Spring Cloud Config

Centralized, versioned configuration management for distributed applications

Objectives

- At the end of this module, you will be able to
  - Explain what Spring Cloud Config is
  - Build and Run and Spring Cloud Config Server
  - Establish a Repository
  - Build, Run, and Configure a Client

Module Outline

- Configuration Management
  - Challenges
  - Desired Solution
- Spring Cloud Config
  - Server Side
  - Client Side
- Repository Organization

What is Application Configuration?

- Applications are more than just code
  - Connections to resources, other applications

- Usually use external configuration to adjust software behavior
  - Where resources are located
  - How to connect to the DB
  - Etc.

Configuration Options

- Package configuration files with application
  - Requires rebuild, restart
- Configuration files in common file system
  - Unavailable in cloud
- Use environment variables
  - Done differently on different platforms
  - Large # of individual variables to manage / duplicate
- Use a cloud-vendor specific solution
  - Coupling application to specific environment

Other Challenges

- Microservices → large # of dependent services
- Dynamic updates
  - Changes to services or environment variables require restage or restart
- Version control

Manual Work, Brittle

Deployment Activities

Traceablity

Desired Solution for Configuration

- Platform/Cloud-Independent solution
  - Language-independent too
- Centralized
  - Or a few discrete sources of our choosing
- Dynamic
  - Ability to update settings while an application is running
- Controllable
  - Same SCM choices we use with software
- Passive
  - Services (Applications) should do most of the work themselves by self-registering

Solution:

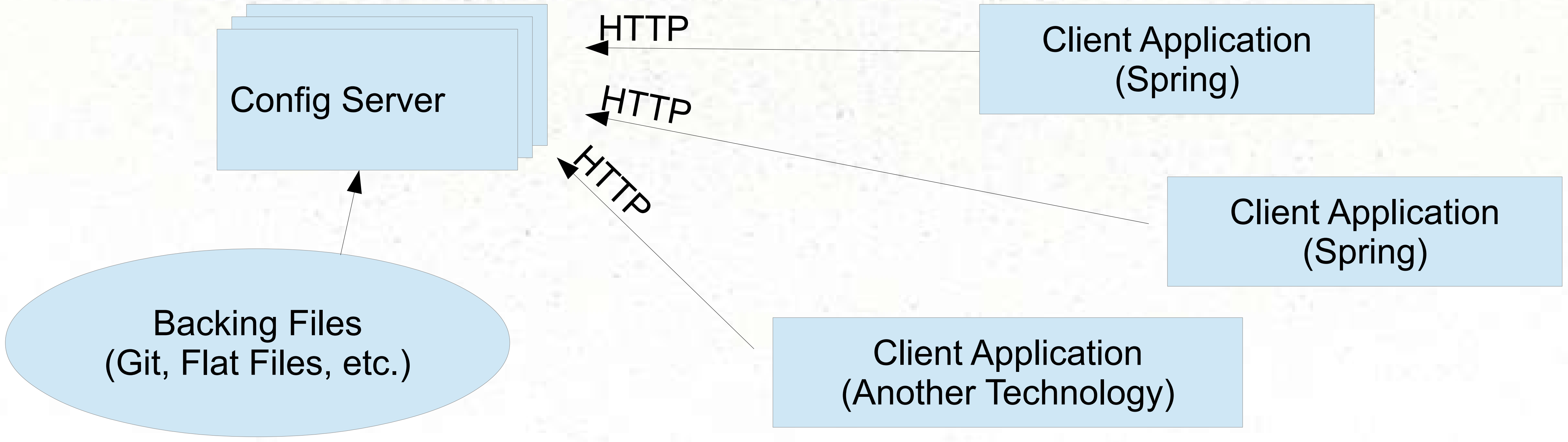
- Spring Cloud Config
  - Provides centralized, externalized, secured, easy-to-reach source of application configuration
- Spring Cloud Bus
  - Provides simple way to notify clients to config changes
- Spring Cloud Netflix Eureka
  - Service Discovery – Allows applications to register themselves as clients

Module Outline

- Configuration Management
  - Challenges
  - Desired Solution
- **Spring Cloud Config**
  - Server Side
  - Client Side
- Repository Organization

Spring Cloud Config

- Designates a centralized server to serve-up configuration information
  - Configuration itself can be backed by source control
- Clients connect over HTTP and retrieve their configuration settings
  - In addition to their own, internal sources of configuration


Config Server
HTTP
HTTP
HTTP
Client Application (Spring)
Client Application (Spring)
Client Application (Another Technology)
Backing Files (Git, Flat Files, etc.)

Module Outline

- Configuration Management
  - Challenges
  - Desired Solution
- Spring Cloud Config
  - **Server Side**
  - Client Side
- Repository Organization

Spring Cloud Config Server

- Source available at GitHub: https://github.com/spring-cloud-samples/configserver
- Or, it is reasonably easy to build your own

Spring Cloud Config Server – Building, part 1

- Include minimal dependencies in your POM (or Gradle)
  - Spring **Cloud** Starter Parent
  - Spring Cloud Config Server

```
<parent>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-parent</artifactId>
  <version>Angel.SR4</version>
</parent>

<dependencies>
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-config-server</artifactId>
  </dependency>
</dependencies>
```

Spring Cloud Config Server – Building, part 2

- application.yml – indicates location of configuration repository

```
---
spring:
  cloud:
    config:
      server:
        git:
          uri: https://github.com/kennyk65/Microservices-With-Spring-Student-Files
          searchPaths: ConfigData
```

  – ...or application.properties

Spring Cloud Config Server – Building, part 3

- Add @EnableConfigServer

```
@SpringBootApplication
@EnableConfigServer
public class Application {

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }

}
```

- That's It!

Module Outline

- Configuration Management
  - Challenges
  - Desired Solution
- Spring Cloud Config
  - Server Side
  - **Client Side**
- Repository Organization

The Client Side – Building part 1

- Use the Spring **Cloud** Starter parent as a Parent POM:

```
<parent>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-parent</artifactId>
    <version>Angel.SR4</version>
</parent>
```

- ...OR use a Dependency management section:

```
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-parent</artifactId>
            <version>Angel.SR4</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
```

The Client Side – Building Part 2

- Include the Spring **Cloud** Starter for config:

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

- Configure application name and server location in bootstrap.properties / yml
    - So it is examined early in the startup process

```
# bootstrap.properties:
spring.application.name: lucky-word
spring.cloud.config.uri: http://localhost:8001
```

- That's It!
    - Client connects at startup for additional configuration settings.

The Client Side

- How Properties work in Spring Applications
  - Spring apps have an `Environment` object
  - `Environment` object contains multiple `PropertySources`
    - Typically populated from environment variables, system properties, JNDI, developer-specified property files, etc.
  - Spring Cloud Config Client library simply adds another `PropertySource`
    - By connecting to server over HTTP
    - http://<server>:<port>/<spring.application.name>/<profile>
  - Result: Properties described by server become part of client application's environment

Module Outline

- Configuration Management
  - Challenges
  - Desired Solution
- Spring Cloud Config
  - Server Side
  - Client Side
- **Repository Organization**

EnvironmentRepository - Choices

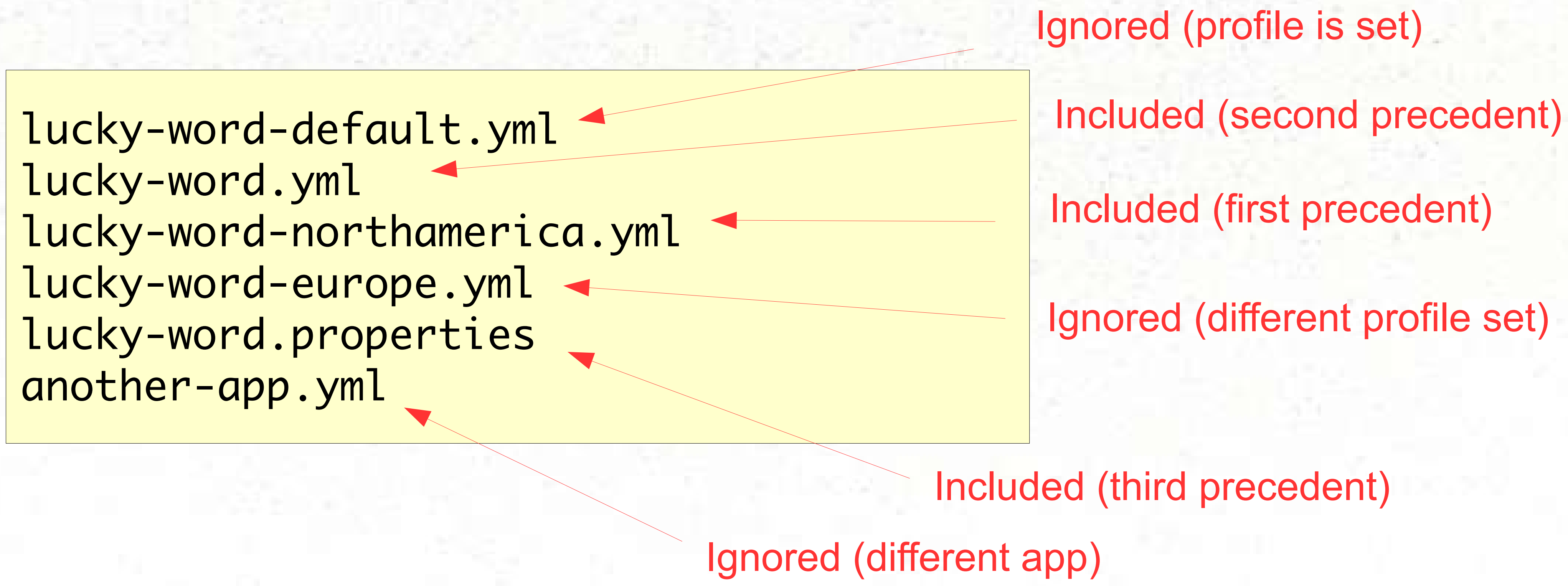
- Spring Cloud Config Server uses an `EnvironmentRepository`
  - Two implementations available: Git and Native (local files)
- Implement `EnvironmentRepository` to use other sources.

Environment Repository - Organization

- Configuration file naming convention:
  - <spring.application.name>-<profile>.yml
    - Or .properties (yml takes precedence)
  - spring.application.name – set by client application's bootstrap.yml (or .properties)
  - profile – Client's spring.profiles.active
    - (set various ways)
- Obtain settings from server:
  - http://<server>:<port>/<spring.application.name>/<profile>
  - Spring Cloud clients do this automatically on startup

Environment Repository – Organization Example

- Assume client application named “lucky-word” and profile set to “northamerica”
  - Spring client (automatically) requests
    - /lucky-word/northamerica



```
lucky-word-default.yml
lucky-word.yml
lucky-word-northamerica.yml
lucky-word-europe.yml
lucky-word.properties
another-app.yml
```

Ignored (profile is set)
Included (second precedent)
Included (first precedent)
Ignored (different profile set)
Included (third precedent)
Ignored (different app)

.yml vs .properties

- Settings can be stored in either YAML or standard Java properties files
  - Both have advantages
  - Config server will favor .yml over .properties

```
# .properties file
spring.config.name=aaa
spring.config.location=bbb
spring.profiles.active=ccc
spring.profiles.include=ddd
some.other.property=fff
```

```
# .yml file
---
spring:
  config:
    name: aaa
    location: bbb
  profiles:
    active: ccc
    include: ddd
some.other.property: fff
```

Profiles

- YAML Format can hold multiple profiles in a single file

```
# lucky-word-east.properties
lucky-word: Clover
```

```
# lucky-word-west.properties
lucky-word: Rabbit's Foot
```

```
# luckyword.yml
---
spring:
  profiles: east
lucky-word: Clover

---
spring:
  profiles: west
lucky-word: Rabbit's Foot
```

What about non-Java / non-Spring Clients?

- Spring Cloud Server exposes properties over simple HTTP interface
  - http://<server>:<port>/<spring.application.name>/<profile>
- Reasonably easy to call server from any application
  - Just not as automated as Spring.

What if the Config Server is Down?

- Spring Cloud Config Server should typically run on several instances
  - So downtime should be a non-issue
- Client application can control policy of how to handle missing config server
  - spring.cloud.config.failFast=true
  - Default is false
- Config Server settings override local settings
  - Strategy: provide local fallback settings.

Summary

- Spring Cloud Config offers centralized, versioned configuration for distributed applications
- Spring Cloud Config Server – Easy to Build
  - Backed by repository (Git or native) with .yml or .properties
- Spring Cloud Config Client –
  - Accesses Server, adds another PropertySource

Exercise

Setup your own Spring Cloud
Config Server, Client, and Repository

Instructions: Student Files, Lab 3