

# Laboratorio 11

## Integración de Hystrix

Incluir Hystrix en una aplicación *Spring Boot* es muy sencillo:

- **Paso 1:** Añadir al fichero ***pom.xml*** del proyecto el *starter* de *Hystrix*:

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

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

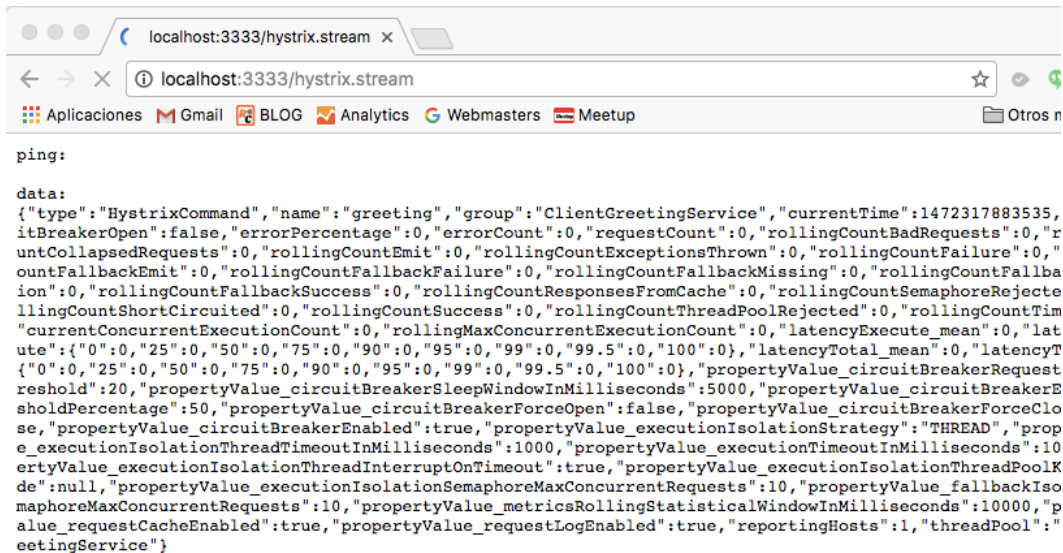
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-turbine</artifactId>
    </dependency>

</dependencies>
```

- **Paso 2:** Habilitar *Hystrix* en la aplicación (***@EnableCircuitBreaker***), lo que permitirá exponer *hystrix stream* a través de la *URI* ***"/hystrix.stream"***

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

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



```
ping:
data:
{"type": "HystrixCommand", "name": "greeting", "group": "ClientGreetingService", "currentTime": 1472317883535,
"itBreakerOpen": false, "errorPercentage": 0, "errorCount": 0, "requestCount": 0, "rollingCountBadRequests": 0, "r
untCollapsedRequests": 0, "rollingCountEmit": 0, "rollingCountExceptionsThrown": 0, "rollingCountFailure": 0, "r
ountFallbackEmit": 0, "rollingCountFallbackFailure": 0, "rollingCountFallbackMissing": 0, "rollingCountFallba
ion": 0, "rollingCountFallbackSuccess": 0, "rollingCountResponsesFromCache": 0, "rollingCountSemaphoreRejecte
llingCountShortCircuited": 0, "rollingCountSuccess": 0, "rollingCountThreadPoolRejected": 0, "rollingCountTim
currentConcurrentExecutionCount": 0, "rollingMaxConcurrentExecutionCount": 0, "latencyExecute_mean": 0, "lat
ute": {"0": 0, "25": 0, "50": 0, "75": 0, "90": 0, "95": 0, "99": 0, "99.5": 0, "100": 0}, "latencyTotal_mean": 0, "latencyT
{"0": 0, "25": 0, "50": 0, "75": 0, "90": 0, "95": 0, "99": 0, "99.5": 0, "100": 0}, "propertyValue_circuitBreakerRequest
reshold": 20, "propertyValue_circuitBreakerSleepWindowInMilliseconds": 5000, "propertyValue_circuitBreakerE
sholdPercentage": 50, "propertyValue_circuitBreakerForceOpen": false, "propertyValue_circuitBreakerForceClo
se", "propertyValue_circuitBreakerEnabled": true, "propertyValue_executionIsolationStrategy": "THREAD", "prop
e_executionIsolationThreadTimeoutInMilliseconds": 1000, "propertyValue_executionTimeoutInMilliseconds": 10
ertyValue_executionIsolationThreadInterruptOnTimeout": true, "propertyValue_executionIsolationThreadPoolK
de": null, "propertyValue_executionIsolationSemaphoreMaxConcurrentRequests": 10, "propertyValue_fallbackIso
maphoreMaxConcurrentRequests": 10, "propertyValue_metricsRollingStatisticalWindowInMilliseconds": 10000, "p
alue_requestCacheEnabled": true, "propertyValue_requestLogEnabled": true, "reportingHosts": 1, "threadPool": "
eetingService"}
```

## Hystrix Dashboard

Como comenté antes, *Hystrix Dashboard* es una consola que nos ofrece *Netflix* para explotar las métricas en tiempo real del estado de los *circuit breakers* de las aplicaciones, procesando los *hystrix stream* que éstas generan y representado los resultados de manera gráfica en un cuadro de mandos.

Partiendo del laboratorio de implementación de un *circuit breaker*, vamos a monitorizar su comportamiento creando una nueva aplicación *Spring Boot* que incluya el *dashboard* de *Hystrix*:

- **Paso 1:** Añadir la siguiente dependencia al fichero *pom.xml* del proyecto:

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

- **Paso 2:** Crear la clase principal de aplicación *Spring Boot* y anotarla con ***@EnableHystrixDashboard***

```

@SpringBootApplication
@EnableHystrixDashboard
public class HystrixDashboardApp {

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

```

- **Paso 3:** Informar las siguientes propiedades en el fichero *application.yml* del proyecto

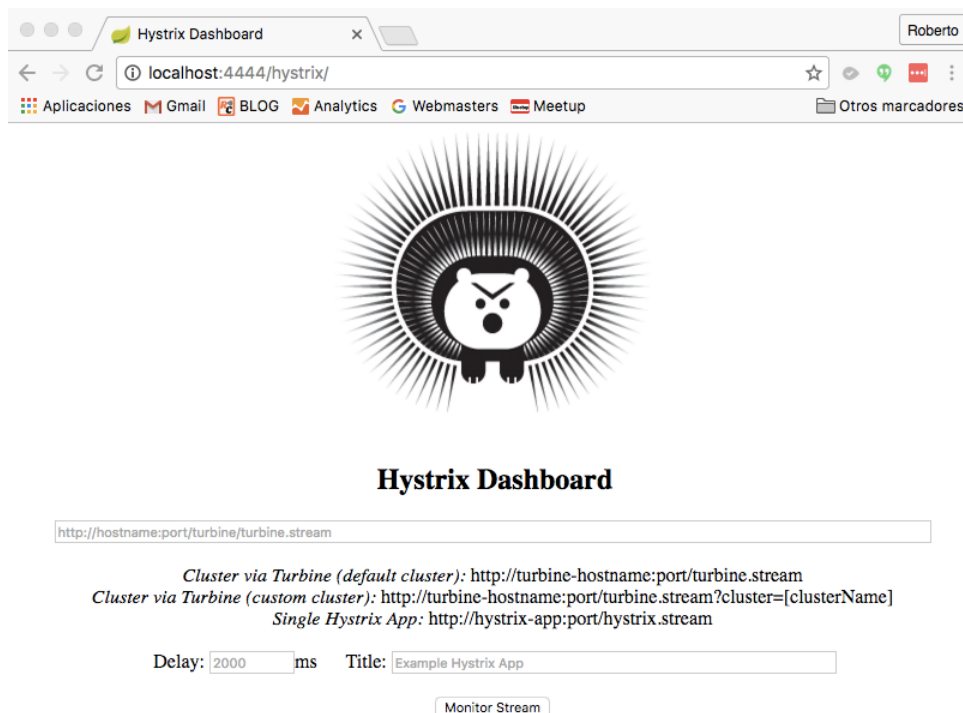
```

# Spring properties
spring:
  application:
    name: hystrix-dashboard-service # Service registers under this name

server:
  port: 4444 # HTTP (Tomcat) port

```

- **Paso 4:** Iniciar la aplicación y accede a la URL <http://localhost:4444/hystrix/>. Si todo fue bien, deberías ver la siguiente pantalla:

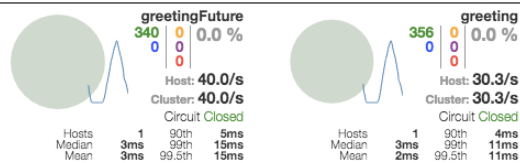


- **Paso 5:** Por último, arranca la aplicación de nuestro ejemplo de partida (*ClientGreetingServer.java*) e introduce en *Hystrix Dashboard* la *uri* donde el microservicio está publicando sus *hystrix stream* (ej: <http://localhost:3333/hystrix.stream>). Haz *click* en el botón *Monitor Stream* y te aparecerá la siguiente consola con información en tiempo real del estado de los *circuit breaker* de la aplicación:

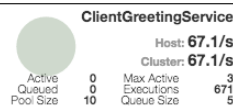
Hystrix Stream: <http://localhost:3333/hystrix.stream>



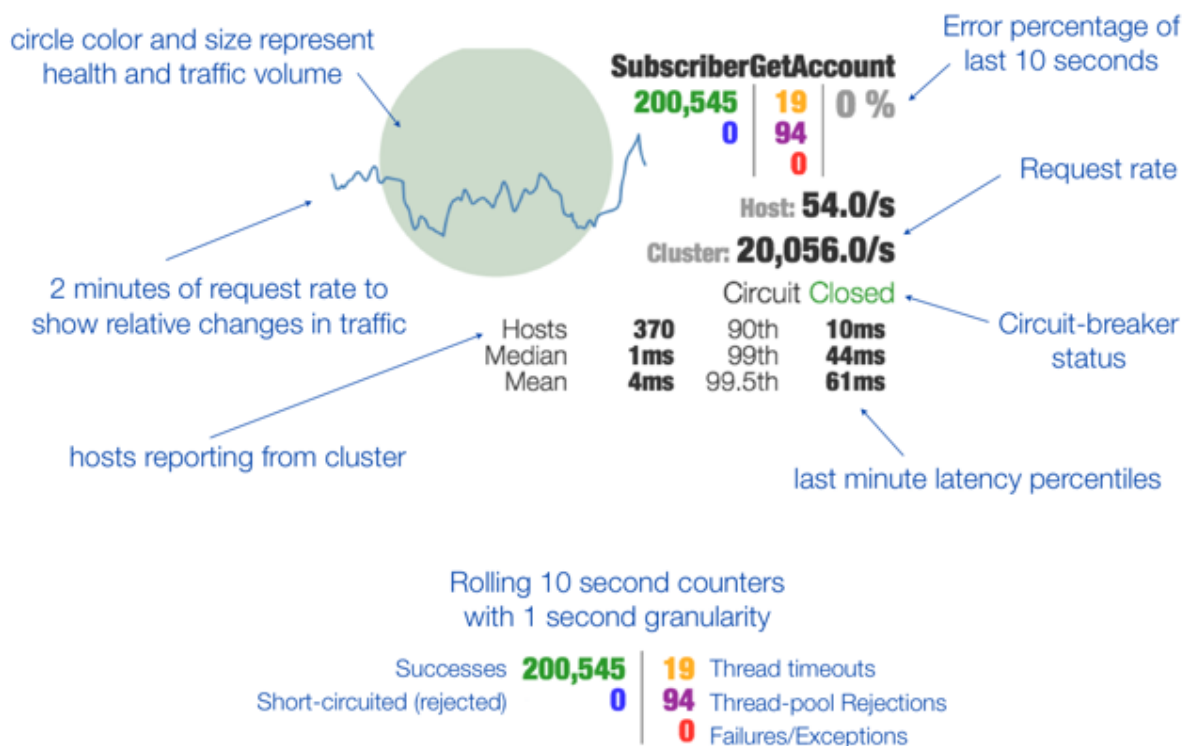
Circuit Sort: [Error then Volume](#) | [Alphabetical](#) | [Volume](#) | [Error](#) | [Mean](#) | [Median](#) | [90](#) | [99](#) | [99.5](#)  
Success | Short-Circuited | Timeout | Rejected | Failure | Error %



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

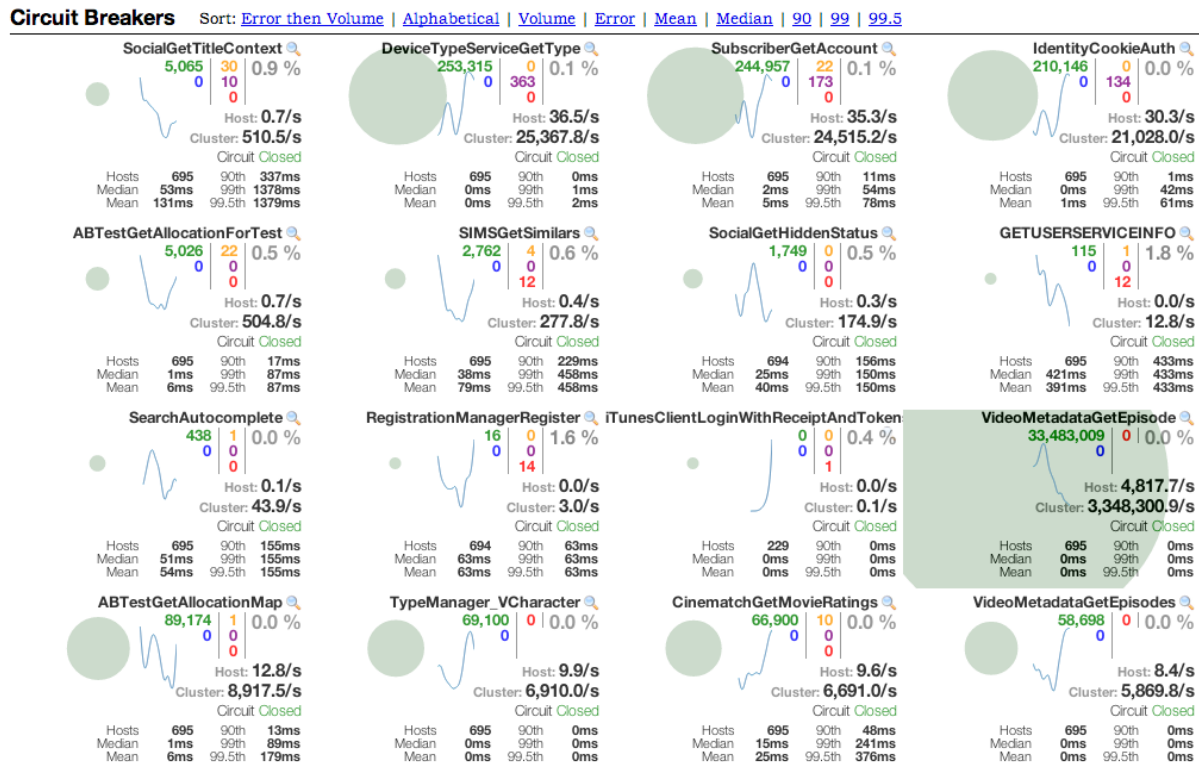


El siguiente gráfico muestra, de forma detallada, el significado de cada una de las métricas que aparecen en *Hystrix Dashboard*:



# Turbine

A diferencia de *Hystrix Stream*, que proporciona información de una única instancia de una aplicación, **Turbine** permite **agregar** los **streams** de las múltiples instancias de dicha aplicación de forma clusterizada. A continuación te muestro un ejemplo de un *Hystrix dashboard* donde se presenta información agregada de varios servicios de una compañía obtenida con *Turbine*.



*Turbine* puede utilizar **Eureka** para localizar los servicios que serán monitorizados.

- **Paso 1:** Añade la siguiente dependencia a tu proyecto:

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

- **Paso 2:** Habilitar *Turbine* en la clase principal de la aplicación *Spring Boot* (*@EnableTurbine*):

```
@SpringBootApplication
@EnableHystrixDashboard
@EnableTurbine
public class HystrixDashboardApp {

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

- **Paso 3:** Establecer las siguientes propiedades en el fichero de configuración del proyecto
  - *turbine.appConfig*: lista de servicios Eureka que Turbine utilizará para localizar las instancias
  - *turbine.aggregator.clusterConfig*: nombre del *cluster* utilizado para agrupar las instancias de cada uno servicio publicado en *Eureka*. Debe coincidir con el nombre utilizado en *appConfig* y estar en mayúsculas

Añadimos al fichero *application.yml* estas dos propiedades y la configuración de acceso a *Eureka*:

```
turbine:
  aggregator:
    clusterConfig: GREETING-CLIENT-SERVICE
    appConfig: greeting-client-service

# Configure this Discovery Server
eureka:
  instance:
    hostname: localhost
  client: # Not a client, don't register with yourself
    registerWithEureka: false
    serviceUrl:
      defaultZone: http://localhost:1111/eureka/
```

- **Paso 4:** Por último, una vez configurado el *cluster*, la monitorización del mismo estará disponible en la URL “/stream?cluster=CLUSTER-NAME”, que en nuestro caso de ejemplo será <http://localhost:4444/turbine.stream?cluster=GREETING-CLIENT-SERVICE>. Introducimos esta URL en *Hystrix Dashboard* para visualizar la información agregada que *Turbine* nos ofrece.

