



Micrometer

Monitorez simplement
votre application



Florentin C.
mardi 27 septembre 2022

Qui suis-je ?



Florentin C.

Arrivé chez Ubik en mai 2021
Développeur Java - Angular

HI



Sommaire

Micrometer



I - Présentation de Micrometer

II - Mise en place avec Spring

III - Métriques par défaut

IV - Métriques personnalisées

V - Visualisation des métriques

VI - Aller plus loin

I - Présentation de Micrometer

Micrometer



Que propose Micrometer ?

Supporte plusieurs plateformes de monitoring



Supporté par plusieurs frameworks



I - Présentation de Micrometer

Les registries

Micrometer



REGISTER NOW

II - Mise en place avec Spring

Micrometer



Dépendance principale pour l'intégration de Micrometer

```
<dependency>
  <groupId>io.micrometer</groupId>
  <artifactId>micrometer-core</artifactId>
</dependency>
```

Debug envoi tags:

logging.level.io.micrometer.datadog.DatadogMeterRegistry = TRACE

II - Mise en place avec Spring

Micrometer



L'intégration avec Spring Boot

Par dépendance dans le pom

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

Par instantiation du composant **MeterRegistry**

```
MeterRegistry registry = new DatadogMeterRegistry(datadogConfig, clock);

Metrics.addRegistry(registry);
```

III - Métriques par défaut

Micrometer



Les métriques recensées par défaut avec Spring Boot

Démarrage et suivi du statut d'activité de l'application



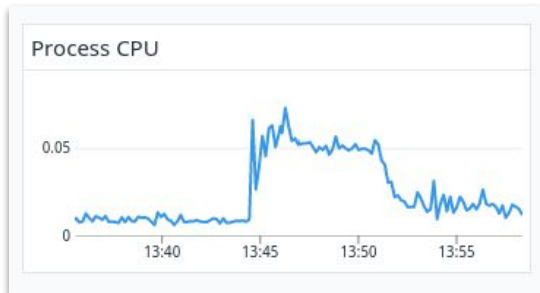
III - Métriques par défaut

Micrometer



Les métriques recensées par défaut avec Spring Boot

Kubernetes



III - Métriques par défaut

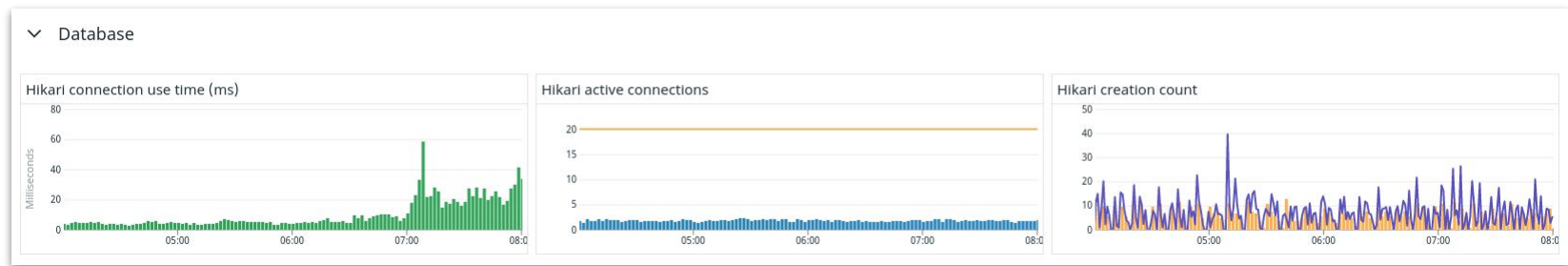
Micrometer



Les métriques recensées par défaut avec Spring Boot

La base de données

Hikari



III - Métriques par défaut

Les métriques recensées par défaut avec Spring Boot

La base de données

Hibernate

Permet de logger les requêtes d'extraction de statistiques:

```
1  # Metrics for Hibernate
2  spring.jpa.properties.hibernate.generate_statistics=true
3  # To see if stats are generated for hibernate
4  logging.level.org.hibernate.stat=debug
```

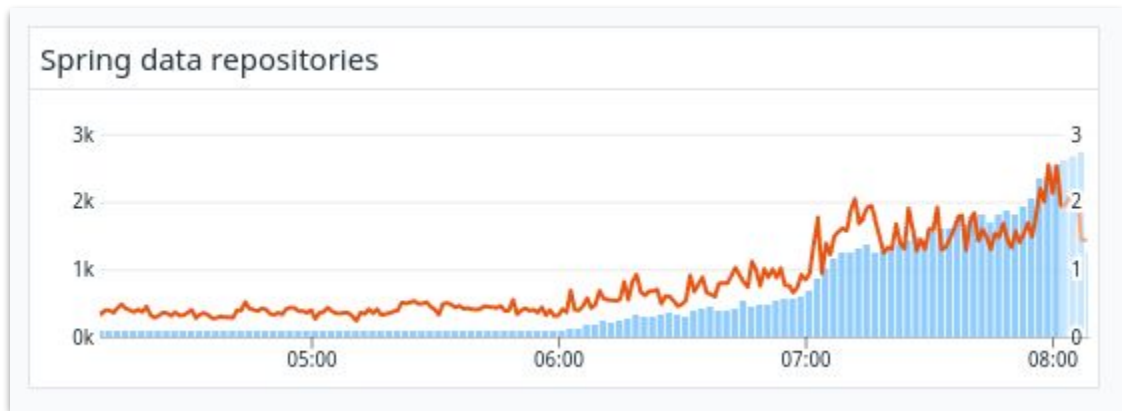
III - Métriques par défaut

Micrometer



Les métriques recensées par défaut avec Spring Boot

Spring data repository



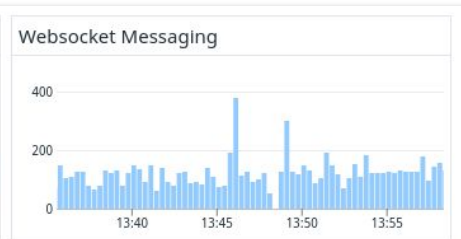
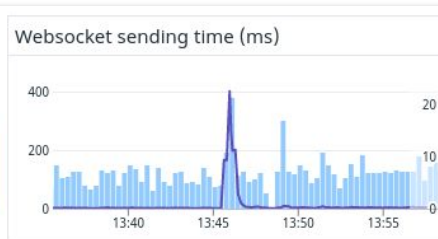
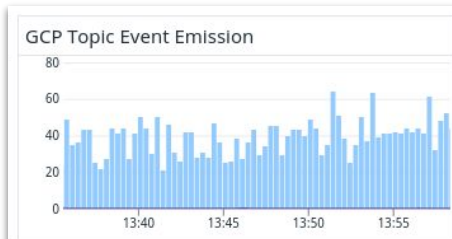
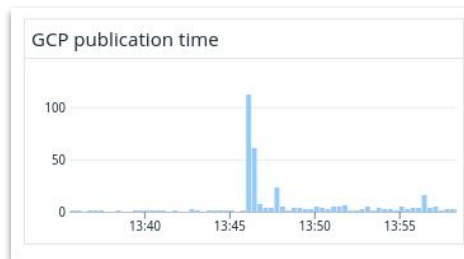
III - Métriques par défaut

Micrometer



Les métriques recensées par défaut avec Spring Boot

GCP



III - Métriques par défaut

Micrometer



Les métriques recensées par défaut avec Spring Boot

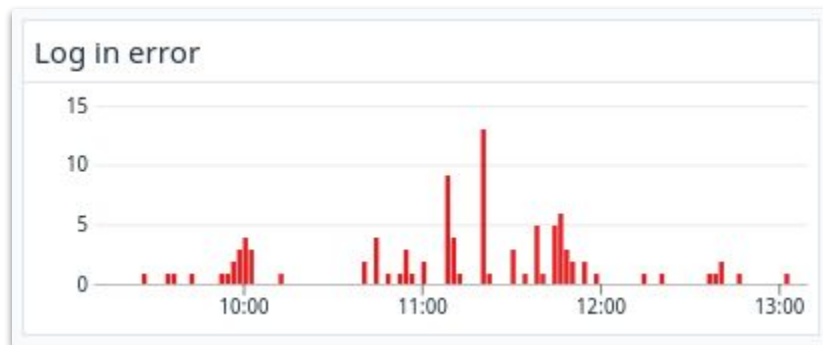
Tomcat



III - Métriques par défaut

Les métriques recensées par défaut avec Spring Boot

Logback



III - Métriques par défaut

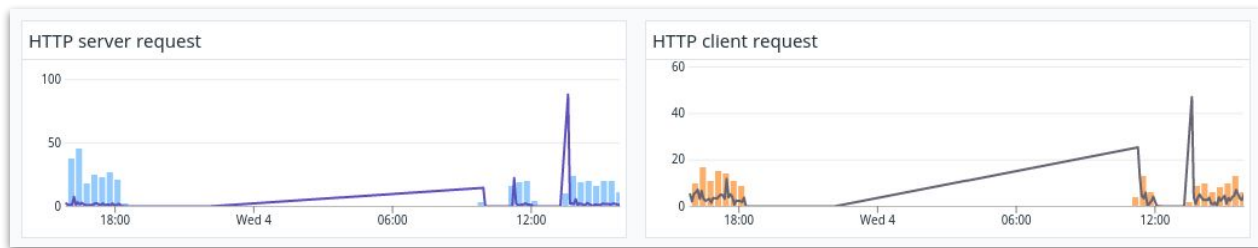
Micrometer



Les métriques recensées par défaut avec Spring Boot

MVC & WebFlux

RestTemplate latencies





Construction des URLs pour RestTemplate

Recommandé :

```
RestTemplate.exchange("https://path.de.mon.appli/service/{param}", ... , Map<String, String> uriVariables)
```

Non recommandé :

```
RestTemplate.exchange("https://path.de.mon.appli/service/" + param.toString(), ...)
```

III - Métriques par défaut

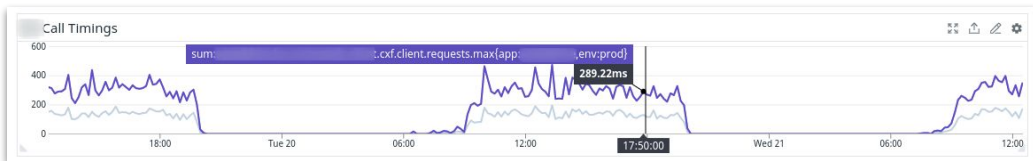
Métriques additionnelles CXF

CXF client side:

```
/**
 * Adds Micrometer metrics to proxyFactory
 * See <a href="https://cxf.apache.org/docs/micrometer.html#Micrometer-IntegrationwithJAX-WS">Integration with JAX-WS</a>
 * @param registry Spring composite registry
 * @param proxyFactory JaxWsProxyFactoryBean
 * @param clientRequestsMetricName
 */
private void customizeFactoryWithMetrics(MeterRegistry registry,
    JaxWsProxyFactoryBean proxyFactory,
    String clientRequestsMetricName) {
    final JaxwsTags jaxwsTags = new JaxwsTags();
    final TagsCustomizer operationsCustomizer = new JaxwsOperationTagsCustomizer(jaxwsTags);
    final TagsCustomizer faultsCustomizer = new JaxwsFaultCodeTagsCustomizer(jaxwsTags, new JaxwsFaultCodeProvider());
    final TagsProvider tagsProvider = new StandardTagsProvider(new DefaultExceptionClassProvider(), new StandardTags());
    final MicrometerMetricsProperties properties = new MicrometerMetricsProperties();
    properties.setClientRequestsMetricName(clientRequestsMetricName);

    final MetricsProvider metricsProvider = new MicrometerMetricsProvider(registry,
        tagsProvider,
        Arrays.asList(operationsCustomizer, faultsCustomizer),
        new DefaultTimedAnnotationProvider(),
        properties);
    proxyFactory.setFeatures(Arrays.asList(new MetricsFeature(metricsProvider)));
}
```

```
<dependency>
  <groupId>org.apache.cxf</groupId>
  <artifactId>cxf-rt-features-metrics</artifactId>
  <version>${cxf.version}</version>
</dependency>
```



- 1 # Metrics for CXF
- 2 # <https://cxf.apache.org/docs/springboot.html>
- 3 cxf.metrics.enabled=true
- 4 # cxf.metrics.jaxws.enabled=true

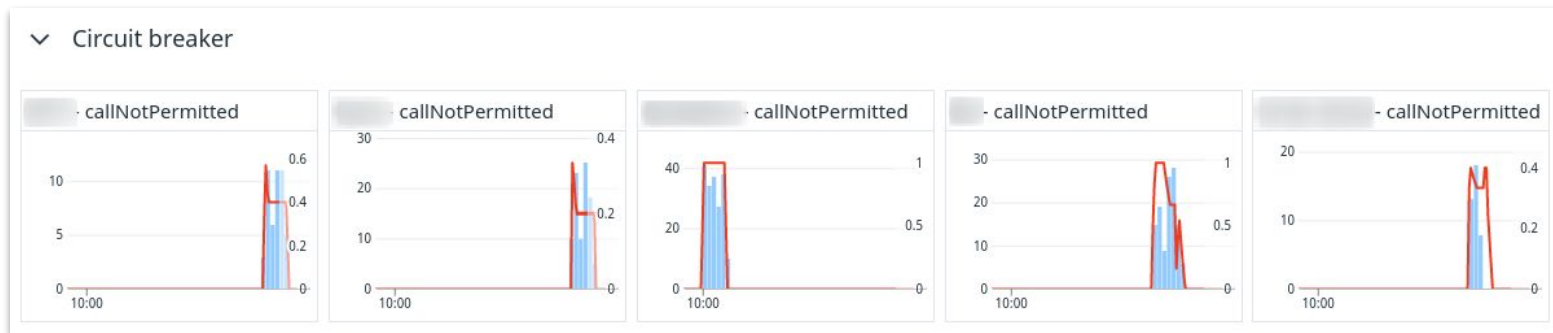
III - Métriques par défaut

Patterns Resilience4j

Exemple des circuits breakers

Attributs “**name : circuitBreakerName**”

“**state : [open, close, half_open, disabled]**”



IV - Métriques personnalisées

Les différentes classes de métriques Micrometer

- **Counter**

Compteur à incrémenter

- **Timer**

Recenser des temps

- **Gauge**

Envoie de statistiques uniquement lorsqu'un événement fixé est observé

Utile pour les statistiques sur les caches ou les collections

- **Tags**

Permet de dissocier une métrique en fonction de critères définis

Exemples d'utilisation

```
@Service
public class CustomMetrics {

    /** Counter */

    // First solution
    // -> indexes the index in Datadog on the launch of the app
    public final Counter firstCounter = Metrics.counter(name: "app.first.metric_name");

    // Second solution
    public void incrementSecondCounter(String tagParam1, String tagParam2) {
        Metrics.counter(name: "app.second.metric_name",
            Arrays.asList(
                Tag.of(key: "tag1", tagParam1),
                Tag.of(key: "tag2", tagParam2))).increment();
    }

    /** Timer */

    // First solution
    // -> indexes the index in Datadog on the launch of the app
    public final Timer firstTimer = Metrics.timer(name: "app.first.timer_name");

    // Second solution
    public void recordSecondTimer(long amountOfTime, String tagParam1) {
        Metrics.timer(name: "app.second.timer_name",
            Arrays.asList(Tag.of(key: "tag1", tagParam1)))
            .record(amountOfTime, TimeUnit.MILLISECONDS);
    }
}
```

IV - Métriques personnalisées

Counter by instantiation

```
@Service
public class CustomMetrics {

    /** Counter */

    // First solution
    // -> indexes the index in Datadog on the launch of the app
    public final Counter firstCounter = Metrics.counter(name: "app.first.metric_name");
}
```

```
@Autowired
private CustomMetrics customMetrics;

public void method1() {
    // blabla code
    customMetrics.firstCounter.increment(); /** increment(value) */
    // blabla code
}
```

IV - Métriques personnalisées

Counter by method

```
@Service
public class CustomMetrics {

    /** Counter */

    // Second solution
    public void incrementSecondCounter(String tagParam1, String tagParam2) {
        Metrics.counter(name: "app.second.metric_name",
            Arrays.asList(
                Tag.of(key: "tag1", tagParam1),
                Tag.of(key: "tag2", tagParam2))).increment();
    }
}
```

```
@Autowired
private CustomMetrics customMetrics;

public void method2() {
    // blabla code
    customMetrics.incrementSecondCounter(tagParam1: "tagParam1", tagParam2: "tagParam2");
    // blabla code
}
```

IV - Métriques personnalisées

Timer by instantiation

```
@Service
public class CustomMetrics {

    /** Timer */

    // First solution
    // -> indexes the index in Datadog on the launch of the app
    public final Timer firstTimer = Metrics.timer(name: "app.first.timer_name");
}
```

```
@Autowired
private CustomMetrics customMetrics;

public void method3() {
    // blabla code
    long amountOfTime = 100;
    customMetrics.firstTimer.record(amountOfTime, TimeUnit.MILLISECONDS);
    // blabla code
}
```


IV - Métriques personnalisées

Timer by method

```
@Service
public class CustomMetrics {

    /** Timer **/

    // Second solution
    public void recordSecondTimer(long amountOfTime, String tagParam1) {
        Metrics.timer(name: "app.second.timer_name",
            Arrays.asList(Tag.of(key: "tag1", tagParam1)))
            .record(amountOfTime, TimeUnit.MILLISECONDS);
    }
}
```

```
@Autowired
private CustomMetrics customMetrics;

public void method4() {
    // blabla code
    long amountOfTime = 100;
    customMetrics.recordSecondTimer(amountOfTime, tagParam1: "tagParam1");
    // blabla code
}
```

V - Visualisation des métriques

Les plateformes de monitoring

Datadog

- Enregistrement sur Datadog par le biais d'une demande de **clé d'API**
- **PUSH** : envoie les métriques de l'application vers la plateforme de monitoring à intervalles fixes
- Métriques prises en charges :
 - Log Management
 - Application Performance Monitoring
 - Security Monitoring
 - Network Monitoring
 - Real User Monitoring

\$

Prometheus

- Définit un **endpoint** de récupération des métriques **"/prometheus"**
- Métriques prises en charges :
 - Multi-dimensional data model

~~\$~~

V - Visualisation des métriques

Configuration de l'export vers Datadog

Faire une demande de clé d'API pour Datadog

```
1  # Metrics Datadog
2  management.metrics.export.datadog.api-key=DEFINED_IN_VAULT
3  management.metrics.export.datadog.enabled=true
4  management.metrics.export.datadog.uri=https://api.datadoghq.eu
5  management.metrics.export.datadog.host-tag=host
6  management.metrics.export.datadog.step=PT10S
```

Définition des tags par défaut:

```
8  management.metrics.tags.app=appname
9  management.metrics.tags.env=DEFINED_BY_ENV
```

V - Visualisation des métriques

Micrometer



Créer un dashboard sur Datadog

- Template variables
- Global Time

Attention à l'ordre d'affichage des courbes sur les graphiques
(première déclaration en arrière plan)

V - Visualisation des métriques

Micrometer



Créer un dashboard sur Datadog

The screenshot shows the Datadog dashboard creation interface. On the left is a dark sidebar with the Datadog logo and navigation links: Go to..., Watchdog, Events, Dashboards, Infrastructure, Monitors, Metrics, Integrations, APM, CI, Notebooks, Logs, Security, and UX Monitoring. At the bottom of the sidebar are links for Contact Support and Help. The main area is titled 'Micrometer Dashboard' with a 'Close Ctrl+E' button. Below the title are search and template variable options. A large dashed purple box in the center contains the text 'Click a tray item to add it here'. On the right, the 'Add Widgets' panel is open, showing a search bar and a grid of widget options. The 'Widgets' tab is selected, displaying various chart types: Timeseries, Query Value, Top List, Table, Heatmap, Distribution, Change, Event Timeline, Scatter Plot, Treemap, Pie Chart, and Funnel. Below these are 'Empty Group' and 'Powerpack' options.

V - Visualisation des métriques

Micrometer



Créer un dashboard sur Datadog

Edit template variables

Tag or Attribute	Name	Default Value	Available Values	Expression Control
app	app	app_name	(all)	0/0 + Add All - Remove All
env	env	prod	(all)	0/0 + Add All - Remove All

+ Add Template Variable

Cancel

Save

V - Visualisation des métriques

Micrometer



Créer un dashboard sur Datadog

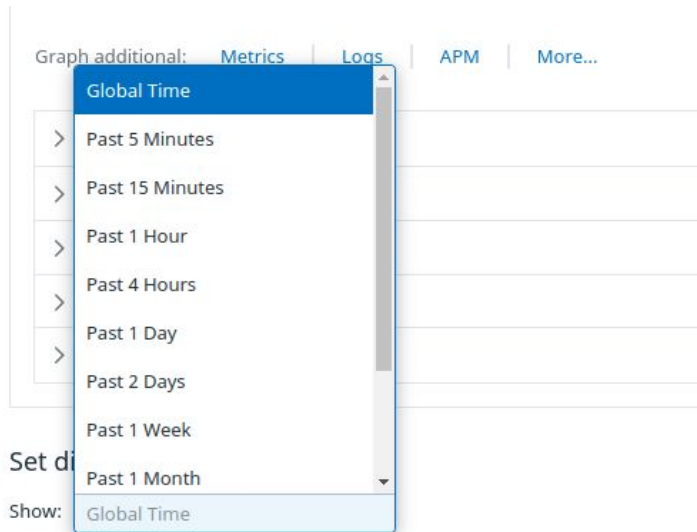


V - Visualisation des métriques

Micrometer



Créer un dashboard sur Datadog



3

Set d

Show:

4

Give your graph a title (or leave blank for suggested title)

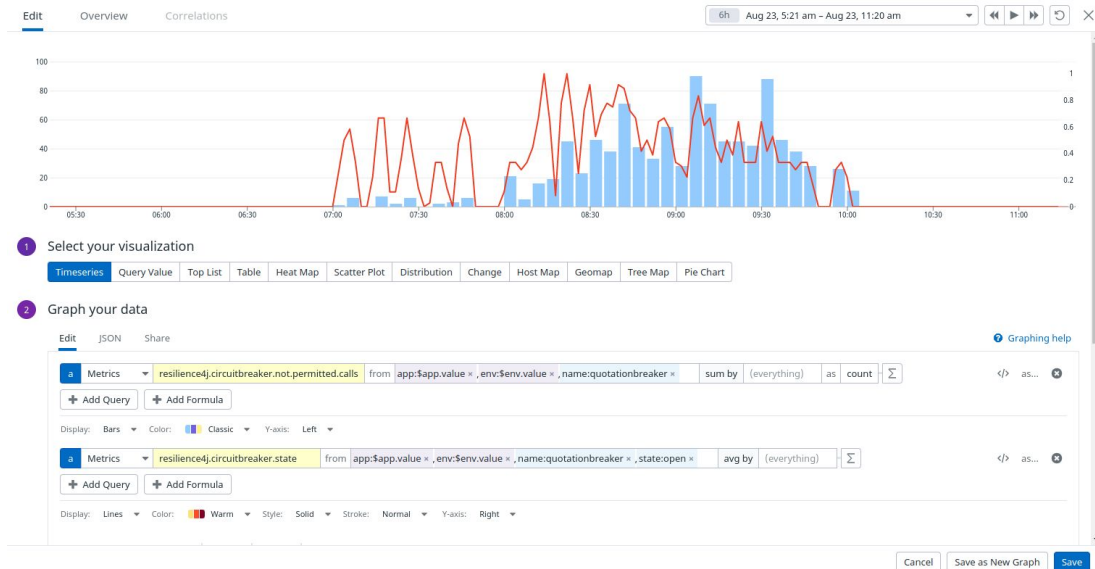
avg:system.cpu.user{app:\$app.value,env:\$env.value}

V - Visualisation des métriques

Micrometer



Créer un dashboard sur Datadog



V - Visualisation des métriques

Micrometer



Créer un dashboard sur Datadog

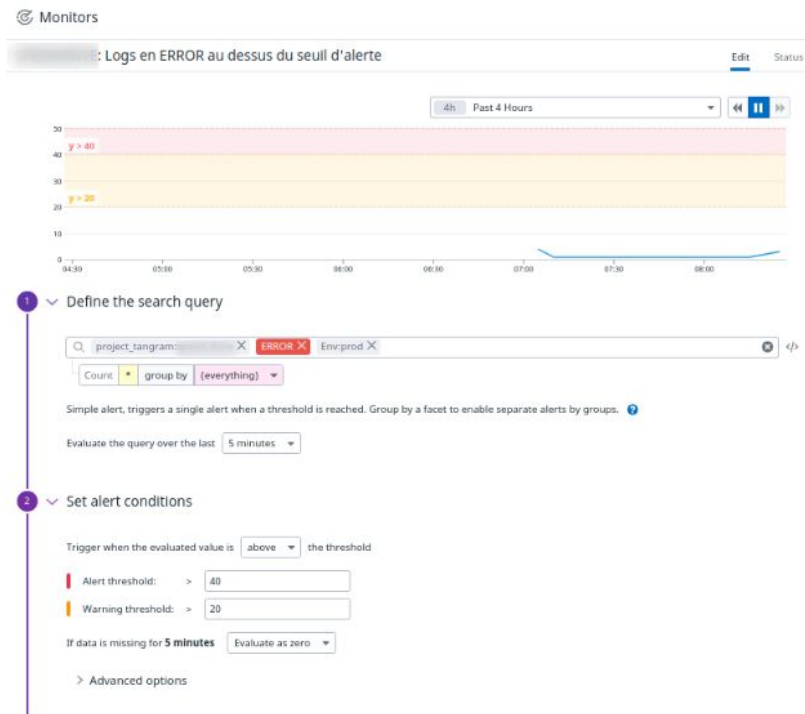


V - Visualisation des métriques

Micrometer



Configurer des alertes



V - Visualisation des métriques

Micrometer



Configurer des alertes

1. Notify your team

Edit Preview Use Message Template Variables

Logs en ERROR au dessus du seuil d'alerte

H B I S P A T L M U W Y Z

@ -G .fr @ -G fr @ -G

Include a sample of 10 logs in the alert notification

Renotification

☒ If this monitor stays in Alert X No Data X renotify every 30 minutes

☐ Stop renotifying after 0 occurrences

Renotification message

Edit Preview H B I S P A T L M U W Y Z

Hello,

Logs in error have exceeded threshold on production:

Go to:

- https://app.datadoghq.eu/logs?query=project_tangram%3A%20env%3Aprod%20status%3Aerror&index=&from_ts=1652109955240&to_ts=1652113555240&live=true

Tags: project_tangram: X

Priority: P3 (Medium)

V - Visualisation des métriques

Micrometer



Configurer des alertes

4 Define permissions and audit notifications

Restrict editing of this monitor to and

Notify If this monitor is modified, notify monitor creator and alert recipients.

Delete

Test Notifications

Export Monitor

Cancel

Save

VI - Aller plus loin

References

- <https://docs.spring.io/spring-boot/docs/current/reference/html/actuator.html#actuator.metrics>
- <https://docs.spring.io/spring-metrics/docs/current/public/prometheus>
- <https://docs.spring.io/spring-metrics/docs/current/public/prometheus#meter-registries>
- <https://spring.io/blog/2018/03/16/micrometer-spring-boot-2-s-new-application-metrics-collector>
- <https://www.baeldung.com/micrometer>
- <https://www.stackstalk.com/2022/03/monitor-spring-boot-app.html>
- <https://cxf.apache.org/docs/springboot.html>
- <https://cxf.apache.org/docs/springboot.html#SpringBoot-SpringBootCXFJAX-WSStarter>
- <https://cxf.apache.org/docs/micrometer.html#Micrometer-IntegrationwithJAX-WS>
- <https://marktjbrown.com/hibernate-statistics> (hors sujet) (doc qui explique l'affichage des requêtes d'extraction des statistiques sur la Database)

Merci