**Step-by-Step Process and Commands:**

**Pre-requisite-1: Install Docker Desktop and Ubuntu in WSL2:**

**1.Install Docker Desktop on Windows:**

* Download Docker Desktop from the official website:  
  Docker Desktop for Windows
* Install Docker Desktop by running the installer and following the on-screen instructions.
* Enable WSL2 backend: During installation, Docker Desktop will ask you to enable the WSL 2 backend. Make sure this option is selected.
* After installation, restart your computer if required.

**2. Install Ubuntu on WSL2:**

* Open Microsoft Store on your Windows machine.
* Search for Ubuntu and choose the version you prefer (e.g., Ubuntu 20.04 or 22.04).
* Click Install and once installed, launch Ubuntu from the Start menu.
* Set Ubuntu as default WSL version 2: If WSL 2 is not set as the default version, run this command in PowerShell to ensure Ubuntu uses WSL2:

**In the Windows powershell run:**

wsl --set-default-version 2

* **If you have already installed Ubuntu using WSL1, you can convert it to WSL2 by running:**

wsl --set-version Ubuntu-20.04 2

* **Open Ubuntu from the Start menu, and it will take a few moments to complete the setup process (if it’s the first time you're opening it).**

**3. Link Docker Desktop with Ubuntu in WSL2:**

* **Open Docker Desktop and make sure it's running. It should automatically detect WSL2 and link to your installed Linux distributions (like Ubuntu).**
* **You can verify if Docker is working by running the following command in the Ubuntu terminal:**

**Inside the wsl terminal:**

docker --version

* **Now, Docker should be available and running on Ubuntu in WSL2.**

**4. Test Docker Setup in Ubuntu (WSL2):**

* **Run the hello-world image to test Docker:**

docker run hello-world

**This should pull the hello-world image from Docker Hub and print a success message, confirming Docker is running inside your WSL2 environment.**

**Pre Requisite -2:To ensure that the Eclipse tar.gz file is in the same folder as the Dockerfile and shellscript.sh, follow these steps:**

1. **Download Eclipse IDE tar.gz:**
   * **Use the provided link to download the Eclipse IDE tarball:** [**Download Eclipse tar.gz**](https://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/2024-12/R/eclipse-jee-2024-12-R-linux-gtk-x86_64.tar.gz&mirror_id=1287)
   * **Once downloaded, place the eclipse-jee-2024-12-R-linux-gtk-x86\_64.tar.gz file in the same directory as your Dockerfile and shellscript.sh.**
2. **Directory Structure:**

**Your project directory should look like this:**

**~/java\_microservices\_project/**

**├── Dockerfile**

**├── shellscript.sh**

**├── eclipse-jee-2024-12-R-linux-gtk-x86\_64.tar.gz**

**Step 1: Building the Docker Image**

1. **Dockerfile:**
   * The Dockerfile specifies the base image, installs dependencies (OpenJDK, MySQL, Tomcat, Eclipse IDE, etc.), and configures the environment.
   * **Explanation:**
     + First, we set up the Ubuntu image as the base.
     + Then, we install OpenJDK 17, essential utilities (curl, git, maven, etc.), and the necessary dependencies for GUI-based applications (such as x11vnc, fluxbox, xvfb for virtual displays).
     + We install Spring Boot CLI, Apache Tomcat, and Eclipse IDE, setting up necessary environment variables.
2. **MySQL Setup:**
   * The Dockerfile also sets up environment variables for MySQL, including user credentials and database name.
   * **Explanation:**
     + The MySQL service is configured to run with the environment variables MYSQL\_ROOT\_PASSWORD, MYSQL\_DATABASE, MYSQL\_USER, and MYSQL\_PASSWORD.
3. **Shell Script (shellscript.sh):**
   * This script is designed to start the services when the container is run, such as:
     + **Xvfb** (for GUI applications in a headless environment),
     + **Fluxbox** (for window management),
     + **VNC Server** (for accessing the GUI via VNC),
     + **Eclipse IDE** (for Java development), and
     + **MySQL Service** (to run MySQL).
   * **Explanation:**
     + The script sets up the virtual display and starts the GUI and MySQL services within the Docker container.
     + It also configures VNC for remote access to the GUI, using x11vnc with a password (my\_secure\_password).
4. **Exposing Ports:**
   * Ports are exposed for VNC (5900), Apache Tomcat (8080), and MySQL (3306).
   * **Explanation:**
     + These ports are exposed to the host system to allow communication with the services running inside the container.

**Step 2: Build the Docker Image**

* **Command:**

docker build -t java .

**Explanation:**

* + This command builds the Docker image using the Dockerfile in the current directory (denoted by .).
  + The -t option tags the image with the name java.

**Step 3: Run the Docker Container**

* **Command:**

docker run --name java -p 5900:5900 -p 8080:8080 -p 3306:3306 java

**Explanation:**

* + This runs the java container and maps ports from the container to the host.
    - -p 5900:5900 maps the VNC port for GUI access.
    - -p 8080:8080 maps Tomcat's default HTTP port.
    - -p 3306:3306 maps MySQL's default port.
  + The container will run the shellscript.sh file as the default command, starting the services inside the container.

**Step 4: Access the VNC and Test Eclipse IDE**

* To access Eclipse via VNC:
  + Open a VNC client and connect to localhost:5900 using the password my\_secure\_password.

**Explanation:**

* + The script sets up a VNC server at port 5900, allowing you to access the GUI (Eclipse IDE) remotely.

**Step 5: Managing Docker Containers**

* **Stop, Restart, and Start the Container:**

docker stop java

docker restart java

docker start java

**Explanation:**

* + docker stop stops the container named java.
  + docker restart stops and then starts the container.
  + docker start starts the container if it's stopped.
* **List Running and All Containers:**

docker ps

docker ps -a

**Explanation:**

* + docker ps lists the currently running containers.
  + docker ps -a lists all containers, including those that are stopped.

**Step 6: Publish Docker Image to Docker Hub and Set Up Docker Swarm**

**In this step, we will:**

* **Publish the Docker image to your Docker Hub repository.**
* **Set up Docker Swarm and perform actions like deploying a service, creating a swarm cluster, configuring failover, and testing scaling.**

a) Publish the Image to Docker Hub:

To publish your Docker image to your Docker Hub repository (thamizh16/microservices\_env), follow these steps:

1. Login to Docker Hub:

If you're not already logged in to Docker Hub, use the following command to log in:

docker login

Enter your Docker Hub credentials (username and password) when prompted.

1. Tag the Docker Image:

Tag your local Docker image to match the format required for your Docker Hub repository. Replace java with the name of the image you built earlier:

docker tag java thamizh16/microservices\_env:v1.0

**Explanation:**

* + This command tags your local java image with the repository name thamizh16/microservices\_env and the tag latest.

1. Push the Image to Docker Hub:

Push the tagged image to your Docker Hub repository using the following command:

docker push thamizh16/microservices\_env:latest

**Explanation:**

* + This command uploads your Docker image to the thamizh16/microservices\_env repository on Docker Hub.

b) Set Up Docker Swarm:

Docker Swarm allows you to create and manage a cluster of Docker nodes and deploy services across them. To set up Docker Swarm:

1. Initialize Docker Swarm:

On the master node (the first node in your cluster), run:

docker swarm init

This will initialize the Swarm mode and return a join token. You can use this token to add additional nodes to the Swarm.

1. Add Worker Nodes to the Swarm:

On each worker node, run the docker swarm join command with the token provided during the swarm init step:

docker swarm join --token <join-token> <master-node-ip>:2377

**Explanation:**

* + Replace <join-token> with the token provided by the swarm init command, and <master-node-ip> with the IP address of the master node.

1. Verify the Swarm Cluster:

To ensure your Swarm nodes are correctly added, run the following on the master node:

docker node ls

This will show the list of nodes in the cluster, including the master and worker nodes.

c) Deploy a Service to the Docker Swarm:

You can now deploy a service to the Docker Swarm using the image you pushed to Docker Hub:

1. Deploy the Service:

On the master node, run the following command to deploy your service:

docker service create --name microservices\_env --replicas 3 thamizh16/microservices\_env:v1.0

**Explanation:**

* + This command creates a service named microservices\_env with 3 replicas running your Docker image from Docker Hub.

1. Verify the Service Deployment:

To check the status of the service, use:

docker service ls

This will display information about the service, including the number of replicas and their status.

d) Configure the Cluster for Failover:

In Docker Swarm, failover is automatically handled. If one of the service replicas fails, Swarm will attempt to restart it on another available node in the cluster. You can test this by stopping one of the service containers:

1. Stop a Running Container:

Run the following command to stop a container in the service:

docker service ps microservices\_env

Then, find the container ID and stop it:

docker stop <container-id>

Docker Swarm will automatically restart the service on another node.

e) Test Scaling:

You can scale the number of replicas of your service at any time:

1. Scale the Service:

To scale your service to 5 replicas, run**:**

docker service scale microservices\_env=5

**Explanation:**

* + This command increases the number of replicas to 5.

1. Verify the Scaled Service:

Check the updated status of the service**:**

docker service ps microservices\_env

This will display the new number of replicas and their status.