



Pivotal

Cloud Native Applications

Spring Cloud Config

Config in a Spring Context

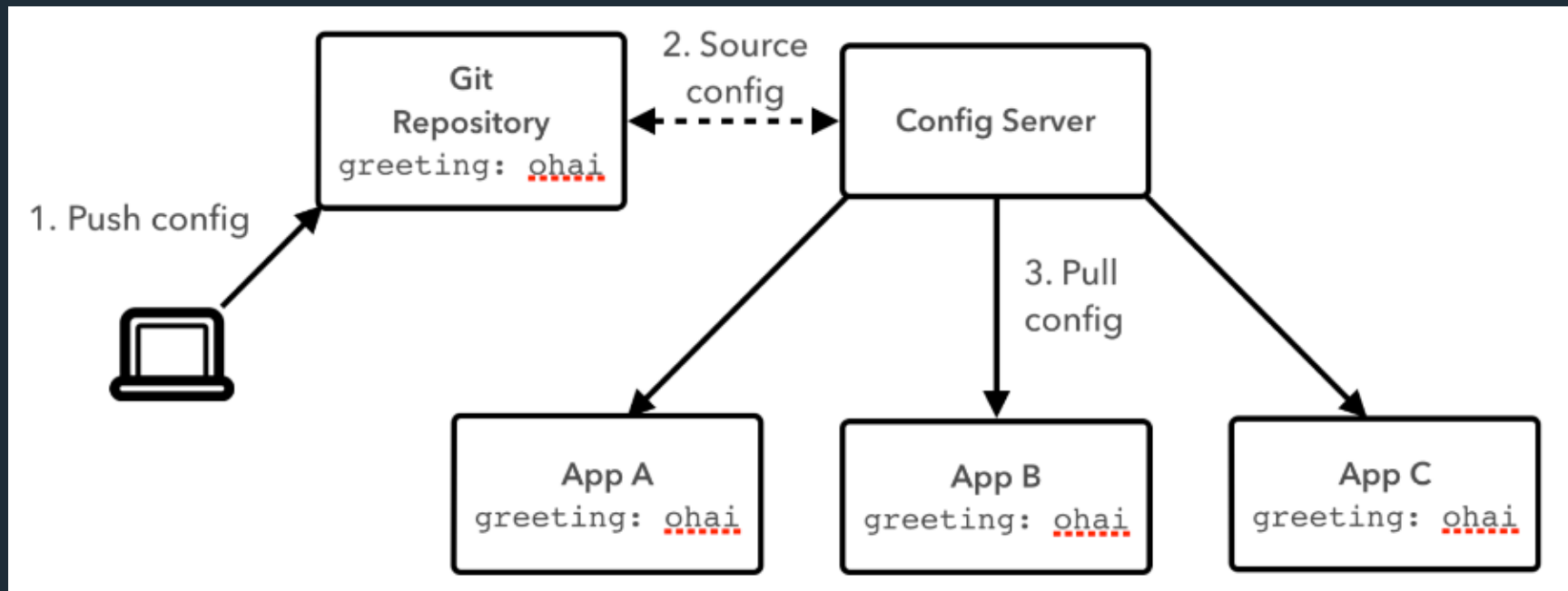
Configuration in a Spring context can usually be described as values that wire up Spring beans.

Spring has provided several approaches to setting config, including externalizing (via Command Line arguments, Env Variables, etc.)

Still, gaps exist:

- Changes to config require restarts
- No audit trail
- Config is de-centralized
- No support for sensitive information (no encryption capabilities)

Spring Cloud Config



Spring Cloud Config Server

- **The Server provides an HTTP, resource-based API for external configuration (name-value pairs, or equivalent YAML content).**
- **Bind to the Config Server and initialize Spring Environment with remote property sources**
- **Embeddable easily in a Spring Boot application using `@EnableConfigServer`**
- **Encrypt and decrypt property values (symmetric or asymmetric)**

Spring Cloud Config Server

```
@SpringBootApplication
@EnableConfigServer ← MAGIC!!
public class ConfigServer {
    public static void main(String[] args) {
        SpringApplication.run(ConfigServer.class, args);
    }
}
```

application.yml

```
spring:
  cloud:
    config:
      server:
        git:
          uri: http://github.com/adamz/config-repo.git
```

<http://github.com/adamz/config-repo/blob/master/demo.yml>

Greeting: Bonjour

Consuming Application

```
@Configuration
@EnableAutoConfiguration ← MAGIC!!
@RestController
public class GreetingService {

    @Autowired Greeter greeter;

    @RequestMapping("/")
    public String home() {
        return String.format("%s World", greeter.greeting);
    }

    @Component
    @RefreshScope
    public class Greeter {
        @Value("${greeting}")
        String name = "World";
    }
}
```

bootstrap.yml

```
spring:
  application
    name: demo

cloud:
  config:
    uri: http://my-config-server.com
```

Refreshing Configuration Context

```
@Component
@RefreshScope ← MAGIC!!
public class Greeter {
    @Value("${greeting}")
    String name = "World";
}
```

1. Update Git Repository
2. Send a POST refresh request to the application(s) to refresh

Curl -X POST http://my-awesome-app.com/refresh

Refreshing Configuration Context

When running many applications, refreshing each one can be cumbersome.

Instead, leverage Spring Cloud Bus pub/sub notification with RabbitMQ.

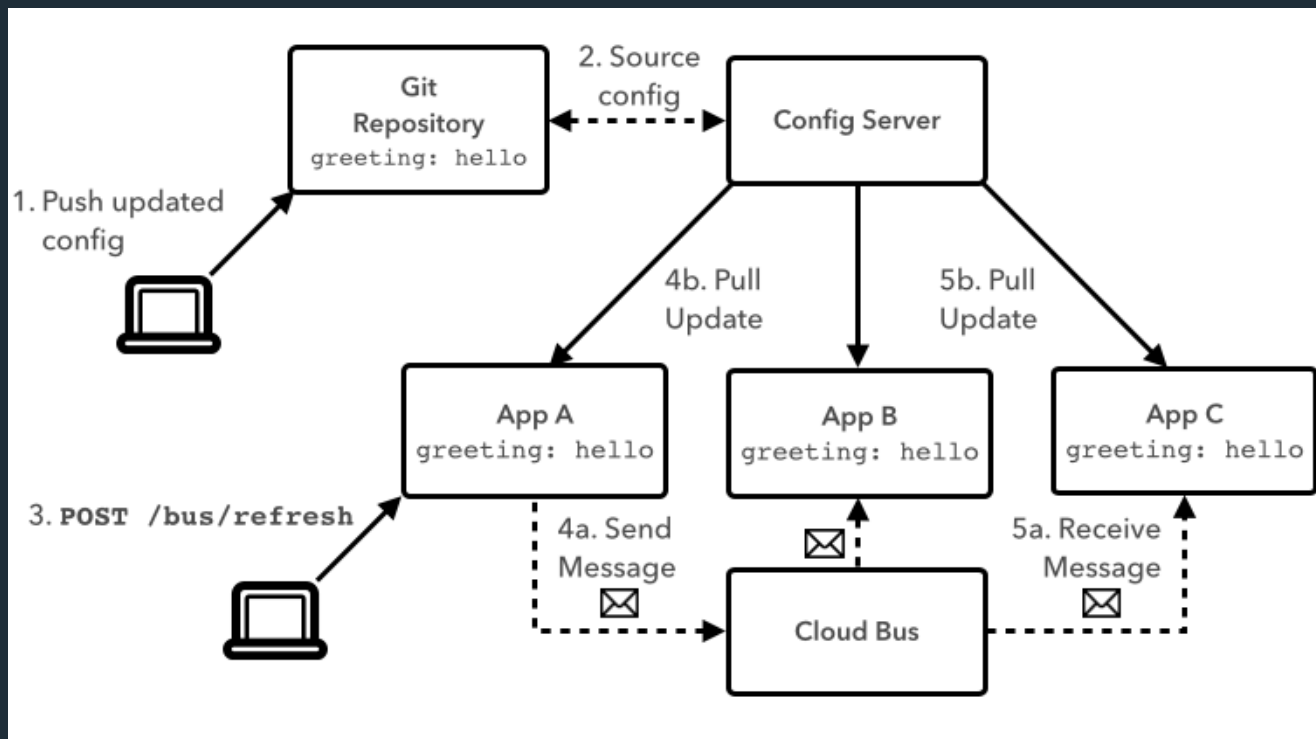
Send a POST request to the refresh endpoint to fetch updated config values:

<http://my-awesome-app.com/bus/refresh>

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-bus-amqp</artifactId>
</dependency>
```



Refreshing Configuration Context



Spring Cloud Services: Config Server

- Automated deployment of server component
- Simply specify Git URL
- Bind into CF application
- *Optionally* Bind in RabbitMQ service for the Cloud Bus



Spring Cloud Services for Pivotal Cloud Foundry

Config Server

Instance ID: b2620e4e-6339-4dfd-a7a8-76656d576616

Configuration Source

☒ Git

☐ Subversion

Git URI

Git Branch (default is 'master')

Pivotal. © 2015 Pivotal Software Inc. All rights reserved.

A dark, high-contrast photograph of a modern interior space, likely a transit hub or a large office building. The scene is filled with the silhouettes of many people walking or standing. In the background, a large wall of windows allows light to filter through, creating a grid-like pattern. A large clock is visible in the upper left corner. The floor is highly reflective, mirroring the silhouettes and the light from the windows. The overall atmosphere is busy and architectural.

LAB