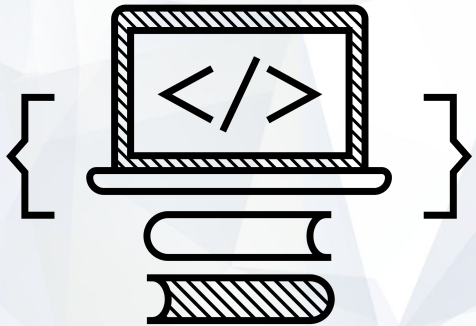
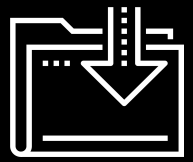


Configuration Server

Course: Java

S1



Learning Outcomes

By the end of this lesson, you will be able to:

01 Explain the advantages of using a Spring configuration server.

02 Create and configure a Spring configuration server.

03 Use a configuration server for the configuration of a Spring Boot REST web service.

Purpose

Purpose

Discuss the following points about the purpose and advantages of externalizing configuration settings:

Settings can be changed without having to rebuild code.

Settings can be changed without having to restart the application.

Settings are centralized.

We can trace changes to settings.

We can provide encryption and decryption of sensitive settings.

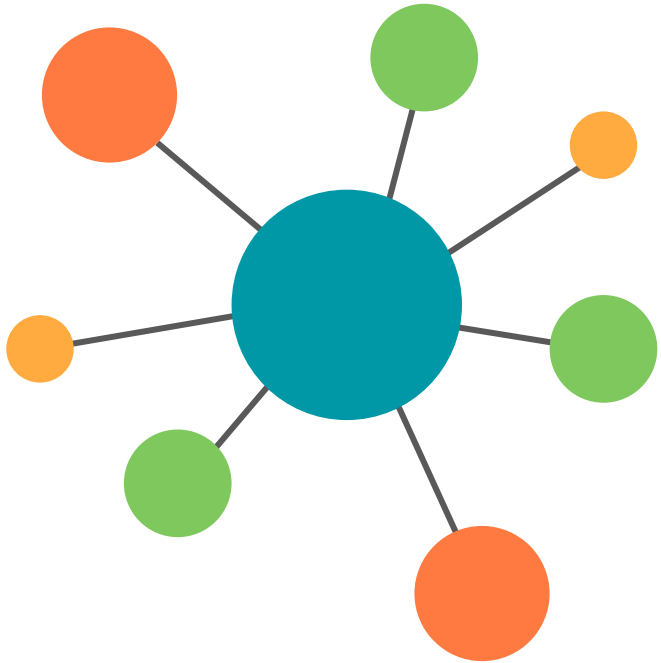
We'll use Spring Cloud Config Server to provide these features.

Configuration Server Setup

Configuration Server Setup

Discuss the following points before stepping through the creation of a configuration server:

- Configuration server creation is similar to the creation of any Spring Boot project. We'll use the Initializr and select the appropriate starter pack. In this case, it’s the Config Server starter pack.
- The configuration server is a web service. We start it up and it runs on a particular port serving configuration settings to configured clients (we'll configure the client in the next step).
- Configuration files for client applications are kept in a Git repository.



Setting Up the Config Server

1. Set up the repo.
2. Add the `@EnableConfigServer` annotation to the `main` class of the application.
3. Set the following values in the `application.properties` file:
    - `server.port`: This is the port that the config server will listen on.
    - `spring.cloud.config.server.git.uri`: This is the location of the Git repository holding all of the client configuration files.
    - `spring.cloud.config.server.git.username`: This is the username for the Git repository.
    - `spring.cloud.config.server.git.password`: This is the password for the Git repository.
    - **Note:** For simplicity, our repositories will not have username/password.
4. After these values are set, we can start the config server as we would any web service.

Time to Code

Create Config Server

Suggested Time:

20 minutes


Break

Using the Configuration Server

Using the Configuration Server

In the upcoming code along, we will create a web service that uses the config server/

01

We enable a web service to use the config server by adding the Cloud Config Client starter dependencies.

02

To enable the ability to refresh the configuration without having to restart the application, we must include the Actuator starter dependencies.

03

We must create a `bootstrap.properties` file in the same location as the `application.properties` file. It gets read before the `application.properties`. This file contains the setup information for the config server so our application can use the config server.

Property Values in bootstrap.properties

We set the following property values in `bootstrap.properties`:

- `spring.application.name`
  - This is the name of our web service. It must match the name of the associated properties file in the Git repo used by the config server.
- `spring.cloud.config.uri`
  - This is the host and port of the config server.

Additional Configuration Settings

One advantage of using the config server is that we can change the config settings without having to restart the application. This requires some additional configuration:

The first step is to include the Actuator dependencies (we did that above).

We must include the `@RefreshScope` annotation on the controller class.

We must include the following entry in our application property file in the Git repo backing the config server:

```
Management.endpoints.web.exposure.
include=*
```

In order to force the application to RefreshScope, we must send an empty POST to the /actuator/refresh endpoint of the client application. The curl command is:

```
$ curl localhost:8080/actuator/
refresh -d {} -H"Content-Type:
application/json"
```

Time to Code

Create Configuration Server Client Application

Suggested Time:

20 minutes


RECAP

Recap

The main takeaways from this lesson are:

01

Spring Cloud Config Server allows us to externalize the configuration settings for our services.

02

The advantages of using Spring Cloud Config Server for this are:

- Settings can be changed without having to rebuild code.
- Settings can be changed without having to restart the application.
- Settings are centralized.
- We can trace changes to settings.
- We can provide encryption and decryption of sensitive settings.

03

We created a Spring Cloud Config Server.

04

We created a web service that uses our Spring Cloud Config Server.