Spring Cloud Netflix: Circuit Breakers

Table of Contents

Spring Cloud Netflix: Circuit Breakers

Estimated Time: 25 minutes

Requirements

Lab Requirements (/spring-cloud-services/requirements)

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`

Exercises

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 $SPRING_CLOUD_SERVICES_LABS_HOME/config-server
$ mvn clean spring-boot:run
```

2) Start the `service-registry`

```
$ cd $SPRING_CLOUD_SERVICES_LABS_HOME/service-registry
$ mvn clean spring-boot:run
```

3) Start the `fortune-service`

```
$ cd $SPRING_CLOUD_SERVICES_LABS_HOME/fortune-service
$ mvn clean spring-boot:run
```

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.

```
@SpringBootApplication
@EnableDiscoveryClient
@EnableCircuitBreaker
public class GreetingHystrixApplication {


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

}
```

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
        @LoadBalanced
        private RestTemplate restTemplate;

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

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



}
```

4) Open a new terminal window. Start the `greeting-hystrix`

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

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

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

7) Restart the `fortune-service`. And 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.

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 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 http://localhost:8080/hystrix.stream (http://localhost:8080/hystrix.stream) to review the metric stream.

localhost:8080/hystrix.stream


```
ping:

data:
{"type":"HystrixCommand","name":"getFortune","group":"FortuneService","currentTime":1443629210165,"isCircuitBreakerOpen":false,"errorPercentage":0,"errorCount":0,"requestCount":0,"rollingCountBa
dRequests":0,"rollingCountCollapsedRequests":0,"rollingCountEmit":0,"rollingCountExceptionsThrown":0,"rollingCountFailure":0,"rollingCountEmit":0,"rollingCountFallbackFailure":0,"rollingCountFal
lbackRejection":0,"rollingCountFallbackSuccess":0,"rollingCountResponsesFromCache":0,"rollingCountSemaphoreRejected":0,"rollingCountShortCircuited":0,"rollingCountSuccess":0,"rollingCountThreadP
oolRejected":0,"rollingCountTimeout":0,"currentConcurrentExecutionCount":0,"rollingMaxConcurrentExecutionCount":0,"latencyExecute_mean":0,"latencyExecute":
{"0":0,"25":0,"50":0,"75":0,"90":0,"95":0,"99":0,"99.5":0,"100":0},"latencyTotal_mean":0,"latencyTotal":
{"0":0,"25":0,"50":0,"75":0,"90":0,"95":0,"99":0,"99.5":0,"100":0},"propertyValue_circuitBreakerRequestVolumeThreshold":20,"propertyValue_circuitBreakerSleepWindowInMilliseconds":5000,"propertyV
alue_circuitBreakerErrorThresholdPercentage":50,"propertyValue_circuitBreakerForceOpen":false,"propertyValue_circuitBreakerForceClosed":false,"propertyValue_circuitBreakerEnabled":true,"property
Value_executionIsolationStrategy":"THREAD","propertyValue_executionIsolationThreadTimeoutInMilliseconds":1000,"propertyValue_executionTimeoutInMilliseconds":1000,"propertyValue_executionIsolatio
nThreadInterruptOnTimeout":true,"propertyValue_executionIsolationThreadPoolKeyOverride":null,"propertyValue_executionIsolationSemaphoreMaxConcurrentRequests":10,"propertyValue_fallbackIsolationS
emaphoreMaxConcurrentRequests":10,"propertyValue_metricsRollingStatisticalWindowInMilliseconds":10000,"propertyValue_requestCacheEnabled":true,"propertyValue_requestLogEnabled":true,"reportingHo
sts":1,"threadPool":"FortuneService"}

data:
{"type":"HystrixThreadPool","name":"FortuneService","currentTime":1443629210165,"currentActiveCount":0,"currentCompletedTaskCount":3,"currentCorePoolSize":10,"currentLargestPoolSize":3,"currentM
aximumPoolSize":10,"currentPoolSize":3,"currentQueueSize":0,"currentTaskCount":3,"rollingCountThreadsExecuted":0,"rollingMaxActiveThreads":0,"rollingCountCommandRejections":0,"propertyValue_queu
eSizeRejectionThreshold":5,"propertyValue_metricsRollingStatisticalWindowInMilliseconds":10000,"reportingHosts":1}

data:
{"type":"HystrixCommand","name":"getFortune","group":"FortuneService","currentTime":1443629210667,"isCircuitBreakerOpen":false,"errorPercentage":0,"errorCount":0,"requestCount":0,"rollingCountBa
dRequests":0,"rollingCountCollapsedRequests":0,"rollingCountEmit":0,"rollingCountExceptionsThrown":0,"rollingCountFailure":0,"rollingCountEmit":0,"rollingCountFallbackFailure":0,"rollingCountFal
lbackRejection":0,"rollingCountFallbackSuccess":0,"rollingCountResponsesFromCache":0,"rollingCountSemaphoreRejected":0,"rollingCountShortCircuited":0,"rollingCountSuccess":0,"rollingCountThreadP
oolRejected":0,"rollingCountTimeout":0,"currentConcurrentExecutionCount":0,"rollingMaxConcurrentExecutionCount":0,"latencyExecute_mean":0,"latencyExecute":
{"0":0,"25":0,"50":0,"75":0,"90":0,"95":0,"99":0,"99.5":0,"100":0},"latencyTotal_mean":0,"latencyTotal":
{"0":0,"25":0,"50":0,"75":0,"90":0,"95":0,"99":0,"99.5":0,"100":0},"propertyValue_circuitBreakerRequestVolumeThreshold":20,"propertyValue_circuitBreakerSleepWindowInMilliseconds":5000,"propertyV
alue_circuitBreakerErrorThresholdPercentage":50,"propertyValue_circuitBreakerForceOpen":false,"propertyValue_circuitBreakerForceClosed":false,"propertyValue_circuitBreakerEnabled":true,"property
Value_executionIsolationStrategy":"THREAD","propertyValue_executionIsolationThreadTimeoutInMilliseconds":1000,"propertyValue_executionTimeoutInMilliseconds":1000,"propertyValue_executionIsolatio
nThreadInterruptOnTimeout":true,"propertyValue_executionIsolationThreadPoolKeyOverride":null,"propertyValue_executionIsolationSemaphoreMaxConcurrentRequests":10,"propertyValue_fallbackIsolationS
emaphoreMaxConcurrentRequests":10,"propertyValue_metricsRollingStatisticalWindowInMilliseconds":10000,"propertyValue_requestCacheEnabled":true,"propertyValue_requestLogEnabled":true,"reportingHo
sts":1,"threadPool":"FortuneService"}

data:
{"type":"HystrixThreadPool","name":"FortuneService","currentTime":1443629210667,"currentActiveCount":0,"currentCompletedTaskCount":3,"currentCorePoolSize":10,"currentLargestPoolSize":3,"currentM
aximumPoolSize":10,"currentPoolSize":3,"currentQueueSize":0,"currentTaskCount":3,"rollingCountThreadsExecuted":0,"rollingMaxActiveThreads":0,"rollingCountCommandRejections":0,"propertyValue_queu
eSizeRejectionThreshold":5,"propertyValue_metricsRollingStatisticalWindowInMilliseconds":10000,"reportingHosts":1}
```

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 is 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.

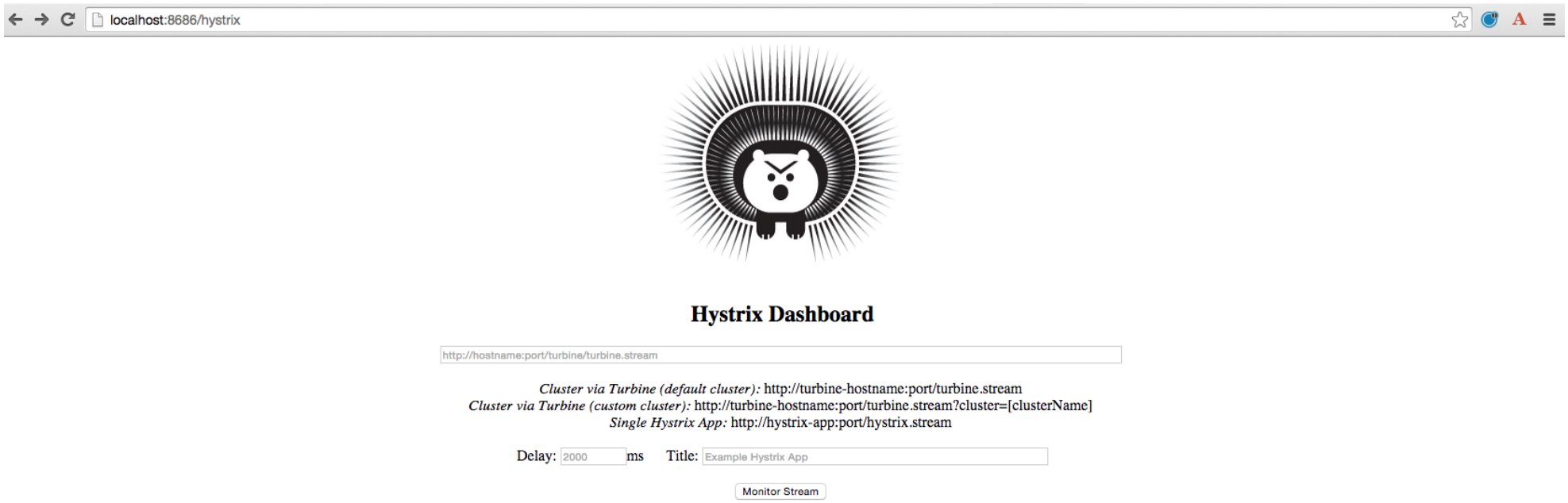
```
@SpringBootApplication
@EnableHystrixDashboard
public class HystrixDashboardApplication {

    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 (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 /` 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.

Back to TOP