Service Discovery

Table of Contents

1. What You Will Learn

- How to embed Eureka in a Spring Boot application
- How to register services (`greeting-service` and `fortune-service`) with Eureka
- How to discover services (`fortune-service`) with Eureka
- How to use Spring Cloud Services to provision a Service Registry

2. Set up the `app-config` Repo

1. In your `app-config` repository, create the file `application.yml` with the following contents:

```yaml
security:
  basic:
    enabled: false

management:
  security:
    enabled: false

logging:
  level:
    io:
      pivotal: DEBUG
```

Then commit and push back to Github.

About application.yml

In a `config-server` backing repository, the file name `application.yml` is special: it's a place to put common configuration that applies to all applications. In this case, we are dropping security on all the endpoints. We're also setting up default logging in one place.

In the Spring Cloud Config Lab, we used application-specific configuration files:

- One based on the application name `greeting-config.yml`
- One based on the application name + profile `greeting-config-qa.yml`

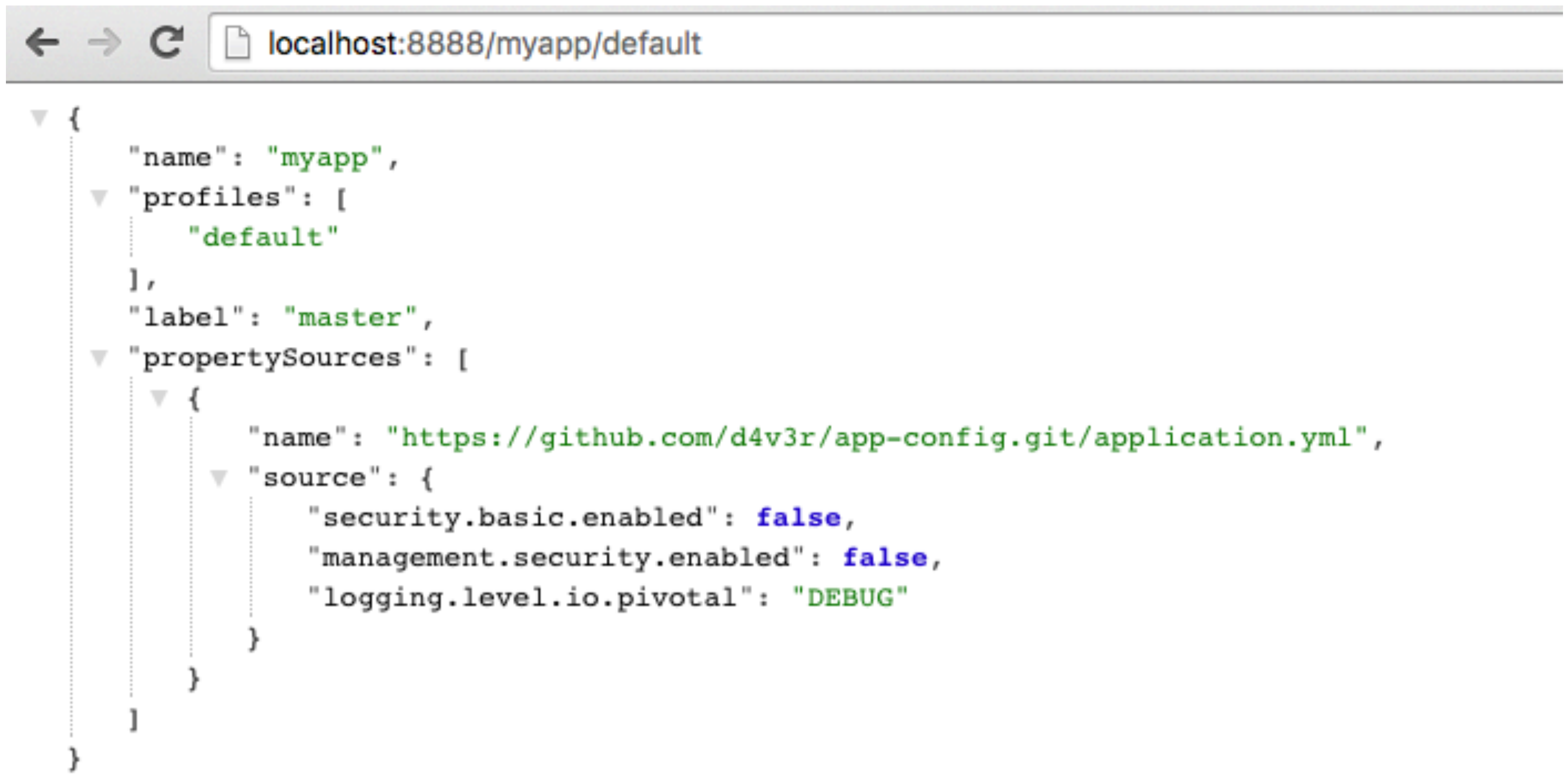
Application-specific files override default configuration settings. So basically the Spring Config Server provides a flexible approach to configuration where profile-specific configuration is overlayed atop app-specific configuration, and at the very bottom we have common settings in `application.yml`.

3. Set up `config-server`

1. Start the `config-server` in a terminal window. You may have a terminal window still open from the previous lab.

```
$ cd config-server
$ mvn spring-boot:run
```

2. Verify the `config-server` is up. Open a browser and fetch http://localhost:8888/myapp/default



```
{
    "name": "myapp",
    "profiles": [
        "default"
    ],
    "label": "master",
    "propertySources": [
        {
            "name": "https://github.com/d4v3r/app-config.git/application.yml",
            "source": {
                "security.basic.enabled": false,
                "management.security.enabled": false,
                "logging.level.io.pivotal": "DEBUG"
            }
        }
    ]
}
```

Note that a random application name was used and it picked up configuration from `application.yml`.

4. Set up `service-registry`

1. Review the `service-registry` project's maven pom file (`pom.xml`).

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

By adding `spring-cloud-starter-eureka-server` as a dependency, this application is eligible to embed a Eureka server.

2. Review the file `ServiceRegistryApplication.java`. Note below, the use of the `@EnableEurekaServer` annotation that makes this application a Eureka server.

   ```java
   package io.pivotal;

   import org.springframework.boot.SpringApplication;
   import org.springframework.boot.autoconfigure.SpringBootApplication;
   import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;

   @SpringBootApplication
   @EnableEurekaServer
   public class ServiceRegistryApplication {

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

3. Review the `application.yml` file:

   ```yaml
   server:
     port: 8761

   eureka:
     instance:
       hostname: localhost
     client:
       registerWithEureka: false
       fetchRegistry: false
       serviceUrl:
         defaultZone: http://${eureka.instance.hostname}:${server.port}/eureka/
   ```

   The above configuration configures Eureka to run in standalone mode.

   ### About Eureka

   Eureka is designed for peer awareness (running multiple instances with knowledge of each other) to further increase availability. Because of this, Eureka is not only a server but a client as well. Therefore, Eureka Servers will be clients to each other. `Eureka Server A` ⇄ `Eureka Server B`.

   For the purposes of this lab, we simplify that configuration to run in standalone mode.

   Standalone mode still offers a high degree of resilience with:

   - Heartbeats between the client and server to keep registrations up to date

- Client side caching, so that clients don’t go to Eureka for every lookup
- By running in Pivotal Cloud Foundry which is designed to keep applications up by design

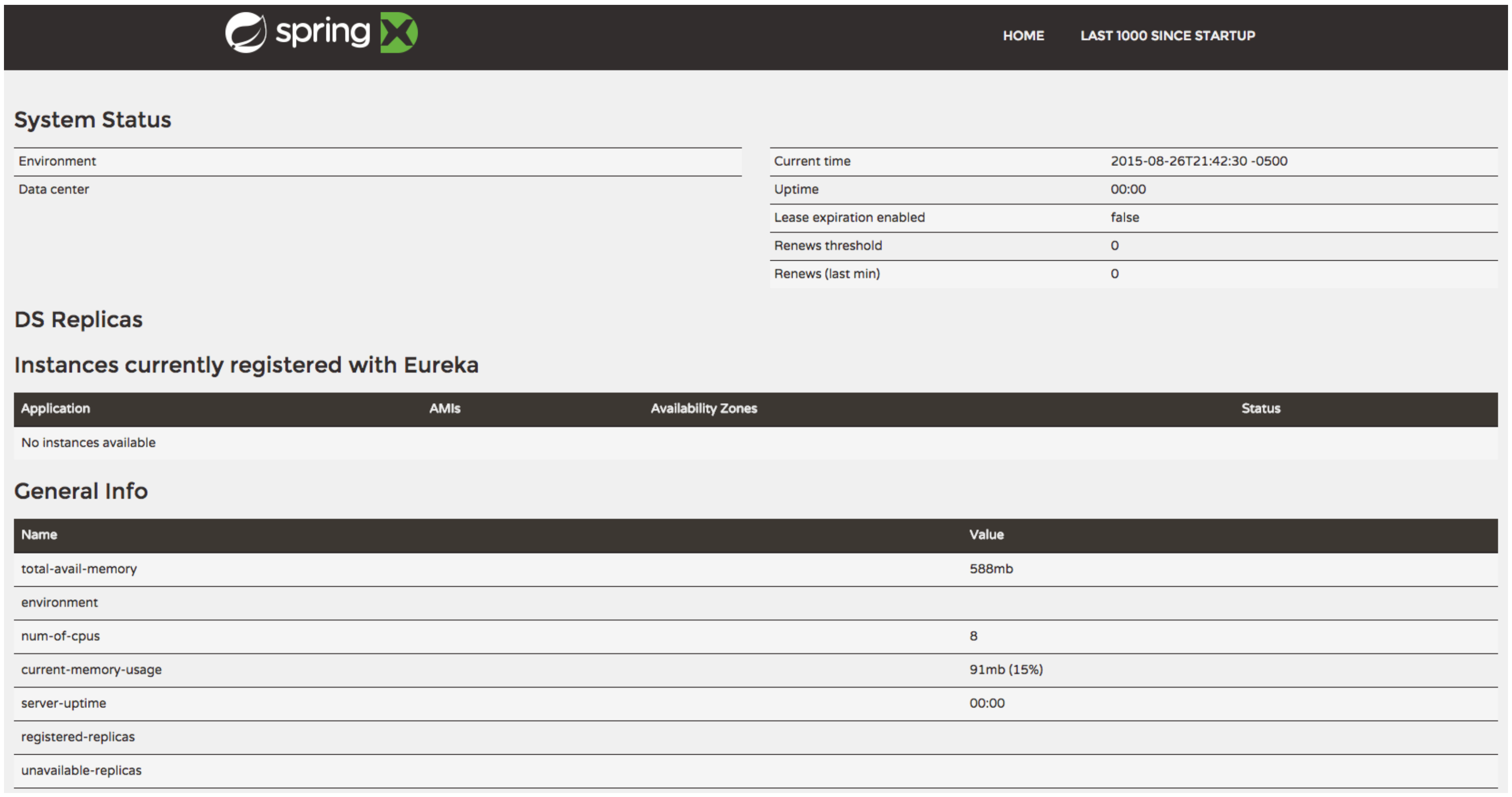
Understanding the configuration parameters

- `eureka.instance.hostname` - the hostname for this service. In this case, what host to use to reach our standalone Eureka instance.
- `eureka.client.registerWithEureka` - should this application (our standalone Eureka instance) register with Eureka
- `eureka.client.fetchRegistry` - should this application (our stand alone Eureka instance) fetch the registry (for how to discover services)
- `eureka.client.serviceUrl.defaultZone` - the Eureka instance to use for registering and discovering services. Notice it is pointing to itself (`localhost`, `8761`).

4. Open a new terminal window. Start the `service-registry`.

```
$ cd service-registry
$ mvn spring-boot:run
```

5. Verify the `service-registry` is up. Browse to http://localhost:8761/



System Status

Environment	
Data center	

Current time	2015-08-26T21:42:30 -0500
Uptime	00:00
Lease expiration enabled	false
Renews threshold	0
Renews (last min)	0

DS Replicas

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
No instances available			

General Info

Name	Value
total-avail-memory	588mb
environment	
num-of-cpus	8
current-memory-usage	91mb (15%)
server-uptime	00:00
registered-replicas	
unavailable-replicas	

5. Set up `fortune-service`

1. Review the `fortune-service` project's `bootstrap.yml` file. This app also uses the `config-server`.

   ```
   server:
     port: 8787
   spring:
     application:
       name: fortune-service
   ```

   `spring.application.name` is the name the application will use when registering with Eureka.

2. Review the project's `pom.xml` file. By adding `spring-cloud-services-starter-service-registry` as a dependency, this application is eligible to register and discover services with the `service-registry`.

   ```
   <dependency>
       <groupId>io.pivotal.spring.cloud</groupId>
       <artifactId>spring-cloud-services-starter-service-registry</artifactId>
   </dependency>
   ```

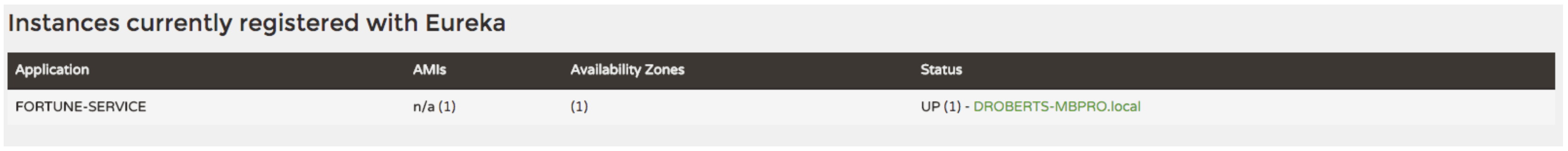
3. Review the file `FortuneServiceApplication.java`. Notice the `@EnableDiscoveryClient`. This enables a discovery client that registers the `fortune-service` with the `service-registry` application.

```
package io.pivotal;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;

@SpringBootApplication
@EnableDiscoveryClient
public class FortuneServiceApplication {

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

}
```

4. Open a new terminal window. Start the `fortune-service`

```
$ cd fortune-service
$ mvn spring-boot:run
```

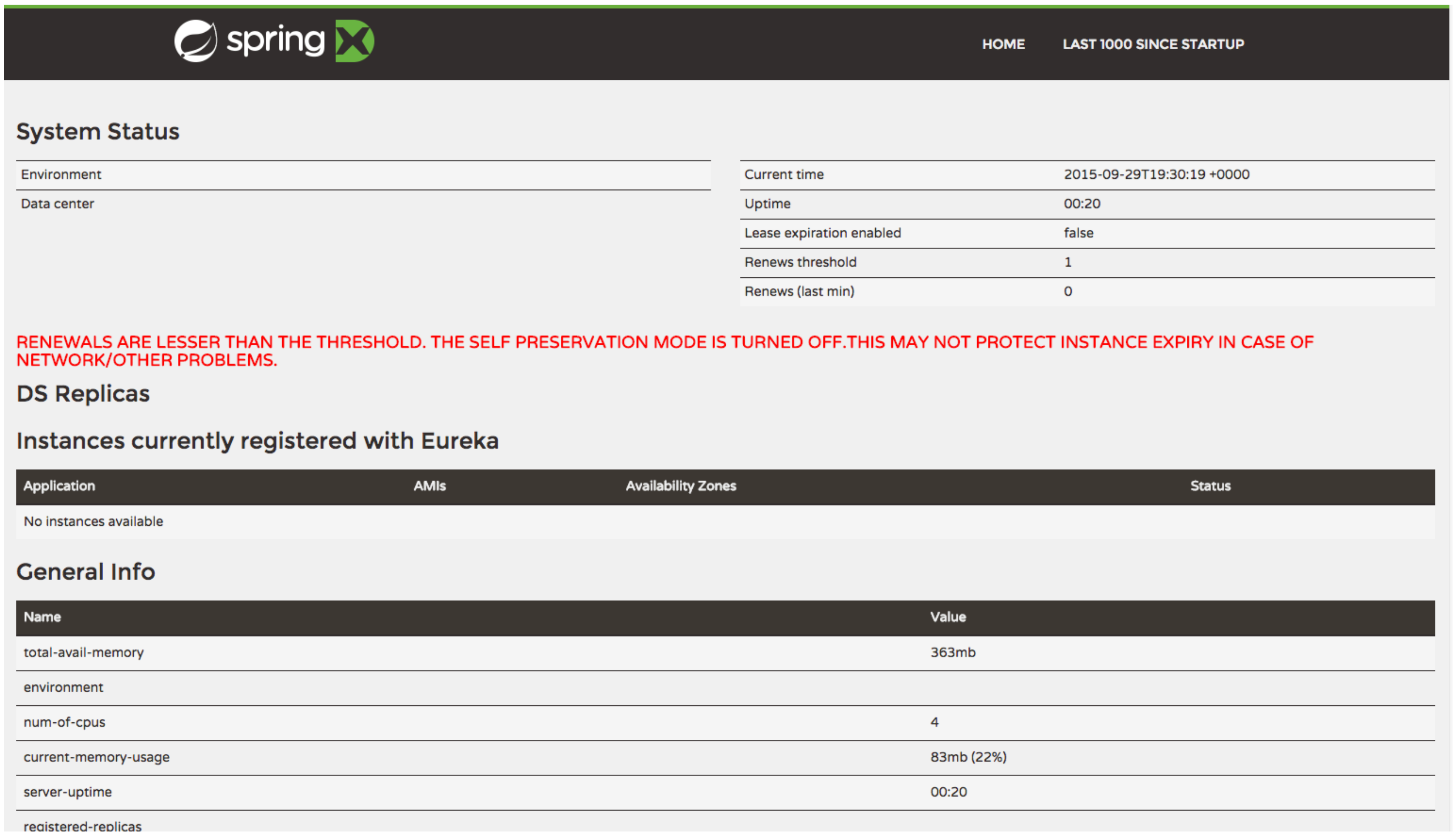
5. After the a few moments, check the `service-registry` dashboard. Confirm the `fortune-service` is registered.



Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
FORTUNE-SERVICE	n/a (1)	(1)	UP (1) - DROBERTS-MBPRO.local

The Eureka Dashboard may report a warning, because we aren't setup with multiple peers. This can safely be ignored.



System Status

Environment	
Data center	

Current time	2015-09-29T19:30:19 +0000
Uptime	00:20
Lease expiration enabled	false
Renews threshold	1
Renews (last min)	0

RENEWALS ARE LESSER THAN THE THRESHOLD. THE SELF PRESERVATION MODE IS TURNED OFF.THIS MAY NOT PROTECT INSTANCE EXPIRY IN CASE OF NETWORK/OTHER PROBLEMS.

DS Replicas

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
No instances available			

General Info

Name	Value
total-avail-memory	363mb
environment	
num-of-cpus	4
current-memory-usage	83mb (22%)
server-uptime	00:20
registered-replicas	

The endpoint http://localhost:8761/eureka/apps provides a raw (xml) view of the application registry that eureka maintains.

6. Set up `greeting-service`

1. Review `greeting-service` project's `bootstrap.yml` file. The name of this app is `greeting-service`. It also uses the `config-server`.

```
spring:
  application:
    name: greeting-service
```

2. Review the `pom.xml` file. By adding `spring-cloud-services-starter-service-registry`, this application is eligible to register and discover services with the `service-registry`.

```
<dependency>
    <groupId>io.pivotal.spring.cloud</groupId>
    <artifactId>spring-cloud-services-starter-service-registry</artifactId>
</dependency>
```

3. Review the file `GreetingServiceApplication.java`. Notice the `@EnableDiscoveryClient`. This enables a discovery client that registers the `greeting-service` app with the `service-registry`.

   ```java
   package io.pivotal;

   import org.springframework.boot.SpringApplication;
   import org.springframework.boot.autoconfigure.SpringBootApplication;
   import org.springframework.cloud.client.discovery.EnableDiscoveryClient;

   @SpringBootApplication
   @EnableDiscoveryClient
   public class GreetingServiceApplication {

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

   }
   ```

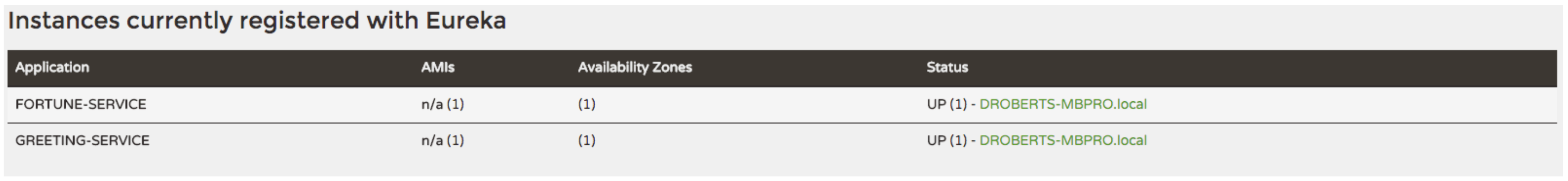
4. Review the file `GreetingController.java`. Notice the `EurekaClient`. The `EurekaClient` is used to discover services registered with the `service-registry`. Review the method `fetchFortuneServiceUrl()` below.

```java
package io.pivotal.greeting;

import com.netflix.appinfo.InstanceInfo;
import com.netflix.discovery.EurekaClient;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.client.RestTemplate;

@Controller
public class GreetingController {

  private final Logger logger = LoggerFactory.getLogger(GreetingController.class);

  private final EurekaClient discoveryClient;

  public GreetingController(EurekaClient discoveryClient) {
    this.discoveryClient = discoveryClient;
  }

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

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

    RestTemplate restTemplate = new RestTemplate();
    String fortune = restTemplate.getForObject(fetchFortuneServiceUrl(), String.class);

    logger.debug("Adding fortune: {}", fortune);
    model.addAttribute("fortune", fortune);

    return "greeting"; // resolves to the greeting.ftl template
  }

  private String fetchFortuneServiceUrl() {
    InstanceInfo instance = discoveryClient.getNextServerFromEureka("FORTUNE-SERVICE", false);
    logger.debug("instanceID: {}", instance.getId());

    String fortuneServiceUrl = instance.getHomePageUrl();
    logger.debug("fortune service homePageUrl: {}", fortuneServiceUrl);

    return fortuneServiceUrl;
  }

}
```

5. Open a new terminal window. Start the `greeting-service` app

```
$ cd greeting-service
$ mvn spring-boot:run
```

6. After the a few moments, check the `service-registry` dashboard http://localhost:8761. Confirm the `greeting-service` app is registered.



Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
FORTUNE-SERVICE	n/a (1)	(1)	UP (1) - DROBERTS-MBPRO.local
GREETING-SERVICE	n/a (1)	(1)	UP (1) - DROBERTS-MBPRO.local

7. Browse to http://localhost:8080/ to the `greeting-service` application. Confirm you are seeing fortunes. Refresh as desired. Also review the terminal output for the `greeting-service`. See the `fortune-service` `instanceId` and `homePageUrl` being logged.

> ### What Just Happened?
>
> The `greeting-service` application was able to discover how to reach the `fortune-service` via the `service-registry` (Eureka).

8. When done, stop the `config-server`, `service-registry`, `fortune-service` and `greeting-service` applications.

7. Update App Config for `fortune-service` and `greeting-service` to run on PCF

The spring cloud services configuration parameter `spring.cloud.services.registrationMethod` provides two distinct ways in which applications can register with Eureka:

- `route`: The application registers using its Cloud Foundry url. This is the default.
- `direct`: The application registers using its host IP and port.

In PCF, it makes sense to think about using the route method. With the route registration method, applications that need the services of other applications deployed in cloud foundry are given their route urls. This ensures that calls to those applications are routed through the cloud foundry GoRouter (https://github.com/cloudfoundry/gorouter). The principal benefit of this option is that the platform takes care of load balancing requests across multiple instances of a scaled microservice.

Even though this option is the default, let's go ahead and set it explicitly. In your `app-config` repository, add a section to the `application.yml` file as shown below (and push back to GitHub):

```yaml
security:
  basic:
    enabled: false

management:
  security:
    enabled: false

logging:
  level:
    io:
      pivotal: DEBUG

spring: # <---NEW SECTION
  cloud:
    services:
      registrationMethod: route
```

8. Deploy the `fortune-service` to PCF

1. Package `fortune-service`

   ```
   $ mvn clean package
   ```

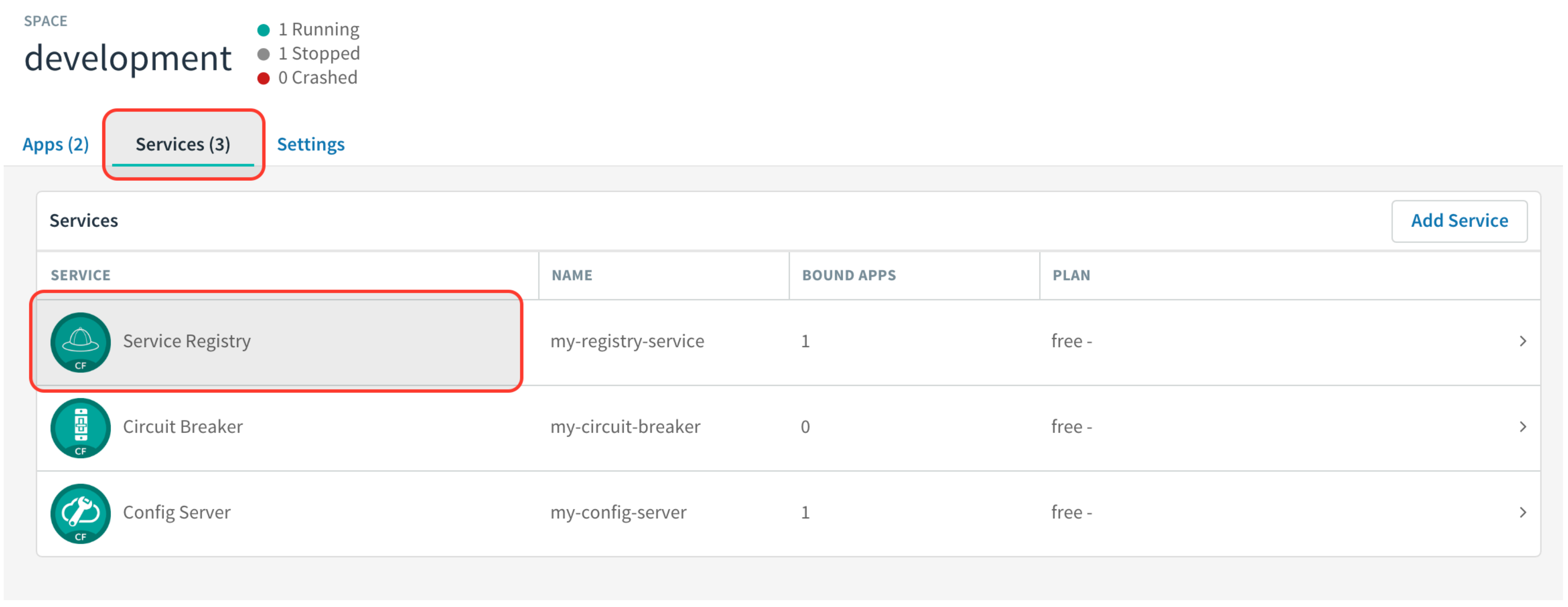
2. Deploy `fortune-service`.

   ```
   $ cf push fortune-service -p target/fortune-service-0.0.1-SNAPSHOT.jar -m 512M --random-route --no-start
   ```

3. Create a Service Registry Service Instance. The `service-registry` service instance will not be immediately bindable. It needs a few moments to initialize.

   ```
   $ cf create-service p-service-registry standard service-registry
   ```

   Click on the **Services** tab and the **Service Registry** entry to navigate to your service.

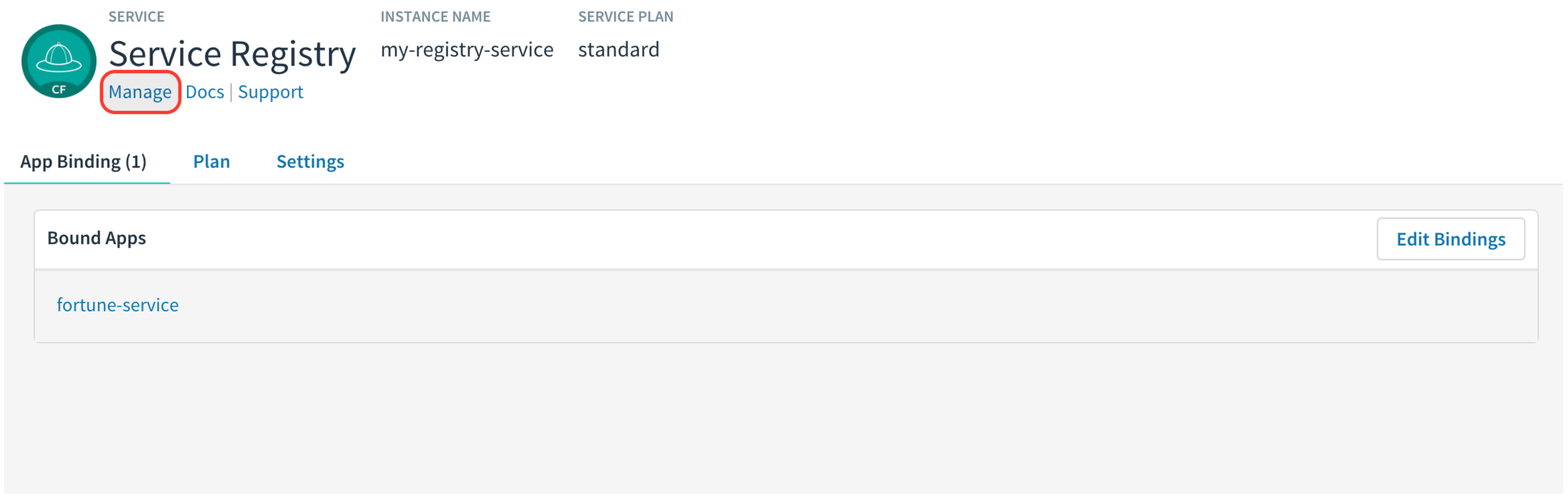

SPACE
development
● 1 Running
● 1 Stopped
● 0 Crashed

Apps (2) | Services (3) | Settings

Services | Add Service

SERVICE	NAME	BOUND APPS	PLAN
Service Registry	my-registry-service	1	free -
Circuit Breaker	my-circuit-breaker	0	free -
Config Server	my-config-server	1	free -

Then, click on the **Manage** link to determine when the `service-registry` is ready.


SERVICE
Service Registry
Manage | Docs | Support

INSTANCE NAME
my-registry-service

SERVICE PLAN
standard

App Binding (1) | Plan | Settings

Bound Apps | Edit Bindings

fortune-service

Spring Cloud Services in PCF are implemented asynchronously. This means that it takes some time after invoking the `create-service` command before the service is online and available. The command `cf services` can be used to monitor the progress of the service creation. You must wait until the service has been created successfully before proceeding with binding applications to these services. If you don’t, you’re likely to see a message similar to this:

```
Binding service service-registry to app fortune-service in org dave / space dev
as droberts@pivotal.io...
FAILED
Server error, status code: 502, error code: 10001, message: Service broker
error: Service instance is not running and available for binding.
```

4. Bind services to the `fortune-service`.

   ```
   $ cf bind-service fortune-service config-server
   ```

   and:

   ```
   $ cf bind-service fortune-service service-registry
   ```

   You can safely ignore the *TIP: Use 'cf restage' to ensure your env variable changes take effect* message from the CLI. We don’t need to restage at this time.

5. Set the `TRUST_CERTS` environment variable for the `fortune-service` application (our PCF instance is using self-signed SSL certificates).

   ```
   $ cf set-env fortune-service TRUST_CERTS api.sys.gcp.esuez.org
   ```

   > Remember, you can find out your api endpoint with the `cf api` command. Furthermore, the value you supply should not include the `https://` prefix, it is strictly a hostname.

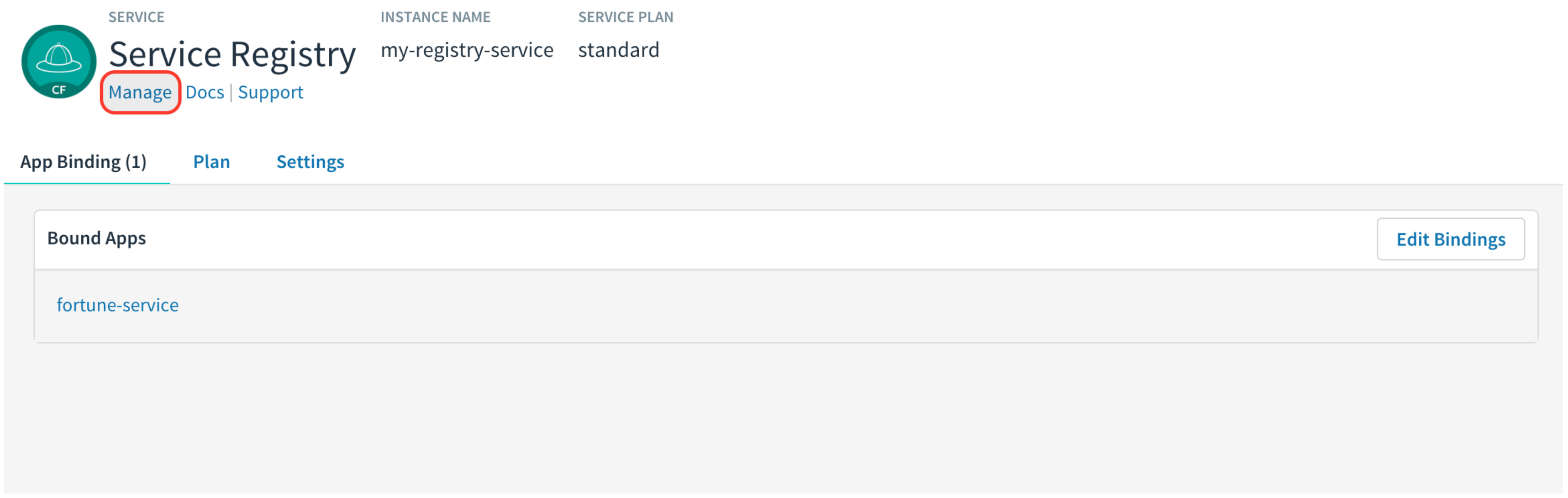
   You can safely ignore the *TIP: Use 'cf restage' to ensure your env variable changes take effect* message from the CLI. We don’t need to restage at this time.

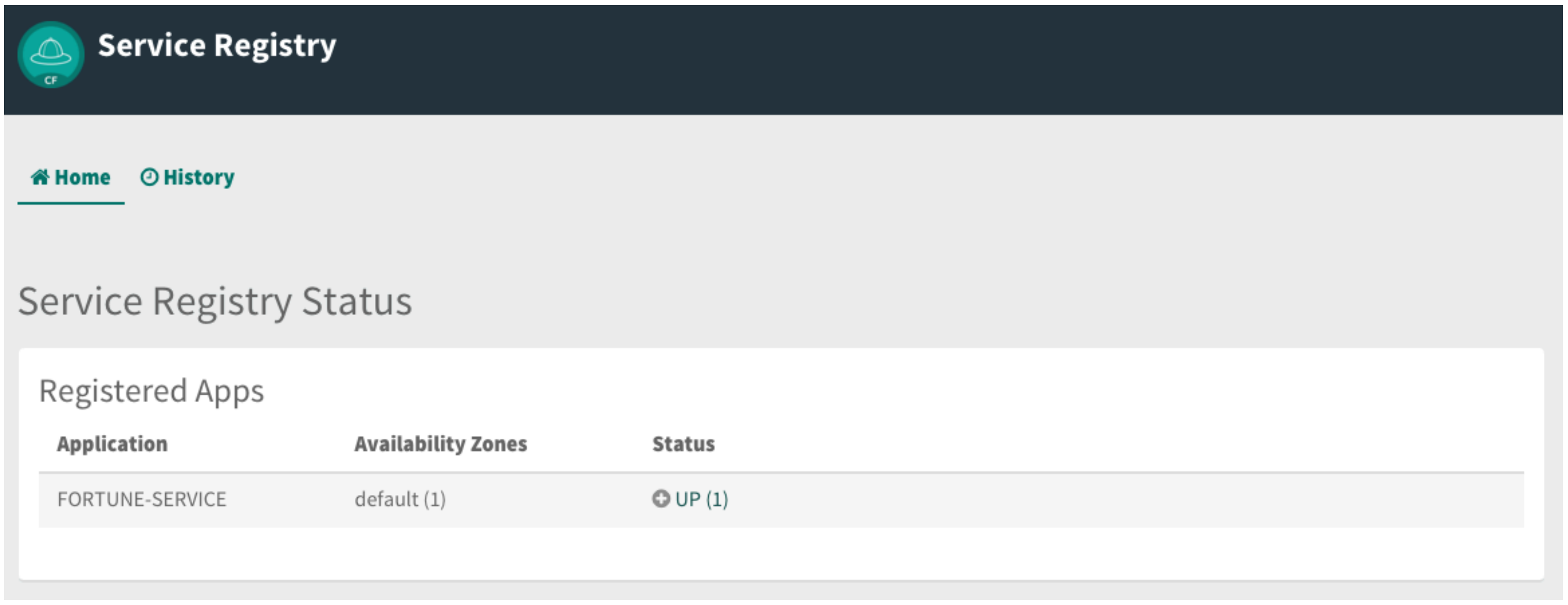
6. Start the `fortune-service` app.

   ```
   $ cf start fortune-service
   ```

7. Confirm `fortune-service` registered with the `service-registry`. This will take a few moments.

   Click on the Manage link for the `service-registry`. You can find it by navigating to the space where your applications are deployed.





Service Registry Status

Registered Apps

Application	Availability Zones	Status
FORTUNE-SERVICE	default (1)	UP (1)

9. Deploy the `greeting-service` app to PCF

1. Package `greeting-service`

```
$ mvn clean package
```

2. Deploy `greeting-service`.

```
$ cf push greeting-service -p target/greeting-service-0.0.1-SNAPSHOT.jar -m 512M
--random-route --no-start
```

3. Bind services for the `greeting-service`.

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

and:

```
$ cf bind-service greeting-service service-registry
```

You can safely ignore the *TIP: Use 'cf restage' to ensure your env variable changes take effect* message from the CLI. We don't need to restage at this time.

4. If using self signed certificates, set the `TRUST_CERTS` environment variable for the `greeting-service` application.

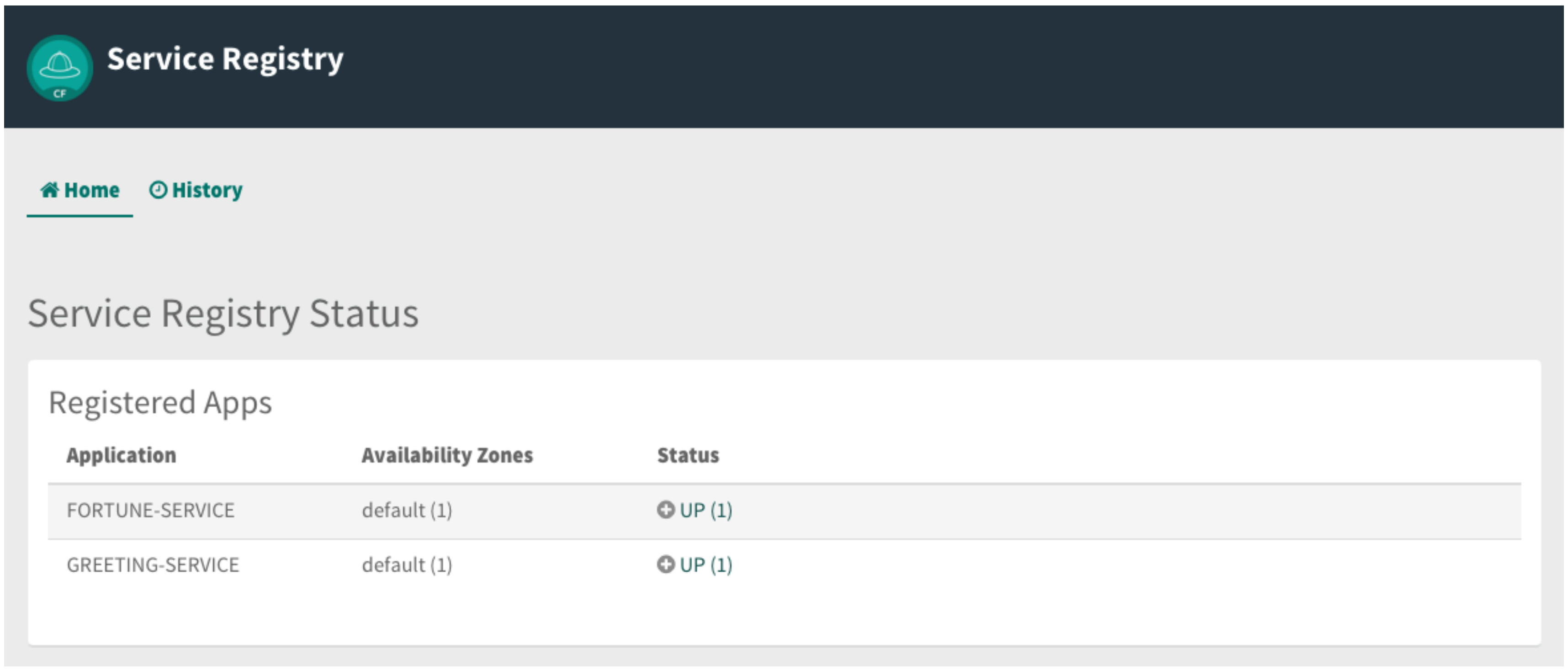
```
$ cf set-env greeting-service TRUST_CERTS api.sys.gcp.esuez.org
```

You can safely ignore the *TIP: Use 'cf restage' to ensure your env variable changes take effect* message from the CLI. We don't need to restage at this time.

5. Start the `greeting-service` app.

```
$ cf start greeting-service
```

6. Confirm `greeting-service` registered with the `service-registry`. This will take a few moments.



Service Registry Status

Registered Apps

Application	Availability Zones	Status
FORTUNE-SERVICE	default (1)	UP (1)
GREETING-SERVICE	default (1)	UP (1)

7. Browse to the `greeting-service` application. Confirm you are seeing fortunes. Refresh as desired.

10. Scale the `fortune-service`

1. Scale the `fortune-service` app instances to 3.

   ```
   $ cf scale fortune-service -i 3
   ```

2. Wait for the new instances to register with the `service-registry`. This will take a few moments.

3. Tail the logs for the `greeting-service` application.

   ```
   $ cf logs greeting-service | grep GreetingController
   ```

4. Refresh the `greeting-service` root endpoint.

5. Observe the log output. Compare the `instanceID` values across log entries. With each refresh, the `GreetingController` obtains a different eureka-registered instance of the `fortune-service` application. Note however that the `homePageUrl` value is the same across multiple instances.

   Log output fragment, abridged

```
[APP/PROC/WEB/0] GreetingController : Adding greeting
[APP/PROC/WEB/0] GreetingController : instanceID: eitan-fortune-service.cfapps.io:5fd41912-9480-
417e-7d3e-6af3
[APP/PROC/WEB/0] GreetingController : Adding fortune: You can always find happiness at work on
Friday
[APP/PROC/WEB/0] GreetingController : fortune service homePageUrl: http://eitan-fortune-
service.cfapps.io:80/
[APP/PROC/WEB/0] GreetingController : Adding greeting
[APP/PROC/WEB/0] GreetingController : instanceID: eitan-fortune-service.cfapps.io:86de654a-a134-
4441-5b8a-94c9
[APP/PROC/WEB/0] GreetingController : fortune service homePageUrl: http://eitan-fortune-
service.cfapps.io:80/
[APP/PROC/WEB/0] GreetingController : Adding fortune: You learn from your mistakes... You will
learn a lot today.
[APP/PROC/WEB/0] GreetingController : Adding greeting
[APP/PROC/WEB/0] GreetingController : instanceID: eitan-fortune-service.cfapps.io:f0f68392-d20b-
4f9d-67d6-cbc4
[APP/PROC/WEB/0] GreetingController : fortune service homePageUrl: http://eitan-fortune-
service.cfapps.io:80/
[APP/PROC/WEB/0] GreetingController : Adding fortune: You will be hungry again in one hour.
[APP/PROC/WEB/0] GreetingController : Adding greeting
[APP/PROC/WEB/0] GreetingController : instanceID: eitan-fortune-service.cfapps.io:5fd41912-9480-
417e-7d3e-6af3
[APP/PROC/WEB/0] GreetingController : fortune service homePageUrl: http://eitan-fortune-
service.cfapps.io:80/
[APP/PROC/WEB/0] GreetingController : Adding fortune: You will be hungry again in one hour.
[APP/PROC/WEB/0] GreetingController : Adding greeting
[APP/PROC/WEB/0] GreetingController : fortune service homePageUrl: http://eitan-fortune-
service.cfapps.io:80/
[APP/PROC/WEB/0] GreetingController : instanceID: eitan-fortune-service.cfapps.io:86de654a-a134-
4441-5b8a-94c9
[APP/PROC/WEB/0] GreetingController : Adding fortune: You learn from your mistakes... You will
learn a lot today.
..
```

If you are not seeing this behavior, make sure that your logging level is set to `DEBUG` and you have refreshed the configurations for the greeting service.

What Just Happened?

The `greeting-service` and `fortune-service` both registered with the `service-registry` (Eureka). The `greeting-service` was able to locate the `fortune-service` via the `service-registry`. In PCF, and using the route registration method, requests to any service instance will use the same url, implying that they are load-balanced by the PCF GoRouter.

Last updated 2017-05-16 03:34:42 CEST