Deployment

Spring Profiles

Spring Profiles

Every enterprise application has many environments, like:

Dev | Test | Stage | Prod

Each environment requires a setting that is specific to them. These environments host specific configurations called Profiles. Use the @Profile annotation to load beans specific to a profile.

Setting up Profiles

Method 1 - via Maven:

```
./mnvw spring-boot:run –Dspring-boot.run.profiles=prod
```

Method 2 - via Environment variables:

```
SPRING_PROFILES_ACTIVE=prod ./mnvw spring-boot:run
```

Method 3 - via JAR files

```
java –Dspring.profiles.active=prod –jar target/<jar file>.jar
```

Deployment

Using Flyway for Database Migration

Hibernate.ddl-auto=update in Production:

- Unpredictable Schema Changes:

Renaming a column in an entity class may result in the creation of a new column in the database rather than renaming the existing one, leading to potential data loss or inconsistency.

- Data Loss Risk:

Removing an entity might cause Hibernate to drop the column or table, which could remove important production data.

- Lack of Control and Irreversible

Spring Boot + Flyway

Why use Flyway:

- Flyway offers structured database schema management and version control.
- It automates migration script execution and maintains a version history for seamless migrations across environments.
- Integration with Spring Boot streamlines deployment, allowing developers to focus on feature development rather than migration management.

Deployment

Homework

Recap

1. Spring Profiles
2. Setting up RDS, EB, Code Pipeline and Code Build in AWS to Deploy Spring Boot code to production!
3. Handling Database Migration with Flyway

Homework

1. Make a list of All the AWS Services and write detailed step by step process to deploy a Spring Boot Application using EB, ELB, RDS, Code Pipeline, Code Deploy
2. Deploy a Spring Boot Application in AWS (optional)
2. Use Flyway to handle Database migrations in dev (prod if possible).