DevOps Concepts & Docker Report

This document provides an overview of Docker and its use in containerizing a Spring Boot application, along with key Docker concepts and commands. Docker has become a popular tool for deploying applications because it allows you to package applications with their dependencies, ensuring they run consistently across different environments.

A **Dockerfile** is a script containing a series of commands to assemble an image. For a Spring Boot application, the Dockerfile typically follows these steps:

1. **Specify the base image**: You usually start with a base image like OpenJDK, which contains the Java runtime environment needed for Spring Boot.
2. **Copy the application's JAR file**: The Dockerfile copies the compiled Spring Boot JAR file from your build directory into the container.
3. **Expose the port**: Spring Boot applications typically run on port 8080, so you expose this port to make it accessible.
4. **Run the application**: The final step is to set the command that runs the JAR file inside the container.

Example steps for creating the Dockerfile:

- Use a base image like `openjdk`.
- Copy the JAR file into the container.
- Expose port 8080.
- Run the JAR file.

2. Docker Image

A **Docker image** is a snapshot of an application or service bundled with its dependencies. After defining the Dockerfile for your Spring Boot application, you build a Docker image from it. The image is immutable, meaning it contains all the files and libraries needed for the application to run.

Key points about Docker images:

- You can build an image using the `docker build` command.
- Once built, the image can be reused to create containers.
- You can store the image in a repository like Docker Hub, from where it can be pulled for deployment.

3. Docker Compose

Docker Compose is a tool that simplifies the management of multi-container applications. Instead of manually starting each container and linking them, Docker Compose uses a `docker-compose.yml` file to define and run multiple services in a single command.

For a Spring Boot application, the `docker-compose.yml` can define not only the Spring Boot service but also related services like databases (e.g., MongoDB, Redis). This enables you to create a fully functioning environment with just one command.

Key features of Docker Compose:

- Define multiple services in a YAML file.
- Manage networks and volumes between services.
- Start all services with a single `docker-compose up` command.

4. Multi-Container Application Using Docker Compose

In a **multi-container application**, different services (e.g., application server, database, cache) run in separate containers. Docker Compose makes it easy to define and manage all these services together.

For example, if your Spring Boot application requires MongoDB and Redis, you can define them in the `docker-compose.yml` file as services alongside the Spring Boot container. Each container runs in isolation, but Docker Compose handles networking between them.

Example of a multi-container setup:

- A Spring Boot service that listens on port 8080.
- A MongoDB service for database storage.
- A Redis service for caching.

Using Docker Compose, you can manage this setup by simply running `docker-compose up`, which will start all services, network them, and ensure they work together.

5. Docker Concepts and Commands

Here are key Docker concepts and the essential commands for managing Docker applications:

Key Docker Concepts

- **Image**: A read-only template that contains the application code and dependencies. It's built from a Dockerfile.
- **Container**: A running instance of a Docker image, providing an isolated environment to run applications.
- **Dockerfile**: A script with instructions on how to build a Docker image.
- **Volumes**: Persistent storage for containers. Data in volumes is not lost when containers are removed.
- **Networks**: Allow communication between containers. Docker provides several network drivers (e.g., bridge, host).
- **Docker Hub**: A cloud-based repository for storing Docker images, enabling you to share and distribute images.

Common Docker Commands

- **Build an image**: Converts a Dockerfile into an image.
  - `docker build -t <image_name> .`
- **Run a container**: Starts a container from an image.
  - `docker run -d -p <host_port>:<container_port> <image_name>`
- **Stop a container**: Stops a running container.
  - `docker stop <container_id>`
- **Remove a container**: Deletes a stopped container.
  - `docker rm <container_id>`
- **List containers**: Shows all running containers.
  - `docker ps`
- **Access a running container**: Opens an interactive terminal inside the container.
  - `docker exec -it <container_id> /bin/bash`
- **List Docker images**: Shows all locally stored images.
  - `docker images`
- **Check logs**: Retrieves logs from a running or stopped container.
  - `docker logs <container_id>`
- **Remove unused containers, images, or volumes**: Frees up disk space.
  - `docker container prune`

Docker Compose Commands

- **Start all services defined in `docker-compose.yml`**:
  - `docker-compose up`
- **Run services in the background**:
  - `docker-compose up -d`
- **Stop services**:
  - `docker-compose down`
- **Rebuild services**:
  - `docker-compose up --build`

- **List running services**:
    - `docker-compose ps`

By understanding these Docker concepts and using the relevant commands, you can efficiently containerize and manage your Spring Boot application along with its supporting services (e.g., databases, caches). Docker Compose helps automate and simplify the management of multi-container applications, making deployment and scaling more efficient.