# **Understanding Spring Boot Configuration**

Rupasri Guruprasad

## [**Rupasri Guruprasad**](https://www.linkedin.com/in/rupasrigu/)

In the world of application development, effective configuration management is crucial for maintaining flexibility and efficiency. Spring Boot streamlines this process, enabling us to focus more on building applications rather than dealing with complex configuration settings. In this article, let's explore how **Spring Boot** makes configuration management seamless using **Profiles**, **Properties**, and **YAML** files.

### **Configuration Management with Spring Boot**

Spring Boot’s configuration system allows us to define settings in files like application.properties or application.yml, which it reads at runtime. This flexible approach means we don’t need to rewrite our code when moving across environments, we simply update or override settings.

For example, **application.properties** is the simpler of the two options, storing each setting as a key-value pair:

server.port=8080  
spring.datasource.url= jdbc:mysql://localhost:3306/mydb

On the other hand, **application.yml** provides a hierarchical, more structured format, which can be especially useful for nested properties:

server:  
 port: 8080  
spring:  
 datasource:  
 url: jdbc:mysql://localhost:3306/mydb  
 username: user  
 password: password

## **Introducing Profiles for Environment-Specific Configurations**

### **What are Profiles in Spring Boot?**

**Profiles** in Spring Boot allow us to define configurations for different environments. For example, a microservice might require different database configurations in development versus production. By setting up profiles, we can quickly switch between these configurations without modifying the core code.

### **How Profiles Work?**

Profiles let you define environment-specific properties in dedicated configuration files. By default, Spring Boot activates the default profile, but custom profiles (like dev, test, or prod) can be specified in the application.properties or application.yml file.

### **Using Profiles for Different Environments**

You can specify different properties for each environment using profile-specific configuration files. Here’s how you might set up configuration files for **development** and **production** environments:

* application-dev.properties or application-dev.yml: Holds configuration settings for the development environment.
* application-prod.properties or application-prod.yml: Holds configuration settings for the production environment.

For example, your application-dev.yml might look like this:  
  
server:  
 port: 8081  
spring:  
 datasource:  
 url: jdbc:mysql://localhost:3306/devdb  
 username: devUser  
 password: devPassword application-prod.yml could contain:  
  
server:  
 port: 8080  
spring:  
 datasource:  
 url: jdbc:mysql://production-server:3306/proddb  
 username: prodUser  
 password: prodPassword

### **Activating a Profile**

To activate a profile, add the following to your application.properties:

spring.profiles.active=dev

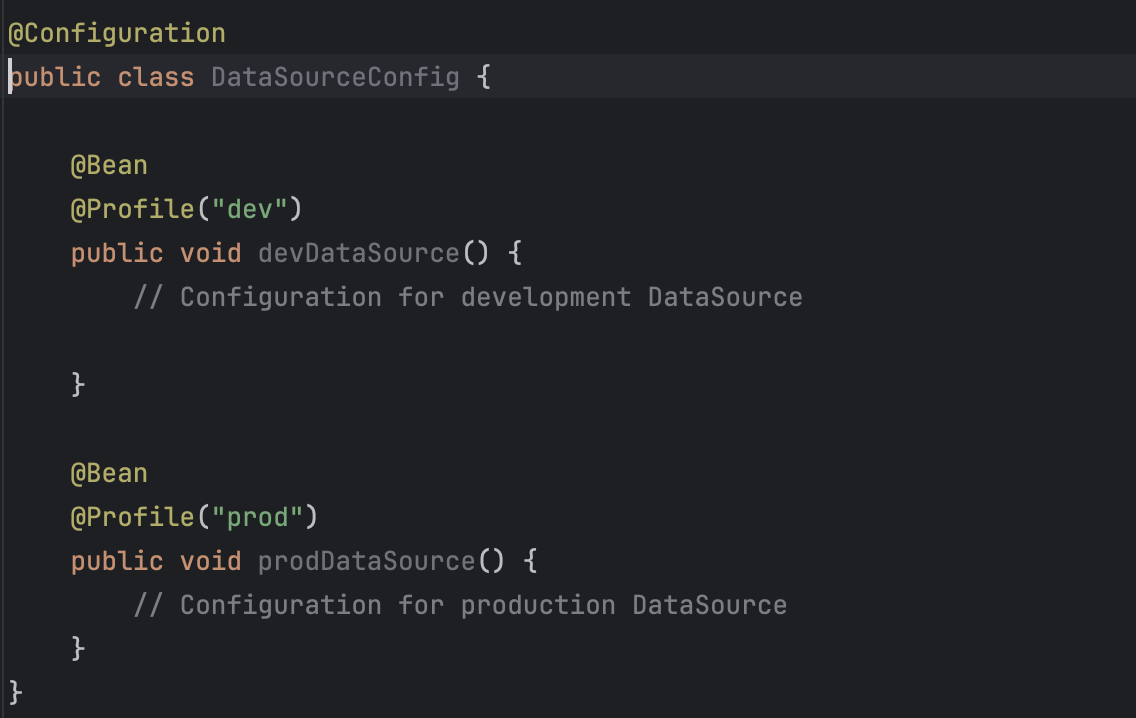
By specifying the profile, Spring Boot loads the respective configuration file (e.g., application-dev.yml), seamlessly adapting the application to the environment it’s running in.

### **When to Use Profiles and Configuration Files**

* **Development**: Use application-dev.yml with configurations suitable for local testing.
* **Testing/QA**: Use application-test.yml to simulate production-like environments and run tests.
* **Production**: Use application-prod.yml with secure and optimized configurations suitable for production load.

### **Using @Profile Annotation for Conditional Beans**

In addition to setting environment-specific configurations, Spring Boot’s @Profile annotation allows you to conditionally load beans based on the active profile. This is useful when certain beans, like a mock database or specialized logging, are needed only in specific environments (e.g., development or testing).



In this example:

* The devDataSource bean is loaded only when the dev profile is active.
* The prodDataSource bean is loaded only when the prod profile is active.

This enables the application to use environment-specific beans, further customizing the configuration.

### **Summary**

Spring Boot’s Profiles, Properties and YAML configurations empower developers to streamline configuration management across multiple environments. This flexibility simplifies setup and testing, allowing you to focus on building and deploying robust microservices.

#SpringBoot #Microservices #ConfigurationManagement #TechLearning #LearningJourney#Day6