Deploying Spring Boot to Render.com

Prerequisites

- Spring Boot application with Git repository
- GitHub, GitLab, or Bitbucket account
- Render.com account (free tier available)

Step 1: Prepare Your Spring Boot Application

1.1 Configure application.properties

properties

```
# For production deployment
server.port=${PORT:8080}
spring.profiles.active=prod

# Database configuration (example for PostgreSQL)
spring.datasource.url=${DATABASE_URL}
spring.datasource.driver-class-name=org.postgresql.Driver
spring.jpa.hibernate.ddl-auto=validate
spring.jpa.database-platform=org.hibernate.dialect.PostgreSQLDialect
```

1.2 Add system.properties (for Java version)

Create `system.properties` in your project root:

properties

```
java.runtime.version=17
```

1.3 Ensure your pom.xml has the Spring Boot Maven plugin

xml

```
<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>
```

1.4 Create a Procfile (optional but recommended)

```
web: java -Dserver.port=$PORT $JAVA_OPTS -jar target/your-app-name-0.0.1-SNAPSHOT.jar
```

Step 2: Push to Git Repository

Ensure your code is in a Git repository on GitHub, GitLab, or Bitbucket.

Step 3: Set Up Render Service

3.1 Create New Web Service

1. Log into Render.com
2. Click "New +" → "Web Service"
3. Connect your Git repository
4. Select your repository and branch

3.2 Configure Build Settings

- **Name**: Your app name
- **Environment**: `Java`
- **Region**: Choose closest to your users
- **Branch**: `main` or your preferred branch
- **Build Command**: `./mvnw clean install -DskipTests`
- **Start Command**: `java -Dserver.port=$PORT $JAVA_OPTS -jar target/your-app-name-0.0.1-SNAPSHOT.jar`

3.3 Environment Variables

Add these in the Render dashboard:

- `JAVA_TOOL_OPTIONS`: `-XX:MaxRAMPercentage=75.0`

- Any database URLs, API keys, etc.

Step 4: Database Setup (if needed)

4.1 Create PostgreSQL Database

1. In Render dashboard: "New +" → "PostgreSQL"
2. Configure database name and settings
3. Copy the database URL from the database dashboard

4.2 Add Database URL to Web Service

- Go to your web service → Environment
- Add `DATABASE_URL` with your PostgreSQL connection string

Step 5: Deploy

1. Click "Create Web Service"
2. Render will automatically build and deploy
3. Monitor the deployment logs for any issues

Common Issues and Solutions

Build Failures

- **Maven wrapper issues**: Ensure `mvnw` has execute permissions:

```bash
git update-index --chmod=+x mvnw
```

- **Memory issues**: Add `MAVEN_OPTS=-Xmx1024m` environment variable

Runtime Issues

- **Port binding**: Always use `${PORT:8080}` in application.properties
- **Profile issues**: Set `SPRING_PROFILES_ACTIVE=prod` environment variable
- **Database connections**: Verify DATABASE_URL format

Performance Optimization

- **Memory settings**: Use `-XX:MaxRAMPercentage=75.0` in JAVA_TOOL_OPTIONS
- **Startup time**: Consider using Spring Boot's lazy initialization:

```properties
spring.main.lazy-initialization=true
```

Free Tier Limitations

- 750 hours/month (sleeps after 15 minutes of inactivity)
- 512MB RAM
- Shared CPU
- Custom domains require paid plan

Example Environment Variables

```
DATABASE_URL=postgresql://user:pass@host:port/dbname
SPRING_PROFILES_ACTIVE=prod
JAVA_TOOL_OPTIONS=-XX:MaxRAMPercentage=75.0
PORT=10000
```

Monitoring Your Deployment

- View logs in Render dashboard
- Set up health check endpoint in your Spring Boot app:

```java
@RestController
public class HealthController {
  @GetMapping("/health")
  public ResponseEntity<String> health() {
    return ResponseEntity.ok("OK");
  }
}
```

Next Steps

- Set up custom domain (paid feature)
- Configure SSL certificates (automatic with custom domains)
- Set up monitoring and alerts
- Consider CI/CD integration for automated deployments