Nadege Gaju

Docker Concepts and Commands

1. Docker Concepts

1.1 What is Docker?

Docker is an open-source platform designed to automate the deployment, scaling, and management of applications in lightweight containers. Containers allow developers to package an application with all its dependencies into a standardized unit that can be deployed consistently across different environments.

1.2 Key Components of Docker

1. **Docker Engine**: The core part of Docker that runs the containers. It consists of:
   - **Server (Docker Daemon)**: Handles tasks such as building, running, and managing Docker containers.
   - **REST API**: Interface that allows Docker to interact with other programs.
   - **Client (Docker CLI)**: Command-line interface that communicates with the Docker daemon.
2. **Docker Images**: Immutable snapshots of the application and its environment. An image is built using a Dockerfile and can be run as a container.
3. **Docker Containers**: Running instances of Docker images. They are lightweight and portable units of the application that include everything the application needs to run.
4. **Dockerfile**: A text file that contains a set of instructions to build a Docker image. It typically includes steps like specifying a base image, installing dependencies, and running the application.
5. **Docker Compose**: A tool used to define and run multi-container Docker applications using a YAML file (docker-compose.yml). It allows you to start, stop, and manage services that consist of multiple containers (e.g., web app, database, and cache).
6. **Docker Hub**: A public registry where Docker images are stored. You can pull base images or share your own images via Docker Hub.

2. Docker Commands

2.1 Basic Docker Commands

1. **Check Docker Version**

```
docker --version
```

This command displays the installed version of Docker.

2. **List Docker Commands**

```
docker --help
```

This displays a list of Docker commands and their descriptions.

2.2 Working with Docker Images

1. **Pull an Image**

```
docker pull <image-name>
```

This downloads a Docker image from Docker Hub.

Example:

```
docker pull postgres
```

2. **List Available Images**

```
docker images
```

This lists all images on the local machine.

3. **Remove an Image**

```
docker rmi <image-id>
```

This removes a specific image by its ID.

2.3 Working with Containers

1. **Run a Container**

```
docker run -d -p <host-port>:<container-port> --name <container-name> <image-name>
```

This starts a container from an image.

Example:

```
docker run -d -p 5432:5432 --name postgres-container postgres
```

2. **List Running Containers**

```
docker ps
```

This lists all running containers.

3. **Stop a Container**

```
docker stop <container-name>
```

This stops a running container.

4. **Remove a Container**

```
docker rm <container-name>
```

This removes a stopped container.

5. **Inspect a Container**

```
docker inspect <container-name>
```

This displays detailed information about the container, including its IP address, volumes, and networking.

6. **View Logs of a Container**

```
docker logs <container-name>
```

This shows the logs generated by the container.

2.4 Building Docker Images

1. **Build a Docker Image**

```
docker build -t <image-name> <path-to-dockerfile>
```

This command builds an image using a Dockerfile.

Example:

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

2. **Tagging an Image**

```
docker tag <image-id> <repository-name>:<tag>
```

Tags a built image for versioning or pushing to a registry.

2.5 Docker Compose Commands

1. **Start Services**

```
docker-compose up
```

This starts all services defined in the docker-compose.yml file. You can also run it with the -d flag to run it in the background (detached mode).

2. **Stop Services**

```
docker-compose down
```

This stops and removes all the containers defined by Docker Compose.

3. **Build and Start Services**

```
docker-compose up --build
```

This builds and starts the services based on the Dockerfile.

4. **View Logs for a Service**

```
docker-compose logs <service-name>
```

This displays logs for a specific service.

5. **Scale Services**

```
docker-compose scale <service-name>=<num-instances>
```

This command allows you to scale the number of instances of a specific service.

3. Example: Dockerizing a Spring Boot Application

Dockerfile

```
# Use an official OpenJDK runtime as a parent image
FROM openjdk:17-jdk-alpine

# Set the working directory in the container
WORKDIR /app

# Copy the JAR file into the container
COPY target/Docker-0.0.1-SNAPSHOT.jar app.jar

# Expose the port the app runs on
EXPOSE 8080

# Run the Spring Boot application
ENTRYPOINT ["java", "-jar", "app.jar"]
```

docker-compose.yml

yaml

Copy code

```yaml
services:
 app:
  build:
   context: .
  ports:
   - "8080:8080"
  depends_on:
   - db
  environment:
   SPRING_DATASOURCE_URL: jdbc:postgresql://db:5432/mydb
   SPRING_DATASOURCE_USERNAME: user
   SPRING_DATASOURCE_PASSWORD: password

 db:
  image: postgres:latest
  environment:
   POSTGRES_DB: mydb
   POSTGRES_USER: user
   POSTGRES_PASSWORD: password
  ports:
   - "5432:5432"
```

4. Summary

- Docker simplifies the development, testing, and deployment of applications by using containers.

- Dockerfiles define how to build images, while Docker Compose allows us to run multi-container applications.
- Docker commands are used to manage images and containers efficiently, allowing rapid development and consistent deployment across environments.