**Use Prometheus and Grafana to Monitor Spring Boot Applications (For Developers)**

**Prometheus**

Prometheus is an open-source systems monitoring and alerting toolkit, and it collects and stores application metrics as time series data. Application metrics are numeric measurements about health and performance about the application. For example, for a web server it might be request times, for a database it might be number of active connections or number of active queries etc. A Prometheus server can act as a central server that scrapes various application metrics from different web applications. All metrics can be queried by PromQL, the Prometheus query language that retrieves, filters, and aggregates metrics data. Prometheus provides user interface that user can access metrics information and dashboard through a browser. Prometheus also provides client libraries in various languages (Java, Python, Go, Ruby etc.) for instrumenting application code, so developers are able to build custom metrics. There are a large number of exporters developed by the community for exporting existing metrics from third-party systems to be the metrics format that Prometheus understands. Prometheus also utilizes a push gateway for supporting gathering metrics from short-lived jobs, such as cron jobs that are automatically kicked off at certain scheduled time periods. Prometheus has an alert manager component to handle alerts, which can be sent through notification system, such as emails.

The following diagram illustrates the architecture of Prometheus and some of its ecosystem components:

Diagram

Description automatically generated

**Spring Boot Actuator**

Spring Boot Actuator is a sub-project of the Spring Boot Framework. It includes a number of additional features that help to monitor and manage the Spring Boot application. Spring Boot Actuator provides several features like health check-up, request traffic, auditing JVM metrics, log information, caching statics, database state etc. It provides the option to use JMX or HTTP end points to manage and monitor applications on the production environment.

To include Spring Boot Actuator library in Spring Boot application, add the following library dependency in maven pom.xml:

<dependency>

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

<artifactId>spring-boot-starter-actuator</artifactId>

</dependency>

To enable all actuator endpoints, add the following line to application.properties file:

management.endpoints.web.exposure.include=\*

To access actuator through web browser, type the URL:

http://<Host IP>:<port>/<Application Context Path>/actuator

The /actuator endpoint reports all available other actuator endpoints in a JSON format：

Table

Description automatically generated

Further accessing those actuator endpoints will display other system and application information in JSON format.

**Use Prometheus to display Metrics of Spring Boot Application**

To expose metrics in a format a Prometheus server can scrape, add the dependency library (Micrometer Prometheus registry) in the pom.xml of the web application:

<dependency>

<groupId>io.micrometer</groupId>

<artifactId>micrometer-registry-prometheus</artifactId>

</dependency>

After restarting the Spring Boot application, the following URL can be used as an endpoint for Prometheus to pull metrics data:

http://<Host IP>:<port>/<Application Context Path>/actuator/prometheus

The following shows the sample Prometheus metrics after accessing the above endpoint:

Text

Description automatically generated

Download Prometheus server from <https://prometheus.io/download/>. Add a metric endpoint in prometheus.yml (Prometheus configuration file). The entry should contain job\_name, metrics\_path (web application endpoint URL for Prometheus metrics), scrape interval, and targets (web application host ip and port). For example:

Graphical user interface, text, application

Description automatically generated

Use the following command to start Prometheus server:

> prometheus --config.file=prometheus.yml

Access the Prometheus console by entering the url <http://localhost:9090> in a browser. The following shows the Prometheus console:

Graphical user interface, text, application, email

Description automatically generated

Click Status drop down menu then click Targets link, the console should display the metric endpoints of targeted web applications with their current statuses:

Graphical user interface, text, application

Description automatically generated

Examples of Prometheus Dashboard

CPU usage metrics:

Graphical user interface

Description automatically generated with medium confidence

Metric “http\_server\_requests\_seconds\_count” in table format:

Table

Description automatically generated

Metric “http\_server\_requests\_seconds\_count” in graph format:

Chart, line chart

Description automatically generated

Use PromQL (Prometheus Query Language) to query Metrics

The following PromQL is an example for showing the change rate of number of the requests that hits /adduser uri in a 5 minute interval.

rate(http\_server\_requests\_seconds\_count{uri='/adduser'}[5m])

Chart, line chart

Description automatically generated

There are quite a few prebuilt metrics generated by Spring Boot Actuator, but developer can instrument application source code to create custom metrics for the application.

**Micrometer for instrumenting application source code**

Developers can instrument application source code to create custom metrics. Micrometer is a metrics instrumentation library for JVM-based applications. With Spring Boot 2, Micrometer is the default instrumentation library. Micrometer has two concepts to produce metrics, one of them is Meter that’s the interface for different metric types such as a counter, timer, histogram etc. The other one is MeterRegistry that’s responsible for creating and keeping meters.

Metric Types:

Counter

A counter is a cumulative metric that represents a single monotonically increasing counter whose value can only increase or be reset to zero on restart. For example, a counter can represent the number of requests served, tasks completed, or errors.

Gauge

A gauge is a metric that represents a single numerical value that can go up and down.

Gauges are typically used for measured values like temperatures or current memory usage.

Histogram

A histogram samples observations (usually, things like request durations or response sizes) and counts them in configurable buckets. It also provides a sum of all observed values.

Summary

Similar to a histogram, a summary samples observations (usually, things like request durations and response sizes). While it also provides a total count of observations and a sum of all observed values, it calculates configurable quantiles over a sliding time window.

Examples for instrumenting source code:

First of all, MeterRegistry needs to be injected into a controller through constructor injection.

Graphical user interface, text, application

Description automatically generated

The following code demonstrates creating a counter metric (usercontroller\_add\_user\_count\_total) that increments when “addUser” method is called, and a gauge metric (usercontroller\_user\_size) for keeping track of the existing users saved in the database.

Graphical user interface, text, application

Description automatically generated

The following code demonstrates creating a counter metric (usercontroller\_delete\_user\_count\_total) that increments when “deleteUser” method is called.

Graphical user interface, text, application, email

Description automatically generated

After rebuilding and restarting the Spring Boot application, try to add and delete users through the front end:

Graphical user interface, application

Description automatically generated

The counter metrics can capture the number of add-user actions and delete-user actions have taken place, and the gauge metric can show current number of remaining users in the database:

Graphical user interface, text, application, email

Description automatically generated

**Grafana**

Grafana is an open-source analytics platform to query, visualize and alert on application metrics. Grafana pulls metrics from metric database, and it supports many metric databases such as Prometheus, Influxdb, Graphite, and other time series databases. Grafana can be downloaded from <https://grafana.com>. To start Grafana server, click grafana-9.0.4\bin\ grafana-server.exe or run it through a command line prompt. To access Grafana console, open <http://localhost:3000> in a browser. Grafana provides richer and fancier UI and dashboards than Prometheus, and multiple visualized metrics can be grouped in a dashboard, for example:

A screenshot of a computer

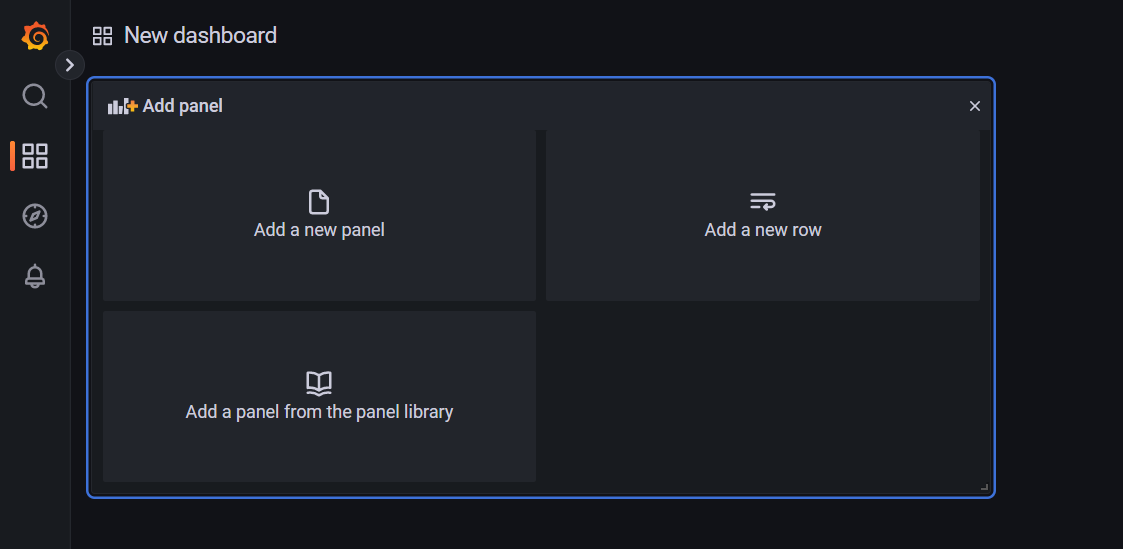
Description automatically generated with medium confidence

To display Prometheus metrics, Grafana needs to add Prometheus as its data source as the following screen shows:

Background pattern

Description automatically generated

Add new dashboard:



When building a dashboard, Prometheus Metrics and PromQL can be specified in query fields:

A screenshot of a computer

Description automatically generated with medium confidence

The following dashboard shows the three custom metrics (usercontroller\_add\_user\_count\_total, usercontroller\_delete\_user\_count\_total, and usercontroller\_user\_size):

A picture containing chart

Description automatically generated

In summary, Prometheus can be used to capture, query, and analyze metrics generated by Spring Boot actuator of a web application, so that application status and performance can be monitored. Micrometer library can be used to instrument application source code to create custom metrics. In addition, Grafana provides fancy dashboard and more advanced visualization for metrics.

Advantages of using Spring Boot Actuator, Prometheus, and Grafana:

1. They are all open-source software and tools, so there is no cost.
2. They become popular in the fast-emerging DevOps software process, so there are strong community supports.
3. Health and performance metrics of multiple web applications can be gathered and visualized in a central place, so that system admins and operational team can easily manage and quickly discover performance issues.
4. Web applications are independent of Prometheus and Grafana, so it means if a web application is down, Prometheus and Grafana are still up and running, and the application “down” status can be quickly captured by Prometheus.