

Circuit Breaker Metric Aggregation

Table of Contents

1. What You Will Learn
 2. Start "all the things"
 3. Set up `turbine`
 4. Deploying to PCF
 5. Deploy `greeting-hystrix` to PCF
-

1. What You Will Learn

- How to aggregate multiple metric streams with `turbine`
- How to use Turbine in Pivotal Cloud Foundry

2. Start "all the things"

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
```



4. Start the `greeting-hystrix`

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

Next, start a **second instance** of `greeting-hystrix` on port 8081

mac, linux

windows

```
$ SERVER_PORT=8081 mvn spring-boot:run
```

5. Start the `hystrix-dashboard`

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

Allow a few moments for `greeting-hystrix` and `fortune-service` to register with the `service-registry`.

3. Set up `turbine`


Looking at individual application instances in the Hystrix Dashboard is not very useful in terms of understanding the overall health of the system. Turbine is an application that aggregates all of the relevant `/hystrix.stream` endpoints into a combined `/turbine.stream` for use in the Hystrix Dashboard.

1. Review the `turbine/pom.xml` file. By adding `spring-cloud-starter-turbine` to the classpath this application is eligible to aggregate metrics via Turbine.

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

2. Review the following file:

`turbine/src/main/java/io/pivotal/TurbineApplication.java`. Note the use of the `@EnableTurbine` annotation. This creates a turbine application.



```
package io.pivotal;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.turbine.EnableTurbine;


@SpringBootApplication
@EnableTurbine
public class TurbineApplication {

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

}
```


3. Review the following file: `turbine/src/main/resources/bootstrap.yml`.

`turbine.appConfig` is a list of Eureka `serviceIds` that Turbine will use to lookup instances. `turbine.aggregator.clusterConfig` is the Turbine cluster these services belong to (how they will be grouped).



```
server:
  port: 8585
spring:
  application:
    name: turbine
turbine:
  aggregator:
    clusterConfig: GREETING-HYSTRIX
  appConfig: greeting-hystrix
```

4. Open a new terminal window. Start the `turbine` app



```
$ cd turbine
$ mvn spring-boot:run
```

5. Wait for the `turbine` application to register with `service-registry` (<http://localhost:8761/>).

6. View the turbine stream in a browser <http://localhost:8585/turbine.stream?cluster=GREETING-HYSTRIX>

```
localhost:8585/turbine.stream?cluster=GREETING-HYSTRIX

: ping
data:
{"rollingCountFallbackSuccess":0,"rollingCountFallbackFailure":0,"propertyValue_circuitBreakerRequestVolumeThreshold":20,"propertyValue_circuitBreakerForceOpen":false,"propertyValue_metricsRollingStatisticalWindowInMilliseconds":10000,"latencyTotal_mean":0,"rollingMaxConcurrentExecutionCount":0,"type":"HystrixCommand","rollingCountResponsesFromCache":0,"rollingCountBadRequests":0,"rollingCountTimeout":0,"propertyValue_executionIsolationStrategy":"THREAD","rollingCountFailure":0,"rollingCountExceptionsThrown":0,"threadPool":"FortuneService","latencyExecute_mean":0,"isCircuitBreakerOpen":false,"errorCount":0,"rollingCountSemaphoreRejected":0,"group":"FortuneService","latencyTotal":{"0":0,"99":0,"100":0,"25":0,"90":0,"50":0,"95":0,"99.5":0,"75":0},"requestCount":0,"rollingCountCollapsedRequests":0,"rollingCountShortCircuited":0,"propertyValue_circuitBreakersSleepWindowInMilliseconds":5000,"latencyExecute":{"0":0,"99":0,"100":0,"25":0,"90":0,"50":0,"95":0,"99.5":0,"75":0},"rollingCountEmit":0,"currentConcurrentExecutionCount":0,"propertyValue_executionIsolationSemaphoreMaxConcurrentRequests":10,"errorPercentage":0,"rollingCountThreadPoolRejected":0,"propertyValue_circuitBreakerEnabled":true,"propertyValue_executionIsolationThreadInterruptOnTimeout":true,"propertyValue_requestCacheEnabled":true,"rollingCountFallbackRejection":0,"propertyValue_requestLogEnabled":true,"rollingCountSuccess":0,"propertyValue_fallbackIsolationSemaphoreMaxConcurrentRequests":10,"propertyValue_circuitBreakerErrorThresholdPercentage":50,"propertyValue_circuitBreakerForceClosed":false,"name":"getFortune","reportingHosts":1,"propertyValue_executionIsolationThreadPoolKeyOverride":"null","propertyValue_executionIsolationThreadTimeoutInMilliseconds":1000}

data: {"reportingHostsLast10Seconds":1,"name":"meta","type":"meta","timestamp":1443645029559}

data:
{"currentCorePoolSize":10,"currentLargestPoolSize":10,"propertyValue_metricsRollingStatisticalWindowInMilliseconds":10000,"currentActiveCount":0,"currentMaximumPoolSize":10,"currentQueueSize":0,"type":"HystrixThreadPool","currentTaskCount":77,"currentCompletedTaskCount":77,"rollingMaxActiveThreads":0,"rollingCountCommandRejections":0,"name":"FortuneService","reportingHosts":1,"currentPoolSize":10,"propertyValue_queueSizeRejectionThreshold":5,"rollingCountThreadsExecuted":0}

: ping
data: {"reportingHostsLast10Seconds":1,"name":"meta","type":"meta","timestamp":1443645032567}

: ping
data: {"reportingHostsLast10Seconds":1,"name":"meta","type":"meta","timestamp":1443645035580}

: ping
data: {"reportingHostsLast10Seconds":1,"name":"meta","type":"meta","timestamp":1443645038589}

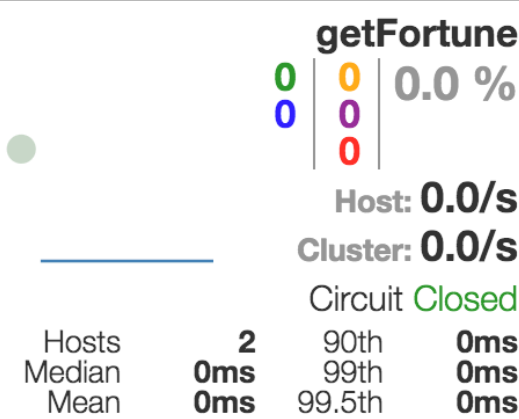
: ping
data: {"reportingHostsLast10Seconds":1,"name":"meta","type":"meta","timestamp":1443645041601}

: ping
data: {"reportingHostsLast10Seconds":1,"name":"meta","type":"meta","timestamp":1443645044601}
```

- 7. Configure the `hystrix-dashboard` (<http://localhost:8686/hystrix>) to consume the turbine stream. Enter `http://localhost:8585/turbine.stream?cluster=GREETING-HYSTRIX`
- 8. Experiment! Refresh the `greeting-hystrix` root endpoint several times. Take down the `fortune-service` app. What does the dashboard do?
- 9. Notice how the hystrix dashboard's `getFortune` circuit reflects that it's monitoring two hosts, as shown here:

Hystrix Stream: Combined Stream

Circuit Sort: [Error then Volume](#) | [Alphabetical](#) | [Volume](#)



Thread Pools Sort: [Alphabetical](#) | [Volume](#) |

- 10. When done, stop the `config-server`, `service-registry`, `fortune-service`, `greeting-hystrix`, `hystrix-dashboard` and `turbine` applications.

What Just Happened?

Turbine discovered the `greeting-hystrix` application through the `service-registry` application. Turbine then consumed the `/hystrix.stream` endpoint and rolled that up into an aggregate `/turbine.stream`. If we had multiple `greeting-hystrix` applications running, all the metrics would be consumed from this single endpoint (`/turbine.stream`)

4. Deploying to PCF

In PCF, the traditional Turbine model of pulling metrics from all the distributed Hystrix enabled applications via HTTP doesn't work when using the `route` `registrationMethod`. Read [here](http://docs.pivotal.io/spring-cloud-services/service-registry/registering-a-service.html) (<http://docs.pivotal.io/spring-cloud-services/service-registry/registering-a-service.html>) for more details on registration methods.

When applications register using the `route` method every application has the same `hostname` (every app instance has the same URL for a given app). Therefore it is unknown from the Turbine perspective if all metrics are properly being collected. The problem is solved with Turbine AMQP. Metrics are published through a message broker. We'll use RabbitMQ.

5. Deploy `greeting-hystrix` to PCF

1. Create a Circuit Breaker Dashboard Service Instance

```
$ cf create-service p-circuit-breaker-dashboard standard circuit-breaker-dashboard
```



When a Circuit Breaker Service instance is created, three items get provisioned:

- Hystrix Dashboard application instance
- Turbine AMQP application instance
- RabbitMQ Service Instance

This process takes some time and won't be immediately available for binding. Give it a couple of minutes.

Click on the **Services** tab and the **Circuit Breaker** entry to navigate to your service.

SPACE
development

- 1 Running
- 1 Stopped
- 0 Crashed




Apps (2)

Services (3)

Settings

Services

Add Service

SERVICE	NAME	BOUND APPS	PLAN	
 Service Registry	my-registry-service	1	free -	>
 Circuit Breaker	my-circuit-breaker	0	free -	>
 Config Server	my-config-server	1	free -	>

Then, click on the **Manage** link to determine when the `circuit-breaker` dashboard is ready.

SERVICE
 **Circuit Breaker**

INSTANCE NAME
my-circuit-breaker

SERVICE PLAN
standard

[Manage](#) [Docs](#) [Support](#)

App Binding (1)

Plan

Settings

Bound Apps

Edit Bindings

greeting-hystrix

2. Package and push the `greeting-hystrix` application.

```
$ mvn clean package
```

..and:

```
$ cf push greeting-hystrix -p target/greeting-hystrix-0.0.1-SNAPSHOT.jar -m 768m  
--random-route --no-start
```

3. Bind services for the `greeting-hystrix`.

```
$ cf bind-service greeting-hystrix config-server  
$ cf bind-service greeting-hystrix service-registry  
$ cf bind-service greeting-hystrix circuit-breaker-dashboard
```

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.

4. 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 api.sys.gcp.esuez.org
```



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. Start the `greeting-hystrix` app.

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



6. Experiment! Refresh the `greeting-hystrix` / endpoint several times. Take down the fortune-service app. Scale the greeting-hystrix app. What does the dashboard do?

What Just Happened?

The `greeting-hystrix` application is publishing metrics via AMQP to RabbitMQ (this can be discovered by looking at `VCAP_SERVICES`). Those metrics are then consumed and aggregated by Turbine. The Circuit Breaker Dashboard then consumes the Turbine endpoint. All of this detail has been abstracted away by using the PCF Circuit Breaker Dashboard Service.