

Monitoring et Observabilité des Microservices Spring Boot avec Prometheus, Grafana et Zipkin

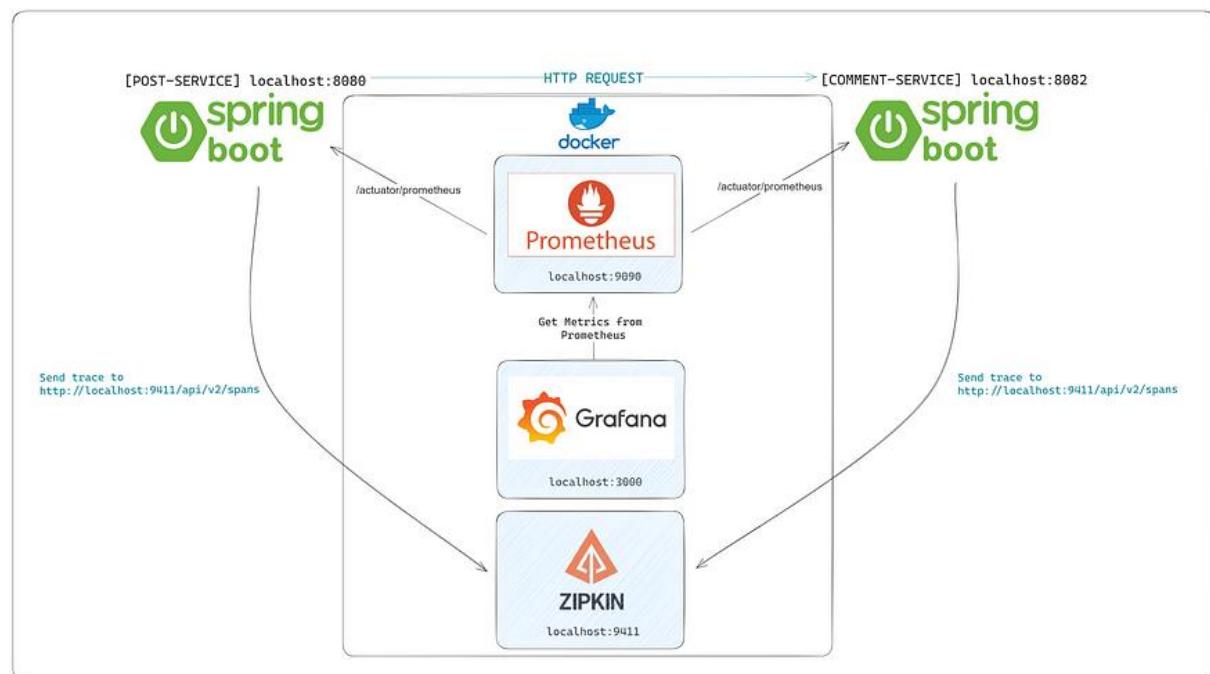
1. Introduction

Les architectures microservices nécessitent des solutions de monitoring et d'observabilité pour garantir leur bon fonctionnement et optimiser leurs performances. Dans un environnement Spring Boot, des outils comme Prometheus, Grafana et Zipkin sont utilisés pour surveiller, visualiser et tracer les services. Prometheus collecte les métriques, Grafana les affiche sous forme de tableaux de bord, et Zipkin assure le suivi des requêtes à travers les services, facilitant ainsi la détection des anomalies et l'analyse de la latence dans les systèmes distribués.

Technologies Utilisées et Justification

- **Spring Boot (3.1)** : Permet de développer des applications microservices rapidement, avec une prise en charge intégrée pour les API REST.
- **Prometheus** : Idéal pour les environnements cloud-native grâce à sa capacité à collecter les données en temps réel.
- **Grafana** : Fournit une interface utilisateur riche pour surveiller les métriques issues de Prometheus.
- **Zipkin** : Conçu pour le traçage distribué, il permet de diagnostiquer les latences et d'optimiser la communication entre les microservices.
- **Docker** : Utilisé pour conteneuriser les applications et faciliter le déploiement.

Architecture



Cette architecture d'une application basée sur des microservices en **Spring Boot** qui utilise des outils de monitoring et Observabilité de traçabilité comme **Prometheus**, **Grafana**, et **Zipkin**. Voici l'explication de chaque composant et de leur rôle dans cette architecture :

Composants de l'architecture

➤ Spring Boot Services

- Il y a deux services Spring Boot distincts : **POST-SERVICE** (sur localhost:8080) et **COMMENT-SERVICE** (sur localhost:8082).
- Ces services communiquent entre eux via des requêtes HTTP.

Exemple de requête HTTP

- notre application s'exécute sur le port **8082** (pour **COMMENT-SERVICE**) et que vous souhaitez obtenir **les commentaires** du **post** ayant l'**ID 1**, l'URL de la requête sera :

<http://localhost:8082/api/v1/comments?postId=1>

La requête devrait retourner une réponse au format JSON avec la liste des commentaires, de **postId=1** comme ceci :



A screenshot of a browser window. The address bar shows the URL: "localhost:8082/api/v1/comments?postId=1". Below the address bar, there is a button labeled "Impression élégante" with a checked checkbox. The main content area displays a JSON array of three objects, each representing a comment:

```
[{"id": 1, "content": "nice post 1", "postId": 1}, {"id": 2, "content": "nice post 2", "postId": 1}, {"id": 3, "content": "nice post 3", "postId": 1}]
```

- notre application s'exécute sur le port **8080** (pour **POST-SERVICE**) et que vous souhaitez obtenir du post ayant l'**ID 1**, l'URL de la requête sera :

<http://localhost:8080/api/v1/posts/1>

La requête devrait retourner une réponse au format JSON avec la liste des posts avec ses commentaires, de **postId=1** comme ceci :

```
1 {
2     "id": 1,
3     "title": "What is the Prometheus?",
4     "content": "Nice tool",
5     "comments": [
6         {
7             "id": 1,
8             "content": "nice post 1",
9             "postId": 1
10        },
11        {
12            "id": 2,
13            "content": "nice post 2",
14            "postId": 1
15        },
16        {
17            "id": 3,
18            "content": "nice post 3",
19            "postId": 1
20        }
21    ]
22 }
```

➤ Prometheus

- **Prometheus** est utilisé pour collecter des métriques des services Spring Boot.
- Les services Spring Boot exposent des points de terminaison /actuator/prometheus, où Prometheus peut récupérer des données de performance et d'utilisation.
- **Prometheus** écoute sur localhost:9090 et collecte ces métriques à intervalles réguliers.

➤ Grafana

- **Grafana** est utilisé pour la visualisation des métriques collectées par Prometheus.
- Il se connecte à Prometheus pour obtenir les données et les afficher sous forme de tableaux de bord interactifs et personnalisés.
- **Grafana** est accessible sur localhost:3000.

➤ Zipkin

- **Zipkin** est un outil de traçabilité distribué qui permet de suivre le parcours des requêtes entre les microservices et de visualiser la latence et les relations de dépendance entre eux.
- Les services Spring Boot envoient des traces de leurs requêtes à Zipkin via le point de terminaison http://localhost:9411/api/v2/spans.
- **Zipkin** est accessible sur localhost:9411.

Les Etapes de configuration

Déploiement Prometheus, Grafana ,Zipkin sur Docker

le fichier [docker-compose.yaml](#) contient

```
=====
version: '3'
services:
  zipkin:
    container_name: zipkin-service
    image: openzipkin/zipkin:latest
    restart: always
    ports:
      - "9411:9411"

  prometheus:
    container_name: prometheus-service
    image: prom/prometheus
    restart: always
    extra_hosts:
      - host.docker.internal:host-gateway
    command:
      - --config.file=/etc/prometheus/prometheus.yml
    volumes:
      - ./docker/prometheus.yml:/etc/prometheus/prometheus.yml
    ports:
      - "9090:9090"

  grafana:
    container_name: grafana-service
    image: grafana/grafana
    ports:
      - "3000:3000"
=====
```

le fichier [prometheus.yaml](#) contient

```
=====
global:
  scrape_interval: 10s
  evaluation_interval: 10s

scrape_configs:
  - job_name: "spring-post-service"
    metrics_path: /actuator/prometheus
    static_configs:
      - targets: ['host.docker.internal:8080']
```

```
- job_name: "spring-comment-service"
  metrics_path: /actuator/prometheus
  static_configs:
    - targets: [ 'host.docker.internal:8082' ]
```

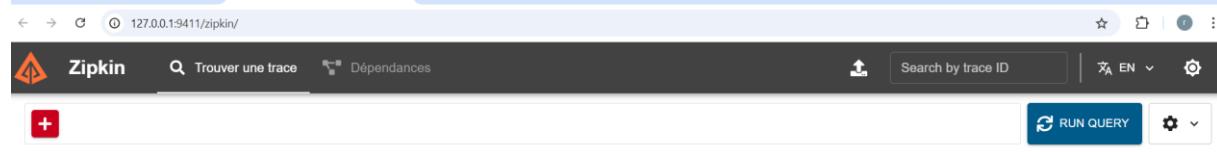
Exécuter la commande suivante : **docker-compose up -d**

```
PS C:\Users\Admin\Desktop\spring-prometheus-grafana> docker-compose up -d
time="2024-11-05T09:18:43+01:00" level=warning msg="C:\\\\Users\\\\Admin\\\\Desktop\\\\spring-prometheus-grafana\\\\docker-compose.yaml: the attribute `version` is obsolete, it will be ignored, please remove it to avoid potential confusion"
[+] Running 3/3
  ✓ Container grafana-service     Started                         0.8s
  ✓ Container prometheus-service  Running                         0.0s
  ✓ Container zipkin-service      Running                         0.0s
PS C:\Users\Admin\Desktop\spring-prometheus-grafana>
```

Tous les conteneurs fonctionnent.

Zipkin UI (pour la vérification)

<http://127.0.0.1:9411>



Search Traces

Please select criteria in the search bar. Then, click the search button.

Grafana UI (pour la vérification)

<http://127.0.0.1:3000>

username: admin

password: admin

Click **Log in** button.

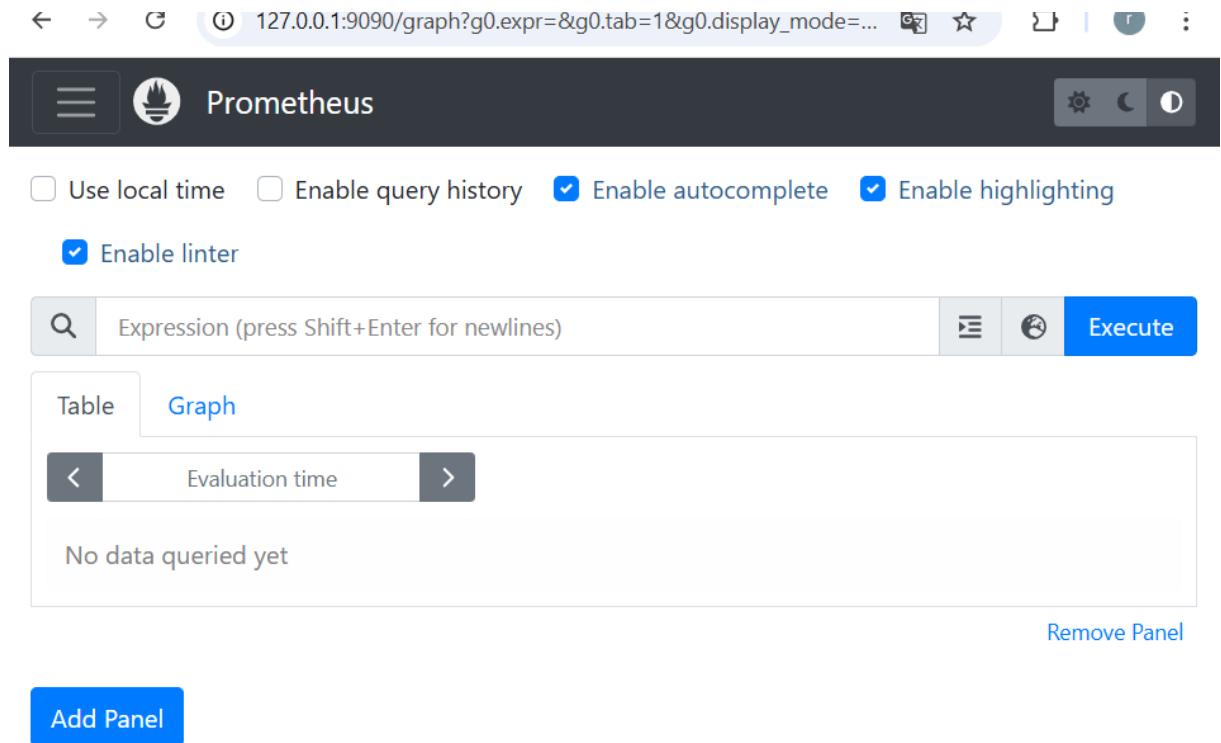


Bienvenue sur le tableau de bord de Grafana.

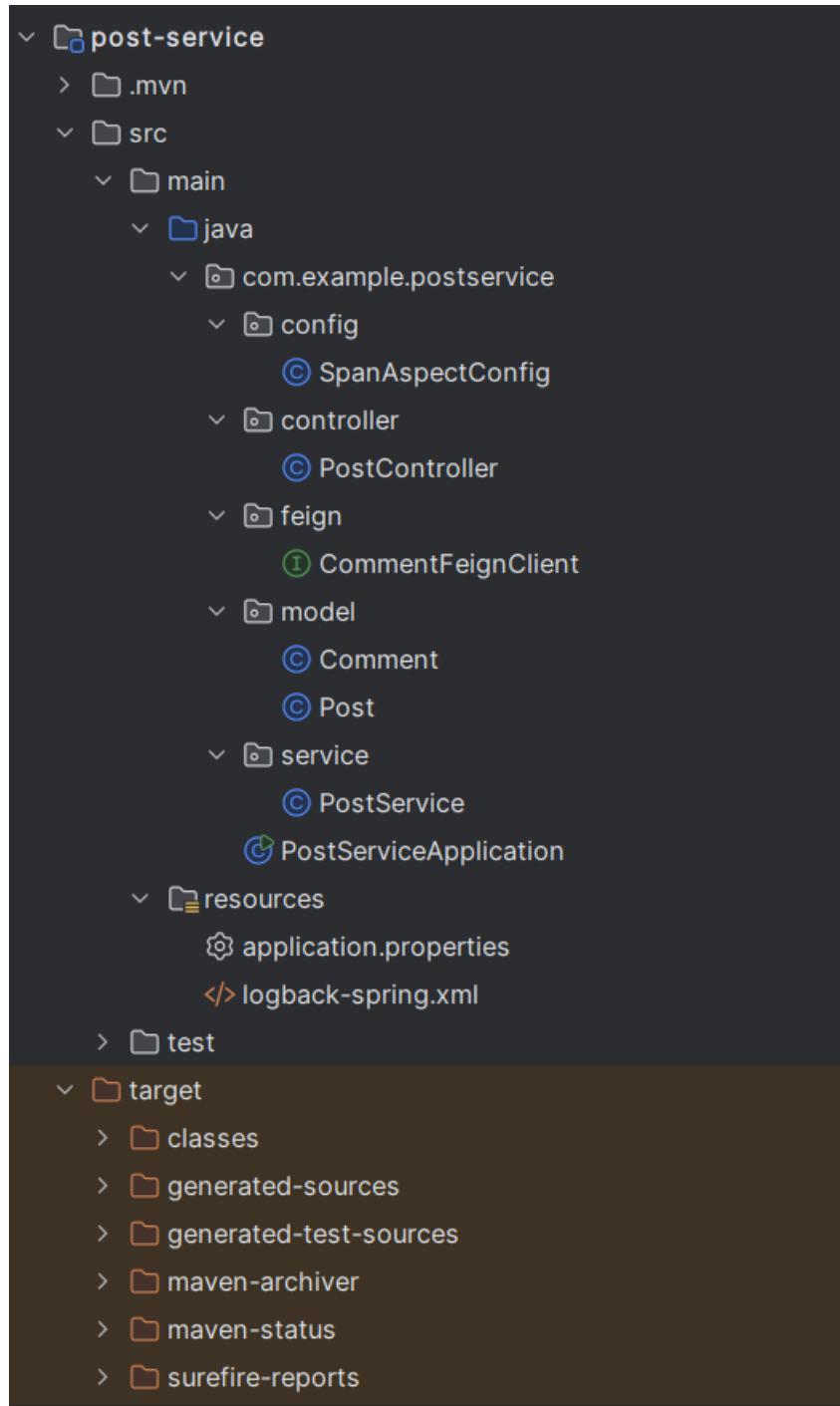
A screenshot of a web browser displaying the Grafana dashboard. The URL in the address bar is "127.0.0.1:3000/?orgId=1&from=now-6h&to=now&timezone=browser". The main header says "Welcome to Grafana". On the left, there's a "Basic" panel with instructions for setting up Grafana. In the center, there are three main panels: "TUTORIAL DATA SOURCE AND DASHBOARDS" (with sub-links "Grafana fundamentals" and "Set up and understand Grafana"), "DATA SOURCES" (with sub-link "Add your first data source" and "Learn how in the docs"), and "DASHBOARDS" (with sub-link "Create your first dashboard" and "Learn how in the docs"). At the bottom left, there are links for "Dashboards", "Starred dashboards", and "Recently viewed dashboards". At the bottom right, there's a "Latest from the blog" section featuring a post about Grafana Labs at KubeCon. The entire interface has a dark theme with light-colored text and buttons.

Prometheus UI (pour la vérification)

<http://127.0.0.1:9090>



Post Service



1) Dependencies (dans le fichier pom.xml)

Java Version: 17

Spring Version: 3.1.0

```
<dependencies>

<!--Web & Lombok-->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
```

```
</dependency>

<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <optional>true</optional>
</dependency>

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

<!--Prometheus, Zipkin & Micrometer-->
<dependency>
    <groupId>io.micrometer</groupId>
    <artifactId>micrometer-registry-prometheus</artifactId>
    <scope>runtime</scope>
    <version>1.11.0</version>
</dependency>

<dependency>
    <groupId>io.zipkin.reporter2</groupId>
    <artifactId>zipkin-reporter-brave</artifactId>
    <version>2.16.3</version>
</dependency>

<dependency>
    <groupId>io.micrometer</groupId>
    <artifactId>micrometer-tracing-bridge-brave</artifactId>
</dependency>

<!--Actuator & AOP-->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
    <version>3.1.0</version>
</dependency>

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

<!--Feign Client-->
<dependency>
    <groupId>org.springframework.cloud</groupId>
```

```

<artifactId>spring-cloud-starter-openfeign</artifactId>
<version>3.1.5</version>
</dependency>

<dependency>
  <groupId>io.github.openfeign</groupId>
  <artifactId>feign-micrometer</artifactId>
  <version>12.3</version>
</dependency>

</dependencies>

<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-dependencies</artifactId>
      <version>2022.0.3</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>

```

2) application.properties

spring.application.name=spring-post-service

```

management.zipkin.tracing.endpoint=http://${ZIPKIN_HOST}:localhost:9411/api/v2/spans
management.tracing.sampling.probability=1.0
management.endpoints.web.exposure.include=health,prometheus,metrics
management.metrics.tags.application=${spring.application.name}

```

logging.pattern.level=%5p [\${spring.application.name:},%X{traceId:-},%X{spanId:-}]

3) PostController

```

@RestController
@RequestMapping("/api/v1/posts")
public class PostController {

  private final PostService postService;

  @Autowired
  public PostController(PostService postService) {
    this.postService = postService;
  }

  @GetMapping("")
  public List<Post> findAllPosts() throws InterruptedException {

```

```

        return postService.findAllPost();
    }

    @GetMapping(path = "/{id}")
    public Post findPostByIdWithComments(@PathVariable int id) throws InterruptedException {
        return postService.findPostByIdWithComments(id);
    }

}

```

4) PostService

```

@Service
@Slf4j
public class PostService {

    private final CommentFeignClient commentFeignClient;

    @Autowired
    public PostService(CommentFeignClient commentFeignClient) {
        this.commentFeignClient = commentFeignClient;
    }

    @NewSpan(value = "post-service-findAllPost-method-span")
    public List<Post> findAllPost() throws InterruptedException {
        Thread.sleep(1000);
        //log.info("find all posts...");
        return List.of( new Post(1, "What is the Prometheus?", "Nice tool", null));
    }

    @NewSpan(value = "post-service-getPostWithComments-span")
    public Post findPostByIdWithComments(int id) {
        List<Comment> comments = commentFeignClient.findCommentsByPostId(id);
        return new Post(1, "What is the Prometheus?", "Nice tool", comments);
    }
}

```

5) CommentFeignClient

```

@FeignClient(value = "comment-service", url = "http://localhost:8082/api/v1/comments")
public interface CommentFeignClient {

    @GetMapping("")
    List<Comment> findCommentsByPostId(@RequestParam int postId);

}

```

6) actuator de Post service

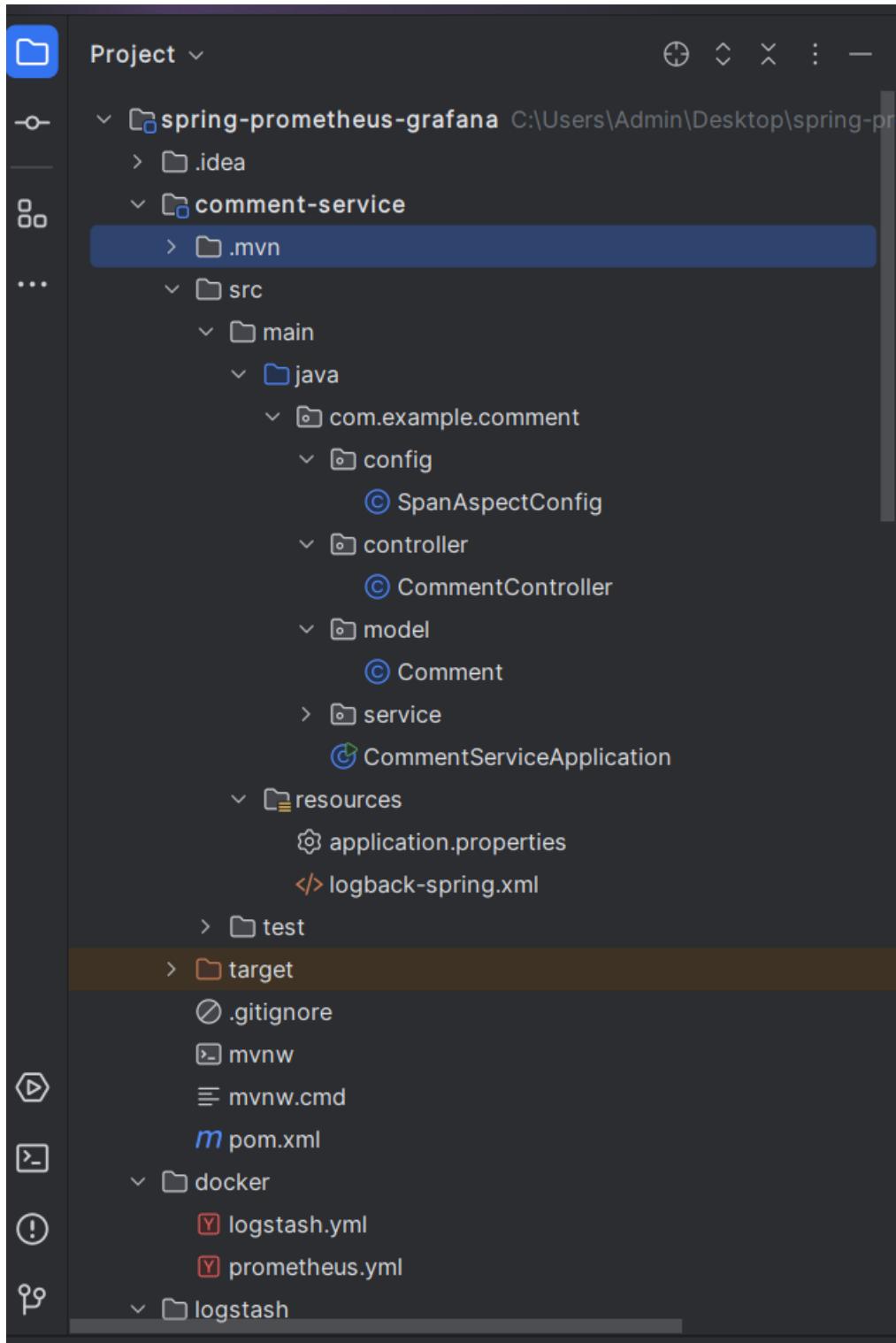
Visit: <http://127.0.0.1:8080/actuator>

Spring Boot Actuator permet de surveiller et gérer les applications Spring Boot en production grâce à des points de contrôle pour la santé et les performances.



```
{  
  "_links": {  
    "self": {  
      "href": "http://127.0.0.1:8080/actuator",  
      "templated": false  
    },  
    "health-path": {  
      "href": "http://127.0.0.1:8080/actuator/health/{*path}",  
      "templated": true  
    },  
    "health": {  
      "href": "http://127.0.0.1:8080/actuator/health",  
      "templated": false  
    },  
    "prometheus": {  
      "href": "http://127.0.0.1:8080/actuator/prometheus",  
      "templated": false  
    },  
    "metrics-requiredMetricName": {  
      "href": "http://127.0.0.1:8080/actuator/metrics/{requiredMetricName}",  
      "templated": true  
    },  
    "metrics": {  
      "href": "http://127.0.0.1:8080/actuator/metrics",  
      "templated": false  
    }  
  }  
}
```

Comment Service



1) Dependencies (dans le fichier pom.xml)

Java Version: 17

Spring Version: 3.1.0

```
<dependencies>

    <!--Web & Lombok-->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <optional>true</optional>
    </dependency>

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

    <!--Prometheus, Zipkin & Micrometer-->
    <dependency>
        <groupId>io.micrometer</groupId>
        <artifactId>micrometer-registry-prometheus</artifactId>
        <scope>runtime</scope>
        <version>1.11.0</version>
    </dependency>

    <dependency>
        <groupId>io.zipkin.reporter2</groupId>
        <artifactId>zipkin-reporter-brave</artifactId>
        <version>2.16.3</version>
    </dependency>

    <dependency>
        <groupId>io.micrometer</groupId>
        <artifactId>micrometer-tracing-bridge-brave</artifactId>
    </dependency>

    <!--Actuator & AOP-->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-actuator</artifactId>
        <version>3.1.0</version>
    </dependency>

    <dependency>
        <groupId>org.springframework.boot</groupId>
```

```
<artifactId>spring-boot-starter-aop</artifactId>
</dependency>

</dependencies>
```

2) application.properties

```
spring.application.name=spring-comment-service
server.port=8082

management.zipkin.tracing.endpoint=http:// ${ZIPKIN_HOST}:localhost:9411/api/v2/spans
management.tracing.sampling.probability=1.0
management.endpoints.web.exposure.include=health,prometheus,metrics
management.metrics.tags.application=${spring.application.name}

logging.pattern.level=%5p [${spring.application.name:-},%X{traceId:-},%X{spanId:-}]
```

3) CommentController

```
@RestController
@RequestMapping("/api/v1/comments")
public class CommentController {

    private final CommentService commentService;

    @Autowired
    public CommentController(CommentService commentService) {
        this.commentService = commentService;
    }

    @GetMapping("")
    public List<Comment> findCommentsByPostId(@RequestParam int postId) {
        return commentService.findCommentsByPostId(postId);
    }
}
```

4) CommentService

```
@Service
@Slf4j
public class CommentService {

    private List<Comment> comments = List.of(
        new Comment(1, "nice post 1", 1),
        new Comment(2, "nice post 2", 1),
        new Comment(3, "nice post 3", 1),
        new Comment(4, "nice post 4", 2),
        new Comment(5, "nice post 5", 2),
        new Comment(6, "nice post 6", 3),
        new Comment(7, "nice post 7", 3),
        new Comment(8, "nice post 8", 3),
```

```

        new Comment(9, "nice post 9", 4)
    );

    @NewSpan(value = "comment-service-findCommentsByPostId-span")
    public List<Comment> findCommentsByPostId(int postId) {
        return comments.stream().filter(comment -> comment.getPostId() == postId).toList();
    }

}

```

5) actuator de Comment Service

Visit: <http://127.0.0.1:8082/actuator>



```

{
  "_links": {
    "self": {
      "href": "http://127.0.0.1:8082/actuator",
      "templated": false
    },
    "health": {
      "href": "http://127.0.0.1:8082/actuator/health",
      "templated": false
    },
    "health-path": {
      "href": "http://127.0.0.1:8082/actuator/health/{*path}",
      "templated": true
    },
    "prometheus": {
      "href": "http://127.0.0.1:8082/actuator/prometheus",
      "templated": false
    },
    "metrics-requiredMetricName": {
      "href": "http://127.0.0.1:8082/actuator/metrics/{requiredMetricName}",
      "templated": true
    },
    "metrics": {
      "href": "http://127.0.0.1:8082/actuator/metrics",
      "templated": false
    }
  }
}

```

Prometheus Targets

Visit <http://127.0.0.1:9090/targets>

Nous voyons nos services dans **Targets**.

The screenshot shows the Prometheus Targets page at <http://127.0.0.1:9090/targets>. There are two sections: "spring-comment-service (1/1 up)" and "spring-post-service (1/1 up)". Both sections show one target each, both of which are marked as "UP". The "Labels" column for the first target in the "spring-comment-service" section shows "instance='host.docker.internal:8082'" and "job='spring-comment-service'". The "Labels" column for the first target in the "spring-post-service" section shows "instance='host.docker.internal:8080'" and "job='spring-post-service'".

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://host.docker.internal:8082/actuator/prometheus	UP	instance="host.docker.internal:8082" job="spring-comment-service"	8.569s ago	157.434ms	

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://host.docker.internal:8080/actuator/prometheus	UP	instance="host.docker.internal:8080" job="spring-post-service"	1.477s ago	44.295ms	

Prometheus a réussi à "scraper" (ou interroger) ces services pour récupérer leurs métriques.

Post Service: UP (actif et fonctionnel)

Comment Service: UP (actif et fonctionnel)

Visit Prometheus UI: <http://127.0.0.1:9090/graph>

The screenshot shows the Prometheus Graph page at <http://127.0.0.1:9090/graph>. The interface includes a toolbar with various icons, a navigation bar with the Prometheus logo, and several configuration checkboxes: "Use local time", "Enable query history", "Enable autocomplete" (checked), "Enable highlighting" (checked), and "Enable linter" (checked). Below the toolbar is a search bar with placeholder text "Expression (press Shift+Enter for newlines)". To the right of the search bar are buttons for "Table" (selected), "Graph", "Execute", and "Remove Panel". A large central area displays the message "No data queried yet". At the bottom left is a blue button labeled "Add Panel".

Exécutez la commande suivante : `http_server_requests_seconds_count`

Cette requête affiche les requêtes HTTP entrantes.

The screenshot shows the Prometheus web interface at the URL `127.0.0.1:9090/graph?g0.expr=http_server_requests_seconds_...`. The interface has a dark-themed header with the Prometheus logo and navigation icons. Below the header, there are several configuration checkboxes: "Use local time" (unchecked), "Enable query history" (unchecked), "Enable autocomplete" (checked), "Enable highlighting" (checked), and "Enable linter" (checked). The search bar contains the query `http_server_requests_seconds_count`, which is underlined in red, indicating it's a metric name. To the right of the search bar are icons for refresh, export, and a dropdown menu, followed by a blue "Execute" button. Below the search bar, there are two tabs: "Table" (selected) and "Graph". The status bar indicates "Load time: 94ms Resolution: 14s Result series: 12". The main content area displays a table of query results with the following data:

Query Details	Count
<code>http_server_requests_seconds_count{application="spring-post-service", error="none", exception="none", instance="host.docker.internal:8080", job="spring-post-service", method="GET", outcome="SUCCESS", status="200", uri="/actuator/prometheus"}</code>	2618
<code>http_server_requests_seconds_count{application="spring-comment-service", error="none", exception="none", instance="host.docker.internal:8082", job="spring-comment-service", method="GET", outcome="SUCCESS", status="200", uri="/actuator/prometheus"}</code>	2618
<code>http_server_requests_seconds_count{application="spring-comment-service", error="none", exception="none", instance="host.docker.internal:8082", job="spring-comment-service", method="GET", outcome="SUCCESS", status="200", uri="/api/v1/comments"}</code>	22
<code>http_server_requests_seconds_count{application="spring-post-service", error="none", exception="none", instance="host.docker.internal:8080", job="spring-post-service", method="GET", outcome="SUCCESS", status="200", uri="/api/v1/posts/{id}"}</code>	16
<code>http_server_requests_seconds_count{application="spring-comment-service", error="none", exception="none", instance="host.docker.internal:8082", job="spring-comment-service", method="GET", outcome="CLIENT_ERROR", status="404", uri="/**"}</code>	2

Affichage sous forme de graphique



Visualisation des métriques dans le tableau de bord Grafana

1) Cliquez sur la case « Ajouter votre première source de données ».

The screenshot shows the Grafana home page at the URL 127.0.0.1:3000/?from=now-6h&to=now&timezone=browser. The page features a navigation bar with icons for back, forward, search, and user profile. Below the bar, there's a header with the Grafana logo and a search input field. A main content area titled "Bienvenue chez Grafana" includes links for "Besoin d'aide?", "Documentation", "Tutoriels", "Communauté", and "Slack public". On the right side, there's a sidebar with a "Basique" section containing text about basic setup steps. The main content area has a card titled "TUTORIEL SOURCE DE DONNÉES ET TABLEAUX DE BORD" with a sub-section "Principes fondamentaux de Grafana". To the right of this card is another card with a red circle highlighting the "Ajoutez votre première source de données" button. This button is part of a section titled "SOURCES DE DONNÉES". Other visible text in this section includes "Supprimer ce panneau" and "Découvrez comment dans la documentation". At the bottom left, there are links for "Tableaux de bord" and "Tableaux de bord étoilés".

2) Sélectionner Prometheus.

The screenshot shows the 'Add data source' page in Grafana. The URL in the browser is 127.0.0.1:3000/connections/datasources/new. The page title is 'Ajouter une source de données'. A sub-header says 'Choisissez un type de source de données'. Below it is a search bar 'Filtrer par nom ou par type'. On the right, there is a 'Annuler' button. The main content area is titled 'Bases de données de séries chronologiques' and lists four options: 'Prométhée', 'Graphite', 'InfluxDB', and 'OpenTSDB'. Each option has a small icon, a name, a description, and a 'Cœur' button. A red box highlights the 'Prométhée' section.

3) Saisir l'URL du serveur Prometheus.

Docker Container: <http://prometheus:9090>

The screenshot shows the 'Edit data source' page for 'prometheus' in Grafana. The URL in the browser is 127.0.0.1:3000/connections/datasources/edit/ae32omofyls74f. The page title is 'Connections > Data sources > prometheus'. It shows a table with columns 'Nom' and 'Défaut'. The 'Nom' column contains 'prometheus' and the 'Défaut' column has a checked toggle. Below the table, a note says 'Avant de pouvoir utiliser la source de données Prometheus, vous devez la configurer ci-dessous ou dans le fichier de configuration. Pour des instructions détaillées, [consultez la documentation](#)'. It also states 'Les champs marqués d'un * sont obligatoires'. The 'Connexion' section has a 'URL du serveur' field containing 'http://prometheus:9090'. The 'Authentification' section has a note 'Choisissez une méthode d'authentification pour accéder à la source de données' and a 'Pas d'authentification' button. A red box highlights the 'Connexion' section.

4) Cliquez sur le bouton « Enregistrer et tester ».

The screenshot shows the Grafana interface for managing data sources. The URL in the browser is 127.0.0.1:3000/connections/datasources/edit/ae32omofyls74f. The page title is "Connections > Data sources > prometheus".

Performance settings:

- Type Prométhée: Choisir
- Niveau de cache: Faible
- Requêtes incrémentales (bêta): Désactivé (switch off)
- Désactiver les règles d'enregistrement (bêta): Désactivé (switch off)

Autre settings:

- Paramètres de requête personnalisés: Exemple : max_source_resolution=5m&timeout=10s
- Méthode HTTP: POSTE

Exemples section:

- + Ajouter

Action buttons at the bottom:

- Supprimer
- Enregistrer et tester (highlighted with a red box)

5)Cliquez sur le bouton « Créeer un tableau de bord ».

The screenshot shows the Grafana interface for managing data sources. The URL in the browser is 127.0.0.1:3000/connections/datasources/edit/ae32omofyls74f. The page title is "Prométhée". The top navigation bar includes a search bar, a connection icon, and various status indicators. Below the title, there are tabs for "Paramètres" (selected) and "Tableaux de bord". A modal window titled "Configurez votre source de données Prometheus ci-dessous" provides instructions about using Prometheus or Grafana Cloud. At the bottom of the page, there is a "Nom" field containing "prometheus", a "Défaut" checkbox (unchecked), and a "Default" toggle switch (unchecked). A red box highlights the "Créer un tableau de bord" button in the top right corner of the main content area.

6) Cliquez sur le bouton « Importer le tableau de bord ».

The screenshot shows the Grafana interface with a dark theme. At the top, there is a navigation bar with icons for back, forward, search, and other settings. Below the navigation bar, the URL is 127.0.0.1:3000/dashboard/new?from=now-6h&to=now&tim... The main content area has a heading 'Panneau d'importation' and a sub-instruction 'Ajoutez des visualisations partagées avec d'autres tableaux de bord.' A blue button '+ Ajouter une visualisation' is visible. Below this, another section titled 'Importer un tableau de bord' contains the instruction 'Importez des tableaux de bord à partir de fichiers ou de grafana.com .' and a blue button with an upward arrow labeled 'Importer le tableau de bord'. A red hand-drawn style circle highlights the 'Importer le tableau de bord' button.

7) Remplissez l'ID en 4701 et cliquez sur le bouton « Load »

Tableau de bord micrométrique JVM (pour importe template de bord):

Visit: <https://grafana.com/grafana/dashboards/4701-jvm-micrometer/>

ID
4701

Datasource
Prometheus

Dependencies
grafana 4.6.5 Graph (old) Singlestat

Published by
mweirauch

Last update
2023-05-04T22:00:06

Grafana Cloud
Free forever plan

← → ⌂ ⓘ 127.0.0.1:3000/dashboard/import

Rechercher ou accéder Ctrl + K

Maison > Tableaux de bord > Importer le tableau de bord

Importer le tableau de bord

Importer un tableau de bord à partir d'un fichier ou de Grafana.com

↑
Télécharger le fichier JSON du tableau de bord
Faites glisser et déposez ici ou cliquez pour parcourir
Types de fichiers acceptés : .json, .txt

Recherchez et importez des tableaux de bord pour les applications courantes sur [grafana.com/tableaux de bord](#)

4701 Charger

Importer via le modèle JSON du tableau de bord

```
{ "title": "Exemple - Variables de dictionnaire répétitives", "uid": "_0HnEoN4z", "panels": [...] ... }
```

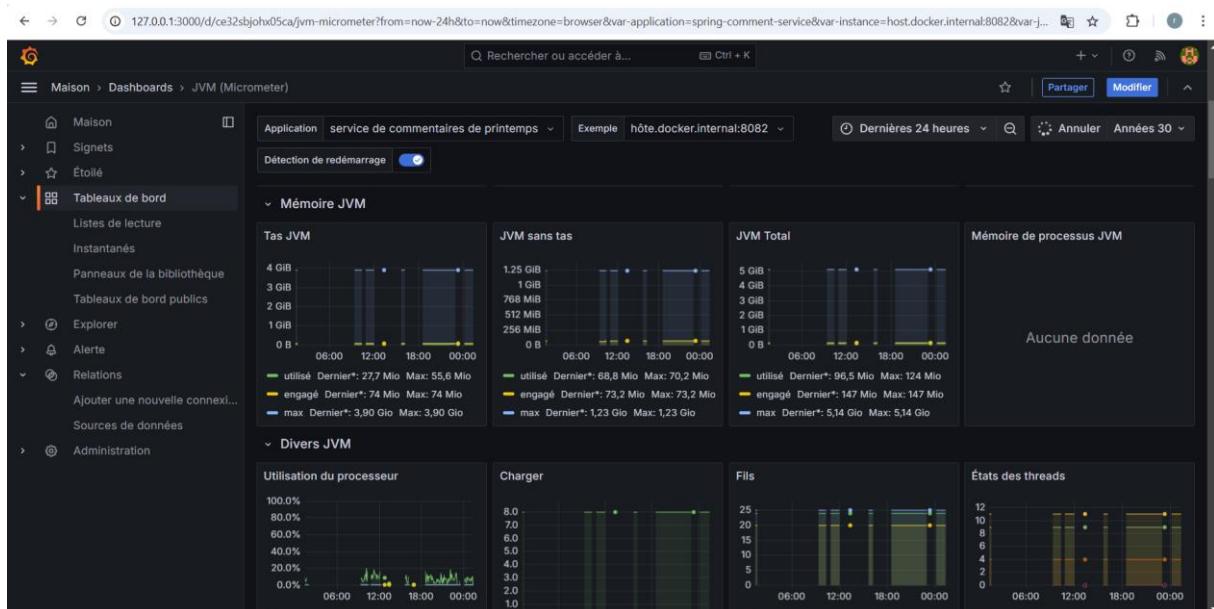
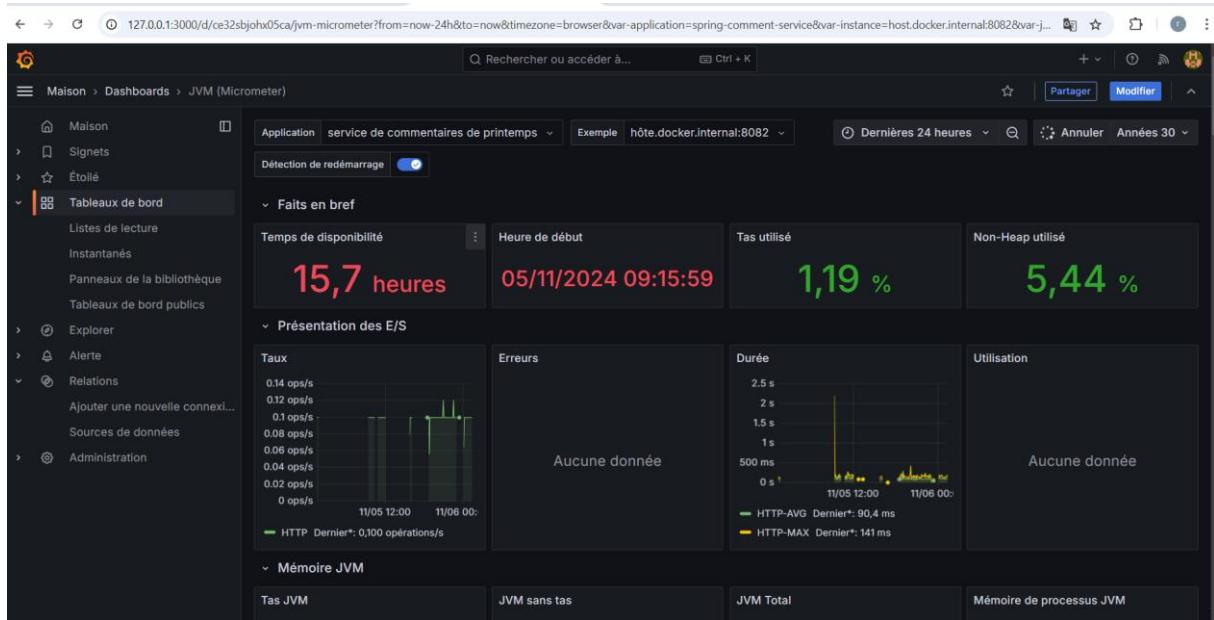


8) Sélectionner le Prometheus et cliquer sur le bouton « Import ».

The screenshot shows the 'Import from Grafana.com' dialog in Grafana. At the top, it displays the URL '127.0.0.1:3000/dashboard/import'. The main title is 'Importer un tableau de bord depuis Grafana.com'. Below it, it shows the author 'Publié par Mweirauch' and the last update 'Mis à jour le 05/05/2023 00:00:06'. The 'Options' section includes fields for 'Nom' (set to 'JVM (Micrometer)'), 'Dossier' (set to 'Tableaux de bord'), and 'Identifiant unique (UID)' (with a note explaining its purpose). A blue 'Changer l'uid' button is visible. The 'Prométhée' section shows 'prometheus' selected. At the bottom are 'Importer' and 'Annuler' buttons.

9) Nous voyons les mesures suivantes dans le tableau de bord créé.

- CPU
- JVM
- Threads
- I/O
- Log Events



10) Ajouter une nouvelle mesure.

The screenshot shows the Grafana interface for editing a dashboard titled 'JVM (Micromètre)'. On the left, there's a sidebar with navigation links like 'Maison', 'Signets', 'Étoilé', 'Tableaux de bord', 'Listes de lecture', 'Instantanés', 'Panneaux de la bibliothèque', 'Tableaux de bord publics', 'Explorer', 'Alerte', 'Relations', 'Administration', and 'Administration'. The main area has tabs for 'Temps de disponibilité' (16,2 heures), 'Heure de début' (05/11/2024 09:15:59), 'Tas utilisé' (0,79 %), and 'Non-Heap utilisé' (5,45 %). Below these are sections for 'Présentation des E/S' (Taux, Erreurs, Durée, Utilisation) and 'Mémoire JVM' (Tas JVM, JVM sans tas, JVM Total, Mémoire de processus JVM). At the top right, there are buttons for 'Ajouter' (highlighted with a red arrow), 'Paramètres', 'Quitter l'édition', and 'Enregistrer le tableau de bord'.

Metric Query : `http_server_requests_seconds_count` Click Save and Apply button.

La visualisation a été ajoutée dans le tableau de bord.

The screenshot shows the 'Edit panel' screen for the 'JVM (Micrometer)' dashboard. The left sidebar includes 'Home', 'Bookmarks', 'Starred', 'Dashboards' (selected), 'Playlists', 'Snapshots', 'Library panels', 'Public dashboards', 'Explore', 'Alerting', 'Connections', 'Data sources', and 'Administration'. The main area features a chart titled 'Panel Title' showing a linear increase in requests over time. Below the chart is a 'Queries' section with a query input field containing `http_server_requests_seconds_count`. To the right, there are sections for 'Time series', 'Value mappings', 'Thresholds', and 'Thresholds mode'. A 'Save dashboard' button is visible at the top right of the main area.

Trouver une trace des requêtes avec zipkin

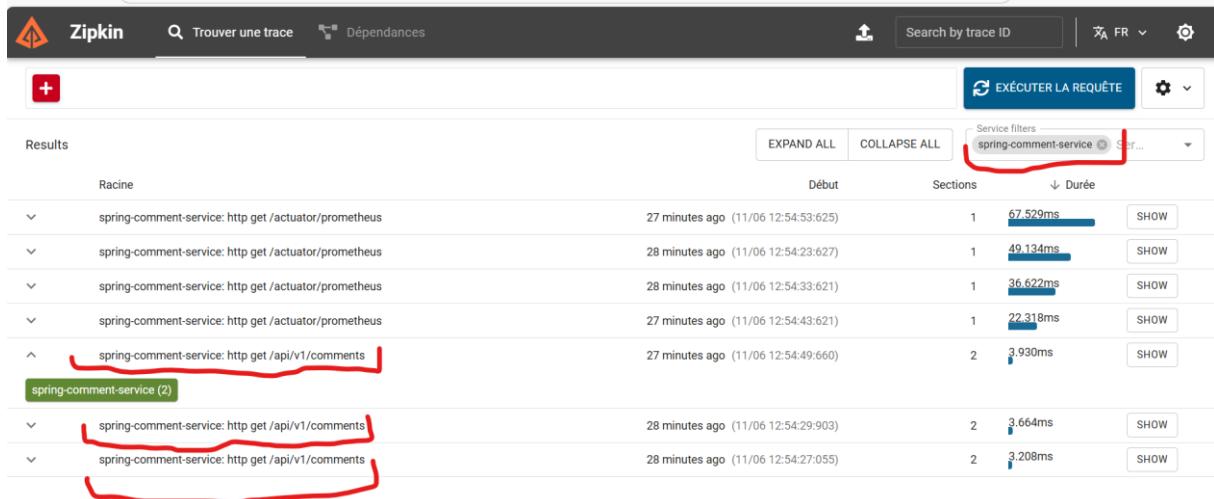
Visite <http://127.0.0.1:9411>

The screenshot shows the Zipkin web interface at localhost:9411/zipkin/. The top navigation bar includes links for "Trouver une trace" (Search a trace) and "Dépendances" (Dependencies). A search bar at the top right contains the placeholder "Search by trace ID". Below the search bar are buttons for "RUN QUERY" and settings. A large magnifying glass icon is centered on the page, with the text "Search Traces" below it. A small note says "Please select criteria in the search bar. Then, click the search button." At the bottom left, there is a link to localhost:9411/zipkin/dependency.

On envoyé cette requête <http://localhost:8080/api/v1/posts/1>

The screenshot shows a browser window at localhost:8080/api/v1/posts/1. The response is a JSON object:

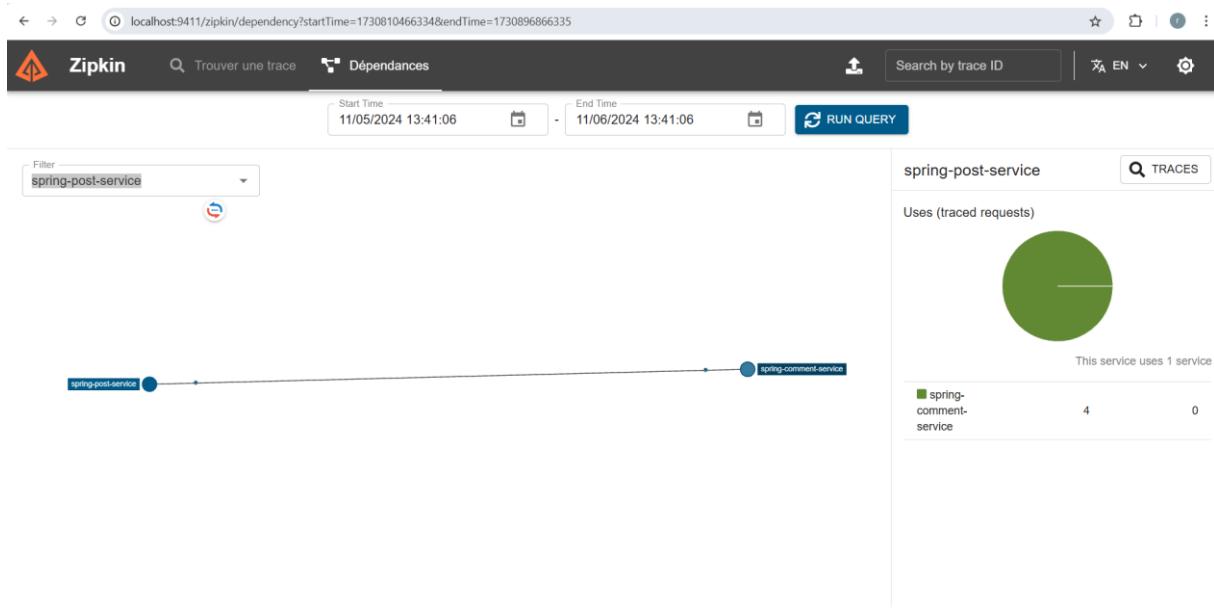
```
[{"id": 1, "title": "What is the Prometheus?", "content": "Nice tool", "comments": [ {"id": 1, "content": "nice post 1", "postId": 1}, {"id": 2, "content": "nice post 2", "postId": 1}, {"id": 3, "content": "nice post 3", "postId": 1} ]}
```



L'image montre une capture d'écran de l'interface Zipkin qui affiche les traces des requêtes HTTP traitées.

+ Détails des Requêtes Capturées :

- Service Source :** La capture d'écran met en évidence le `spring-comment-service` qui répond aux requêtes HTTP, notamment celles dirigées vers des points de terminaison tels que `/actuator/prometheus` et `/api/v1/comments`.
- Durée :** La durée de chaque trace est affichée, indiquant le temps total pris par le service pour traiter chaque requête. Par exemple, certaines traces montrent une durée de 49 ms et 36 ms, ce qui donne un aperçu de la performance et du temps de réponse des endpoints du service.
- Segments et Séctions :** Chaque trace est divisée en segments ou sections, permettant d'identifier les étapes spécifiques dans le traitement d'une requête, y compris les appels internes, les opérations de bases de données, ou d'autres sous-processus.



Cette image est une capture d'écran de l'interface Zipkin, un outil de traçage distribué. Elle montre les dépendances entre différents services, en l'occurrence ici le service "spring-post-service" qui utilise le service "spring-comment-service".

L'image affiche les informations suivantes :

- La période de temps sélectionnée, du 11/05/2024 à 13h41 au 11/06/2024 à 13h41.
- Le filtre appliqué, qui se concentre sur le service "spring-post-service".
- Un graphique montrant l'utilisation du service "spring-comment-service" par le service "spring-post-service" au cours de cette période.
- Des statistiques indiquant que ce service utilise 1 autre service.

Cette visualisation permet d'analyser les interactions entre les différents composants d'une application distribuée, afin d'identifier d'éventuels problèmes de performance ou de fiabilité.

Conclusion

La mise en place de Prometheus, Grafana et Zipkin dans une architecture de microservices Spring Boot permet d'obtenir une visibilité accrue sur les performances et la santé des services. Prometheus assure la collecte de métriques, Grafana offre des visualisations en temps réel pour faciliter l'analyse, et Zipkin trace les requêtes afin d'identifier les goulots d'étranglement et d'autres anomalies. Grâce à cette solution, les équipes de développement et d'exploitation peuvent réagir rapidement aux problèmes, optimiser les performances et garantir une meilleure fiabilité du système global. L'utilisation de ces outils de monitoring et d'observabilité devient ainsi essentielle pour maintenir une architecture de microservices évolutive et résiliente.

Centralisation des logs Spring Boot dans ELK (Elasticsearch, Logstash, Kibana)

1. Introduction

La centralisation des logs permet de gérer et de visualiser efficacement les journaux d'applications à partir d'une seule plateforme. Le trio Elasticsearch, Logstash et Kibana (ELK) est un choix populaire pour ce processus.

- **Elasticsearch** stocke les logs et offre des capacités de recherche performantes.
- **Logstash** collecte, transforme et envoie les logs à Elasticsearch.
- **Kibana** permet de visualiser les logs et d'analyser les données via des dashboards.

Ce document détaille le processus de centralisation des logs d'une application Spring Boot dans la stack ELK.

2. Pré-requis et configuration de l'environnement

Pour ce projet, vous avez besoin de :

- Docker Compose pour gérer facilement les conteneurs.
- Une application Spring Boot générant des logs structurés.
- Les services Elasticsearch, Logstash et Kibana configurés avec Docker Compose.

3. Configuration de Docker Compose pour ELK

Voici la configuration Docker Compose pour les services nécessaires de ELK :

docker-compose.yml

```
////////////////////////////////////////////////////////////////////////
```

Elasticsearch for log storage

elasticsearch:

image: docker.elastic.co/elasticsearch/elasticsearch:7.10.1 # Image Elasticsearch version 7.1

environment:

- discovery.type=single-node # Mode de déploiement en tant que nœud unique.

- ES_JAVA_OPTS=-Xms512m -Xmx512m # Limitation de la mémoire JVM à 512 Mo.

ports:

- "9200:9200" # Expose le port 9200 pour interagir avec Elasticsearch.

volumes:

- es_data:/usr/share/elasticsearch/data # Volume pour stocker les données persistantes.

networks:

- elk # Connecte Elasticsearch au réseau "elk".

Kibana for log visualization

kibana:

image: docker.elastic.co/kibana/kibana:7.10.1 # Image Kibana version 7.10.1

ports:

- "5601:5601" # Expose le port 5601 pour accéder à l'interface web Kibana.

environment:

- ELASTICSEARCH_HOSTS=http://elasticsearch:9200 # Configure Kibana pour se connecter à Elasticsearch.

depends_on:

```
- elasticsearch # Démarre Elasticsearch avant Kibana.
```

networks:

```
- elk # Connecte Kibana au réseau "elk".
```

```
# Logstash for log processing
```

logstash:

```
image: docker.elastic.co/logstash/logstash:7.10.1 # Image Logstash version 7.10.1
```

volumes:

```
- ./logstash/pipeline/logstash.conf:/usr/share/logstash/pipeline/logstash.conf # Fichier de configuration du pipeline Logstash.
```

```
- ./logstash/config/logstash.yml:/usr/share/logstash/config/logstash.yml # Fichier de configuration principal Logstash.
```

ports:

```
- "5000:5000" # Expose le port 5000 pour recevoir les logs de Spring Boot.
```

depends_on:

```
- elasticsearch # Démarre Elasticsearch avant Logstash.
```

networks:

```
- elk # Connecte Logstash au réseau "elk".
```

4. Configuration de Logstash pour la collecte de logs Spring Boot

logstash.yml

```
http.host: "0.0.0.0" # Permet à Logstash d'écouter sur toutes les interfaces réseau.
```

```
path.config: /usr/share/logstash/pipeline # Chemin vers les fichiers de configuration des pipelines Logstash.
```

```
xpack.monitoring.elasticsearch.hosts: [ "localhost:9200" ] # Configure la connexion pour surveiller via Elasticsearch.
```

Pipeline de Logstash (pipeline/logstash.conf)

Créez un fichier logstash.conf dans le répertoire pipeline pour définir la collecte des logs Spring Boot.

```
input {  
    tcp {  
        port => 5000 # Logstash écoute les logs entrants sur le port 5000.  
        codec => json # Les données reçues sont au format JSON.  
    }  
}  
  
output {  
    elasticsearch {  
        hosts => ["http://elasticsearch:9200"] # Les logs sont envoyés à Elasticsearch, hébergé sur ce serveur.  
        index => "spring-boot-logs-%{+YYYY.MM.dd}" # Les logs sont stockés dans un index nommé par date.  
    }  
}
```

5. Configuration de Spring Boot pour envoyer des logs vers Logstash

Dans l'application Spring Boot, configurez les logs pour qu'ils soient envoyés à Logstash en JSON.

Logback-spring.yml

```
<configuration>  
  
    <!-- Niveau de log minimum -->  
    <root level="INFO">  
        <appender-ref ref="LOGSTASH"/>  
    </root>  
  
    <!-- Configuration du appender pour envoyer les logs à Logstash -->  
    <appender name="LOGSTASH"  
    class="net.logstash.logback.appender.LogstashTcpSocketAppender">  
        <destination>localhost:5000</destination> <!-- Logstash écoute sur le port 5000 -->  
        <encoder class="net.logstash.logback.encoder.LoggingEventCompositeJsonEncoder">  
            <providers>  
                <timestamp>  
                    <pattern>yyyy-MM-dd'T'HH:mm:ss.SSSZZ</pattern>  
                </timestamp>  
                <message/>  
                <loggerName/>  
            </providers>  
        </encoder>  
    </appender>  
</configuration>
```

```

<threadName/>
<logLevel/>
<stackTrace/>
</providers>
</encoder>
</appender>

</configuration>
```

Cela enverra les logs à Logstash sur le port 5000, pour qu'ils soient traités et envoyés à Elasticsearch

6. Tests et Configuration de Kibana pour la visualisation

1. Démarrage des services : Lancez docker-compose up -d pour démarrer tous les services.

```

will be ignored, please remove it to avoid potential confusion"
+] Running 7/0
✓ Container prometheus-service           Running   0.0s
✓ Container zipkin-service               Running   0.0s
✓ Container spring-prometheus-grafana-elasticsearch-1 Running   0.0s
✓ Container spring-prometheus-grafana-kibana-1    Running   0.0s
✓ Container grafana-service             Running   0.0s
✓ Container spring-prometheus-grafana-logstash-1  Running   0.0s
✓ Container spring-prometheus-grafana-post-service-1 Running   0.0s
PS C:\Users\Admin\Desktop\spring-prometheus-grafana> 
```

2. Vérifiez Elasticsearch : Accédez à <http://localhost:9200> pour vous assurer qu'Elasticsearch est en marche .



3. Configuration de Kibana

-Lancez Kibana et connectez-vous à <http://localhost:5601>

-Dans **Index Patterns** de Kibana, créez un index pattern `spring-boot-logs-*`.

-Sélectionnez le champ de temps `@timestamp` pour permettre le filtrage temporel.

localhost:5601/app/management/kibana/spaces

Elastic

Stack Management / Spaces

Ingest

- Ingest Node Pipelines

Data

- Index Management
- Index Lifecycle Policies
- Snapshot and Restore
- Rollup Jobs
- Transforms
- Remote Clusters

Alerts and Insights

- Alerts and Actions
- Reporting

Kibana

- Index Patterns**
- Saved Objects
- Spaces**
- Advanced Settings

Stack

- License Management
- 8.0 Upgrade Assistant

Spaces

Organize your dashboards and other saved objects into meaningful categories.

Create a space

Space	Description	Features	Identifier	Actions
Default	This is your default space!	All features visible		

Rows per page: 10

Your data is not secure

Don't lose one bit. Enable our free security features.

Don't show again

Enable security **Dismiss**

localhost:5601/app/management/kibana/indexPatterns

Elastic

Stack Management / Index patterns

Ingest

- Ingest Node Pipelines

Data

- Index Management
- Index Lifecycle Policies
- Snapshot and Restore
- Rollup Jobs
- Transforms
- Remote Clusters

Alerts and Insights

- Alerts and Actions
- Reporting

Kibana

- Index Patterns**
- Saved Objects
- Spaces
- Advanced Settings

Stack

- License Management
- 8.0 Upgrade Assistant

Index patterns

Create and manage the index patterns that help you retrieve your data from Elasticsearch.

Create index pattern

spring

Pattern

spring-boot-logs Default

Rows per page: 10

Your data is not secure

Don't lose one bit. Enable our free security features.

Don't show again

Enable security **Dismiss**

localhost:5601/app/management/kibana/indexPatterns/create

Create index pattern

An index pattern can match a single source, for example, `filebeat-4-3-22`, or **multiple** data sources, `filebeat-*`. [Read documentation](#)

Step 1 of 2: Define an index pattern

Index pattern name

Use an asterisk (*) to match multiple indices. Spaces and the characters \, /, ?, *, <, >, | are not allowed.

Include system and hidden indices

✓ Your index pattern matches 2 sources.

spring-boot-logs-2024.11.08	Index
spring-boot-logs-2024.11.09	Index

Rows per page: 10

Your data is not secure
Don't lose one bit. Enable our free security features.
 Don't show again [Enable security](#) [Dismiss](#)

Next step >

localhost:5601/app/management/kibana/indexPatterns/create

Create index pattern

An index pattern can match a single source, for example, `filebeat-4-3-22`, or **multiple** data sources, `filebeat-*`. [Read documentation](#)

Step 2 of 2: Configure settings

Specify settings for your **spring*** index pattern.

Select a primary time field for use with the global time filter.

Time field Refresh

> Show advanced settings

< Back [Create index pattern](#)

Stack Management / Index patterns / Create index pattern

localhost:5601/app/management/kibana/indexPatterns/patterns/27806600-9eda-11ef-b9d9-2d60089c2243#?_a=(tab:indexedFields)

Elastic

Stack Management / Index patterns / spring*

Ingest Node Pipelines

spring*

Time field: '@timestamp'

This page lists every field in the **spring*** index and the field's associated core type as recorded by Elasticsearch. To change a field type, use the Elasticsearch [Mapping API](#).

Fields (25) Scripted fields (0) Source filters (0)

Name	Type	Format	Searchable	Aggregatable	Excluded
@timestamp	date		●	●	
@version	string		●		
@version.keyword	string		●	●	
_id	string		●	●	
_index	string		●	●	
_score	number				
_source	_source				
_type	string		●	●	
app	string		●		

localhost:5601/app/management/

Elastic

Stack Management

Home

Recently viewed

No recently viewed items

Kibana

Overview

Discover

Dashboard

Canvas

Maps

Machine Learning

Visualize

Enterprise Search

Overview

App Search

Workplace Search

Observability

Overview

Logs

Welcome to Stack Management 7.10.1

Manage your indices, index patterns, saved objects, Kibana settings, and more.

A complete list of apps is in the menu on the left.

The screenshot shows the Elastic search interface at [localhost:5601/app/discover/?_g=\(filters:!{refreshInterval:\(pause!t,value:0\),time:\(from:now-15m,to:now\)\)&_a=\(columns!:!{_source}.filters:!{index:27806600-9eda-11ef-b9d9-2...](http://localhost:5601/app/discover/?_g=(filters:!{refreshInterval:(pause!t,value:0),time:(from:now-15m,to:now))&_a=(columns!:!{_source}.filters:!{index:27806600-9eda-11ef-b9d9-2...). The search bar contains 'spring*'. A modal window titled 'CHANGE INDEX PATTERN' is open, showing filter options: s*, sp*, and spring*. The main search results area displays the message: 'No results match your search criteria'. Below it, a note says: 'Expand your time range' and 'One or more of the indices you're looking at contains a date field. Your query may not match anything in the current time range, or there may not be any data at all in the currently selected time range. You can try changing the time range to one which contains data.'

J'ai spécifié les logs dans post-service pour faire le test

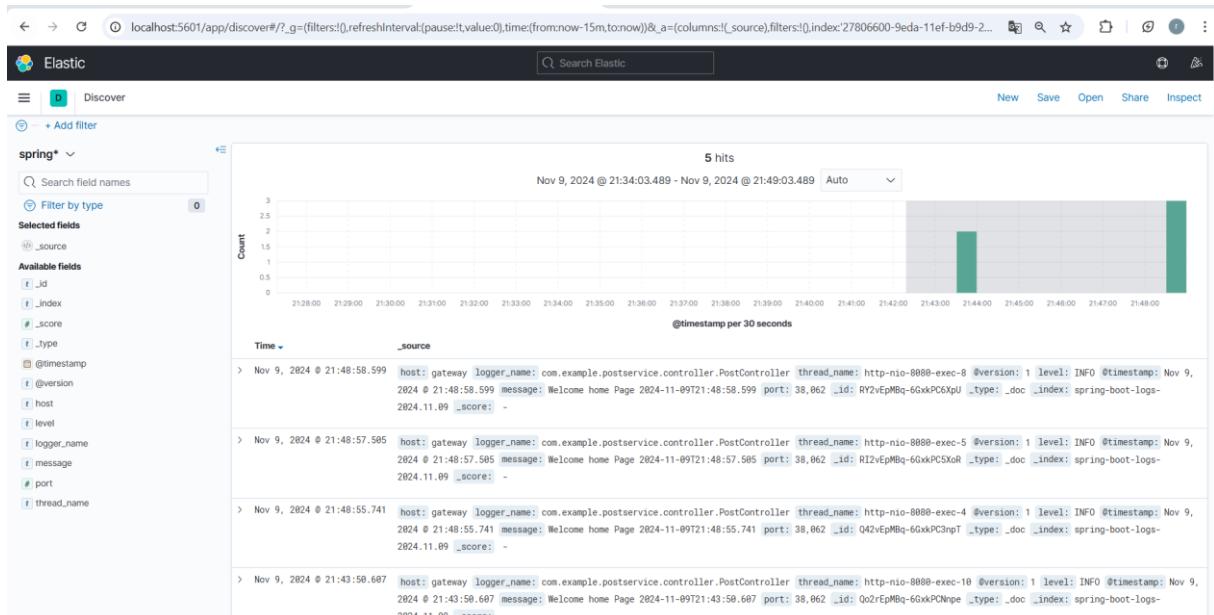
The screenshot shows the IntelliJ IDEA code editor with the file `PostController.java` open. The code defines a `PostController` class with four methods: `HomePage()`, `LogsPage()`, `WarnPage()`, and `ErrorPage()`. Each method contains a `log.info` statement. Red arrows point from the right margin to each of these `log.info` statements, highlighting them. The code editor also shows other files like `PostServiceApplication.java`, `logstash.conf`, and `PostServiceApplicationTests.java`.

```
public class PostController {
    // Ajoute pour ele et kibana
    @GetMapping("/wel")
    public String HomePage(){
        LocalDateTime localDateTime = LocalDateTime.now();
        log.info("Welcome home Page " + localDateTime);
        return "Welcome to Home page post";
    }

    @GetMapping("/Logs")
    public String LogsPage(){
        LocalDateTime localDateTime = LocalDateTime.now();
        log.info("This Logs page " + localDateTime);
        return "Welcome to logs page post";
    }

    @GetMapping("/warn")
    public String WarnPage(){
        LocalDateTime localDateTime = LocalDateTime.now();
        log.warn("This warn page " + localDateTime);
        return "Welcome to warn page post";
    }

    @GetMapping("/er")
    public String ErrorPage(){
        LocalDateTime localDateTime = LocalDateTime.now();
        log.error("This error page " + localDateTime);
        return "Welcome to error page post";
    }
}
```

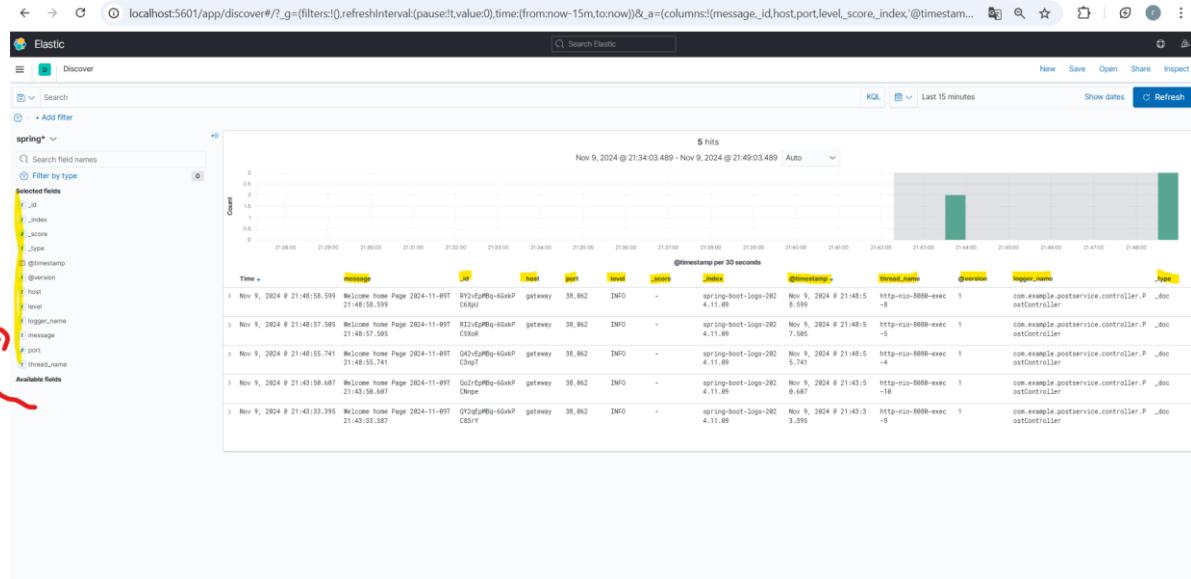


- j'ai fait le test pour le log de type **info** « voit le code contrôler de post-service pour comprendre localhost:8080/api/v1/posts/wel



On clique sur Refresh et ajoute les champs disponibles (message, Level....) Ces champs sont essentiels pour comprendre et analyser les données de vos logs.

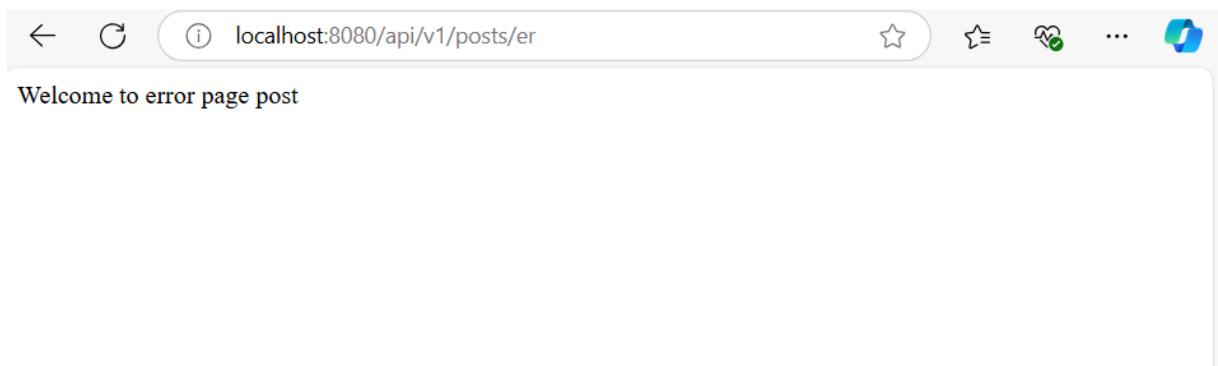
- **_id** : Identifiant unique du document dans Elasticsearch.
- **_index** : Index auquel le document appartient, utilisé pour organiser les logs.
- **_score** : Score de pertinence, surtout pour les recherches.
- **_type** : Type de document (souvent "`_doc`" dans les versions récentes).
- **@timestamp** : Date et heure de création du log, essentiel pour le suivi temporel.
- **@version** : Version du schéma de log (généralement fixé à 1).
- **host** : Source de l'hôte (nom ou adresse IP d'où le log est généré).
- **level** : Niveau de gravité du log (INFO, ERROR, WARN, etc.).
- **logger_name** : Nom du logger ayant généré le log, utile pour identifier le module source.
- **message** : Message principal du log, décrivant l'événement ou l'erreur.
- **port** : Port de l'application générant le log, utile pour distinguer les services si plusieurs sont utilisés.
- **thread_name** : Nom du thread où le log a été généré, pertinent pour le suivi des processus concurrents.
- **app** : Nom de l'application générant le log.
- **stack_trace** : Détail de l'erreur ou de l'exception, pour le débogage.
- **tags** : Étiquettes personnalisées pour organiser ou filtrer les logs selon des critères spécifiques.



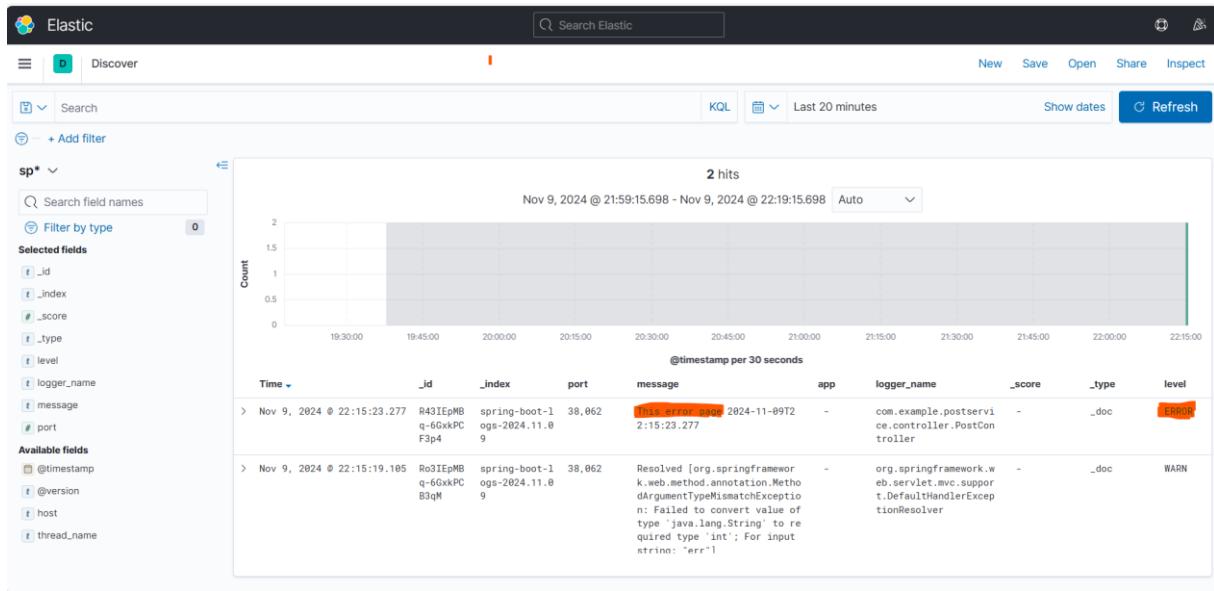
Voilà, on obtient le message de log le type info

Time	message	_id	host	port	level	_score	_index	@timestamp	thread_name	@version	logger_name	_type
Nov 9, 2024 @ 21:48:58.599	Welcome home Page 2024-11-09T21:48:58.599	Rt2vEpM8q-6G	gateway	38,062	INFO	-	spring-boot-logs-2024-11-09	Nov 9, 2024 @ 21:48:58.599	http-nio-8080-exec-8	1	com.example.postservice.controller.PostController	_doc
Nov 9, 2024 @ 21:48:57.595	Welcome home Page 2024-11-09T21:48:57.595	Rt2vEpM8q-6G	gateway	38,062	INFO	-	spring-boot-logs-2024-11-09	Nov 9, 2024 @ 21:48:57.595	http-nio-8080-exec-5	1	com.example.postservice.controller.PostController	_doc
Nov 9, 2024 @ 21:48:55.741	Welcome home Page 2024-11-09T21:48:55.741	Qo2vEpM8q-6G	gateway	38,062	INFO	-	spring-boot-logs-2024-11-09	Nov 9, 2024 @ 21:48:55.741	http-nio-8080-exec-4	1	com.example.postservice.controller.PostController	_doc
Nov 9, 2024 @ 21:43:58.687	Welcome home Page 2024-11-09T21:43:58.687	Qo2vEpM8q-6G	gateway	38,062	INFO	-	spring-boot-logs-2024-11-09	Nov 9, 2024 @ 21:43:58.687	http-nio-8080-exec-10	1	com.example.postservice.controller.PostController	_doc
Nov 9, 2024 @ 21:43:33.995	Welcome home Page 2024-11-09T21:43:33.995	Qo2vEpM8q-6G	gateway	38,062	INFO	-	spring-boot-logs-2024-11-09	Nov 9, 2024 @ 21:43:33.995	http-nio-8080-exec-9	1	com.example.postservice.controller.PostController	_doc

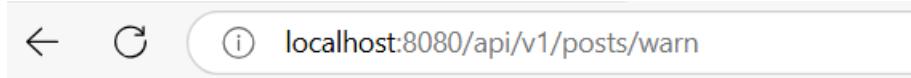
- j'ai fait le test pour le log de type erreur « voit le code contrôler de post-service pour comprendre <localhost:8080/api/v1/posts/er>



Voilà, on obtient le message de log le type erreur

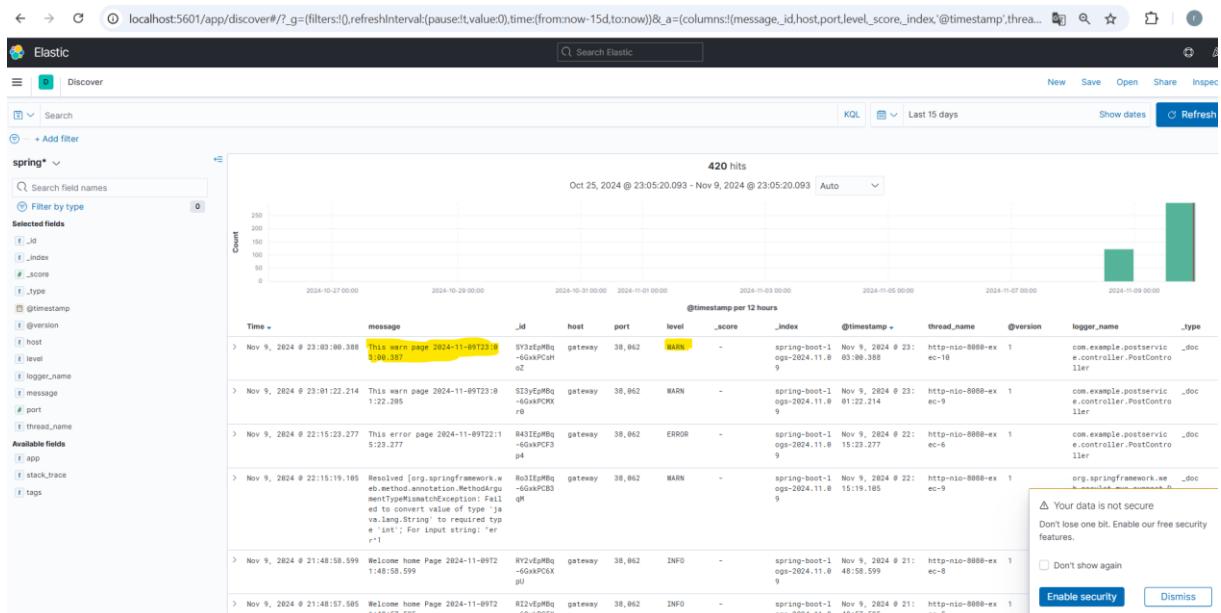


- Mêmes étapes pour log de WARN juste cette fois en utilisant <localhost:8080/api/v1/posts/warn>



Welcome to warn page post

Voilà, on obtient le message de log le type WARN



Conclusion

La centralisation des logs d'une application Spring Boot dans la stack ELK (Elasticsearch, Logstash, Kibana) permet de gérer, visualiser et analyser efficacement les journaux d'événements en un seul endroit. En configurant Spring Boot pour envoyer des logs structurés vers Logstash, qui les transfère ensuite à Elasticsearch, il devient possible d'exploiter la puissance de Kibana pour créer des tableaux de bord et analyser les tendances des données en temps réel.

Cette solution améliore la surveillance de l'application en offrant une vue d'ensemble des logs, permettant aux équipes de réagir rapidement aux problèmes, d'optimiser les performances et de faciliter le débogage des applications complexes.

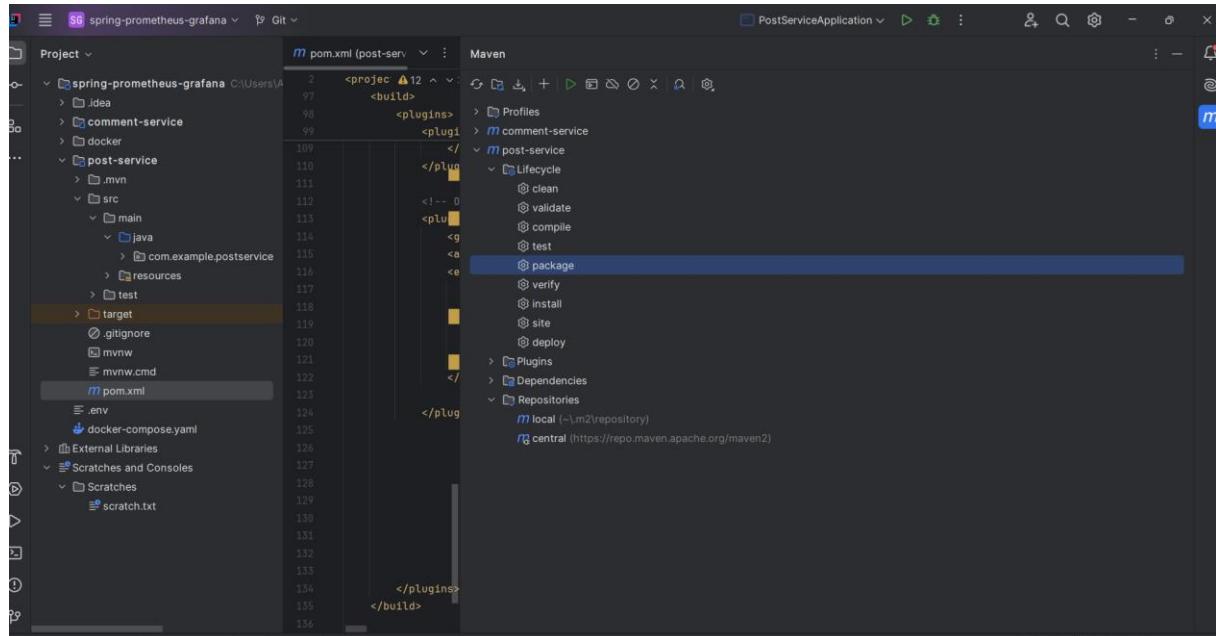
Remédier aux Vulnérabilités avec la Gestion des Dépendances Maven dans Spring Boot

1. Introduction

La gestion des dépendances Maven est essentielle pour garantir que les dépendances de votre application Spring Boot sont sécurisées, à jour et exemptes de vulnérabilités connues. Remédier aux vulnérabilités implique de gérer et de mettre à jour ces dépendances pour éviter les risques de sécurité. Cela fait partie de l'approche **DevSecOps**, qui intègre les pratiques de sécurité dans le pipeline DevOps, en se concentrant sur la détection précoce des vulnérabilités.

Voici comment vous pouvez remédier aux vulnérabilités en utilisant Maven dans un projet Spring Boot :

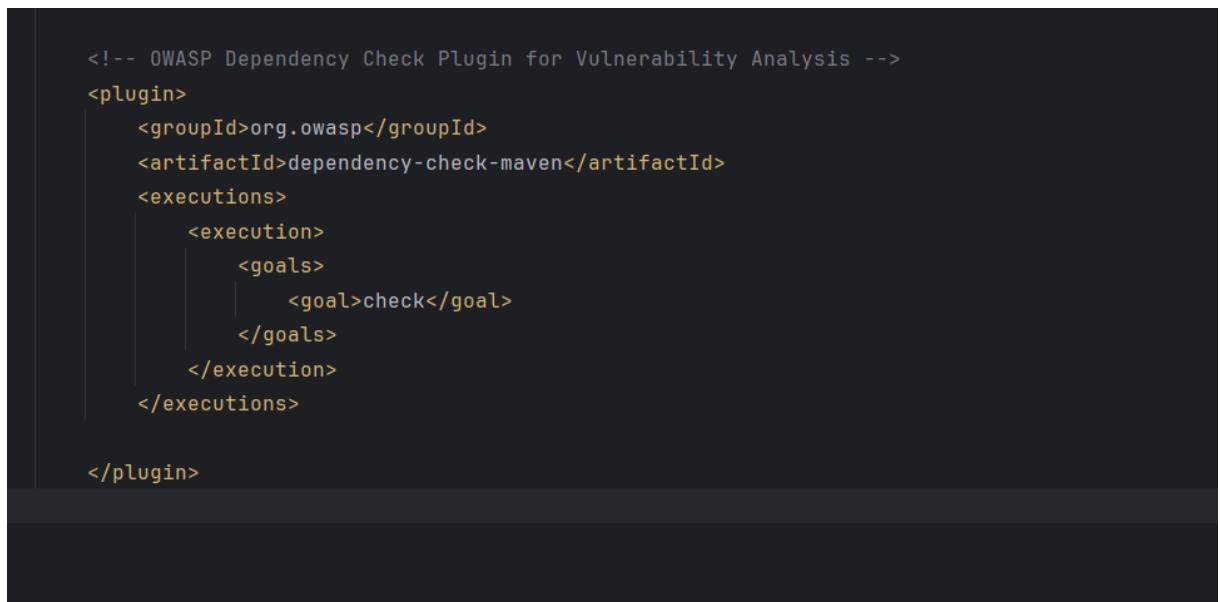
2. Utiliser le Maven package



3. Identifier les Vulnérabilités des Dépendances

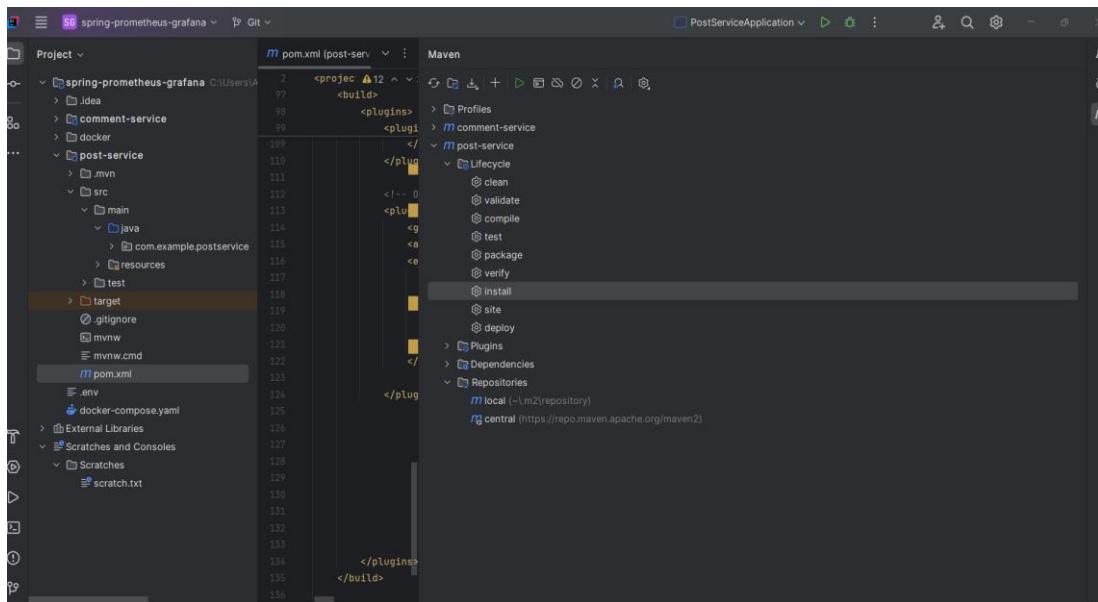
Avant de remédier aux vulnérabilités, vous devez les identifier dans les dépendances de votre projet. Vous pouvez utiliser des outils comme **OWASP Dependency-Check** pour analyser notre projet(j'ai appliqué sur service post-service) à la recherche de vulnérabilités connues.

- Ajoutez le plugin **dependency-check-maven** à notre fichier **pom.xml** de post-service.



```
<!-- OWASP Dependency Check Plugin for Vulnerability Analysis -->
<plugin>
    <groupId>org.owasp</groupId>
    <artifactId>dependency-check-maven</artifactId>
    <executions>
        <execution>
            <goals>
                <goal>check</goal>
            </goals>
        </execution>
    </executions>
</plugin>
```

4. Installer l'application pour générer le rapport des Vulnérabilités



The screenshot shows the IntelliJ IDEA interface with the Maven tool window open. The project structure on the left includes 'spring-prometheus-grafana' and 'comment-service'. The central pane displays the output of a 'post-service [install]' Maven command. The log output shows various INFO and WARNING messages related to the dependency check process, including file analysis, version filtering, and suppression rule processing. A note at the bottom indicates that Dependency-Check is an open-source tool performing a best-effort analysis.

```
2024-11-07T20:11:43.370+01:00 INFO [spring-post-service,,] 1636 --- [main] o.s.b.a.e.web.EndpointLinksResolver
2024-11-07T20:11:43.509+01:00 INFO [spring-post-service,,] 1636 --- [main] c.e.p.PostServiceApplicationTests
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 11.104 s - in com.example.postservice.PostServiceApp
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO]
[INFO] --- jar:3.3.0:jar (default-jar) @ post-service ---
[INFO]
[INFO] --- spring-boot:3.1.0:repackage (repackage) @ post-service ---
[INFO] Replacing main artifact C:\Users\Admin\Desktop\spring-prometheus-grafana\post-service\target\post-service-0.0.1-SNAPSHOT.jar with C:\Users\Admin\Desktop\spring-prometheus-grafana\post-service\target\post-service-0.0.1-SNAPSHOT.jar
[INFO] The original artifact has been renamed to C:\Users\Admin\Desktop\spring-prometheus-grafana\post-service\target\post-service-0.0.1-SNAPSHOT.jar
[INFO]
[INFO] --- dependency-check:11.1.0:check (default) @ post-service ---
[WARNING] Explicitly loaded driver org.h2.Driver from classpath; if JDBCv4 service loading is supported by the driver you can skip this check
[INFO] Checking for updates
[INFO] Skipping the NVD API Update as it was completed within the last 240 minutes
[INFO] Skipping Known Exploited Vulnerabilities update check since last check was within 24 hours.
[INFO] Check for updates complete (1652 ms)
[WARNING] Explicitly loaded driver org.h2.Driver from classpath; if JDBCv4 service loading is supported by the driver you can skip this check
[INFO]

Dependency-Check is an open source tool performing a best effort analysis of 3rd party dependencies; false positives and false negatives may occur.
```

The screenshot shows a terminal window displaying the execution of the 'dependency-check' command. The output shows the progress of the analysis, starting with file analyzers like Archive, File Name, Jar, and Dependency Merging. It then moves on to Hint, Version Filter, and CPE index creation. Following this, it performs CPE, False Positive, NVD CVE, Sonatype OSS Index, and Vulnerability Suppression analyses. It also handles Known Exploited Vulnerability and Dependency Bundling analysis. The process concludes with suppression rule processing, unused suppression rule analysis, and finally, the completion of the analysis and the writing of an HTML report to the specified target directory.

```
[INFO] Analysis Started
[INFO] Finished Archive Analyzer (1 seconds)
[INFO] Finished File Name Analyzer (0 seconds)
[INFO] Finished Jar Analyzer (1 seconds)
[INFO] Finished Dependency Merging Analyzer (0 seconds)
[INFO] Finished Hint Analyzer (0 seconds)
[INFO] Finished Version Filter Analyzer (0 seconds)
[INFO] Created CPE Index (5 seconds)
[INFO] Finished CPE Analyzer (8 seconds)
[INFO] Finished False Positive Analyzer (0 seconds)
[INFO] Finished NVD CVE Analyzer (0 seconds)
[INFO] Finished Sonatype OSS Index Analyzer (1 seconds)
[INFO] Finished Vulnerability Suppression Analyzer (0 seconds)
[INFO] Finished Known Exploited Vulnerability Analyzer (0 seconds)
[INFO] Finished Dependency Bundling Analyzer (0 seconds)
[INFO] Suppression Rule had zero matches: SuppressionRule{packageUrl=PropertyType{value=^pkg:maven/io\.etcdd/jetcd-[a-z]*.jar}, severity=INFO}
[INFO] Suppression Rule had zero matches: SuppressionRule{packageUrl=PropertyType{value=^pkg:maven/io\.etcdd/jetcd-grpc@.*$}, severity=INFO}
[INFO] Finished Unused Suppression Rule Analyzer (0 seconds)
[INFO] Analysis Complete (14 seconds)
[INFO] Writing HTML report to: C:\Users\Admin\Desktop\spring-prometheus-grafana\post-service\target\dependency-check-report
[WARNING]
```

5. le dossier Target contient le rapport des Vulnérabilités

This document contains very long lines. Soft wraps were enabled to improve editor performance.

```
<html>
  <head>
    <title>Dependency-Check Report</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <link rel="shortcut icon" href="data:base64,>
      _1vB0Rw0KGoAAANnUhEugAACAAAAGCAYAAEbzenn0AAAAAAANSROIars4c6QAAAAnQUB1AAcCxjw8YQAAAALSBVBFhVzdbFRFEMB3913v9Vr,>
      _xUNRntrdrewgllg0E6g9YJsw0MwSPkGdwZlghR+UY2geIuEje+R+rIfE+eULrl+QYAwhBQmkFQDp01xd,>
      _vPbaSu9Uy767r0dS515vXsqZcsb3u9C5zsu1m81Ea5fhyJylzRYCz7Utl+wbnQL+Nd0TeoyePLFgf,>
      _o19TnWq9y6WuHnWtZ4Jte1nXKuqNzJzsuJHxMhE4cjy9p6etTIPco2zbxFH016V9yCrhMjC,>
      _xF53oG6Q6KwxsicsDqNzXNyWbNSVYUFAFkyAx2XmAx1HlGpukJr1oCDV0Ks0admgBaxL7y8CkuBuq0lHlnKufFk,>
      _Gdlm0q6cTpmBq16dXZRLtCkR8rgedIxkgNyXm151A2ewhNPW84qAx2hewno6wyFH0AZOLAc6PrBu,>
      _sd20YrbmpbPTgjplQzga6mBq04HKL3LzVbc1/Tl02+gqlxV9yFWN0QdC0Kf3Jm1,>
      _C2p2WwkGc621p0cDb7hShgJ0cWUAG8sbV1KtEuSqFBd0lKmuPzK2z6mz8B8XzB5irD8uP1fzQp61+V1y9eAS9vxjYAzV9ez8p2a0ck,>
      _737KQPFrAr8Vtpx7k001Y1Lmpcv3zr1w1syFkyaIFSZXzzA1zqzg9R0+7t11021zVwN7mQ0Q1ldmneyrsqBq401tWt9qzGt6zhfb5s,>
      _dHrpw-THPFgvdhA8xdzj84pRdz010nq1YukR0M31GkVREKHmKsJwAdmlw+TuJuxor3N16z6L9B1w3MSMav1,>
      _FW9w5Ntgcl8ayTP0bjx3tL7TKxqJyK52y4nv5u4f7stzqN,>
      _HwnOrc5Q8J0yBoDwryN07JVLVt9x07400pV7f6nd5C071vEmD01Vqhz9yR0+7t11021zVwN7mQ0Q1ldmneyrsqBq401tWt9qzGt6zhfb5s,>
      _sLEAs2G09n0WzMyhd/Lt,>
      _sd0Umthuzun3C8drdnyFStcrpA01AE1cv16lmjkC5d1Sw8zB3zH0Vl020tKiz70tBjAM4kopVgoseB2n1JyFexJFrxG0524ds+xWmN9t1,>
      _7t8sDALX7Vvvjjd1cChg140yhDzUsy1PGkA1PaL1u743+/v365bpc4PhBSxZsM9qJzXWbG1xLzqsfkWLN5jcuVUbU1+D360uP1PPWnR,>
      _wZAW3HERRHs5y4C3Lehdo1An368b6xdhxKLrvN5nOpfZ2Fg7J00f50y9RjLkV5WgYJvnk2yRpUkfBz6Q0yMfaAdW0+0xXrdSE9E8V3wAmFD70se0,>
      _XNHNLLhJmXuJyJvUdJyJnre1CvXz62k20y4Pm7K4nSmxtE231r1v1Rt80UBf1B2fyaYzcp7guefERtix9,>
      _Mj6nN5DpM21zq0pd0841N8BxRvaWYyRgE77Ad4vhzXh4615812e1t092R/+dN0Z1lqbsW+VN,>
      _S2x12yIXy94xb2d5lQsUdC9z613tL1Wns6tveK91y0LzCmmh7E731TiFyqAp+/>
      _4L2CSUYIXf7ffmB86k6nxJkn0zB61Q80ja1FPTXU0m5vzs/6770ngdK0XLZ7nNnGhuXmn,>
      _w9y9Tytw2yU1efz1B883C2s514m="AAAABAJRSURkBTggg==" />
    <script type="text/javascript">
      /*! jQuery v3.7.1 | (c) OpenJS Foundation and other contributors | jquery.org/license */
      !function(e,t){"use strict";"object"==typeof module&&"object"==typeof module.exports&&module.exports=e.document.exports=t(e,!0)
    
```

-Ouvrir le fichier dependency-check-report.html dans le navigateur puis on a obtenu des Vulnérabilités détaillées de notre service post-service.



DEPENDENCY-CHECK

Dependency-Check is an open source tool performing a best effort analysis of 3rd party dependencies. False positives and false negatives may exist in the analysis performed by the tool. Use of the tool and the reporting provided constitutes acceptance for use in an AS IS condition, and there are NO warranties, implied or otherwise, with regard to the analysis or its use. Any use of the tool and the reporting provided is at the user's risk. In no event shall the copyright holder or OWASP be held liable for any damages whatsoever arising out of or in connection with the use of this tool, the analysis performed, or the resulting report.

[How to read the report](#) | [Suppressing false positives](#) | [Getting Help: github issues](#)

[!\[\]\(04f079d792c65fca73a2705e3ffd94ce_img.jpg\) Sponsor](#)

Project: post-service

com.example.post-service:0.0.1-SNAPSHOT

Scan Information ([show all](#)):

- dependency-check version: 11.1.0
- Report Generated On The: 7 Nov 2024 20:12:12 +0100
- Dependencies Scanned: 77 (51 unique)
- Vulnerable Dependencies: 15
- Vulnerabilities Found: 34
- Vulnerabilities Suppressed: 0
- ...

Summary

Display [Showing Vulnerable Dependencies \(click to show all\)](#)

Dependency	Vulnerability IDs	Package	Highest Severity	CVE Count	Confidence	Evidence Count
bcprov-jdk15on-1.69.jar	cpe:2.3:a:bouncycastle:bouncy_castle_for_java:1.69:***** cpe:2.3:a:bouncycastle:bouncy_castle_crypto:package:1.69:***** cpe:2.3:a:bouncycastle:bouncy_castle_crypto_package:1.69:***** cpe:2.3:a:bouncycastle:bouncy_castle_for_java:1.69:***** cpe:2.3:a:bouncycastle:legion_of_the-bouncy-castle-java-cryptography-api:1.69:*****	pkg:maven/org/bouncycastle/bcprov-jdk15on@1.69 pkg:maven/org/bouncycastle/bcprov-jdk15on@1.69	MEDIUM	1	Highest	66
bcprov-jdk15on-1.69.jar			HIGH	5	Highest	60

Ce rapport de **Dependency-Check** pour le **projet post-service** identifie les vulnérabilités présentes dans les bibliothèques utilisées.

Résumé des points clés :

- **77 dépendances** ont été analysées dans le projet.
 - **15 dépendances vulnérables** ont été trouvées.

- **34 vulnérabilités** ont été identifiées au total, avec des niveaux de严重性 allant de moyen à élevé.

Chaque dépendance vulnérable est listée avec des informations sur la严重性 des vulnérabilités détectées et le nombre de preuves qui justifient chaque vulnérabilité. Ce rapport permet aux développeurs de repérer et corriger les failles de sécurité en mettant à jour ou en remplaçant les dépendances vulnérables.

Summary						
Dependency	Vulnerability IDs	Package	Highest Severity	CVE Count	Confidence	Evidence Count
bcpkix-jdk15on-1.69.jar	cpe:2.3:a:bouncycastle:bouncy_castle_for_java:1.69:*****	pkg:maven:org.bouncycastle:bcpkix-jdk15on@1.69	MEDIUM	1	Highest	66
bcprov-jdk15on-1.69.jar	cpe:2.3:a:bouncycastle:bouncy_castle_crypto_package:1.69:***** cpe:2.3:a:bouncycastle:bouncy_castle_for_java:1.69:***** cpe:2.3:a:bouncycastle:legion-of-the-bouncy-castle-java-cryptography-api:1.69:***** cpe:2.3:a:bouncycastle:the_bouncy_castle_crypto_package_for_java:1.69:*****	pkg:maven:org.bouncycastle:bcprov-jdk15on@1.69	HIGH	5	Highest	60
bcutil-jdk15on-1.69.jar	cpe:2.3:a:bouncycastle:bouncy_castle_for_java:1.69:*****	pkg:maven:org.bouncycastle:bcutil-jdk15on@1.69	MEDIUM	1	Highest	50
commons-fileupload-1.4.jar	cpe:2.3:a:apache:commons_fileupload:1.4:*****	pkg:maven:commons-fileupload@1.4	HIGH	1	Highest	115
jackson-databind-2.15.0.jar	cpe:2.3:a:fasterxml:jackson-databind:2.15.0:***** cpe:2.3:a:fasterxml:jackson-modules-java8:2.15.0:*****	pkg:maven:com.fasterxml.jackson.core:jackson-databind@2.15.0	MEDIUM	1	Highest	41
logback-core-1.4.7.jar	cpe:2.3:org:logback:logback-core:1.4.7:*****	pkg:maven:ch.qos.logback:logback-core@1.4.7	HIGH	1	Highest	36
snakeyaml-1.33.jar	cpe:2.3:a:snakeyaml_project:snakeyaml:1.33:*****	pkg:maven:org.yaml:snakeyaml@1.33	CRITICAL	1	Highest	40
spring-aop-6.0.9.jar	cpe:2.3:a:pivotal_software:spring_framework:6.0.9:***** cpe:2.3:a:springsource:spring_framework:6.0.9:***** cpe:2.3:a:vmware:spring_Framework:6.0.9:*****	pkg:maven:org.springframework:spring-aop@6.0.9	HIGH	2	Highest	35
spring-boot-3.1.0.jar	cpe:2.3:a:vmware:spring_boot:3.1.0:*****	pkg:maven:org.springframework.boot:spring-boot@3.1.0	MEDIUM	1	Highest	38
spring-boot-starter-web-3.1.0.jar	cpe:2.3:a:vmware:spring_web_project:web:3.1.0:***** cpe:2.3:a:web_project:web:3.1.0:*****	pkg:maven:org.springframework.boot:spring-boot-starter-web@3.1.0	MEDIUM	1	Highest	36
spring-core-6.0.9.jar	cpe:2.3:a:pivotal_software:spring_framework:6.0.9:***** cpe:2.3:a:springsource:spring_framework:6.0.9:***** cpe:2.3:a:vmware:spring_Framework:6.0.9:*****	pkg:maven:org.springframework:spring-core@6.0.9	HIGH	3	Highest	37
spring-security-crypto-6.1.0.jar	cpe:2.3:a:pivotal_software:spring_security:6.1.0:***** cpe:2.3:a:vmware:spring_security:6.1.0:*****	pkg:maven:org.springframework.security:spring-security-crypto@6.1.0	CRITICAL	2	Highest	38
spring-web-6.0.9.jar	cpe:2.3:a:pivotal_software:spring_framework:6.0.9:***** cpe:2.3:a:springsource:spring_framework:6.0.9:***** cpe:2.3:a:vmware:spring_Framework:6.0.9:***** cpe:2.3:a:web_project:web:6.0.9:*****	pkg:maven:org.springframework:spring-web@6.0.9	HIGH	5	Highest	35
spring-webmvc-6.0.9.jar	cpe:2.3:a:pivotal_software:spring_framework:6.0.9:***** cpe:2.3:a:springsource:spring_framework:6.0.9:***** cpe:2.3:a:vmware:spring_Framework:6.0.9:***** cpe:2.3:a:web_project:web:6.0.9:*****	pkg:maven:org.springframework:spring-webmvc@6.0.9	HIGH	3	Highest	37
tomcat-embed-core-10.1.8.jar	cpe:2.3:a:apache:tomcat:tomcat-embed-core@10.1.8:***** cpe:2.3:a:apache_tomcat:apache_tomcat-embed-core@10.1.8:*****	pkg:maven:org.apache.tomcat.embed:tomcat-embed-core@10.1.8	HIGH*	6	Highest	63

Ce tableau du rapport **Dependency-Check** affiche les détails des dépendances vulnérables dans le service **post-service**

- **Dependency** : Liste les bibliothèques utilisées dans le projet avec leur version (exemple : lucene-core-8.11.1.jar).
- **Vulnerability IDs** : Identifiants des vulnérabilités associées à chaque dépendance (souvent des CVE).
- **Package** : Référence Maven ou source de la dépendance (exemple : org.apache.lucene/lucene-core@8.11.1).
- **Highest Severity** : Niveau de gravité maximal des vulnérabilités (CRITICAL, HIGH, MEDIUM, LOW).
- **CVE Count** : Nombre total de vulnérabilités (CVE) détectées pour la dépendance.
- **Confidence** : Niveau de certitude de l'outil concernant la vulnérabilité (exemple : Highest pour une confiance élevée).
- **Evidence Count** : Nombre de preuves identifiées pour confirmer la vulnérabilité.

Dependencies (vulnerable)

bcpkix-jdk15on-1.69.jar

Description:
The Bouncy Castle Java APIs for CMS, PKCS, EAC, TSP, CMP, CRMF, OCSP, and certificate generation. This jar contains APIs for JDK 1.5 and up. The APIs can be used in conjunction with a JCE/JCA provider such as the one provided with the Bouncy Castle Cryptography APIs.

License:
Bouncy Castle Licence: <https://www.bouncycastle.org/licence.html>

File Path: C:\Users\Admin\m2repository\org\bouncycastle\bcpkix-jdk15on\1.69\bcpkix-jdk15on-1.69.jar
MD5: cdff7e841c1c2cbbeede75f50fe8d72ee
SHA1: 45c36fb72fafb069303a97956ec28305795e6d
SHA256: 3a336f67c68a75329049fb7d28ed2
Referenced In Project/Scope: post-service compile
Included by: pkg.maven.org:springframework.cloud:spring-cloud-starter-openfeign@3.1.5

Evidence

Identifiers

- pkg.maven.org:bouncycastle:bcpkix-jdk15on@1.69 (Confidence High)
- cve.2.3.a.bouncycastle.bouncy_castle_for_java.1.69***** (Confidence Highest) [suppress](#)

Published Vulnerabilities

CVE-2023-33202 [suppress](#)

Bouncy Castle for Java before 1.73 contains a potential Denial of Service (DoS) issue within the Bouncy Castle org.bouncycastle.openssl PEMParser class. This class parses OpenSSL PEM encoded streams containing X.509 certificates, PKCS# encoded keys, and PKCS7 objects. Parsing a file that has crafted ASN.1 data through the PEMParser causes an OutOfMemoryError, which can enable a denial of service attack. (For users of the FIPS Java API: BC-FJA 1.0.2.3 and earlier are affected; BC-FJA 1.0.2.4 is fixed.)

CWE-400 Uncontrolled Resource Consumption

- Base Score: MEDIUM (5.5)
- Vector: CVSS3.1/AV:L/AC:L/PR:N/UI:N/I:N/A:H/E:1.8/RC:R/MAV:A

References:

- cve@mitre.org: [EXPLOIT_THIRD_PARTY ADVISED](#)

La bibliothèque **Bouncy Castle (bcpkix-jdk15on-1.69.jar)** a une faille de sécurité qui peut permettre une attaque pour bloquer l'application. **Solution :** Mettez-la à jour pour corriger cette faille.

Conclusion

La gestion des dépendances Maven dans un projet Spring Boot est un élément clé pour garantir la sécurité et la stabilité de l'application. En utilisant des outils comme **OWASP Dependency-Check**, vous pouvez identifier et remédier aux vulnérabilités des dépendances avant qu'elles n'affectent votre application en production. Cette approche fait partie intégrante de la méthodologie **DevSecOps**, qui vise à intégrer les pratiques de sécurité dès le début du cycle de développement.