Circuit Breakers

Table of Contents

Estimated Time: 25 minutes

Requirements

- Provided by e-mail last week
- Completion of Spring Cloud Config Lab, Service Discovery Lab

---

What You Will Learn

- How to protect your application (`greeting-hystrix`) from failures or latency with the circuit breaker pattern
- How to publish circuit-breaking metrics from your application (`greeting-hystrix`)
- How to use Turbine in Pivotal Cloud Foundry to consume metric streams and view them with the `hystrix-dashboard`

Exercises

Start the `config-server`, `service-registry`, and `fortune-service`

1) Make sure config-server, service-registry, and fortune-service are running, as per the previous labs.

Set up `greeting-hystrix`

1) Review the `$SPRING_CLOUD_SERVICES_LABS_HOME/greeting-hystrix/pom.xml` file. By adding `spring-cloud-services-starter-circuit-breaker` to the classpath this application is eligible to use circuit breakers via Hystrix.

```
<dependency>
        <groupId>io.pivotal.spring.cloud</groupId>
        <artifactId>spring-cloud-services-starter-circuit-breaker</artifactId>
 </dependency>
```

2) Review the following file: `$SPRING_CLOUD_SERVICES_LABS_HOME/greeting-hystrix/src/main/java/io/pivotal/GreetingHystrixApplication.java`. Note the use of the `@EnableCircuitBreaker` annotation. This allows the application to create circuit breakers. Note also how we again configure our `RestTemplate` bean to be load-balanced.

```
@SpringBootApplication
@EnableDiscoveryClient
@EnableCircuitBreaker
public class GreetingHystrixApplication {
      public static void main(String[] args) {
```

```
        SpringApplication.run(GreetingHystrixApplication.class, args);
    }
@LoadBalanced
@Bean
RestTemplate restTemplate() {
        return new RestTemplate();
    }
}
```

3) Review the following file: `$SPRING_CLOUD_SERVICES_LABS_HOME/greeting-hystrix/src/main/java/io/pivotal/fortune/FortuneService.java`. Note the use of the `@HystrixCommand`. This is our circuit breaker. If `getFortune()` fails, a fallback method `defaultFortune` will be invoked.

```
@Service
public class FortuneService {
    Logger logger = LoggerFactory.getLogger(FortuneService.class);

    @Autowired
    private RestTemplate restTemplate;

    @HystrixCommand(fallbackMethod = "defaultFortune")
    public String getFortune() {
     return restTemplate.getForObject("http://fortune-service", String.class);
   }

    public String defaultFortune() {
     logger.debug("Default fortune used.");
     return "This fortune is no good. Try another.";
   }
}
```

4) Package and push the `greeting-hystrix` application.

```
$ mvn clean package

$ cf push greeting-hystrix -p target/greeting-hystrix-0.0.1-SNAPSHOT.jar -m 1G
 --random-route --no-start
```

5) Bind services for the `greeting-hystrix`.

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

$ cf bind-service greeting-hystrix 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.

6) Set the `TRUST_CERTS` environment variable for the `greeting-hystrix` application (our PCF instance is using self-signed SSL certificates).

```
$ cf set-env greeting-hystrix 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.

7) Start the `greeting-hystrix` app.

```
$ cf start greeting-hystrix
```

8) Refresh the `greeting-hystrix` `/` endpoint. You should get fortunes from the `fortune-service`.

6) Stop the `fortune-service`. Refresh the `greeting-hystrix` `/` endpoint again. The default fortune is given.

7) Restart the `fortune-service`. Refresh the `greeting-hystrix` `/` endpoint again. After some time, fortunes from the `fortune-service` are back.

What Just Happened?

The circuit breaker insulated `greeting-hystrix` from failures when the `fortune-service` was not available. This results in a better experience for our users and can also prevent cascading failures.

View the `greeting-hystrix` metric stream

Being able to monitor the state of our circuit breakers is highly valuable. To make this possible, the `greeting-hystrix` application must expose the relevant metrics.

This is accomplished by including the `actuator` dependency in the `greeting-hystrix` `pom.xml`.

1) Review the `$SPRING_CLOUD_SERVICES_LABS_HOME/greeting-hystrix/pom.xml` file. By adding `spring-boot-starter-actuator` to the classpath this application will publish metrics at the `/hystrix.stream` endpoint.

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

2) Browse to the `/hystrix.stream` endpoint of your `greeting-hystrix` application and review the hystrix stream.

Set up `hystrix-dashboard`

Consuming the metric stream is difficult to interpret on our own. The metric stream can be visualized with the Hystrix Dashboard.

1) Review the `$SPRING_CLOUD_SERVICES_LABS_HOME/hystrix-dashboard/pom.xml` file. By adding `spring-cloud-starter-hystrix-dashboard` to the classpath this application exposes a Hystrix Dashboard.

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

2) Review the following file: `$SPRING_CLOUD_SERVICES_LABS_HOME/hystrix-dashboard/src/main/java/io/pivotal/HystrixDashboardApplication.java`. Note the use of the `@EnableHystrixDashboard`annotation. This creates a Hystrix Dashboard.

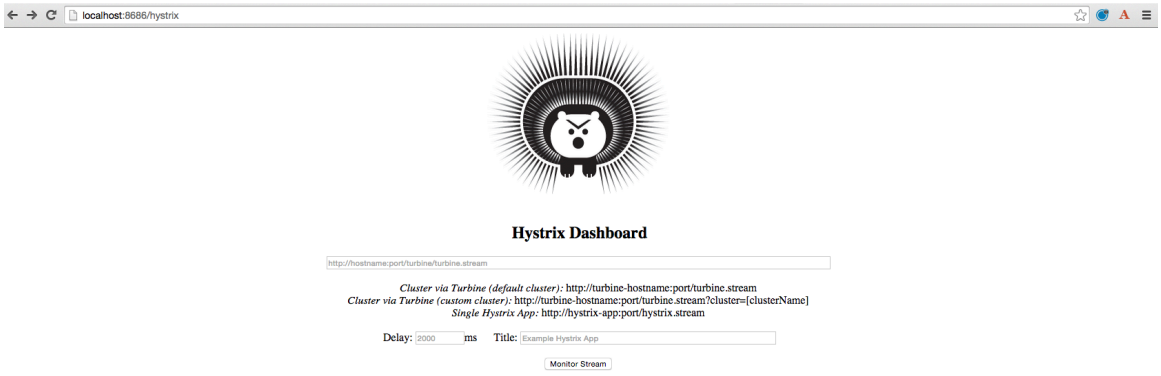
```
@SpringBootApplication
@EnableHystrixDashboard
public class HystrixDashboardApplication {
```

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

3) Open a new terminal window. Start the `hystrix-dashboard`

```
$ cd $SPRING_CLOUD_SERVICES_LABS_HOME/hystrix-dashboard
$ mvn clean spring-boot:run
```

4) Open a browser to http://localhost:8686/hystrix



5) Link the `hystrix-dashboard` to the `greeting-hystrix` app. Enter `<your greeting-hystrix url>/hystrix.stream` as the stream to monitor.

6) Experiment! Refresh the `greeting-hystrix /` endpoint several times. Take down the `fortune-service` app. What does the dashboard do? Review the dashboard doc for an explanation on metrics.

7) Stop your local hystrix dashboard process

Back to TOP