:
      - "5000:5000"

  app2:
    build: ./app2
    networks:
      - app-network
    depends_on:
      - app1

networks:
  app-network:
    driver: bridge
```

Explanation:

- Networks → Both apps are on the same network (app-network) to communicate.
- Depends_on → Ensures App 1 starts before App 2.
- app1:5000 → App 2 can communicate with App 1 using the service name app1.

Step 4: Build and Run the Containers

1. **Build the Docker images**:

   ```
   docker-compose build
   ```

2. **Start the containers**:

   docker-compose up

Step 5: Access the Applications

- **Check logs from App 2** to see the response from App 1:

  docker-compose logs app2

  You should see:

  Response from App 1: Hello from App 1!

- **Manually test App 1** by opening a browser and visiting:

  http://localhost:5000