Docker- Containerising Services using Docker

Docker is a platform for developing, shipping, and running applications in a consistent, isolated environment called containers. Containers encapsulate everything needed for an application to run, allowing developers to build applications that work seamlessly across different environments.

Docker Image

An **image** is a read-only template with instructions for creating a Docker container. It contains everything your app needs to run, like code, dependencies, and configurations.

Docker Container

A container is a runtime instance of an image. It's an isolated environment with everything needed to run an application.

Key Commands:

- **Pull an Image**: You can pull a pre-built image from Docker Hub (Docker's public image repository).

  ```
  docker pull <image_name>
  ```

  Example:

  ```
  docker pull nginx
  ```

Build an Image: To build a custom image, you create a **Dockerfile** with instructions and use the `docker build` command.

Example `Dockerfile`:

```
FROM openjdk:17-jdk-slim
WORKDIR /app
COPY target/my-app.jar my-app.jar
EXPOSE 8080
ENTRYPOINT ["java", "-jar", "my-app.jar"]
```

Build the Image

:

```
docker build -t my-spring-app .
```

List Docker Images:

```
docker images
```

Run a Container:

```
docker run -d -p 8080:8080 my-spring-app
```

- **List Running Containers**:

  ```
  docker ps
  ```

- **Stop a Container**:

```
docker stop <container_id>
```

- **Remove a Container**:

```
docker rm <container_id>
```

Docker Volumes

- **Concept**: Volumes provide persistent storage outside the container's file system, ideal for databases or configuration files.
- **Example**: Mount a volume to save data from a Spring Boot application.
- **Commands**:
  - **Create a Volume**:

    ```
    docker volume create spring-app-data
    ```

  - **Attach Volume to Container**:

    ```
    docker run -d -p 8080:8080 -v spring-app-data:/data my-spring-app
    ```

    Here, `/data` in the container is mounted to the `spring-app-data` volume on the host, ensuring persistent data storage.

Docker Networking

- **Concept**: Docker networking allows containers to communicate. Docker provides bridge networks for isolated networks, but you can create custom networks.
- **Example**: Create a network for multiple services to communicate, e.g., Spring Boot app and MySQL database.
- **Commands**:
  - **Create a Network**:

    ```
    docker network create spring-net
    ```

  - **Run Containers on the Same Network**:

    ```
    docker run -d --network spring-net --name app my-spring-app
    docker run -d --network spring-net --name db mysql:8.0
    ```

    Now, `app` can access `db` by the hostname `db` in the `spring-net` network.

Docker Compose

- **Concept**: Docker Compose simplifies running multi-container applications by defining services, networks, and volumes in a YAML file.
- **Example**: Use Docker Compose to run a Spring Boot app with a MySQL database.
- **Commands**:
  - **docker-compose.yml**:

    ```
    version: '3'
    services:
      app:
        image: my-spring-app
    ```

```
    ports:
      - "8080:8080"
    environment:
      SPRING_DATASOURCE_URL: jdbc:mysql://db:3306/mydatabase
      SPRING_DATASOURCE_USERNAME: root
      SPRING_DATASOURCE_PASSWORD: password
    depends_on:
      - db

  db:
    image: mysql:8.0
    environment:
      MYSQL_ROOT_PASSWORD: password
      MYSQL_DATABASE: mydatabase
    ports:
      - "3306:3306"
```

- **Run Docker Compose**:

```
docker-compose up
```

- **Stop Docker Compose**:

```
docker-compose down
```

7. Docker Commands Cheat Sheet

Command	Description
`docker pull <image_name>`	Pull an image from Docker Hub
`docker build -t <image_name> .`	Build an image from Dockerfile
`docker run -d -p <host>:<container>`	Run a container in detached mode
`docker ps`	List running containers
`docker stop <container_id>`	Stop a container

`docker rm <container_id>`	Remove a container
`docker volume create <volume_name>`	Create a Docker volume
`docker network create <network_name>`	Create a Docker network
`docker-compose up`	Start all services defined in docker-compose.yml
`docker-compose down`	Stop and remove services defined in docker-compose.yml

Working with spring boot project

Package your application in jar file.

```
mvn clean install
```

run directory jar file to check

```
mvn spring-boot:run
```

build docker file

```
# Use a lightweight JRE image for runtime
FROM openjdk:21-slim-buster
MAINTAINER substring.technologies

# Copy the built jar from the builder stage
COPY  target/category-service-0.0.1-SNAPSHOT.jar category-service-0.0.1-S


# Run the application
ENTRYPOINT ["java", "-jar", "category-service-0.0.1-SNAPSHOT.jar"]
```

build the docker images

```
 docker build . -t batchlcwd/category-service:v1
```

run container

```
docker run -p 9091:9091 batchlcwd/category-service:v1
```

Create image using buildpacks

configure image name in pom.xml and then run the below command

```
mvn spring-boot:build-image
```

Push the image to dockerhub

```
docker image push docker.io/batchlcwd/category-service:v1
```

Lets introduce docker-compose:

Docker Compose is a tool for defining and running multi-container applications. It is the key to unlocking a streamlined and efficient development and deployment experience.

Compose simplifies the control of your entire application stack, making it easy to manage services, networks, and volumes in a single, comprehensible YAML configuration file. Then, with a single command, you create and start all the services from your configuration file.

Compose works in all environments; production, staging, development, testing, as well as CI workflows. It also has commands for managing the whole lifecycle of your application:

- Start, stop, and rebuild services

- View the status of running services
- Stream the log output of running services
- Run a one-off command on a service

Configure services to docker compose file

```
# mysql,phpmyadmin,postgress, pgadmin, mongo and mongo-express

version: '3.8'

services:
  category:
    image: "batchlcwd/notification-service:v1"
    container_name: order_ms
    ports:
      - "9098:9097"
    networks:
      - batchnetwork
    deploy:
      resources:
        limits:
          memory: 700m

  mysql-db:
    #name of service
    image: mysql:8.0
    container_name: mysql-container
    environment:
      MYSQL_ROOT_PASSWORD: admin
      MYSQL_DATABASE: categorydb
      MYSQL_USER: user
      MYSQL_PASSWORD: user123
    ports:
      - "3307:3306"
    volumes:
      - ./mysql-data:/var/lib/mysql
```

```
# php my admin service: db client
phpmyadmin:
  image: phpmyadmin:latest
  container_name: phpmyadmin
  environment:
    PMA_HOST: mysql-db
  ports:
    - "8081:80"
  depends_on:
    - mysql-db

# PostgreSQL Service
postgres-db:
  image: postgres:13
  container_name: postgres-db
  environment:
    POSTGRES_USER: user
    POSTGRES_PASSWORD: user123
    POSTGRES_DB: coursedb
  ports:
    - "5432:5432"
  volumes:
    - ./postgres-data:/var/lib/postgresql/data

# pgAdmin for PostgreSQL
pgadmin:
  image: dpage/pgadmin4
  container_name: pgadmin
  environment:
    PGADMIN_DEFAULT_EMAIL: admin@gmail.com
    PGADMIN_DEFAULT_PASSWORD: admin
  ports:
    - "8082:80"
  depends_on:
    - postgres-db

# MongoDB Service
mongo:
    image: mongo:latest
    container_name: mongo
    ports:
      - "27017:27017"
    volumes:
      - ./mongo-data:/data/db

  # Mongo Express for MongoDB
  mongo-express:
    image: mongo-express:latest
    container_name: mongo-express
    ports:
      - "8083:8081"
    environment:
      ME_CONFIG_MONGODB_SERVER: mongo
      # MONGO_INITDB_DATABASE: videodb
    depends_on:
      - mongo

networks:
  batchnetwork:
    driver: "bridge"
volumes:
  mysql-data:
  postgres-data:
  mongo-data:
```

Configure Apache Kafka and Redis using docker.

```
version: '3.8'

services:
  redis:
    image: redis:latest
    container_name: redis_service
    ports:
      - "6379:6379"
    restart: always

  zookeeper:
    image: confluentinc/cp-zookeeper:latest
    container_name: zookeeper
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181
      ZOOKEEPER_TICK_TIME: 2000
    ports:
      - "2181:2181"
    restart: always

  kafka:
    image: confluentinc/cp-kafka:latest
    container_name: kafka
    depends_on:
      - zookeeper
    ports:
      - "9092:9092"
    environment:
      KAFKA_BROKER_ID: 1
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: PLAINTEXT:PLAINTEXT
      KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://kafka:9092
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
    restart: always

  app:
    image: your_app_image_name
    container_name: app_service
    depends_on:
      - redis
      - kafka
    ports:
      - "8080:8080"
    environment:
```

```
    - REDIS_HOST=redis
    - REDIS_PORT=6379
    - KAFKA_BROKER=kafka:9092
  restart: always
```