Spring framework 5 - Metrics

Actuator

Para configurar actuator debemos incluir la siguiente dependencia:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

Configuración

Una vez incluida la dependencia, debemos especificar los recursos que deseamos incluir en actuator agregando lo siguiente en el application.properties:

```
management.endpoints.web.exposure.include=health,info,metrics,prometheus
```

Configuración de métricas

Es posible crear métricas propias para analizar el funcionamiento de mi aplicación, para esto incluirémos la siguiente dependencia:

```
<dependency>
   <groupId>io.micrometer</groupId>
   <artifactId>micrometer-registry-prometheus</artifactId>
</dependency>
```

Uso de métricas

Una vez configurado micrometer, debemos definir los servicios a los que deseamos monitorear utilizando la siguiente anotación:

```
 @Timed(value ="get.characters")
public ResponseEntity<List<String>> getCharacters() {
    log.info("Characters");
     returnnew ResponseEntity<List<String>>(characters,
HttpStatus.OK);
}
```

Una vez definida la anotación @Timed, las métricas de uso del endpoint serán expuestas en **/actuator/metrics**.

Métricas

Una vez hecho lo anterior podremos consultar el endpoint de **/actuator/metrics**:

```
"names": [
    "jvm.memory.max",
    "process.files.max",
    "get.characters"
    ........ //Más métricas
  ]
```

En esta salida veremos la nueva métrica get.characters, si se desea consultar el valor de la métrica se debe consultar **/actuator/metrics/get.characters**:

```
{
  "name": "get.characters",
  "description": null,
  "baseUnit": "seconds",
  "measurements": [
    {
      "statistic": "COUNT",
      "value": 3.0
    },
    {
      "statistic": "TOTAL_TIME",
      "value": 0.04963493
    },
    {
      "statistic": "MAX",
      "value": 0.04235978
    }
  ]
]
}
```

Prometheus

Las méticas se pueden consultar siguiendo la estructura de prometheus, para esto consultaremos utilizando la siguiente url:

/actuator/prometheus

Hecho lo anterior veremos una salida como la siguiente:

```
# HELP get_characters_seconds   # TYPE
get_characters_seconds summary
get_characters_seconds_count
{exception="None",method="GET",
status="200",uri="/characters",} 1.0
get_characters_seconds_sum
{exception="None",method="GET",
status="200",uri="/characters",} 0.009020125
```

Descarga de prometheus

Puedes descargar prometheus del siguiente enlace:

https://prometheus.io/download/

Una vez descargado, lo debes descomprimir entrar a la carpeta y ejecutar:

```
./prometheus
--config.file=prometheus.yml
```

Una vez iniciado puedes acceder a la siguiente URL:

http://localhost:9090/graph

Modificaremos el archivo prometheus.yml para que utilice las métricas que estamos generando en nuestra aplicación como se muestra a continuación:

```
scrape_configs:
  - job_name: 'prometheus'
    metrics_path: /actuator/prometheus

    scrape_interval: 5s
    static_configs:
      - targets: ['localhost:8080']
```

Una vez hecho lo anterior entra de nuevo a la Url y verás las métricas de tu aplicación

Descarga grafana

Puedes descargar grafana en :

https://grafana.com/grafana/download

Una vez descargado entra al folder en la carpeta de bin y ejecuta:

```
./grafana-server
```

Y abre la URL http://localhost:3000/

Selecciona en el side bar:

```
    - Configuración
    - Datasources
    - Add datasource
    - Selecciona Prometheus
        -Name = Prometheus
        -Url = http://localhost:9000
```

DEVS4J
Hecho por Devs para Devs

Creación de un dashboard

Para crear un dashboard de grafana selecciona el símbolo + de la barra de la izquierda y sigue los siguientes pasos:

- Crea un panel

- En la sección de Metrics, selecciona Prometheus y coloca el query que deseas utilizar, en el ejemplo utilizaremos :

rate(get_characters_seconds_count[5m])

- En el panel de la derecha puedes definir el título y el tipo de gráfica que deseas representar