(../..)

Purpose of this lab

- Set up a client to pull configuration from a *Config Server* with a git backend.
- Use `@ConfigurationProperties` to create type-safe configuration.
- Use `@RefreshScope` to live-reload configuration.
- Notify horizontally-scaled applications to refresh configuration with *Cloud Bus*.

Set up the `app-config` repository

1. Create a local directory and initialize a git repo.

   ```
   mkdir ~/workspace/app-config
   cd ~/workspace/app-config
   git init
   ```

2. Create a repository named `app-config` in your GitHub account. Copy the ssh URL and add a remote to your local repo.

   ```
   git remote add origin https://github.com/<YOUR-USERNAME>/app-confi
   g.git
   ```

Set up your Config Server

1. Clone the spring cloud services repo and change to the *Config Server* folder.

```
git clone https://github.com/platform-acceleration-lab/apps-spring
-cloud-services-code.git
cd apps-spring-cloud-services-code/config-server
```

2. Note that in the pom.xml, the `spring-cloud-config-server` has been added to the classpath. This enables this application to embed a `config-server`.

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

3. Review the following file. Pay special attention to the `@EnableConfigServer` annotation.

   Hide ConfigServerApplication.java

   config-server/src/main/java/io/pivotal/ConfigServerApplication.java (https://github.com/platform-acceleration-lab/apps-spring-cloud-services-code/blob/master/config-server/src/main/java/io/pivotal/ConfigServerApplication.java)

   view on **GitHub** (https://github.com/platform-acceleration-lab/apps-spring-cloud-services-code/blob/master/config-server/src/main/java/io/pivotal/ConfigServerApplication.java)

```
package io.pivotal;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplicatio
n;
import org.springframework.cloud.config.server.EnableConfigServer;

@SpringBootApplication
@EnableConfigServer
public class ConfigServerApplication {

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

4. Configure the `config-server` application to get configuration from your `apps-config` repository. Edit your `application.yml` to point to your repository.

⊖ Hide application.yml

config-server/src/main/resources/application.yml (https://github.com/platform-acceleration-lab/apps-spring-cloud-services-code/blob/master/config-server/src/main/resources/application.yml)

view on **GitHub** (https://github.com/platform-acceleration-lab/apps-spring-cloud-services-code/blob/master/config-server/src/main/resources/application.yml)

```
server:
  port: 8888

spring:
  cloud:
    config:
      server:
        git:
          uri: https://github.com/<github-username>/app-config.git
```

5. Start the `config-server` and leave it running.

```
./mvnw clean spring-boot:run
```

6. To add some configuration, create a file called `hello-world.yml` in your `app-config` repo. Add the content below to the file and push the changes back to GitHub. Be sure to substitute your name for `<Your name>`.

```
name: <Your Name>
```

7. Confirm the `config-server` is up and configured with a backing git repository by navigating to http://localhost:8888/hello-world/default (http://localhost:8888/hello-world/default).

   The `config-server` exposes several endpoints (http://projects.spring.io/spring-cloud/docs/1.0.3/spring-cloud.html#_quick_start) to fetch configuration.

   In this case, we are manually calling one of those endpoints (`/{application}/{profile}[/{label}]`) to fetch configuration. We substituted our example client application `hello-world` as the `{application}` and the `default` profile as the `{profile}`. We did not specify the label to use so `master` is assumed.

   In the returned document, we see the configuration file `hello-world.yml` listed as a `propertySource` with the associated key/value pair.

Set up `greeting-config`

1. In a new terminal window, go to the Greeting Config app

   ```
   cd ~/workspace/apps-spring-cloud-services-code/greeting-config
   ```

   This application will consume configuration from the `config-server`.

   Review the `pom.xml` in the `greeting-config` app. Note the addition of `spring-cloud-services-starter-config-client` to the classpath.

   ```
   <dependency>
       <groupId>io.pivotal.spring.cloud</groupId>
       <artifactId>spring-cloud-services-starter-config-client</artif
   actId>
   </dependency>
   ```

2. Review the following file/

   ● Hide bootstrap.yml
   greeting-config/src/main/resources/bootstrap.yml (https://github.com/platform-acceleration-lab/apps-spring-cloud-services-code/blob/master/greeting-config/src/main/resources/bootstrap.yml)

   view on **GitHub** (https://github.com/platform-acceleration-lab/apps-spring-cloud-services-code/blob/master/greeting-config/src/main/resources/bootstrap.yml)

   ```
   spring:
     application:
       name: greeting-config
   ```

   The application name is defined by `spring.application.name`. This name will be used to locate config files for the `greeting-config` application.

   Absent from the bootstrap.yml is the `spring.cloud.config.uri`, which defines how `greeting-config` reaches the `config-server`. Since there is no `spring.cloud.config.uri` defined in this file, the default value of `http://localhost:8888` is used. Notice that this is the same host and port of the `config-server` application.

3. Start the `greeting-config` application.

   ```
    ./mvnw clean spring-boot:run
   ```

4. Confirm the `greeting-config` app is up by browsing to http://localhost:8080 (http://localhost:8080). You will be prompted to authenticate because `spring-cloud-services-starter-config-client` has a dependency on Spring Security (http://projects.spring.io/spring-security/).

   By default, this causes all endpoints to be protected by HTTP Basic Authentication.

5. For these labs we do not need Spring Security's default behavior of securing every endpoint, so we will turn it off. Add a `greeting-config.yml` to your `app-config` repo with the following content:

   ```
   security:
     basic:
       enabled: false

   management:
     security:
       enabled: false
   ```

   Push the changes to GitHub and restart the greeting config application.

6. Navigate to localhost:8080 (http://localhost:8080).

   At this point, you connected the `greeting-config` application with the `config-server`. This can be confirmed by reviewing the logs of the `greeting-config` application.

   We will next add additional configuration parameters/values and see the effects in out client application `greeting-config`.

Changing Logging Levels

Next you will change the logging level of the `greeting-config` application.

1. View the `getGreeting()` method of the `GreetingController` class.

   ⊖ Hide GreetingController.java
   greeting-config/src/main/java/io/pivotal/greeting/GreetingController.java (https://github.com/platform-acceleration-lab/apps-spring-cloud-services-code/blob/master/greeting-config/src/main/java/io/pivotal/greeting/GreetingController.java)

   view on **GitHub** (https://github.com/platform-acceleration-lab/apps-spring-cloud-services-code/blob/master/greeting-config/src/main/java/io/pivotal/greeting/GreetingController.java)

   ```
   package io.pivotal.greeting;

   import io.pivotal.fortune.FortuneService;
   ```

```java
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.context.properties.EnableConfigura
tionProperties;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
@EnableConfigurationProperties(GreetingProperties.class)
public class GreetingController {

    private Logger logger = LoggerFactory.getLogger(getClass());
    private GreetingProperties greetingProperties;
    private FortuneService fortuneService;

    @Autowired
    public GreetingController(GreetingProperties greetingPropertie
s, FortuneService fortuneService) {
        this.greetingProperties = greetingProperties;
        this.fortuneService = fortuneService;
    }



    @RequestMapping("/")
    public String getGreeting(Model model) {

        logger.debug("Adding greeting");
        model.addAttribute("msg", "Greetings!!!");

        if (greetingProperties.isDisplayFortune()) {
            logger.debug("Adding fortune");
            model.addAttribute("fortune", fortuneService.getFortun
e());
        }

        //resolves to the greeting.vm velocity template
        return "greeting";
    }
}
```

We want to see these debug messages, but `DEBUG` logs are not shown by default. Next we will change the log level to `DEBUG` using configuration.

All log output will be directed to `System.out` & `System.error` by default, so logs will be output to the terminal window(s).

2. In your `app-config` repo, add the following content to the `greeting-config.yml` file:

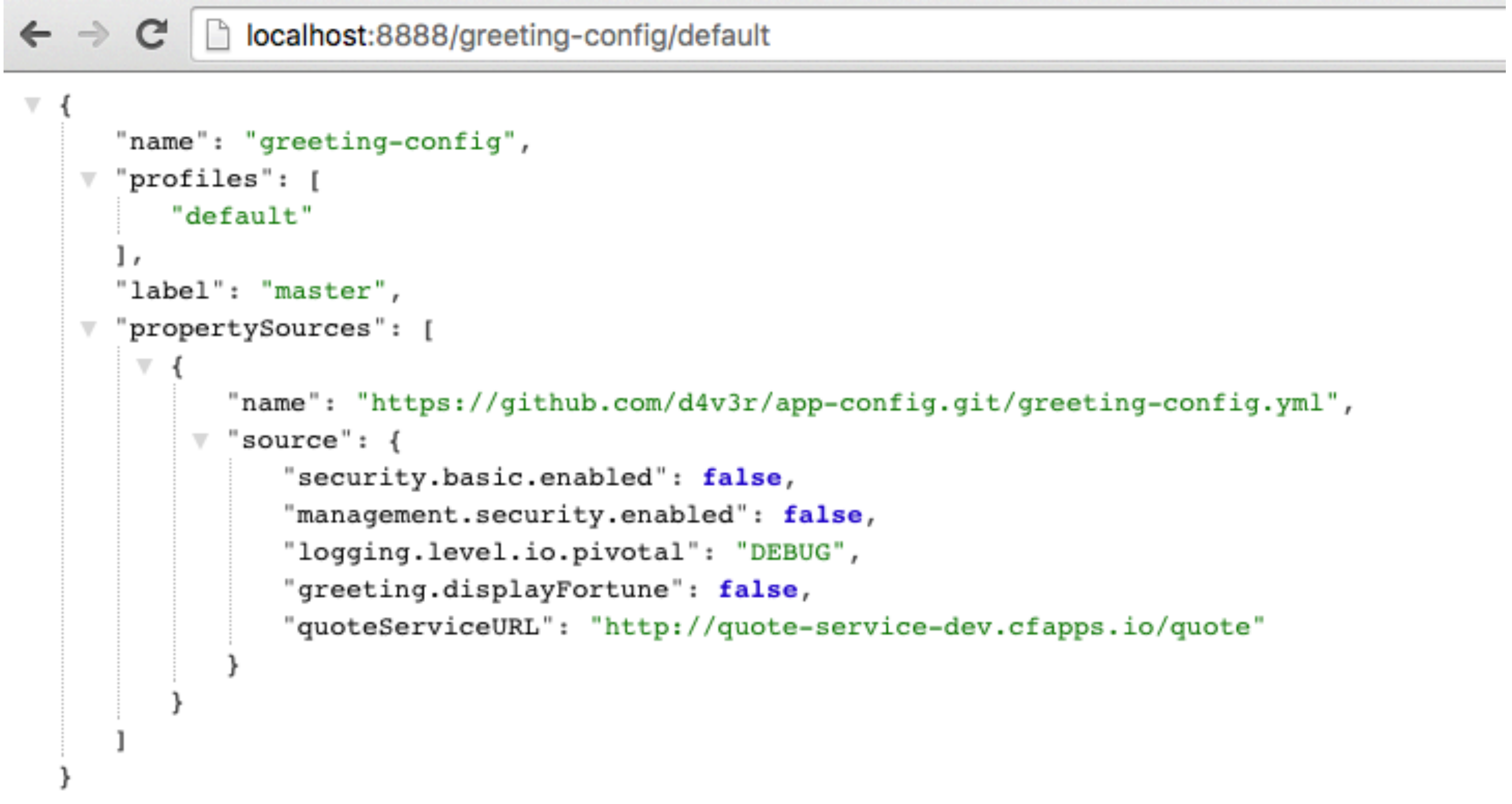
   ```
   security:
     basic:
       enabled: false

   management:
     security:
       enabled: false

   logging:
     level:
       io:
         pivotal: DEBUG

   greeting:
     displayFortune: false

   quoteServiceURL: http://quote-service-dev.cfapps.io/quote
   ```

   Push the changes back to GitHub.

   We have added several configuration parameters that will be used throughout this lab. For this exercise, we have set the log level for classes in the `io.pivotal` package to `DEBUG`.

3. While watching the `greeting-config` terminal, refresh the http://localhost:8080 (http://localhost:8080/) url. Notice there are no `DEBUG` logs yet.

4. Does the `config-server` see the change in your git repo? Check what the `config-server` is serving by browsing to http://localhost:8888/greeting-config/default (http://localhost:8888/greeting-config/default)



```
{
    "name": "greeting-config",
    "profiles": [
        "default"
    ],
    "label": "master",
    "propertySources": [
        {
            "name": "https://github.com/d4v3r/app-config.git/greeting-config.yml",
            "source": {
                "security.basic.enabled": false,
                "management.security.enabled": false,
                "logging.level.io.pivotal": "DEBUG",
                "greeting.displayFortune": false,
                "quoteServiceURL": "http://quote-service-dev.cfapps.io/quote"
            }
        }
    ]
}
```

The propertySources value has changed! The `config-server` has picked up the changes to the git repo. (If you do not see the change, verify that you have pushed your `app-config` repo to GitHub.)

5. Review the following file:

   Hide pom.xml

   greeting-config/pom.xml (https://github.com/platform-acceleration-lab/apps-spring-cloud-services-code/blob/master/greeting-config/pom.xml)

   view on **GitHub** (https://github.com/platform-acceleration-lab/apps-spring-cloud-services-code/blob/master/greeting-config/pom.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http
://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http
://maven.apache.org/xsd/maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>

        <groupId>io.pivotal</groupId>
        <artifactId>greeting-config</artifactId>
        <version>0.0.1-SNAPSHOT</version>
        <packaging>jar</packaging>

        <name>greeting-config</name>
        <description>Spring Cloud Config Client: Greeting Config</
```

```
description>

        <parent>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-parent</artifactId
>
                <version>1.4.2.RELEASE</version>
                <relativePath /> <!-- lookup parent from repositor
y -->
        </parent>


        <properties>
                <project.build.sourceEncoding>UTF-8</project.build
.sourceEncoding>
                <project.reporting.outputEncoding>UTF-8</project.r
eporting.outputEncoding>
                <java.version>1.8</java.version>
        </properties>

        <dependencies>

                <dependency>
                        <groupId>io.pivotal.spring.cloud</groupId>
                        <artifactId>spring-cloud-services-starter-
config-client</artifactId>
                </dependency>


                <dependency>
                        <groupId>org.springframework.cloud</groupI
d>
                        <artifactId>spring-cloud-starter-config</a
rtifactId>
                </dependency>

                <dependency>
                        <groupId>org.springframework.boot</groupId
>
                        <artifactId>spring-boot-starter-actuator</
artifactId>
                </dependency>
                <dependency>
                        <groupId>org.springframework.boot</groupId
```

```
>
                        <artifactId>spring-boot-starter-web</artif
actId>
                </dependency>
                <dependency>
                        <groupId>org.springframework.boot</groupId
>
                        <artifactId>spring-boot-starter-velocity</
artifactId>
                </dependency>
                <dependency>
                        <groupId>org.springframework.boot</groupId
>
                        <artifactId>spring-boot-starter-test</arti
factId>
                        <scope>test</scope>
                </dependency>
                <dependency>
                        <groupId>org.springframework.boot</groupId
>
                        <artifactId>spring-boot-configuration-proc
essor</artifactId>
                        <optional>true</optional>
                </dependency>
        </dependencies>

        <dependencyManagement>
                <dependencies>
                        <dependency>
                                <groupId>org.springframework.cloud
</groupId>
                                <artifactId>spring-cloud-dependenc
ies</artifactId>
                                <version>Camden.SR3</version>
                                <type>pom</type>
                                <scope>import</scope>
                        </dependency>
                        <dependency>
                                <groupId>io.pivotal.spring.cloud</
groupId>
                                <artifactId>spring-cloud-services-
dependencies</artifactId>
                                <version>1.3.1.RELEASE</version>
                                <type>pom</type>
                                <scope>import</scope>
                        </dependency>
                </dependencies>
        </dependencyManagement>

        <build>
                <plugins>
                        <plugin>
                                <groupId>org.springframework.boot<
/groupId>
                                <artifactId>spring-boot-maven-plug
in</artifactId>
                        </plugin>
                </plugins>
        </build>


</project>
```

Note the inclusion of the `actuator` dependency. The `actuator` adds several additional endpoints to the application for operational visibility and tasks that need to be carried out. In this case, we have added the actuator so that we can use the `/refresh` endpoint, which allows us to refresh the application config on demand.

6. For the `greeting-config` application to pick up new configuration changes, it must be told to do so. Notify `greeting-config` app to pick up the new config by sending a POST request to the `greeting-config` `/refresh` endpoint. Run the following command:

```
curl -X POST http://localhost:8080/refresh
```

7. Refresh the `greeting-config` localhost:8080 (http://localhost:8080/) url while viewing the `greeting-config` terminal. You should now see the debug log message `Adding greeting`.

Turning on a Feature with `@ConfigurationProperties`

Use of `@ConfigurationProperties` is a common way to provide type-safe configuration in Spring applications. Beans annotated with `@ConfigurationProperties` are automatically reloaded when the application config is refreshed.

1. Review the following file:

● Hide GreetingProperties.java

greeting-config/src/main/java/io/pivotal/greeting/GreetingProperties.java (https://github.com/platform-acceleration-lab/apps-spring-cloud-services-code/blob/master/greeting-config/src/main/java/io/pivotal/greeting/GreetingProperties.java)

view on **GitHub** (https://github.com/platform-acceleration-lab/apps-spring-cloud-services-code/blob/master/greeting-config/src/main/java/io/pivotal/greeting/GreetingProperties.java)

```
package io.pivotal.greeting;

import org.springframework.boot.context.properties.ConfigurationPr
operties;

@ConfigurationProperties(prefix = "greeting")
public class GreetingProperties {

    private boolean displayFortune;

    public boolean isDisplayFortune() {
        return displayFortune;
    }

    public void setDisplayFortune(boolean displayFortune) {
        this.displayFortune = displayFortune;
    }
}
```

Use of the `@ConfigurationProperties` annotation allows for reading of configuration values. Configuration keys are a combination of the `prefix` and the field names. In this case, there is one field (`displayFortune`). Therefore `greeting.displayFortune` is used to turn the display of fortunes on/off.

2. Review the greeting controller.

● Hide GreetingController.java

greeting-config/src/main/java/io/pivotal/greeting/GreetingController.java (https://github.com/platform-acceleration-lab/apps-spring-cloud-services-code/blob/master/greeting-config/src/main/java/io/pivotal/greeting/GreetingController.java)

view on **GitHub** (https://github.com/platform-acceleration-lab/apps-spring-cloud-services-code/blob/master/greeting-config/src/main/java/io/pivotal/greeting/GreetingController.java)

```
package io.pivotal.greeting;

import io.pivotal.fortune.FortuneService;
```

```java
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.context.properties.EnableConfigura
tionProperties;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
@EnableConfigurationProperties(GreetingProperties.class)
public class GreetingController {

    private Logger logger = LoggerFactory.getLogger(getClass());
    private GreetingProperties greetingProperties;
    private FortuneService fortuneService;

    @Autowired
    public GreetingController(GreetingProperties greetingPropertie
s, FortuneService fortuneService) {
        this.greetingProperties = greetingProperties;
        this.fortuneService = fortuneService;
    }



    @RequestMapping("/")
    public String getGreeting(Model model) {

        logger.debug("Adding greeting");
        model.addAttribute("msg", "Greetings!!!");

        if (greetingProperties.isDisplayFortune()) {
            logger.debug("Adding fortune");
            model.addAttribute("fortune", fortuneService.getFortun
e());
        }

        //resolves to the greeting.vm velocity template
        return "greeting";
    }
}
```

Note how the `greetingProperties.isDisplayFortune()` is used to turn the display of fortunes on/off. There are times when you want to turn features on/off on demand. In this case, we want the fortune feature "on" with our greeting.

3. In your `app-config` repo change `greeting.displayFortune` from `false` to `true` in the `greeting-config.yml`.

   ```
   greeting:
     displayFortune: true
   ```

   Push the changes to GitHub.

4. Notify the `greeting-config` app to pick up the new config by POSTing to the `/refresh` endpoint.

   ```
   curl -X POST http://localhost:8080/refresh
   ```

5. Refresh the localhost:8080 (http://localhost:8080/) url and see the fortune included.

Reinitializing Beans with `@RefreshScope`

Next we will use the `config-server` to obtain a service URI.

1. Review the `QuoteService`.

   ● Hide QuoteService.java

   greeting-config/src/main/java/io/pivotal/quote/QuoteService.java (https://github.com/platform-acceleration-lab/apps-spring-cloud-services-code/blob/master/greeting-config/src/main/java/io/pivotal/quote/QuoteService.java)

   view on **GitHub** (https://github.com/platform-acceleration-lab/apps-spring-cloud-services-code/blob/master/greeting-config/src/main/java/io/pivotal/quote/QuoteService.java)

```
package io.pivotal.quote;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.cloud.context.config.annotation.Refresh
Scope;
import org.springframework.stereotype.Service;
import org.springframework.web.client.RestTemplate;

@Service
@RefreshScope
public class QuoteService {

    private Logger logger = LoggerFactory.getLogger(getClass());
    private String quoteServiceURL;
    private RestTemplate restTemplate;

    @Autowired
    public QuoteService(@Value("${quoteServiceURL:}") String quote
ServiceURL) {
        this.quoteServiceURL = quoteServiceURL;
        this.restTemplate = new RestTemplate();
    }


    public String getQuoteServiceURL() {
        return quoteServiceURL;
    }

    public Quote getQuote() {
        logger.info("quoteServiceURL: {}", quoteServiceURL);
        return restTemplate.getForObject(quoteServiceURL, Quote.cl
ass);
    }
}
```

Note that the `QuoteService` uses the `@RefreshScope` annotation. Beans with the `@RefreshScope` annotation will be recreated when configuration is refreshed. The `@Value` annotation allows for injecting the config value named `quoteServiceURL`.

In this case, we are using a third party service to get quotes. We want to keep our environments aligned with the third party, so we are going to override configuration values by profile (next section).

2. Review the `QuoteController`, which uses the `QuoteService`.

   ● Hide QuoteController.java
   greeting-config/src/main/java/io/pivotal/quote/QuoteController.java (https://github.com/platform-acceleration-lab/apps-spring-cloud-services-code/blob/master/greeting-config/src/main/java/io/pivotal/quote/QuoteController.java)

   view on **GitHub** (https://github.com/platform-acceleration-lab/apps-spring-cloud-services-code/blob/master/greeting-config/src/main/java/io/pivotal/quote/QuoteController.java)

```
package io.pivotal.quote;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class QuoteController {

    private QuoteService quoteService;

    @Autowired
    public QuoteController(QuoteService quoteService) {
        this.quoteService = quoteService;
    }


    @RequestMapping("/random-quote")
    public String getView(Model model) {
        model.addAttribute("quote", quoteService.getQuote());
        model.addAttribute("uri", quoteService.getQuoteServiceURL(
));
        return "quote";
    }
}
```

3. Navigate to localhost:8080/random-quote (http://localhost:8080/random-quote). Note the uri that is serving the quotes: `http://quote-service-dev.cfapps.io/quote`

Override Configuration Values By Profile

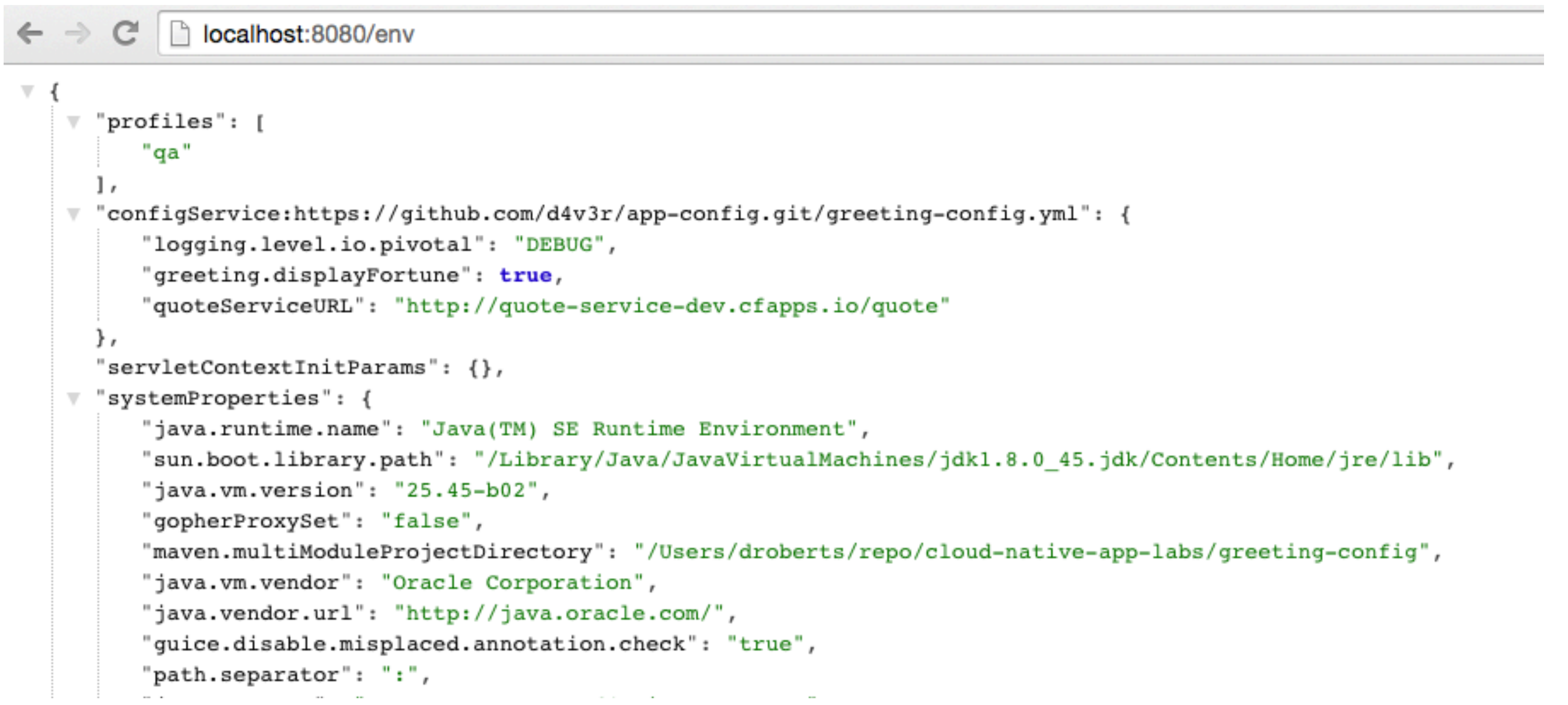
1. Stop the `greeting-config` application.
2. Set the active profile to qa for the `greeting-config` application. In the example below, we use an environment variable to set the active profile.
   - MacOS:

     ```
     SPRING_PROFILES_ACTIVE=qa  ./mvnw clean spring-boot:run
     ```

   - Windows:

     ```
     set SPRING_PROFILES_ACTIVE=qa
      ./mvnw clean spring-boot:run
     ```

3. Make sure the profile is set by browsing to the localhost:8080/env (http://localhost:8080/env) endpoint (provided by `actuator`). Under profiles `qa` should be listed.

   localhost:8080/env

   ```
   {
     "profiles": [
       "qa"
     ],
     "configService:https://github.com/d4v3r/app-config.git/greeting-config.yml": {
       "logging.level.io.pivotal": "DEBUG",
       "greeting.displayFortune": true,
       "quoteServiceURL": "http://quote-service-dev.cfapps.io/quote"
     },
     "servletContextInitParams": {},
     "systemProperties": {
       "java.runtime.name": "Java(TM) SE Runtime Environment",
       "sun.boot.library.path": "/Library/Java/JavaVirtualMachines/jdk1.8.0_45.jdk/Contents/Home/jre/lib",
       "java.vm.version": "25.45-b02",
       "gopherProxySet": "false",
       "maven.multiModuleProjectDirectory": "/Users/droberts/repo/cloud-native-app-labs/greeting-config",
       "java.vm.vendor": "Oracle Corporation",
       "java.vendor.url": "http://java.oracle.com/",
       "guice.disable.misplaced.annotation.check": "true",
       "path.separator": ":",
   ```

4. In your fork of the `app-config` repository, create a new file: `greeting-config-qa.yml`. Fill it in with the following content:

   ```
   quoteServiceURL: http://quote-service-qa.cfapps.io/quote
   ```

   Make sure to commit and push to GitHub.

5. Browse to localhost:8080/random-quote (http://localhost:8080/random-quote). Quotes are still being served from `http://quote-service-dev.cfapps.io/quote`.

6. Refresh the application configuration values

   ```
   curl -X POST http://localhost:8080/refresh
   ```

7. Refresh the localhost:8080/random-quote (http://localhost:8080/random-quote) url. Quotes are now being served from QA.

8. Stop both the `config-server` and `greeting-config` applications.

   Configuration from `greeting-config.yml` was overridden by a configuration file that was more specific (`greeting-config-qa.yml`).

Deploy the `greeting-config` Application to PCF

1. Package the `greeting-config` application. Execute the following from the `greeting-config` directory:

   ```
    ./mvnw clean package
   ```

2. Deploy the `greeting-config` application to PCF, without starting the application:

   ```
   cf push greeting-config -p target/greeting-config-0.0.1-SNAPSHOT.j
   ar -m 512M --random-route --no-start
   ```

3. Create a Config Server Service Instance

   Using the cf cli, do the following (for help review the docs (http://docs.pivotal.io/spring-cloud-services/config-server/creating-an-instance.html)):

   Create a config server:

   First, create a `cfg-svr-config.json` file in the `greeting-config` directory. The contents of the file look something like this:

   ```
   { "git": { "uri": "https://github.com/<username>/app-config.git" }
   }
   ```

   Then:

```
cf create-service p-config-server standard config-server -c ./cfg-svr-config.json
```

Feel free to name your service anything you like (it does not have to be named `config-server`). The Config Server instance will take a few moments to initialize and then be ready for use.

You can invoke either the `cf services` or `cf service` commands to view the status of the service you just created.

In addition, you can visit your Config Server's service dashboard in the Apps Manager to view its configuration and status:

In a browser, navigate to the apps manager and to your space. You should be able to find your config server service and its status there (it may be in a separate tab named `services`).

4. Bind the `config-server` service to the `greeting-config` app. This will enable the `greeting-config` app to read configuration values from the `config-server`.

   ```
   cf bind-service greeting-config config-server
   ```

   You can safely ignore the *TIP: Use 'cf restage' to ensure your env variable changes take effect* message from the CLI. Our app does not need to be restaged at this time because it is not currently running.

5. If you are using PWS or if your PCF instance is using self-signed SSL certificates, you should set the `TRUST_CERTS` environment variable to the API endpoint of your Elastic Runtime instance. You can quickly retrieve the API endpoint by running the command `cf api`.

   ```
   cf set-env greeting-config TRUST_CERTS <your api endpoint>
   ```

   Be sure to specify the api endpoint as a hostname and not a url, i.e. without the leading `https://` scheme. You can safely ignore the *TIP: Use 'cf restage' to ensure your env variable changes take effect* message from the CLI. Our app does not need to be restaged at this time.

   NOTE: All communication between Spring Cloud Services components are made through HTTPS. If you are on an environment that uses self-signed certs, the Java SSL trust store will not have those certificates. By adding the `TRUST_CERTS` environment variable a trusted domain is added to the Java trust store. For more information see the this portion (https://docs.pivotal.io/spring-cloud-services/config-server/writing-client-applications.html#self-signed-ssl-certificate) of the SCS documentation.

6. Start the `greeting-config` app.

```
cf start greeting-config
```

7. Browse to your `greeting-config` application. Are your configuration settings that were set when developing locally mirrored on PCF?

    - Is the log level for `io.pivotal` package set to `DEBUG`? Yes, this can be confirmed with `cf logs` command while refreshing the `greeting-config` `/` endpoint (`http://<your-random-greeting-config-url/`).
    - Is `greeting-config` app displaying the fortune? Yes, this can be confirmed by visiting the `greeting-config` `/` endpoint.
    - Is the `greeting-config` app serving quotes from `http://quote-service-qa.cfapps.io/quote`? No, this can be confirmed by visiting the `greeting-config` `/random-quote` endpoint. Why not? When developing locally we used an environment variable to set the active profile, we need to do the same on PCF.

    ```
    cf set-env greeting-config SPRING_PROFILES_ACTIVE qa
    cf restart greeting-config
    ```

    You can safely ignore the *TIP: Use 'cf restage' to ensure your env variable changes take effect* message from the CLI. Our app does not need to be restaged but just re-started.

    Then confirm quotes are being served from `http://quote-service-qa.cfapps.io/quote`

Refreshing Application Configuration at Scale with Cloud Bus

Until now you have been notifying your application to pick up new configuration by POSTing to the `/refresh` endpoint.

When running several instances of your application, this poses several problems:

- Refreshing each individual instance is time consuming and too much overhead
- When running on Cloud Foundry you do not have control over which instances you hit when sending the POST request due to load balancing provided by the `router`

Cloud Bus addresses the issues listed above by providing a single endpoint to refresh all application instances via a pub/sub notification.

1. Create a RabbitMQ service instance *or* a CloudAMQP service instance and bind it to `greeting-config`

   ```
   cf create-service p.rabbitmq standard cloud-bus
   cf bind-service greeting-config cloud-bus
   ```

   You can safely ignore the *TIP: Use 'cf restage' to ensure your env variable changes take effect* message from the CLI. Our app does not need to be restaged. We will push it again with new functionality in a moment.

2. Include the cloud bus dependency in the `greeting-config/pom.xml`. *You will need to paste this in your file.*

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

3. Repackage the `greeting-config` application:

   ```
    ./mvnw clean package
   ```

4. Deploy the application and scale the number of instances.

   ```
   cf push greeting-config -p target/greeting-config-0.0.1-SNAPSHOT.j
   ar -i 3
   ```

5. Observe the logs that are generated by refreshing the `greeting-config` `/` endpoint several times in your browser and tailing the logs. Allow this process to run through the next few steps.

   - MacOS

     ```
     cf logs greeting-config | grep GreetingController
     ```

   - Windows

     ```
     cf logs greeting-config
     ```

All app instances are creating debug statements. Notice the `[App/X]`. It denotes which app instance is logging.

6. Turn logging down. In your fork of the `app-config` repo edit the `greeting-config.yml`. Set the log level to `INFO` and push to Github.

   ```
   logging:
    level:
      io:
        pivotal: INFO
   ```

7. Notify applications to pickup the change. Open a new terminal window. Send a POST to the `greeting-config` `/bus/refresh` endpoint. Use your `greeting-config` URL not the literal below.

   ```
   curl -X POST http://greeting-config-hypodermal-subcortex.cfapps.io
   /bus/refresh
   ```

8. Refresh the `greeting-config` `/` endpoint several times in your browser. No more logs!