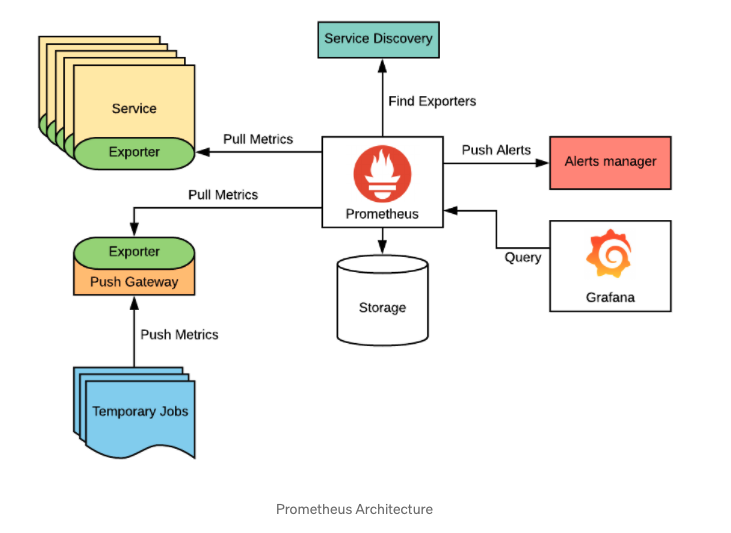
Monitoring Spring Boot with Micrometer and Prometheus

This POC demonstrate:

* How to monitor **Spring Boot** Application metrics using **Prometheus**
* How to **visualize metrics** using **Grafana Dashboard**
* How to trigger Alerts using the **Alert Manager** component
* How to publish metrics to **Prometheus** using **Prometheus Push Gateway**
* How to scrap application log metrics using **prometheus-simpleclient\_log4j2** library
* How to publish RabbitMQ metrics to **Prometheus** and **Grafana**
* How to **notify alert** via **email** and **webhook** and **sms** etc.

Teams

Description automatically generated  
  
**SIMPLE Architecture**   
  
  
  
  
  
  
  
  
1. Spring Boot Actuator module includes a number of built-in endpoints that exposes app metrics.

2. Spring Boot Actuator provides dependency management and auto-configuration for Micrometer.  
  
Timeline

Description automatically generated

3. Micrometer, an application metrics facade that supports numerous monitoring systems, including: prometheus

4. we are going to leverage HTTP endpoints to see metrics.

5. Micrometer job is to expose our application metrics to external monitoring systems while prometheus job is to scape and store metrics information exposed by it.

Diagram

Description automatically generated

6. If multiple applications publish metrics data, Spring Boot can use a tag to identify the application name. It does it by registering MeterRegistry.  
  
Graphical user interface, application

Description automatically generated  
  
  
7. The micrometer framework provides several types of the custom metrics or meters such as Counter, Gauge, Timer, DistributionSummary that you can use to create custom metrics

• Count the number of courses created using the Counter metric

• Count the number of courses created using the Gauge metric.

• Capture the time taken to create the course using the Timer metric

• Capture the distribution summary of the course ratings using the DistributionSummary metric  
  
Diagram

Description automatically generated

**Prometheus** is a monitoring system and time-series database that allows us to store time-series data, which includes the metrics of an application over time, a simple way to visualize the metrics, or setting up alerts on different metrics.

We will now configure Prometheus to scrape Spring Boot metrics. To do that, open the **prometheus.yml** file and add the following job\_name in the **scrape\_configs section**:

Text

Description automatically generated

Following Prometheus URL to see all registered endpoints: http://localhost:9090/targets

## A picture containing graphical user interface Description automatically generated **Capturing Spring Boot Metrics**

We can collection various metrics exposed by Spring Boot application:

### Chart, line chart Description automatically generated Within the Path *“actuator/prometheus”*there are many interesting metrics Graphical user interface, text, application, email Description automatically generated Text Description automatically generated A picture containing text Description automatically generated WE can use PQL to narrow our specific metric data: Graphical user interface, text, application, email Description automatically generated Graphical user interface, application Description automatically generated **Grafana** allows you to obtain data from various data sources such as Prometheus and visualize it through exquisite graphics

A screenshot of a video game

Description automatically generated with medium confidence

**Prometheus Push Gateway :** The Prometheus Pushgateway exists to allow ephemeral and batch jobs to expose their metrics to Prometheus. Since these kinds of jobs may not exist long enough to be scraped, they can instead push their metrics to a Pushgateway. The Pushgateway then exposes these metrics to Prometheus.  
  
Text

Description automatically generated

Simple Spring Boot Scheduler function:

Text

Description automatically generated  
Push metrics to pushgateway agent.

Graphical user interface

Description automatically generated

**prometheus-simpleclient\_log4j2** : It’s a Metrics collector for log4j2 appender logging.   
Graphical user interface, text, application

Description automatically generated  
  
To register the Logback collector can be added to the root level in log4j2-spring.xml file.  
  
Graphical user interface, text, application

Description automatically generated

Text

Description automatically generated  
  
  
**Alertmanager** manages alerts sent by Prometheus server. It takes care of grouping alerts, and routing them to the correct receiver. For example an email receiver, pager, webhook, sms etc.

Alerts are generally included in a separate file, in our case it’s rules.yml

Text

Description automatically generated

**RabbitMQ metric exporter to Prometheus:**While RabbitMQ management UI also provides access to a subset of metrics, it by design doesn't try to be a long term metric collection solution.   
  
As of 3.8.0, RabbitMQ ships with built-in Prometheus & Grafana support.

With the rabbitmq\_prometheus plugin, rabbitMq can ship metrics to default URL URL15692/metrics

A screenshot of a computer

Description automatically generated with medium confidence

### **Using the Alert Manager to signal Spring Boot metrics**

You can reach Prometheus Alert UI from Prometheus dashboard and see the configured alert

Graphical user interface, text, application, email

Description automatically generated  
  
  
In case our Spring boot application is down alert agent triggers the alert to appropriate route.  
  
Graphical user interface, text, application

Description automatically generated

### **Routing Alerts via email**

Finally, we can configure the Alert Manager to send mails whenever an alert reaches the firing state. To do that, we need to add the below configuration in **alertmanager.yml**

Text

Description automatically generated  
  
Diagram

Description automatically generated  
  
  
  
Another important component is the Alert Manager: that shows the list of alerts that were fired,

Graphical user interface, text, application, email

Description automatically generated

**Spring Boot’s Prometheus Metrics**: http://localhost:8080/actuator/prometheus

**Prometheus Target Endpoints**: <http://localhost:9090/targets>  
**Prometheus Alerts**: <http://localhost:9090/alerts>  
**Grafana server:** [**http://localhost:3000**](http://localhost:3000) **Alert Manager:** [**http://localhost:9093**](http://localhost:9093) **GitHub Repo**: <https://github.com/royalespn/SpringBootPrometheus>  
  
Run the application : mvn spring-boot:run  
Run services : docker-compose up

Text

Description automatically generated

Reference:   
Prometheus: Up & Running by Brian Brazil by by Brian Brazil, ISBN: 9781492034148  
  
Spring Boot Actuautor: <https://docs.spring.io/spring-boot/docs/current/reference/html/actuator.html#actuator.metrics>

**Extra : configure short lived job.**