(../..)

Purpose of this lab

- 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
- Estimated Time: 45 minutes

Clone the `spring-cloud-services-code` repo

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

Add `application.yml` to the `app-config` repo

Create an `application.yml` file in your `app-config` repo with the following content:

```
security:
  basic:
    enabled: false

management:
  security:
    enabled: false

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

Commit and push to Github.

A note about the application.yml

When the `config-server`'s backing repository contains an `application.yml` it is shared with all applications. Therefore, it is a great place to put common configuration for all applications. In this case, we have dropped security on all the endpoints and setup logging.

In the Spring Cloud Config Lab (../spring-cloud-config), 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 configuration settings in the `application.yml`.

Set up the 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
   ./mvnw clean spring-boot:run
   ```

2. Verify that the Config Server is up. Open a browser and fetch http://localhost:8888/myapp/default (http://localhost:8888/myapp/default)

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 the `application.yml`.

Set up the Service Registry

1. Review the `service-registry/pom.xml` file. The addition of `spring-cloud-starter-eureka-server` to the classpath makes this application eligible to embed a Eureka server.

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

2. Review the following file: `service-registry/src/main/java/io/pivotal/ServiceRegistryApplication.java`. Note the use of the `@EnableEurekaServer` annotation that makes this application a Eureka server.

```
@SpringBootApplication
@EnableEurekaServer
public class ServiceRegistryApplication {

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

3. Review the following file: `service-registry/src/main/resources/application.yml`

```
server:
  port: 8761

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

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 have simplified that configuration by setting Eureka 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.

Configuration parameters

- `eureka.instance.hostname` - the hostname for this service. In this case, the host to use to

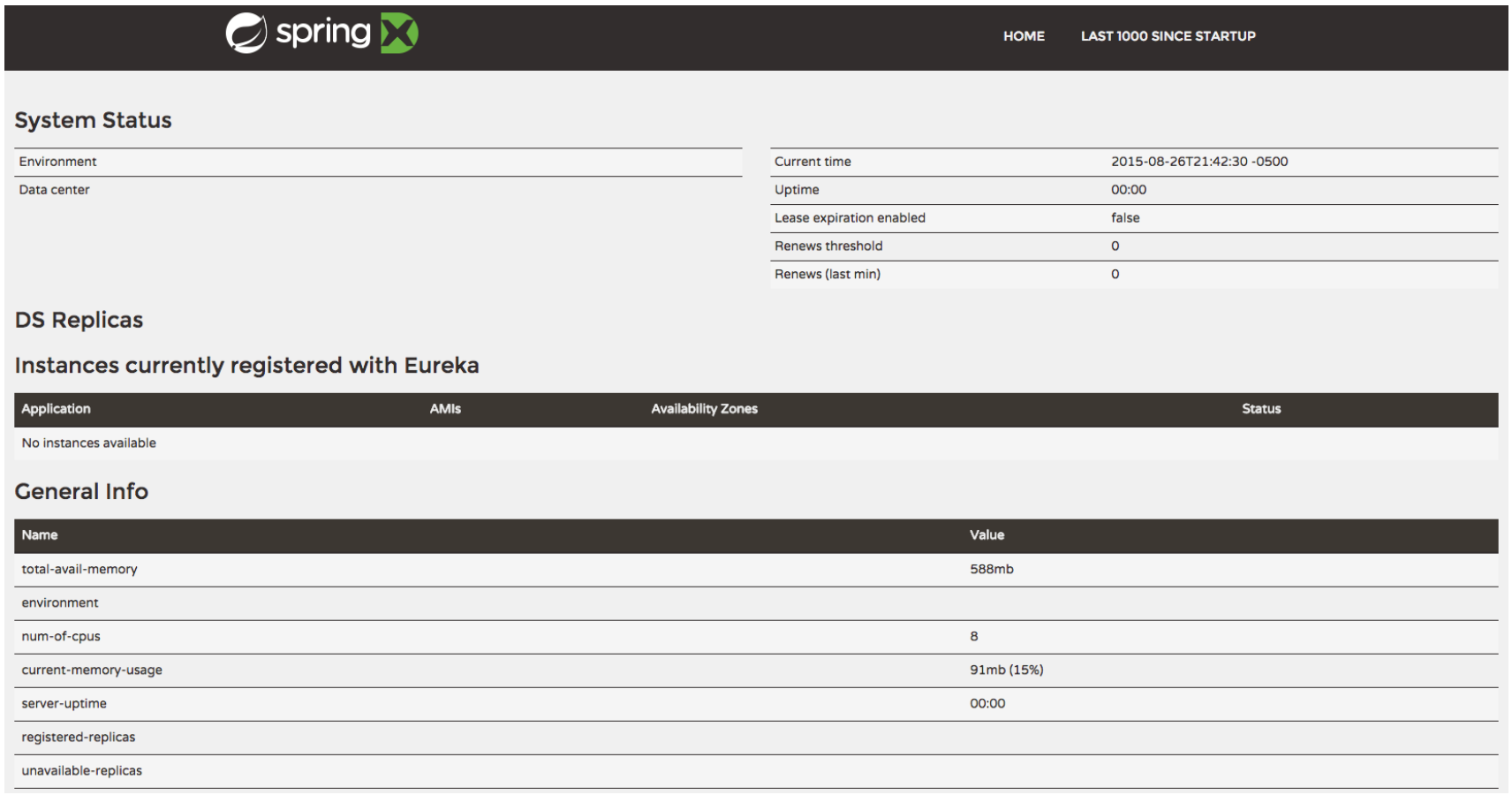
reach our standalone Eureka instance.

- `eureka.client.registerWithEureka` - should this application (our standalone Eureka instance) register with Eureka.
- `eureka.client.fetchRegistry` - should this application (our standalone Eureka instance) fetch the registry (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`).

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

```
cd service-registry
./mvnw clean spring-boot:run
```

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



Set up the Fortune Service

1. Review the `fortune-service/src/main/resources/bootstrap.yml` file. The name of this app is `fortune-service`. It 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 `fortune-service/pom.xml` file. By adding `spring-cloud-services-starter-service-registry` to the classpath 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</ar
tifactId>
</dependency>
```

3. Review the following file: `fortune-service/src/main/java/io/pivotal/FortuneServiceApplication.java`. Notice the `@EnableDiscoveryClient`. This enables a discovery client that registers the `fortune-service` with the `service-registry` application.

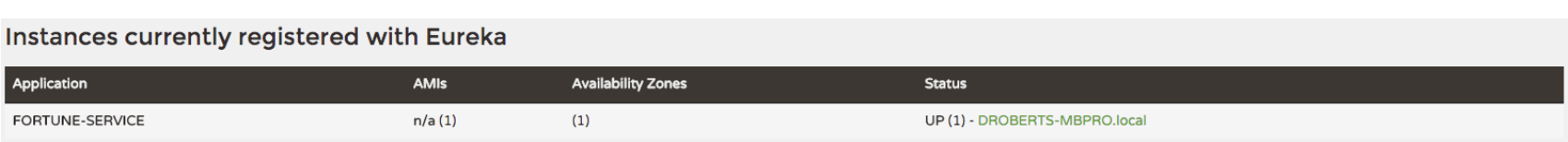
```
@SpringBootApplication
@EnableDiscoveryClient
public class FortuneServiceApplication {

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

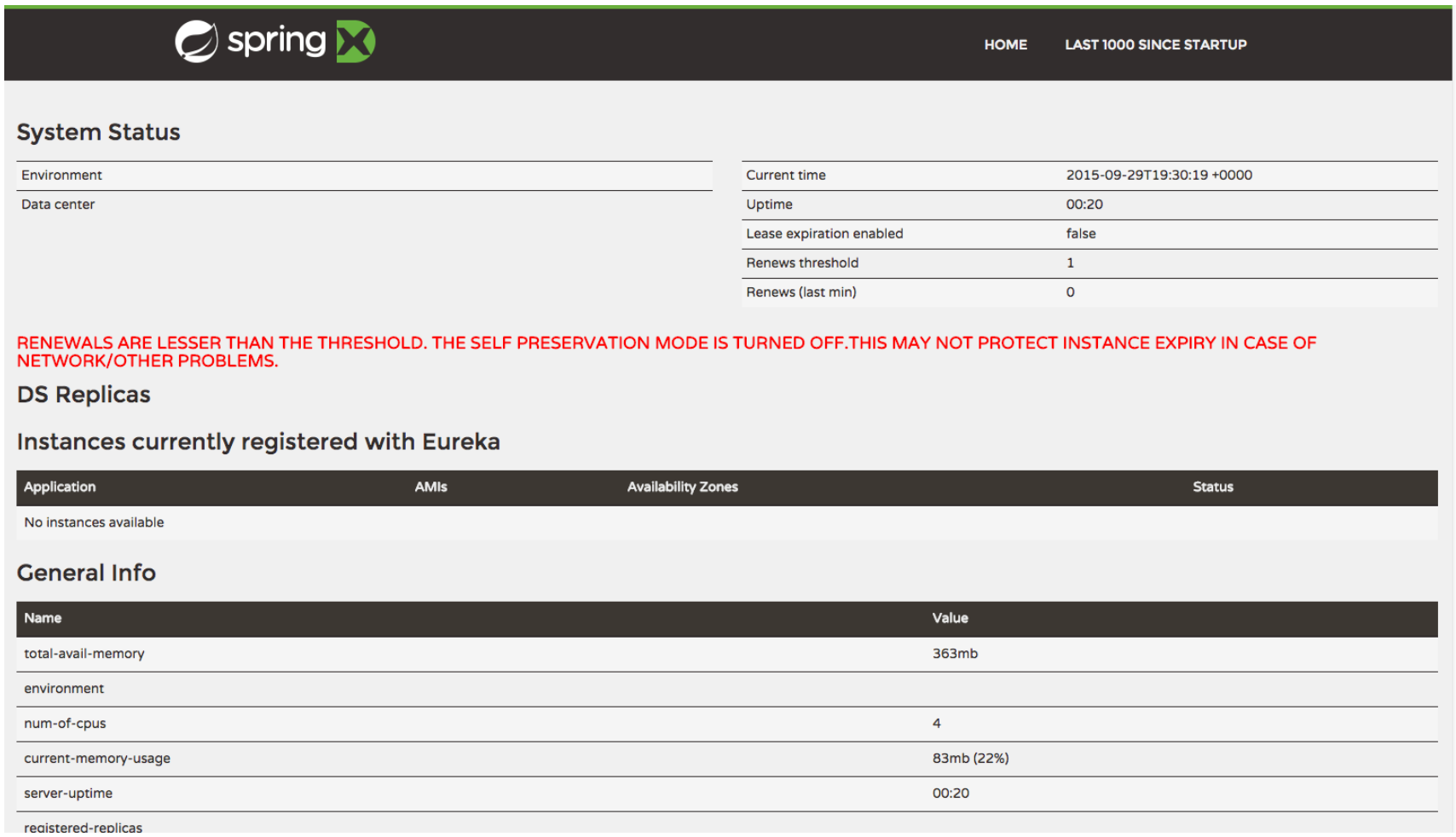
4. Open a new terminal window. Start the Fortune Service

```
cd fortune-service
./mvnw clean spring-boot:run
```

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



The Eureka Dashboard may report a warning, because we are not set-up with multiple peers. This can safely be ignored.



Set up the Greeting Service

1. Review the `greeting-service/src/main/resources/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 `greeting-service/pom.xml` file. Note that the application has `spring-cloud-services-starter-service-registry` on the classpath, making it eligible to register and discover services with the `service-registry`.

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

3. Review the following file: `greeting-service/src/main/java/io/pivotal/GreetingServiceApplication.java`. Notice the `@EnableDiscoveryClient`. This enables a discovery client that registers the `greeting-`

`service` app with the `service-registry`.

```
@SpringBootApplication
@EnableDiscoveryClient
public class GreetingServiceApplication {

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

}
```

4. Review the the following file: `greeting-service/src/main/java/io/pivotal/greeting/GreetingController.java`. Notice the `DiscoveryClient`. `DiscoveryClient` is used to discover services registered with the `service-registry`. See `fetchFortuneServiceUrl()`.

```
@Controller
public class GreetingController {

    private Logger logger = LoggerFactory.getLogger(getClass());
    private EurekaClient discoveryClient;
    private RestTemplate restTemplate;

    @Autowired
    public GreetingController(EurekaClient discoveryClient) {
        this.discoveryClient = discoveryClient;
        this.restTemplate = new RestTemplate();
    }


    @RequestMapping("/")
    public String getGreeting(Model model) {
        logger.debug("Adding greeting");
        model.addAttribute("msg", "Greetings!!!");

        String fortune = restTemplate.getForObject(fetchFortuneSer
viceUrl(), String.class);

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

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


    private String fetchFortuneServiceUrl() {
        InstanceInfo instance = discoveryClient.getNextServerFromE
ureka("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
./mvnw clean spring-boot:run
```

6. After the a few moments, check the `service-registry` dashboard http://localhost:8761 (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/ (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.

   The `greeting-service` application was able to discover how to reach the `fortune-service` via the `service-registry` (Eureka).

8. Stop the `config-server`, `service-registry`, `fortune-service` and `greeting-service` applications.

Deploy the Fortune Service to PCF

1. Package `fortune-service`.

```
./mvnw clean package
```

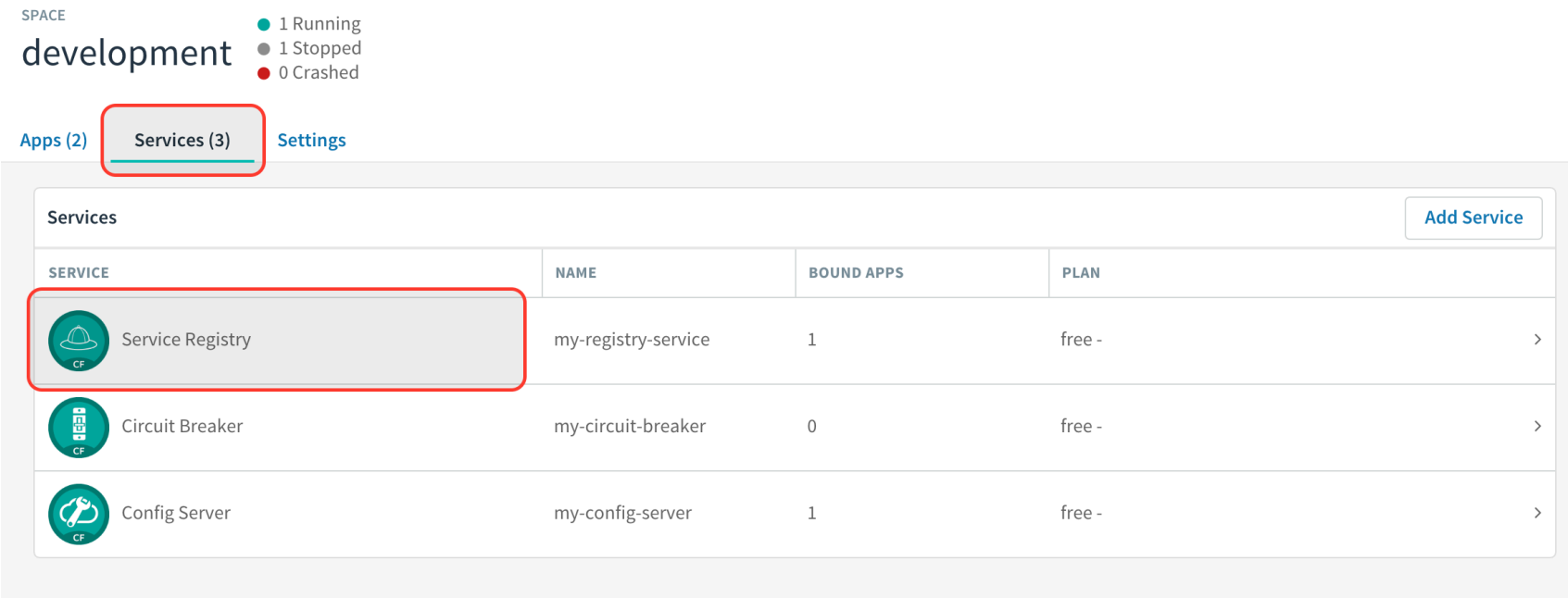
2. Deploy `fortune-service`.

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

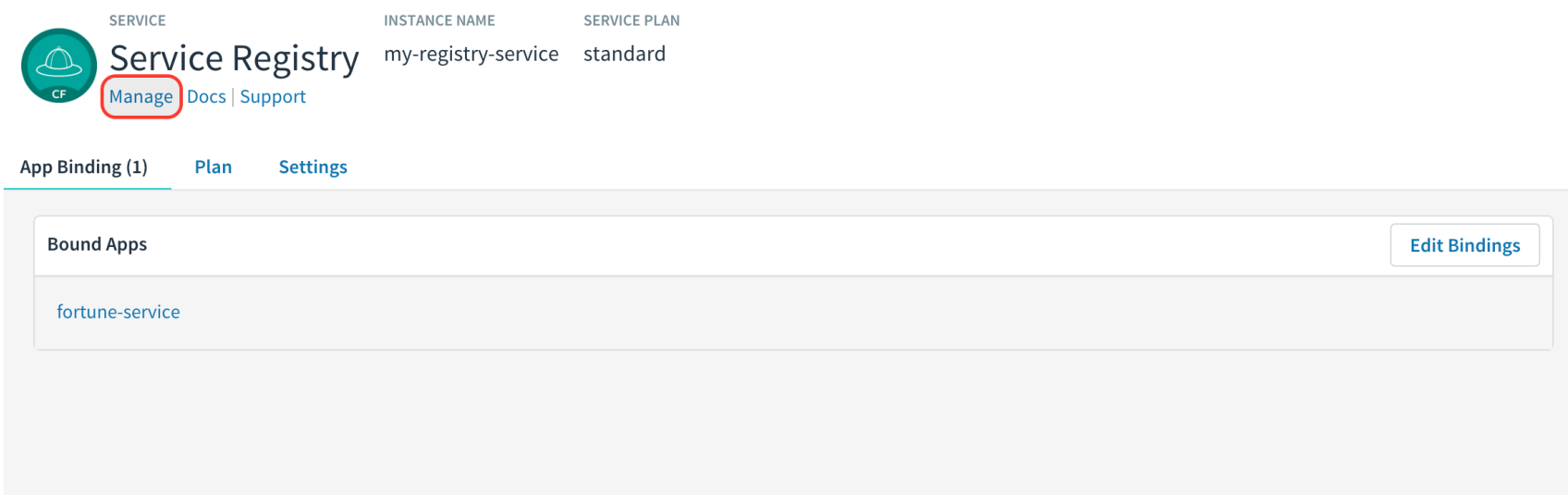
3. Create a Service Registry Service Instance. The `service-registry` service instance will not be immediately bindable. It needs a few moments to initialize.

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

From apps manager, you can monitor the status of your service registry. Click on the *Services* tab navigate to your service.



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



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

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

You will need to wait and try again if you see the following message when binding the `service-registry`:

```
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 bin
ding.
```

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. If using self-signed certificates, set the `TRUST_CERTS` environment variable for the Fortune Service application.

```
cf set-env fortune-service TRUST_CERTS <your api endpoint>
```

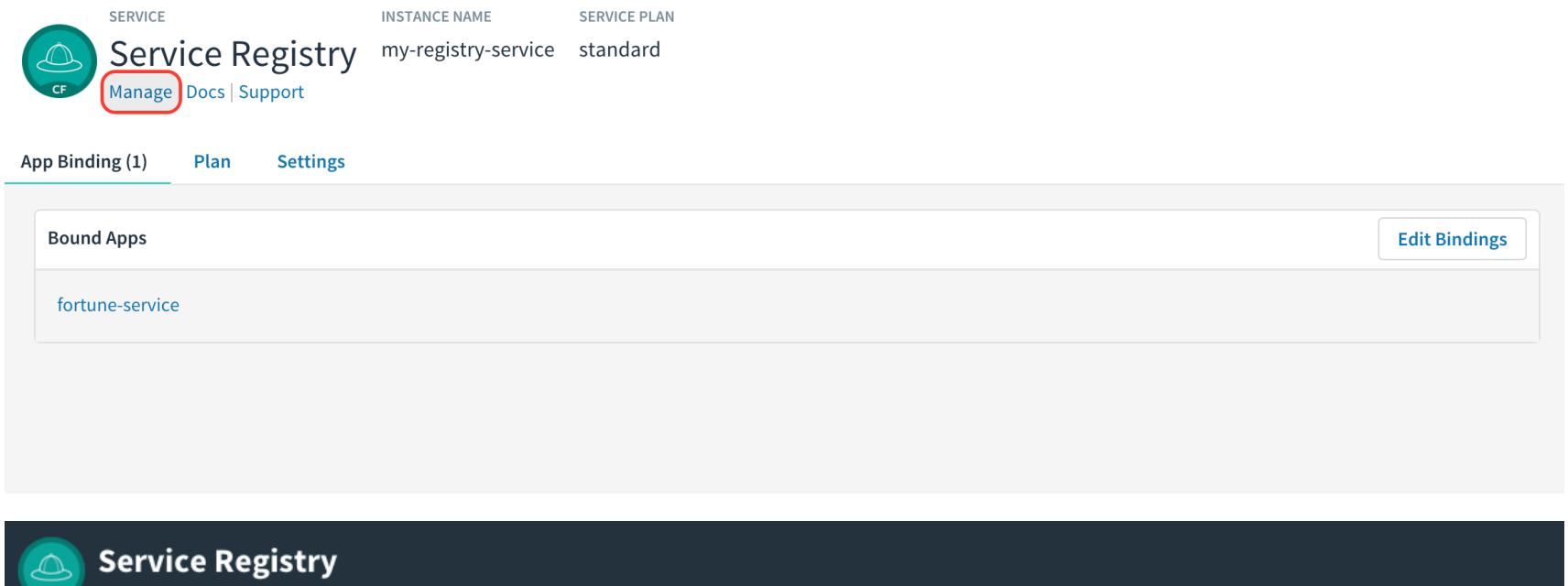
   You can safely ignore the *TIP: Use 'cf restage' to ensure your env variable changes take effect* message from the CLI. We don't need to restage at this time.

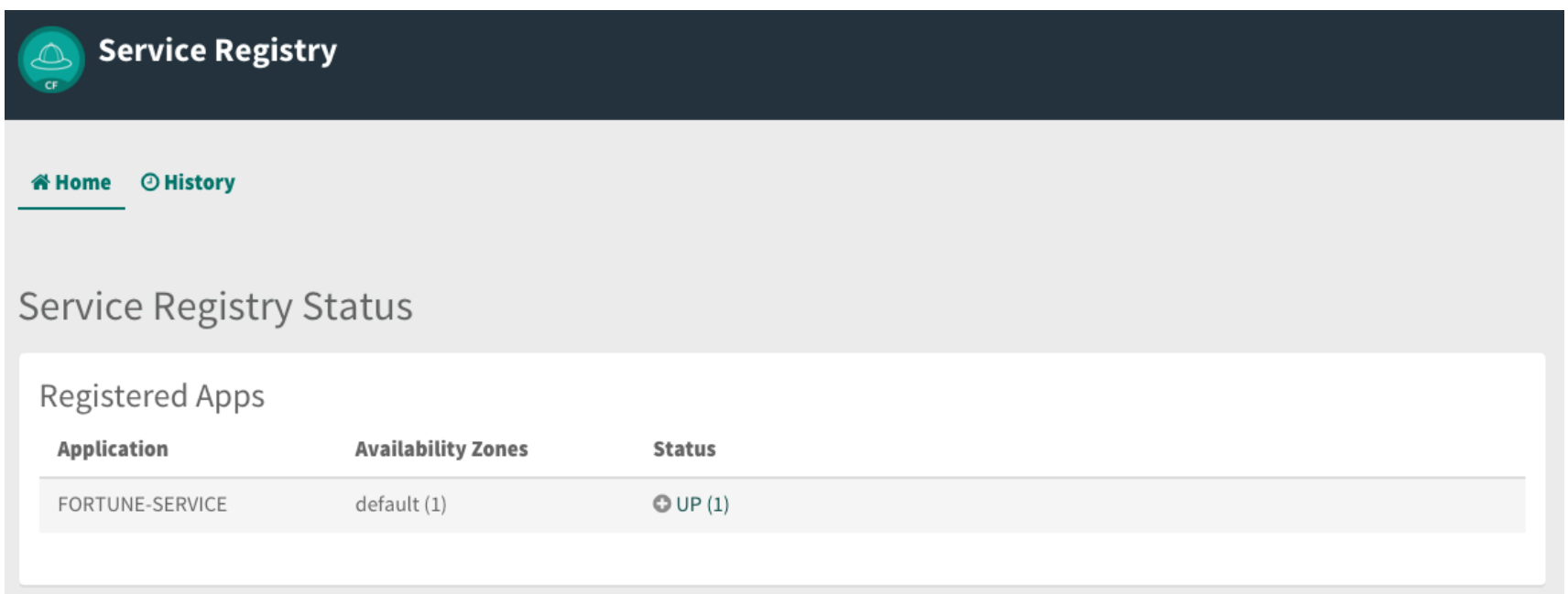
6. Start the Fortune Service app.

```
cf start fortune-service
```

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

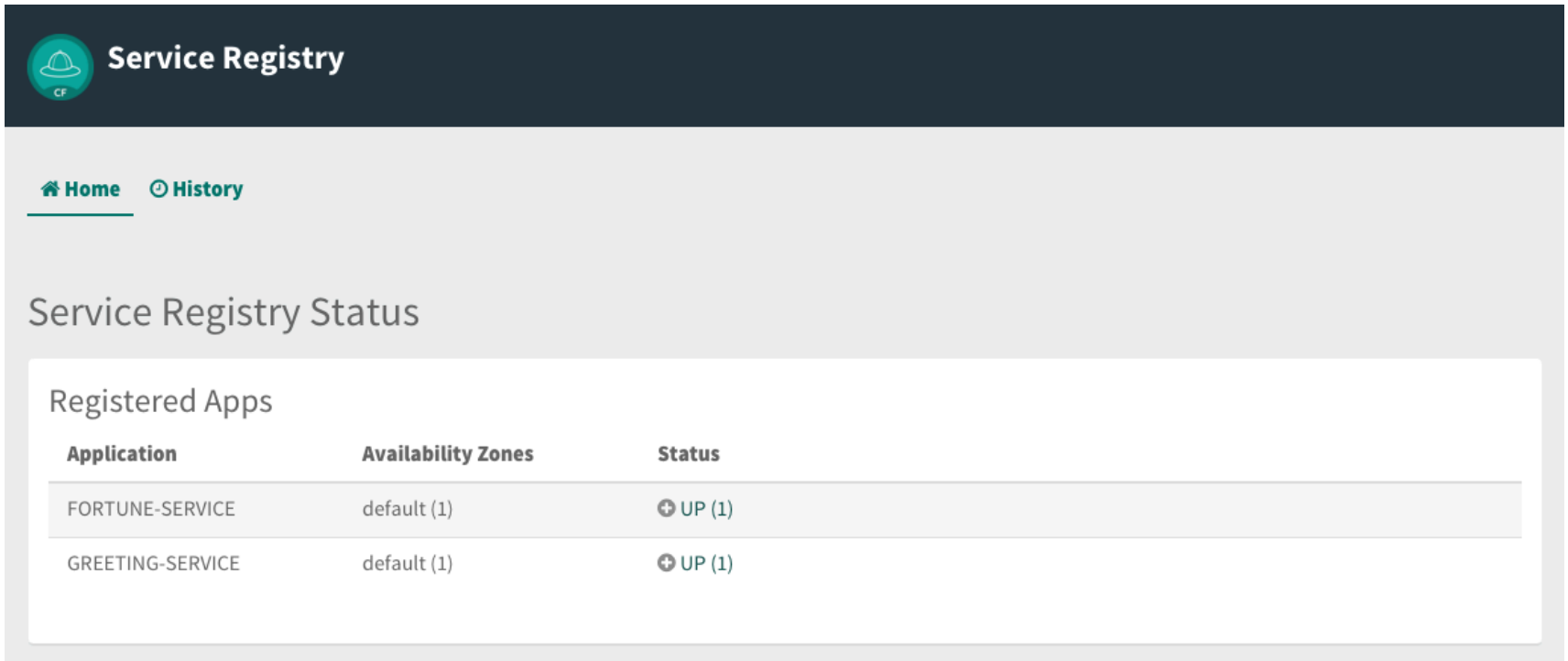
   Click on the *Manage* link for the `service-registry`. You can find it by navigating to the space where your applications are deployed.





Deploy the Greeting Service app to PCF

1. Use the process described above to deploy the Greeting Service to PCF.
2. Confirm `greeting-service` registered with the `service-registry`. This will take a few moments.



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

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

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

4. Refresh the `greeting-service` `/` endpoint.
5. Observe the log output. Compare the `instanceId` and `homePageUrl` being logged across log entries. The `discoveryClient` round robins between the Fortune Service instances.

   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.

The Fortune Service and Greeting Service both registered with the Service Registry (Eureka). The Greeting Service was able to locate the Fortune Service via the Service Registry.

(https://pivotal.io)