



Spring Cloud

Spring Cloud Hystrix Circuit Breaker Pattern:

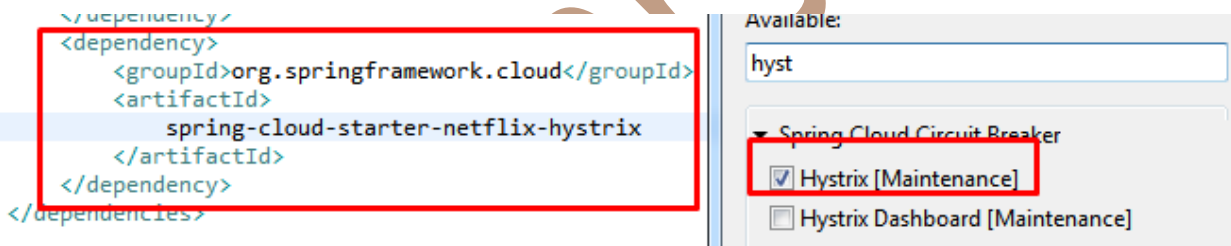
Spring Cloud Hystrix as another Netflix library used for a **fault-tolerant and latency-tolerant** in microservice implementation based on circuit breaker pattern. Hystrix is based on the principle: **fault-tolerant and latency-tolerant** in microservice implementation.

Enabling Circuit Breaker

Step-1:

Update the doctor-service as we are calling patient-service from there using feign client .As per Circuit Breaker pattern, if the patient-service is down or busy at the moment when doctor-service is calling it, then rather getting exception at doctor-service, an alternate or fall back method should get fired automatically and have to send some reasonable message to the client. Hence we need to enabled the Hystrix here to get the circuit breaker pattern to work.

Add the Hystrix dependency to the service. If developing from scratch, select the following libraries:



Step-2:

In the **DoctorServiceApplication** class add **@EnableCircuitBreaker** as shown below. This command will tell Spring Cloud Hystrix to enable a circuit breaker for this application. It also exposes the /hystrix.stream endpoint for metrics collection

```
@SpringBootApplication
@EnableSwagger2
@EnableFeignClients
@EnableDiscoveryClient
@EnableCircuitBreaker
public class DoctorServiceApplication
```



Spring Cloud

Step-3:

Add annotation on trips Controllers service method I.e. getAllTrips() with @HystrixCommand with the fall back method name as shown below.

```
@GetMapping(value="/trips")
@HystrixCommand(fallbackMethod="viewTripsSecondBoard",
    commandKey="primaryCommandTC",
    threadPoolKey="primaryThreadTC",
    commandProperties= {
        @HystrixProperty(name="circuitBreaker.requestVolumeThreshold", value="2"),
        @HystrixProperty(name="circuitBreaker.sleepWindowInMilliseconds", value="10000"),
        @HystrixProperty(name="execution.isolation.thread.timeoutInMilliseconds", value="5000")
    },
    threadPoolProperties= {
        @HystrixProperty(name="coreSize",value="4"),
        @HystrixProperty(name="maxQueueSize",value="10")
    })

public ResponseEntity<List<TripBean>> viewTripsBoard(@RequestHeader(name="Authorization") String authToken) {
    logger.info(new Date()+"i am in primary board");
    logger.info("authToken:\t"+authToken);

    String vehicleData = outboundCommunicator.fetchVechileInfo(authToken,1);

    List<TripBean> tripsBeanList = tripsService.findAll(authToken);
    if(tripsBeanList != null && tripsBeanList.size() >0) {
        return ResponseEntity.ok(tripsBeanList);
    }else {
        return ResponseEntity.badRequest().build();
    }
}

// @HystrixCommand(fallbackMethod="viewTripsFailFast",
//     commandKey="getSecTripsKey",
//     threadPoolKey="getSecTripsThread",
//     commandProperties= {
//         @HystrixProperty(name="circuitBreaker.requestVolumeThreshold",value="2"),
//         @HystrixProperty(name="circuitBreaker.sleepWindowInMilliseconds",value="1000")
//     },
//     threadPoolProperties= {
//         @HystrixProperty(name="coreSize",value="4"),
//         @HystrixProperty(name="maxQueueSize",value="10")
//     })
// @GetMapping(value="/trips")
public ResponseEntity<List<TripBean>> viewTripsFromSecondarySource() {
    logger.info("i am in Trips Board");
    logger.info(new Date()+"Fecthing Trips from Secondary Method");
    List<TripBean> tripsBeanList = getTripsFromSecondarySource();
    return ResponseEntity.ok(tripsBeanList);
}

// @GetMapping(value="/trips")
public ResponseEntity<List<TripBean>> viewTripsFailFast() {
    logger.info("i am in Trips Board");
    logger.info(new Date()+"Secondary source is down too. Fetching defaults trips");
    List<TripBean> tripsList = new ArrayList<>();
    TripBean trip = new TripBean();
    trip.setId(0);
    tripsList.add(trip);
    return ResponseEntity.ok(tripsList);
}
```



Spring Cloud

```
@HystrixCommand(fallbackMethod="viewTripsFromSecondarySource",
    commandKey="getTripsKey", threadPoolKey="getTripsThread",

    commandProperties= {
        @HystrixProperty(name="circuitBreaker.requestVolumeThreshold", value="2"),

        @HystrixProperty(name="circuitBreaker.sleepWindowInMilliseconds", value="1000")
    },

    threadPoolProperties= {

        @HystrixProperty(name="coreSize", value="4"),

        @HystrixProperty(name="maxQueueSize", value="10")

    })

@GetMapping(value="/trips")
public ResponseEntity<List<TripBean>> viewTripsBoard() {

    Logger.info("i am in Trips Board");

    Logger.info(new Date()+"Fecthing Trips from Primary Method");

    List<TripBean> tripsBeanList = tripsService.findAll();

    return ResponseEntity.ok(tripsBeanList);

}
```

This tells Spring that this method is prone to failure. Spring Cloud libraries wrap these methods to handle fault tolerance and latency tolerance by enabling circuit breaker. The Hystrix command typically follows with a fallback method. In case of failure, Hystrix automatically enables the fallback method mentioned and diverts traffic to the fallback method.

Step-4:

Restart the doctor-service, and give the request to register patient by doctor as shown below. We could see the message written fall back method when there is not availability of patient-service.



Spring Cloud

View Results Tree

Name:

Comments:

Write results to file / Read from file

Filename: Log/Display Only: ☐ Errors ☐ Successes

Search: ☐ Case sensitive ☐ Regular exp.

Text

HTTP Request

CIRCUIT BRAKER ENABLED No Response from the doctor-service at this moment. service will be back shortly-Sun Nov 26 22:41:41 IST 2017

Enabling the Hystrix Dashboard

Step-1: create a separate application with the name “hystrix-dashboard” as shown below:

- hystrix-dashboard [boot]
 - .mvn
 - .settings
 - src
 - target
 - .classpath
 - .gitignore
 - .project
 - mvnw
 - mvnw.cmd
 - pom.xml

With the startes below:



Spring Cloud

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

  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
  </dependency>
</dependencies>
```

Note:

Hystrix dashboard will use freemarker template engine to render its dashboard. Hence it will download the latest version of free marker dependency in to our maven local repository. If it is not downloadable by maven, the explicitly pass it as maven dependency in pom .xml file as shown below

```
<!-- https://mvnrepository.com/artifact/org.freemarker/freemarker -->
<dependency>
  <groupId>org.freemarker</groupId>
  <artifactId>freemarker</artifactId>
  <version>2.3.28</version>
</dependency>
```

Step-3: add `@EnableHystrixDashboard` annotation on application java class as shown below.



Spring Cloud

```
2
3 import org.springframework.boot.SpringApplication;
4
5
6 @SpringBootApplication
7 @EnableHystrixDashboard
8 public class HystricsDashboardApplication {
9
10     public static void main(String[] args) {
11         SpringApplication.run(HystricsDashboardApplication.class, args);
12     }
13 }
```

Step-4: configure the application name and port as shown below. Hystrix dashboard will start up on the default port 9999.

```
1 spring.application.name=hystrics-dashboard
2 server.port=9999
```

Step-5:

Hystrix Dashboard

As we have added hystrix dashboard dependency, hystrix has provided a nice Dashboard and a Hystrix Stream in the bellow URLs:

<http://localhost:9095/hystrix.stream> –Here as doctor-service is enabled with Circuit Breaker, a continuous stream Hystrix will generates on it to check health and being monitored by Hystrix.



Spring Cloud

← → ↻ localhost:9095/hystrix.stream

Apps spring-boot-excellent angular-js-excellent converts and formats Are Banks Open? Post Office Open? Spring Boot hello world java - Is it possible to e-books HMS-Pro


ping:

```
data:
{"type":"HystrixCommand","name":"addPatient","group":"DoctorController","currentTime":1540373188985,"isCircuitBreakerOpen":false,"errorPercentage":100,"errorCount":1,"requestCount":1,"rollingCountBadRequests":0,"rollingCountCollapsedRequests":0,"rollingCountEmit":0,"rollingCountExceptionsThrown":0,"rollingCountFailure":0,"rollingCountFallbackEmit":0,"rollingCountFallbackFailure":0,"rollingCountFallbackMissing":0,"rollingCountFallbackRejection":0,"rollingCountFallbackSuccess":0,"rollingCountResponsesFromCache":0,"rollingCountSemaphoreRejected":0,"rollingCountShortCircuited":0,"rollingCountSuccess":0,"rollingCountThreadPoolRejected":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,"propertyValue_circuitBreakerErrorThresholdPercentage":50,"propertyValue_circuitBreakerForceOpen":false,"propertyValue_circuitBreakerForceClosed":false,"propertyValue_circuitBreakerEnabled":true,"propertyValue_executionIsolationStrategy":"THREAD","propertyValue_executionIsolationThreadTimeoutInMilliseconds":1000,"propertyValue_executionTimeoutInMilliseconds":1000,"propertyValue_executionIsolationThreadInterruptOnTimeout":true,"propertyValue_executionIsolationThreadPoolKeyOverride":null,"propertyValue_executionIsolationSemaphoreMaxConcurrentRequests":10,"propertyValue_fallbackIsolationSemaphoreMaxConcurrentRequests":10,"propertyValue_metricsRollingStatisticalWindowInMilliseconds":10000,"propertyValue_requestCacheEnabled":true,"propertyValue_requestLogEnabled":true,"reportingHosts":1,"threadPool":"DoctorController"}
```

```
data:
{"type":"HystrixThreadPool","name":"DoctorController","currentTime":1540373188986,"currentActiveCount":0,"currentCompletedTaskCount":1,"currentCorePoolSize":10,"currentLargestPoolSize":1,"currentMaximumPoolSize":10,"currentPoolSize":1,"currentQueueSize":0,"currentTaskCount":1,"rollingCountThreadsExecuted":0,"rollingMaxActiveThreads":0,"rollingCountCommandRejections":0,"propertyValue_queueSizeRejectionThreshold":5,"propertyValue_metricsRollingStatisticalWindowInMilliseconds":10000,"reportingHosts":1}
```

run the service and open the url <http://localhost:9999/hystrix>. the following dash board will be opened.

← → ↻ localhost:9999/hystrix



Hystrix Dashboard

http://localhost:7070/hystrix.stream

Cluster via Turbine (default cluster): <http://turbine-hostname:port/turbine.stream>
Cluster via Turbine (custom cluster): [http://turbine-hostname:port/turbine.stream?cluster=\[clusterName\]](http://turbine-hostname:port/turbine.stream?cluster=[clusterName])
Single Hystrix App: <http://hystrix-app:port/hystrix.stream>

Delay: ms Title:



Spring Cloud

Here in dash board to see the stream of doctor-service I.e. failure services etc, give the doctor-service stream url I.e. <http://localhost:9095/hystrix.stream> in the bar above and give the application-name I.e.doctor-service as Title and click on Monitor stream. Now we should see the stream as below:



If there are continuous failures, then the circuit status will be changed to open. This can be done by hitting the preceding URL a number of times. In the open state, the original service will no longer be checked. The Hystrix Dashboard will show the status of the circuit as **Open** and **Closed**. Once a circuit is opened, periodically, the system will check for the original service status for recovery. When the original service is back, the circuit breaker will fall back to the original service and the status will be set to **Closed**.