**Docker Swarm**

Docker Swarm is a container orchestration tool that allows you to manage a cluster of Docker nodes as a single logical system. It provides several benefits, such as scalability, high availability, load balancing, and simplified deployment. Here are some use cases and examples of how Docker Swarm can be utilized:

### **1. High Availability Web Application**

**Use Case:** Deploying a web application that requires high availability and redundancy.

**Example:**

* Create a Swarm cluster with multiple manager and worker nodes.
* Deploy a replicated service for the web application.
* Docker Swarm ensures that if one node fails, another node takes over, maintaining the application's availability.

**Steps:**

**Initialize Swarm:**  
docker swarm init --advertise-addr <MANAGER-IP>

**Add Worker Nodes:** On each worker node:  
  
docker swarm join --token <WORKER-TOKEN> <MANAGER-IP>:2377

**Deploy a Web Application:**  
docker service create --name webapp --replicas 3 -p 80:80 nginx

**Check Service Status:**  
docker service ls

### **2. Continuous Integration/Continuous Deployment (CI/CD) Pipeline**

**Use Case:** Automating the deployment of applications with a CI/CD pipeline.

**Example:**

* Use Docker Swarm to deploy applications automatically when new code is committed.
* Integrate with CI/CD tools like Jenkins, GitLab CI, or GitHub Actions.

**Steps:**

**Initialize Swarm and Deploy Jenkins:**  
docker swarm init

docker service create --name jenkins --replicas 1 -p 8080:8080 jenkins/jenkins

1. **Configure Jenkins to Deploy to Swarm:**
   * Set up Jenkins with necessary plugins for Docker and Docker Swarm.
   * Create a Jenkins pipeline that builds Docker images and deploys them to the Swarm cluster.
2. **Automate Deployment:**
   * Configure Jenkins to trigger builds and deployments on code changes.

### **3. Load Balancing and Scaling Services**

**Use Case:** Distributing traffic across multiple instances of a service for load balancing and scaling.

**Example:**

* Deploy a service with multiple replicas.
* Docker Swarm automatically load balances requests across these replicas.

**Steps:**

**Initialize Swarm:**  
docker swarm init

**Deploy a Service with Load Balancing:**  
docker service create --name myservice --replicas 5 -p 8080:80 nginx

**Scale the Service:**  
docker service scale myservice=10

### **4. Microservices Architecture**

**Use Case:** Deploying a microservices-based application with multiple interdependent services.

**Example:**

* Use Docker Swarm to manage the deployment and scaling of each microservice.
* Ensure communication between services through the Swarm network.

**Steps:**

**Initialize Swarm:**  
docker swarm init

**Deploy Microservices:**  
docker service create --name service1 --replicas 3 -p 5000:5000 my\_microservice1

docker service create --name service2 --replicas 2 -p 5001:5001 my\_microservice2

1. **Ensure Services Communicate:**
   * Use Docker Swarm's service discovery to enable communication between services using their service names.

### **Docker Logs**

To view the logs of a container, you can use the following command:

docker logs <container\_name\_or\_id>

### **Options**

Here are some useful options for the docker logs command:

* **-f, --follow**: Follow log output (similar to tail -f).
* **--tail**: Show only the last N lines of log output.
* **-t, --timestamps**: Show timestamps for each log entry.
* **--since**: Show logs since a specific time (e.g., 2022-07-01T13:23:37 or 10m for last 10 minutes).
* **--until**: Show logs up until a specific time.

#### **1. Viewing Logs of a Container**

docker logs my\_container

#### **2. Following Logs in Real-Time**

docker logs -f my\_container

#### **3. Showing the Last 10 Lines of Logs**

docker logs --tail 10 my\_container

#### **4. Showing Logs with Timestamps**

docker logs -t my\_container

#### **5. Showing Logs Since a Specific Time**

docker logs --since "2023-07-11T15:00:00" my\_container

#### **6. Combining Options**

docker logs -f --tail 10 --since "10m" my\_container

## **Before proceeding with the project, kindly fill out the feedback form for week 1 below:**

<https://forms.office.com/r/4F2xy5HvPY>

**Do not hesitate in sharing what you liked the most and what you think can be done to improve your learning experience.**

## **Project 01**

### **Objectives:**

* Create and manage Docker volumes for data persistence.
* Set up a Docker network for container communication.
* Use Docker Compose to manage multi-container applications.
* View and manage Docker logs.
* Deploy the application using Docker Swarm.

### **Project Outline:**

1. **Create Docker Volumes**
2. **Create a Docker Network**
3. **Write a Docker Compose File**
4. **Deploy the Application with Docker Compose**
5. **Manage Docker Logs**
6. **Deploy the Application Using Docker Swarm**

### **Step-by-Step Guide**

#### **1. Create Docker Volumes**

Docker volumes are used to persist data generated by and used by Docker containers.

docker volume create wordpress\_data

docker volume create mysql\_data

#### **2. Create a Docker Network**

Create a custom network for the containers to communicate.

docker network create wordpress\_network

#### **3. Write a Docker Compose File**

Create a docker-compose.yml file to define and manage the services.

version: '3.3'

services:

db:

image: mysql:5.7

volumes:

- mysql\_data:/var/lib/mysql

networks:

- wordpress\_network

environment:

MYSQL\_ROOT\_PASSWORD: example

MYSQL\_DATABASE: wordpress

MYSQL\_USER: wordpress

MYSQL\_PASSWORD: wordpress

wordpress:

image: wordpress:latest

volumes:

- wordpress\_data:/var/www/html

networks:

- wordpress\_network

ports:

- "8000:80"

environment:

WORDPRESS\_DB\_HOST: db:3306

WORDPRESS\_DB\_USER: wordpress

WORDPRESS\_DB\_PASSWORD: wordpress

WORDPRESS\_DB\_NAME: wordpress

volumes:

mysql\_data:

wordpress\_data:

networks:

wordpress\_network:

#### **4. Deploy the Application with Docker Compose**

Run the following command to start the services defined in the docker-compose.yml file.

docker-compose up -d

* Verify that the containers are running.

docker-compose ps

* Access the WordPress setup by navigating to http://localhost:8000.

#### **5. Manage Docker Logs**

* View logs for a specific service.

docker-compose logs wordpress

* Follow logs for real-time updates.

docker-compose logs -f wordpress

#### **6. Deploy the Application Using Docker Swarm**

Docker Swarm is a native clustering and orchestration tool for Docker.

* Initialize Docker Swarm.

docker swarm init

* Convert the Docker Compose file to a Docker Stack file, docker-stack.yml.

version: '3.3'

services:

db:

image: mysql:5.7

volumes:

- mysql\_data:/var/lib/mysql

networks:

- wordpress\_network

environment:

MYSQL\_ROOT\_PASSWORD: example

MYSQL\_DATABASE: wordpress

MYSQL\_USER: wordpress

MYSQL\_PASSWORD: wordpress

deploy:

replicas: 1

wordpress:

image: wordpress:latest

volumes:

- wordpress\_data:/var/www/html

networks:

- wordpress\_network

ports:

- "8000:80"

environment:

WORDPRESS\_DB\_HOST: db:3306

WORDPRESS\_DB\_USER: wordpress

WORDPRESS\_DB\_PASSWORD: wordpress

WORDPRESS\_DB\_NAME: wordpress

deploy:

replicas: 1

volumes:

mysql\_data:

wordpress\_data:

networks:

wordpress\_network:

* Deploy the stack using Docker Swarm.

docker stack deploy -c docker-stack.yml wordpress\_stack

* Verify the stack is running.

docker stack services wordpress\_stack

## **Project 02:**

## **Objectives:**

* Deploy an application across multiple Docker Swarm worker nodes.
* Place specific components on designated nodes.
* Monitor and troubleshoot using Docker logs.
* Modify and redeploy the application.

### **Project Outline:**

1. **Initialize Docker Swarm and Join Worker Nodes**
2. **Label Nodes for Specific Component Placement**
3. **Create a Docker Stack File**
4. **Deploy the Application**
5. **Monitor and Troubleshoot Using Docker Logs**
6. **Modify and Redeploy the Application**

### **Step-by-Step Guide**

#### **1. Initialize Docker Swarm and Join Worker Nodes**

On the manager node, initialize Docker Swarm:

docker swarm init --advertise-addr <MANAGER-IP>

Join the worker nodes to the swarm. On each worker node, run the command provided by the docker swarm init output:

docker swarm join --token <SWARM-TOKEN> <MANAGER-IP>:2377

Verify the nodes have joined:

docker node ls

#### **2. Label Nodes for Specific Component Placement**

Label nodes to specify where certain components should run. For example, label a node for the database service:

docker node update --label-add db=true <NODE-ID>

Label another node for the application service:

docker node update --label-add app=true <NODE-ID>

Verify the labels:

docker node inspect <NODE-ID>

#### **3. Create a Docker Stack File**

Create a docker-stack.yml file to define the services and node placement constraints:

version: '3.8'

services:

db:

image: mysql:5.7

volumes:

- mysql\_data:/var/lib/mysql

networks:

- app\_network

environment:

MYSQL\_ROOT\_PASSWORD: example

MYSQL\_DATABASE: appdb

MYSQL\_USER: user

MYSQL\_PASSWORD: password

deploy:

placement:

constraints:

- node.labels.db == true

app:

image: your-app-image

networks:

- app\_network

ports:

- "8000:80"

environment:

DB\_HOST: db

deploy:

replicas: 2

placement:

constraints:

- node.labels.app == true

volumes:

mysql\_data:

networks:

app\_network:

#### **4. Deploy the Application**

Deploy the stack using Docker Swarm:

docker stack deploy -c docker-stack.yml app\_stack

docker stack services app\_stack

#### **5. Monitor and Troubleshoot Using Docker Logs**

Check the logs for the services:

docker service logs app\_stack\_db

docker service logs app\_stack\_app

Follow the logs in real-time to monitor issues:

docker service logs -f app\_stack\_app

#### **6. Modify and Redeploy the Application**

Make modifications to the application or the stack file as needed. For example, change the number of replicas:

services:

app:

deploy:

replicas: 3

Update the stack with the new configuration:

docker stack deploy -c docker-stack.yml app\_stack

Verify the changes:

docker stack services app\_stack