Simplifying Environment Management with Centralized Configuration

Richard Seroter
SENIOR DIRECTOR OF PRODUCT, PIVOTAL

@rseroter

Overview

The role of configuration in microservices

Problems with the status quo

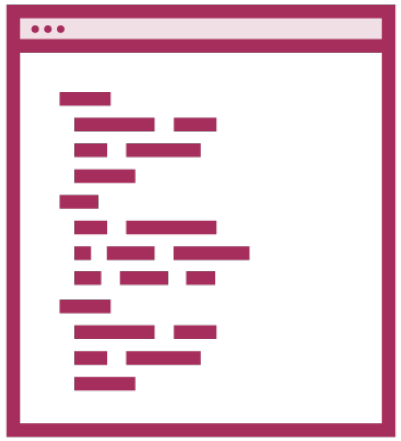
Describing Spring Cloud Config

Creating a configuration server

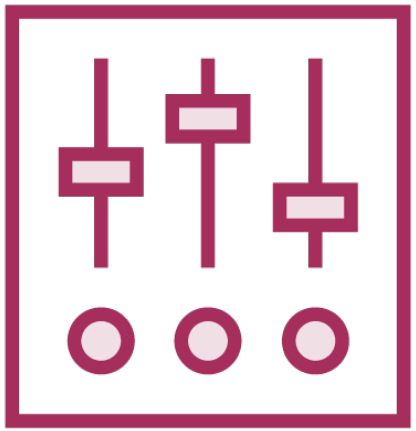
Consuming configurations in apps

Summary

The Role of Configuration in Microservices



Removing "settings" from compiled code



Change runtime behaviors

Enforce consistency across elastic services

Cache values to reduce load on databases

Problems with the Status Quo

- Local configuration files fall out of sync
- No history of changes with env variables
- Configuration changes require restart
- Challenges with sensitive information
- Inconsistent usage across teams

Spring Cloud Config

HTTP access to git or file based configurations.

Creating the Config Server

Choose a config source

Add config files

Build the Spring project

Secure the configurations

Creating the Config Server: Choosing a Source

Local Files	Git-based Repository
Points to classpath or file system	Points to git repo
Multiple search locations possible	Multiple search locations possible
No audit trail	Full change history
Supports labelling	Supports labelling
Support for placeholders in URI	Support for placeholders in URI
Relies on “native” profile	Multiple profiles possible
Dev/test only, unless set up in reliable, shared fashion	Local git for dev/test highly available file system or service for production

Setting up Configuration Files

- Native support for YML, properties files
- Can serve out *any* text file
- File name contains app, optionally profile
- Nested folders supported
- All matching files returned

Creating the Config Server: The Spring Project

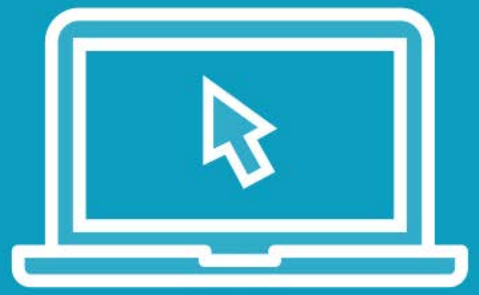
1. Use start.spring.io, Spring Tool Suite or chosen IDE to generate scaffolding.

2. See POM dependency on spring-cloud-config-server and spring-boot-starter-actuator.

3. Add @EnableConfigServer annotation to class.

4. Create application properties (or YAML) with server port, app name, and profile.

Demo



Create a Spring Starter project

Annotate the main class

Set the application properties

Add **local** configuration files

Run as a Spring Boot app

Query for configurations

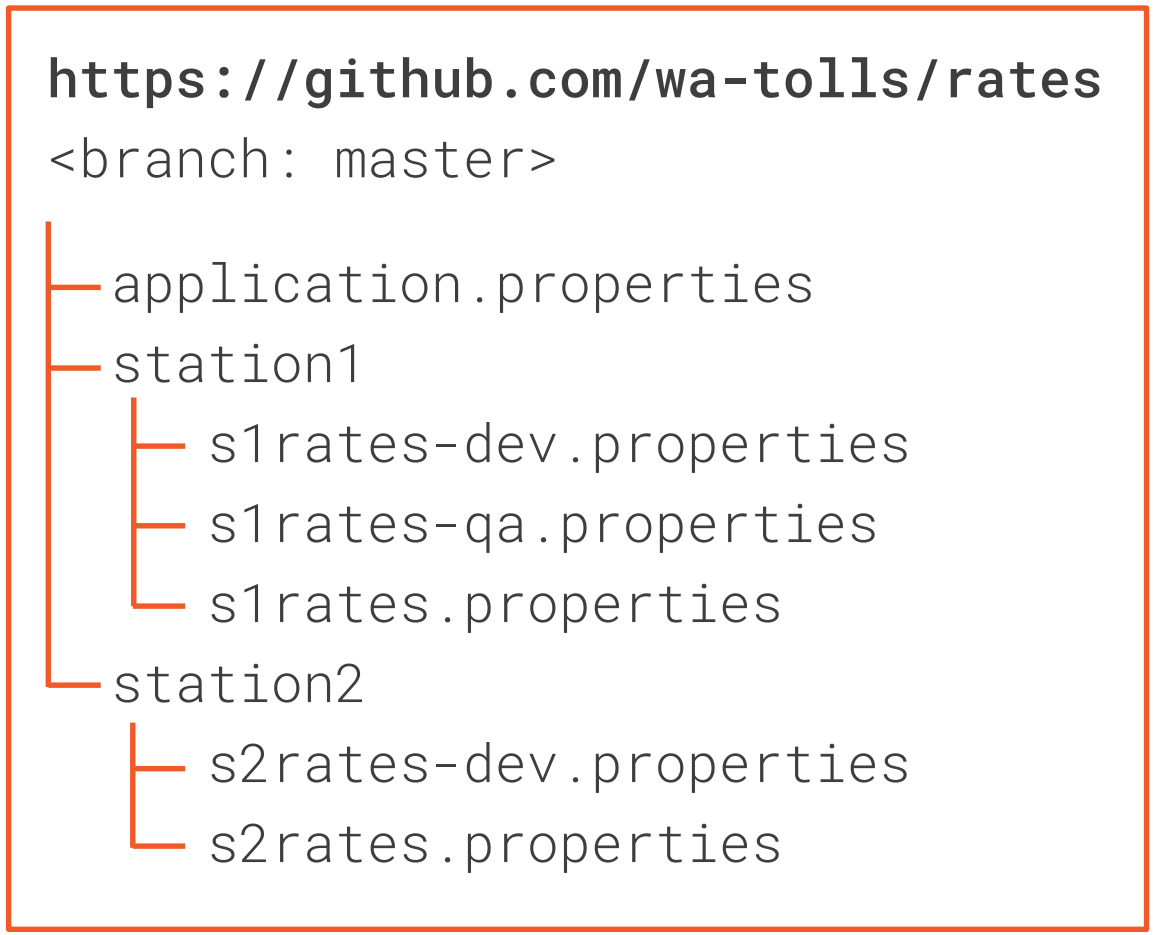
```
---
spring:
 cloud:
  config:
   server:
    git:
     uri: https://github.com/wa-
tolls/rates
     searchPaths: 'station*'
     repos:
      prod:
       pattern: '*/prod'
       uri: https://github.com/wa-
tolls/rates-prod
```

Creating the Config Server

- Point to git location
- Pattern to search sub directories
- Points to alternate repos
- Pattern to go to alternate repo
- URI for alternate repo

Creating the Config Server: Endpoints

https://github.com/wa-tolls/rates
<branch: master>



/{application}/{profile}[/{label}]

-required- -required- -optional-

Creating the Config Server: Endpoints

https://github.com/wa-tolls/rates
<branch: master>

- **application.properties**
- station1
  - s1rates-dev.properties
  - s1rates-qa.properties
  - **s1rates.properties**
- station2
  - s2rates-dev.properties
  - s2rates.properties

/{application}/{profile}[/{label}]

-required- -required- -optional-

/s1rates/default

Creating the Config Server: Endpoints

https://github.com/wa-tolls/rates
<branch: master>

- **application.properties**
- station1
  - **s1rates-dev.properties**
  - s1rates-qa.properties
  - **s1rates.properties**
- station2
  - s2rates-dev.properties
  - s2rates.properties

/{application}/{profile}[/{label}]

-required- -required- -optional-

/s1rates/dev

Creating the Config Server: Endpoints

https://github.com/wa-tolls/rates
<branch: master>

- **application.properties**
- station1
  - s1rates-dev.properties
  - s1rates-qa.properties
  - s1rates.properties
- station2
  - s2rates-dev.properties
  - **s2rates.properties**

/{application}/{profile}[/{label}]

-required- -required- -optional-

/s2rates/qa

Creating the Config Server: Endpoints

https://github.com/wa-tolls/rates
<branch: master>

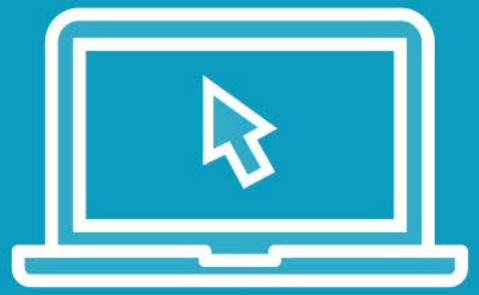
- **application.properties**
- station1
  - s1rates-dev.properties
  - s1rates-qa.properties
  - s1rates.properties
- station2
  - s2rates-dev.properties
  - s2rates.properties

/{application}/{profile}[/{label}]

-required- -required- -optional-

/s3rates/default

Demo



- Create GitHub repo with files
- Create a Spring Starter project
- Annotate the main class
- Set git URL in application YAML
- Run as a Spring Boot app
- Experiment with search paths, queries

Consuming Configurations

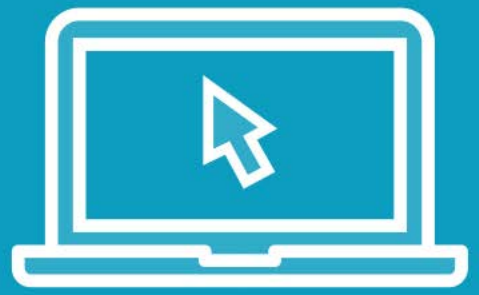
Spring apps use Config Servers as a property source

Loads values based on app name, Spring profile, label.

Annotate code with @Value attribute

Can consume from non-Spring apps via URL

Demo



Create a Spring Starter project

Add application and bootstrap files

Create controller with annotations

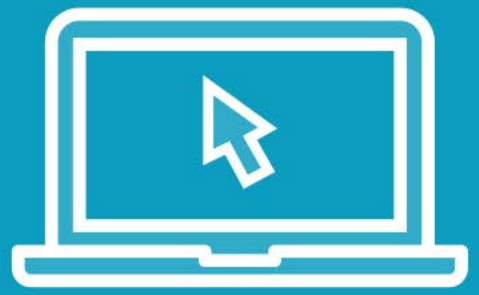
Return values derived from properties

Experiment with different name, profiles

Applying Access Security to Configurations

- Integrated security via Spring Security
- Default HTTP Basic, but other options like OAuth2
- Configured in properties, YAML files
- Could be unique per profile
- Look to also secure with network security, API gateways

Demo



Add POM dependency for springboot-starter-security

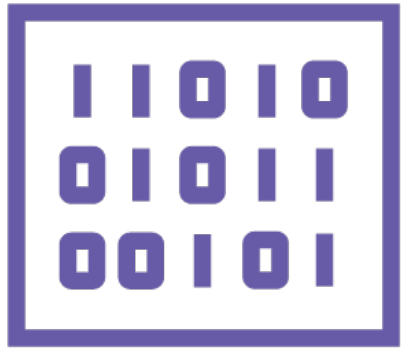
Test project and get authentication error

Add Basic Auth credentials

Call API with valid credentials

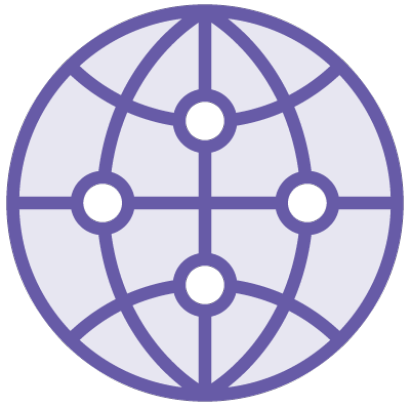
Update client app with credentials

Encrypting and Decrypting Configurations

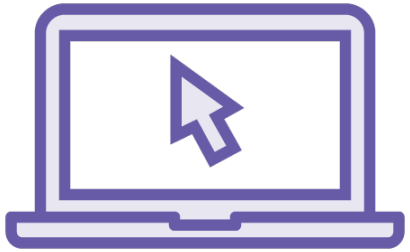


Property values not stored in plain text

Symmetric or asymmetric options

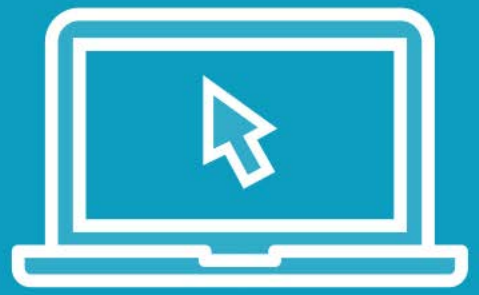


Server offers /encrypt and /decrypt endpoints



Can decrypt on server or in the client

Demo



- Download full-strength JCE
- Add key to bootstrap file
- Generate encrypted value and add to properties file
- Retrieve configuration via API
- Test client app with server-side decrypted value
- Update server to require client-side decryption
- Change client to decrypt

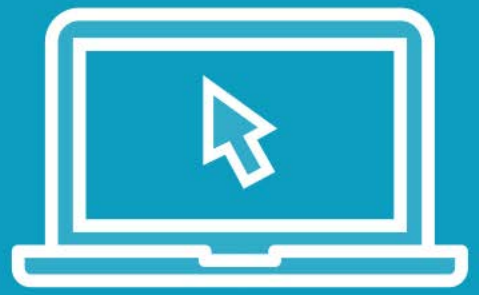
Advanced Settings and Property Refresh

Configure for "fail fast" to fail service if it cannot connect to Config Server

Can add client retry if Config Server occasionally unavailable

Refresh clients individually or in bulk

Demo



- Add RefreshScope to controller
- Start server and client apps
- Change a property in GitHub
- Trigger client refresh
- See new value without requiring a restart

Summary

- Overview
- The role of configuration in microservices
- Problems with the status quo
- Describing Spring Cloud Config
- Creating a configuration server
- Consuming configurations in apps