Circuit Breakers

Table of Contents

1. 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 consume metric streams with the `hystrix-dashboard`

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

1. Start the `config-server` in a terminal window. You may have terminal windows still open from previous labs. They may be reused for this lab.

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

2. Start the `service-registry`

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

3. Start the `fortune-service`

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

3. Set up `greeting-hystrix`

1. Review the `greeting-hystrix` project's `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 class `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.

```
package io.pivotal;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.circuitbreaker.EnableCircuitBreaker;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
import org.springframework.cloud.client.loadbalancer.LoadBalanced;
import org.springframework.context.annotation.Bean;
import org.springframework.web.client.RestTemplate;

@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 class file `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.

```
package io.pivotal.fortune;

import com.netflix.hystrix.contrib.javanica.annotation.HystrixCommand;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Service;
import org.springframework.web.client.RestTemplate;

@Service
public class FortuneService {

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

  private final RestTemplate restTemplate;

  public FortuneService(RestTemplate restTemplate) {
    this.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.";
  }

}
```

the fallback method signature must match the signature (return type, method arguments) of the method it stands in for.

4. Open a new terminal window, and launch `greeting-hystrix`:

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

5. Refresh the `greeting-hystrix` root endpoint. You should get fortunes from the `fortune-service`.

6. Stop the `fortune-service`. Now refresh the `greeting-hystrix` root endpoint again. The default fortune is returned.

7. Restart the `fortune-service`. And refresh the `greeting-hystrix` root 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.

4. Set up the `greeting-hystrix` metric stream

Being able to monitor the state of our circuit breakers is highly valuable, but first the `greeting-hystrix` application must expose the metrics. This is accomplished by including the `actuator` dependency in the `greeting-hystrix` project's build file.

1. Review the `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 http://localhost:8080/hystrix.stream to review the metric stream.

5. 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 `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>
```

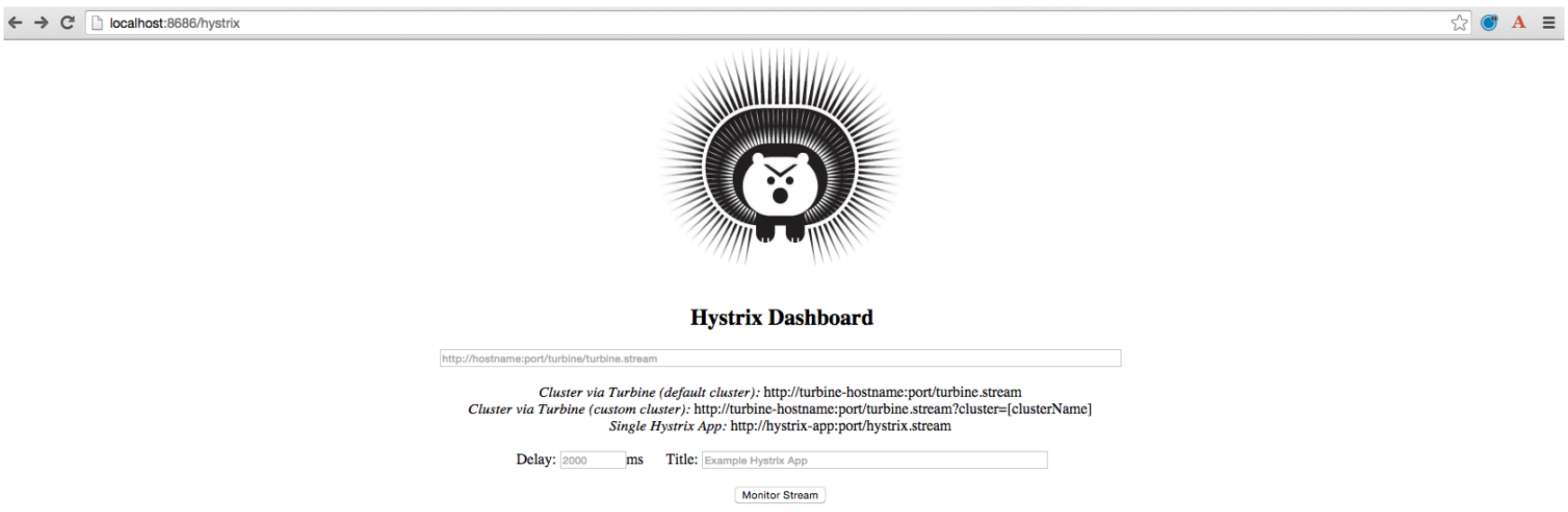
2. Review the file `hystrix-dashboard/src/main/java/io/pivotal/HystrixDashboardApplication.java`. Note the use of the `@EnableHystrixDashboard` annotation. This creates a Hystrix Dashboard.

```
package io.pivotal;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.hystrix.dashboard.EnableHystrixDashboard;

@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`

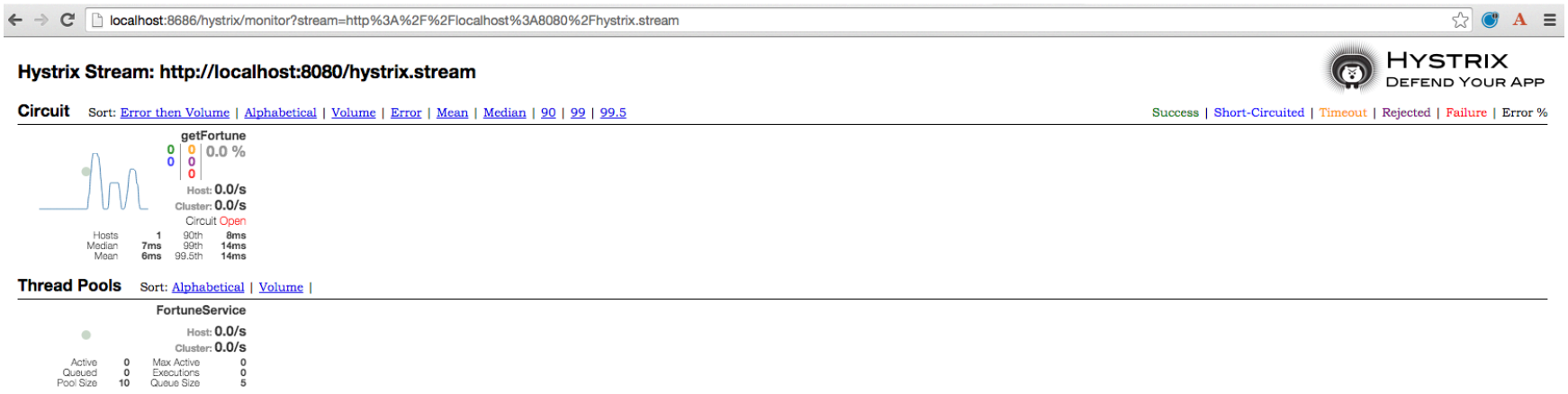
```
$ cd hystrix-dashboard
$ mvn spring-boot:run
```

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



5. Link the `hystrix-dashboard` to the `greeting-hystrix` app. Enter `http://localhost:8080/hystrix.stream` as the stream to monitor.

6. Experiment! Refresh the `greeting-hystrix` root endpoint several times. Take down the `fortune-service` app. What does the dashboard do? Review the dashboard doc (https://github.com/Netflix/Hystrix/wiki/Dashboard) for an explanation on metrics.



It's not always convenient to refresh your application's endpoint multiple times to attempt to get a circuit to open or close. Using a tool such as ab (https://httpd.apache.org/docs/2.4/programs/ab.html) is often more practical.

Last updated 2017-05-05 21:51:05 CEST