**Monitoring Spring Boot Apps with Micrometer, Prometheus, and Grafana**

In this, we will cover how to monitor Spring Boot web applications. We will be using three projects to achieve this:

Micrometer : Exposes the metrics from our application

Prometheus : Stores our metric data

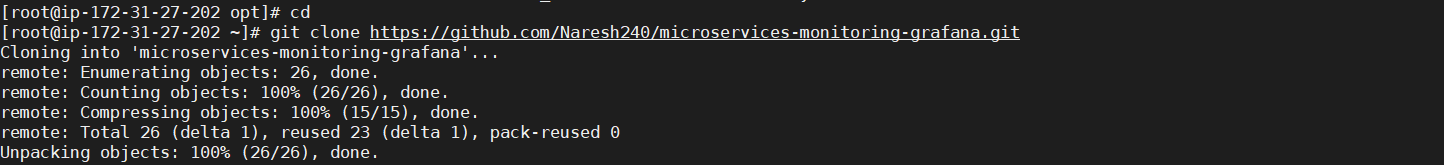
Grafana : Visualizes our data in graphs

Pre-requisites:

* Install Git
* Install Maven
* Install Docker

Clone SCM:

git clone <https://github.com/Naresh240/microservices-monitoring-grafana.git>



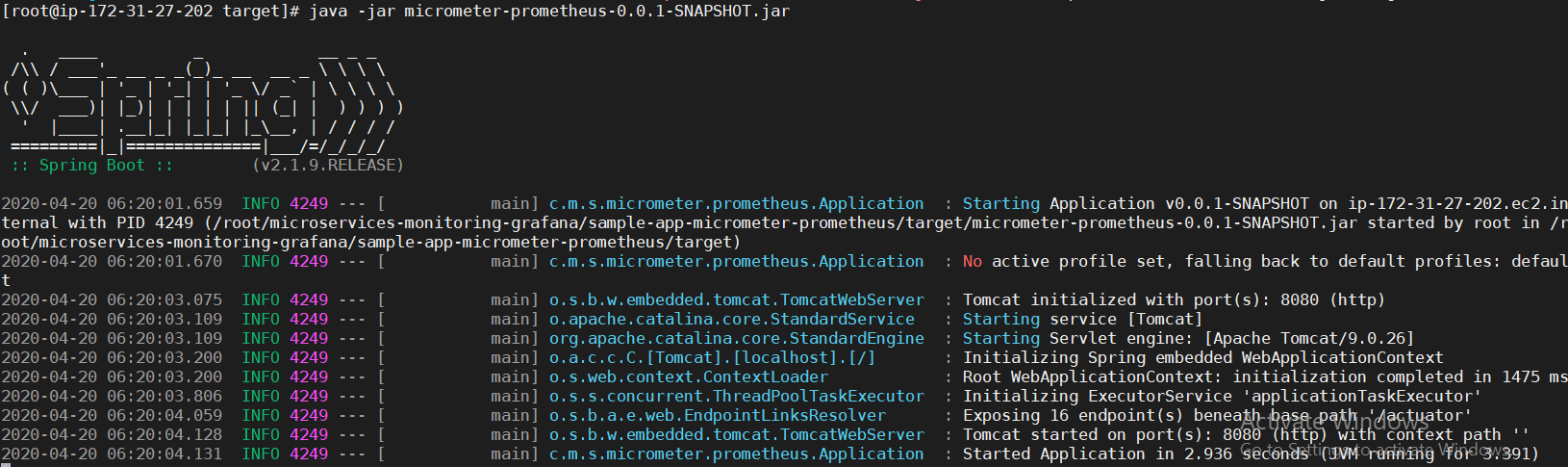
Build Artifacts for Springboot Application:

cd microservices-monitoring-grafana/

mvn clean install

Now run artifact:

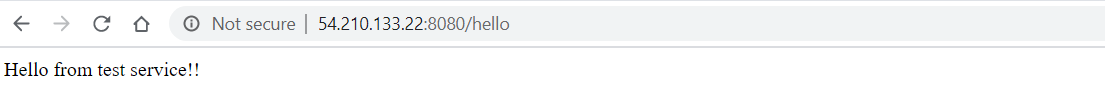
java -jar micrometer-prometheus-0.0.1-SNAPSHOT.jar



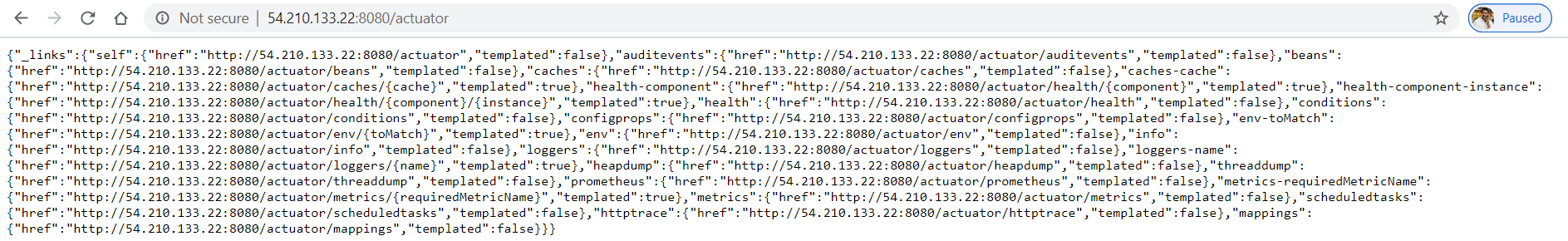
We'll start off with a simple REST service using Spring Initializr that contains a single endpoint of /hello and running on the default port of 8080.

Besides, this application also has the spring-boot-starter-actuator dependency, which provides production-ready endpoints that you can use for your application. These endpoints fall under a common prefix of /actuator and are, by default, protected.

To check, let's navigate our browser to http://<ip-address>:8080/hello

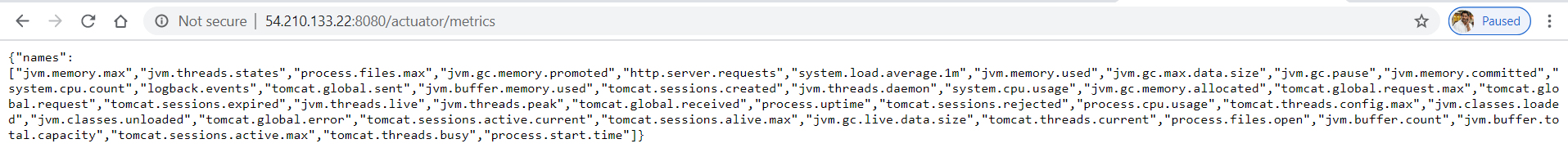


To check, let's navigate our browser to http://<ip-address>:8080/actuator:



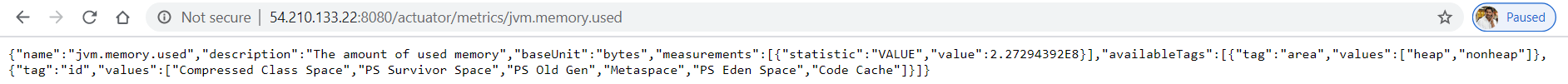
You can see all the endpoints that Actuator exposes such as /health, /metrics, /mappings, etc. Let's open up the /metrics endpoint of the Actuator by navigating our browser:

http://<ip-address>:8080/actuator/metrics:



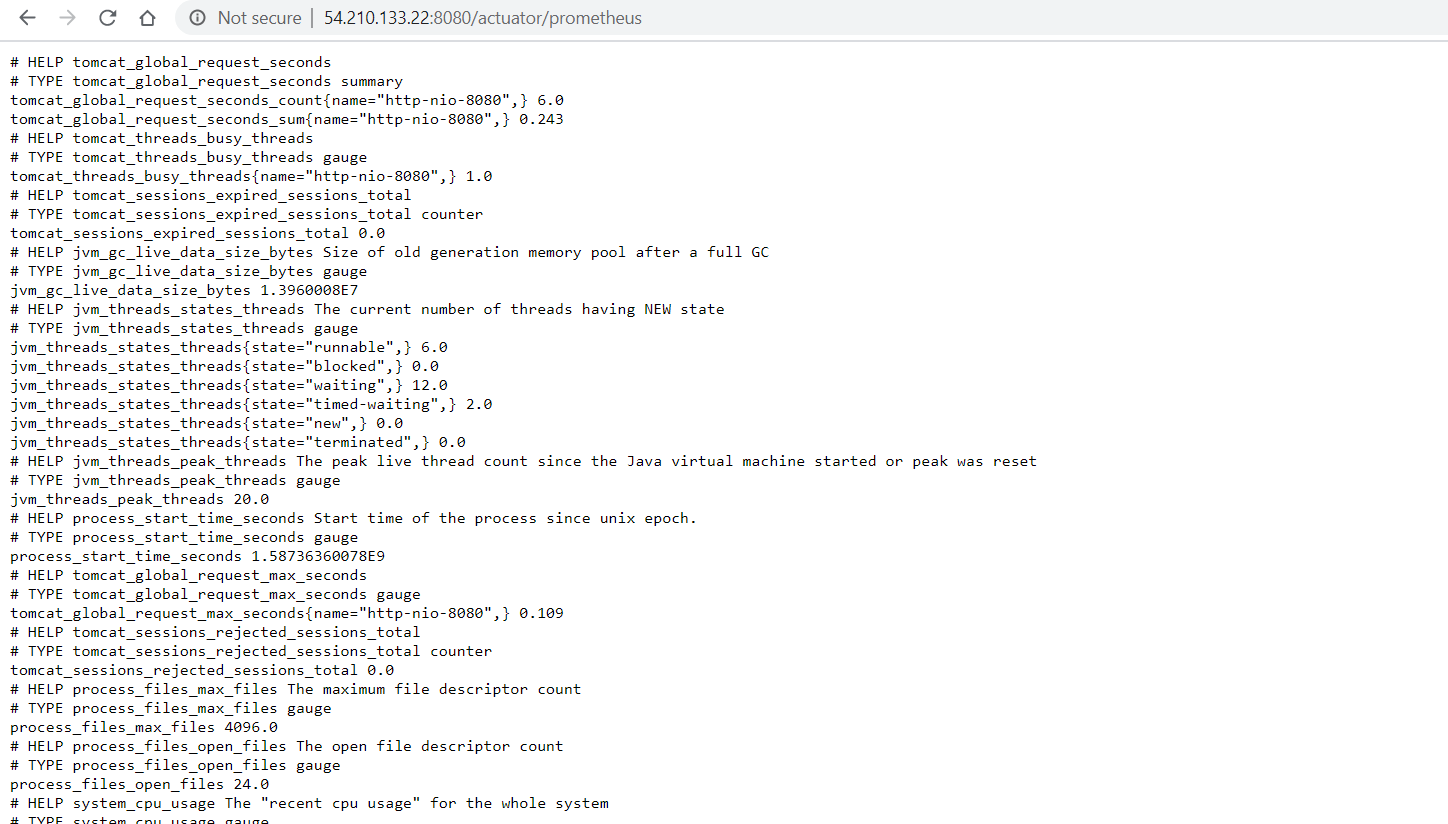
As you can see, there's a bunch of information about our application here, such as information about threads, Tomcat sessions, classes, the buffer, etc. Let's go deeper and retrieve information about the JVM memory used:

http://<ip-address>:8080/actuator/metrics/jvm.memory.used:



you will see data formatted specific for Prometheus:

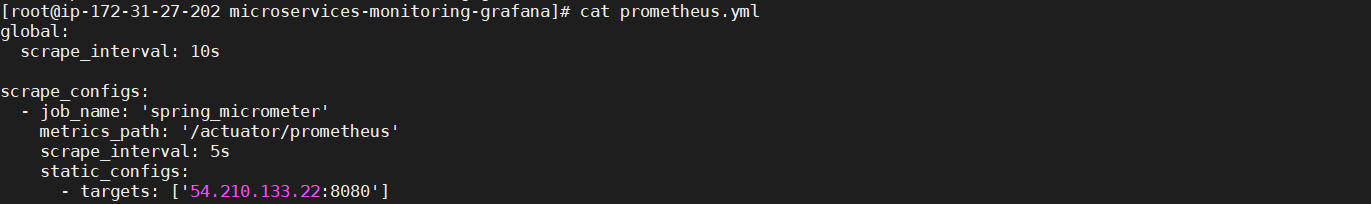
http://<ip-address>:8080/actuator/prometheus



Prometheus:

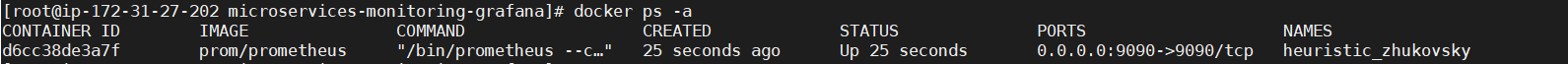
Prometheus is a time-series database that stores our metric data by pulling it (using a built-in data scraper) periodically over HTTP. The intervals between pulls can be configured, of course, and we have to provide the URL to pull from. It also has a simple user interface where we can visualize/query on all of the collected metrics.

Let's configure Prometheus, and more precisely the scrape interval, the targets, etc. To do that, we'll be using the prometheus.yml file:

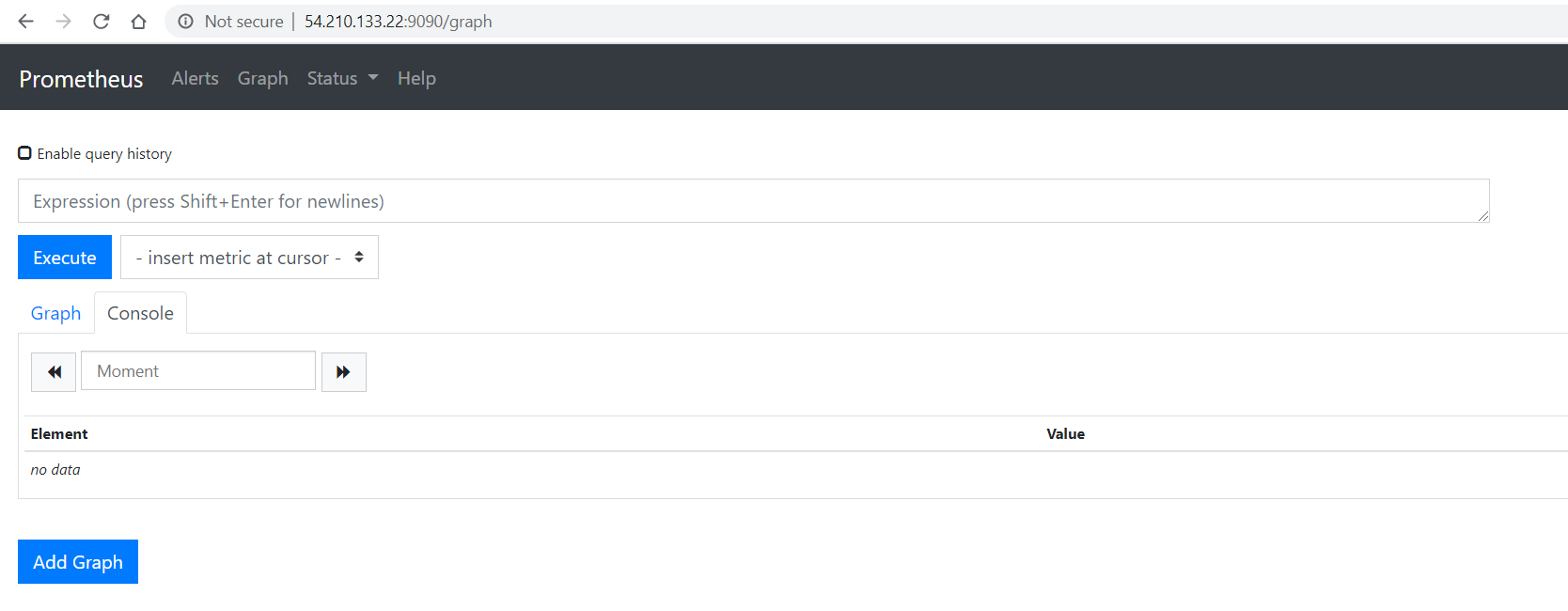


Now, we can run Prometheus using the Docker command:

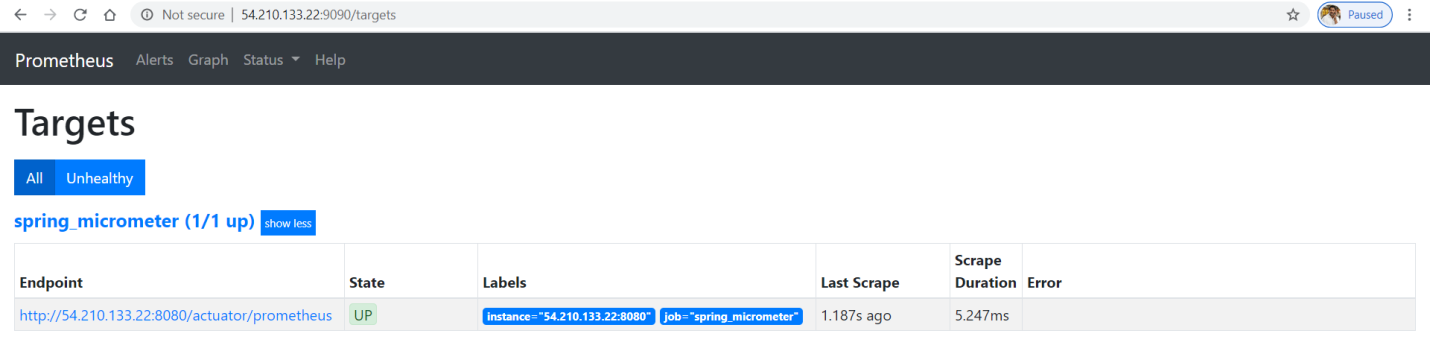
docker run -d -p 9090:9090 -v $PWD/prometheus.yml:/etc/prometheus/prometheus.yml prom/Prometheus



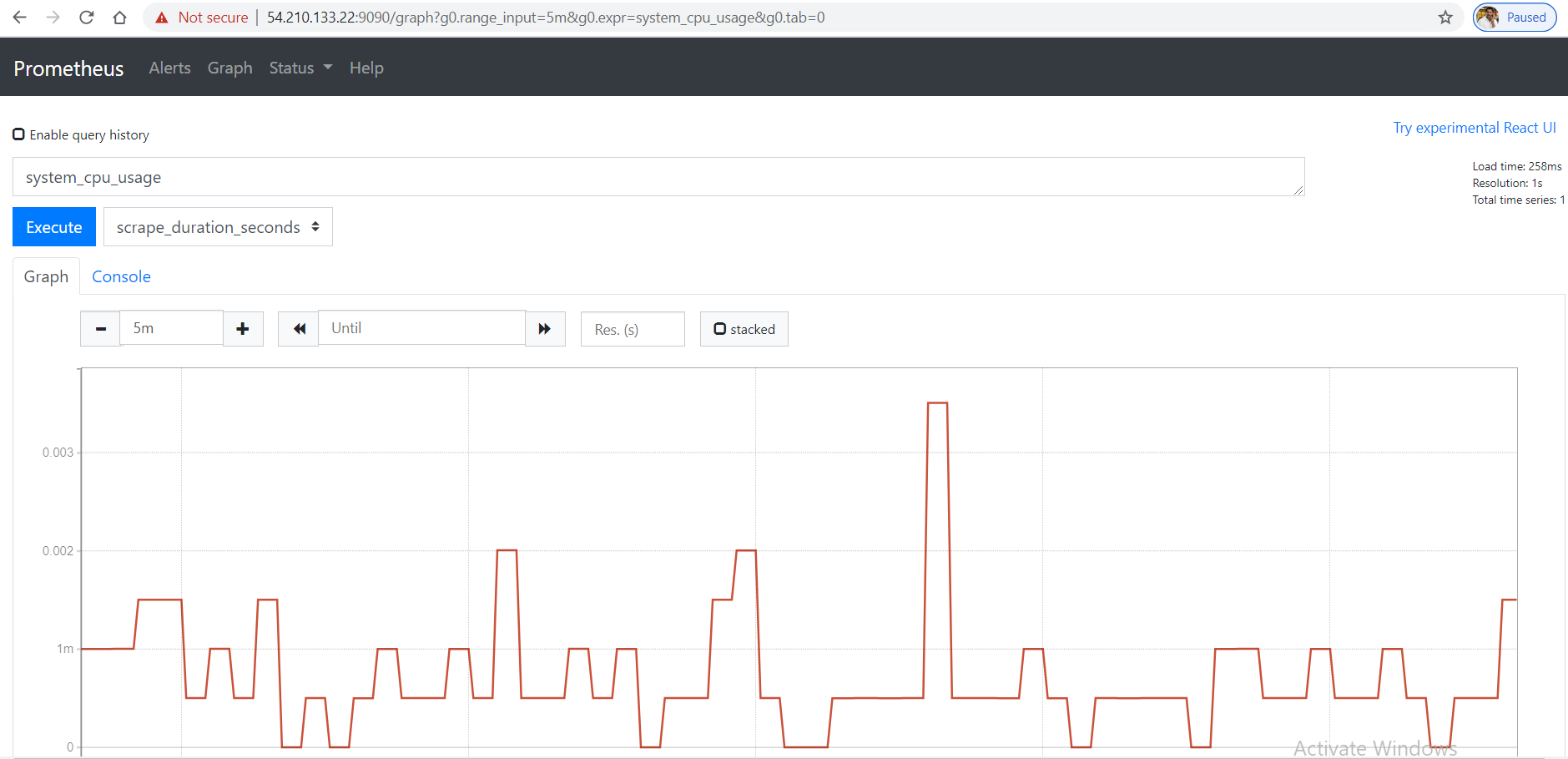
To see Prometheus dashboard, navigate your browser to http://<ip-address>:9090:



To check if Prometheus is actually listening to the Spring app, you can go to the /targets endpoint:



Let's go back to the home page and select a metric from the list and click Execute:



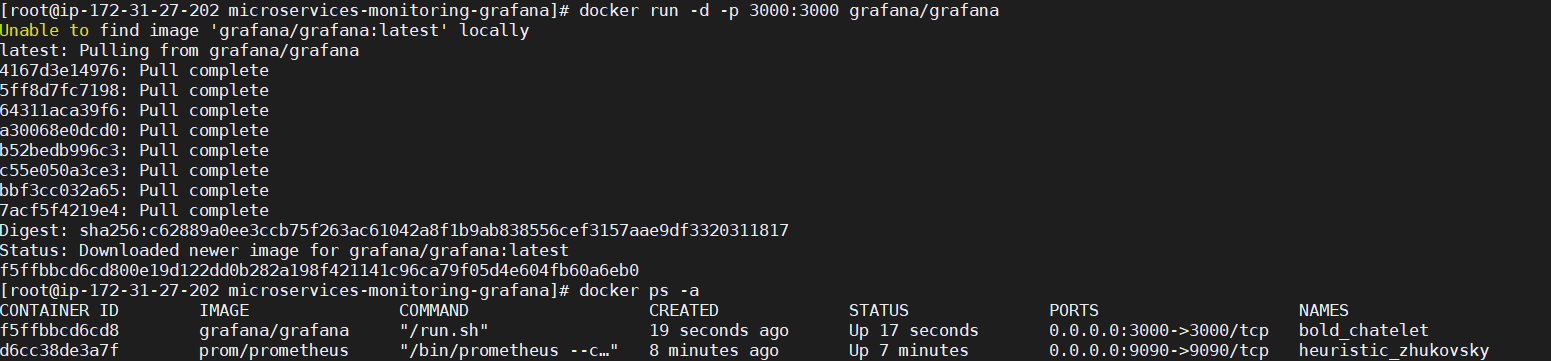
Grafana:

While Prometheus does provide some crude visualization, Grafana offers a rich UI where you can build up custom graphs quickly and create a dashboard out of many graphs in no time. You can also import many community built dashboards for free and get going.

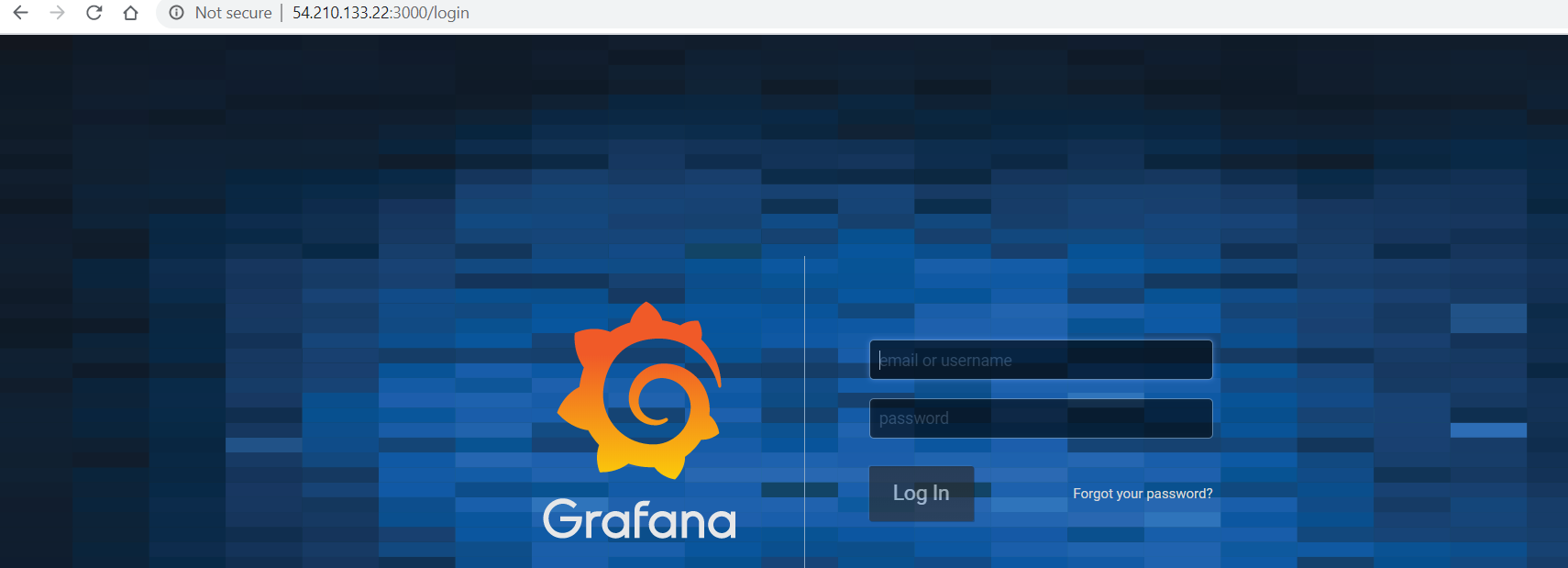
Let's start off by running Grafana using Docker:

docker run -d -p 3000:3000 grafana/Grafana

docker ps -a



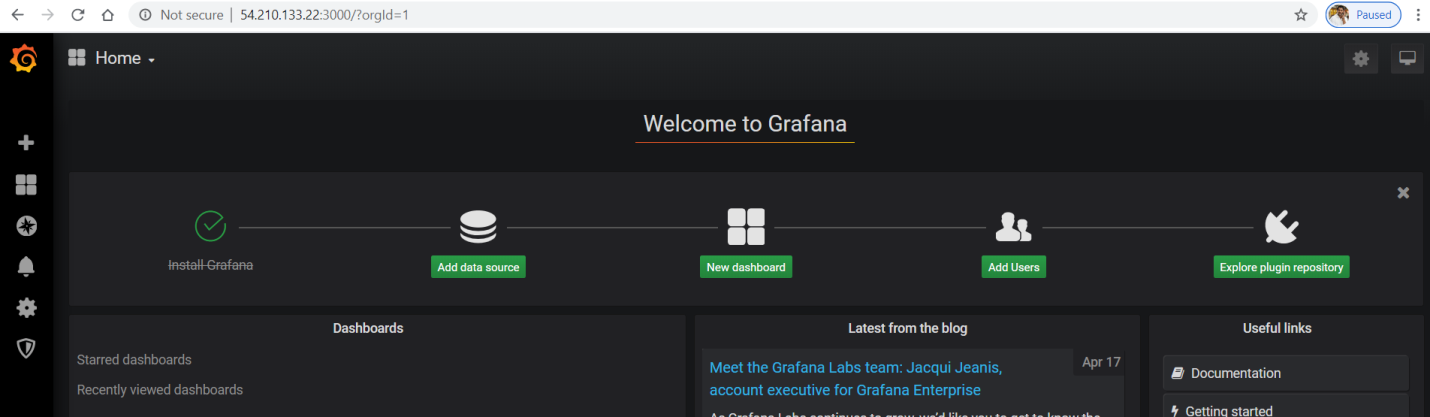
If you visit http://<ip-address>:3000, you will be redirected to a login page:



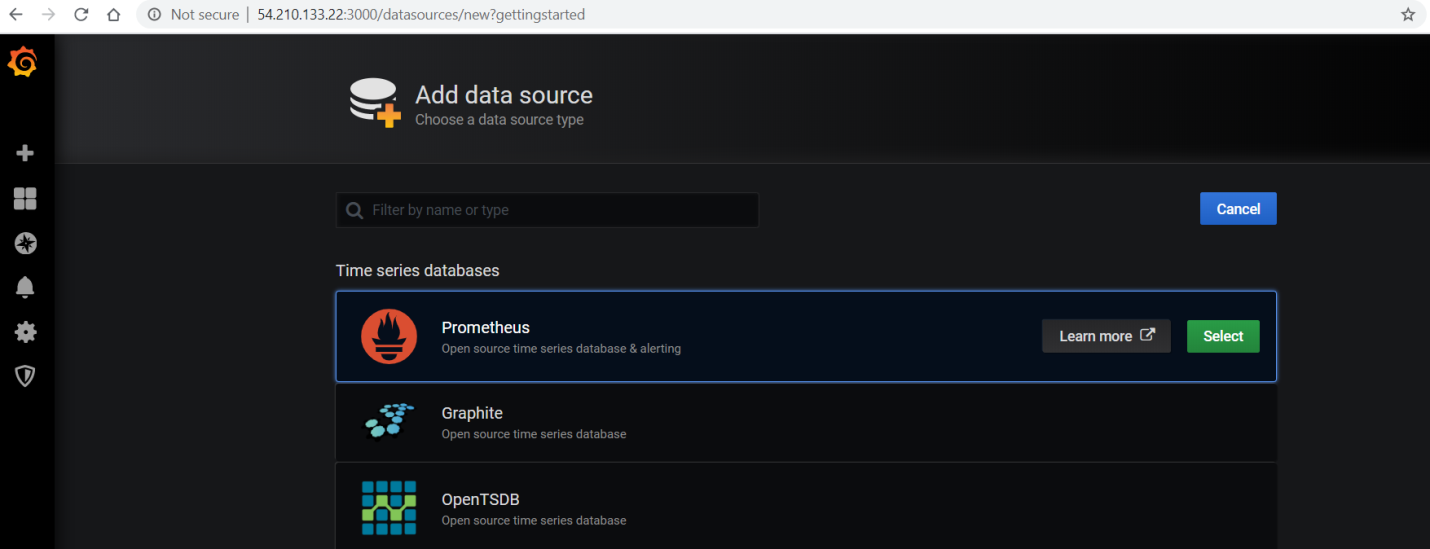
Username : admin

Password : admin

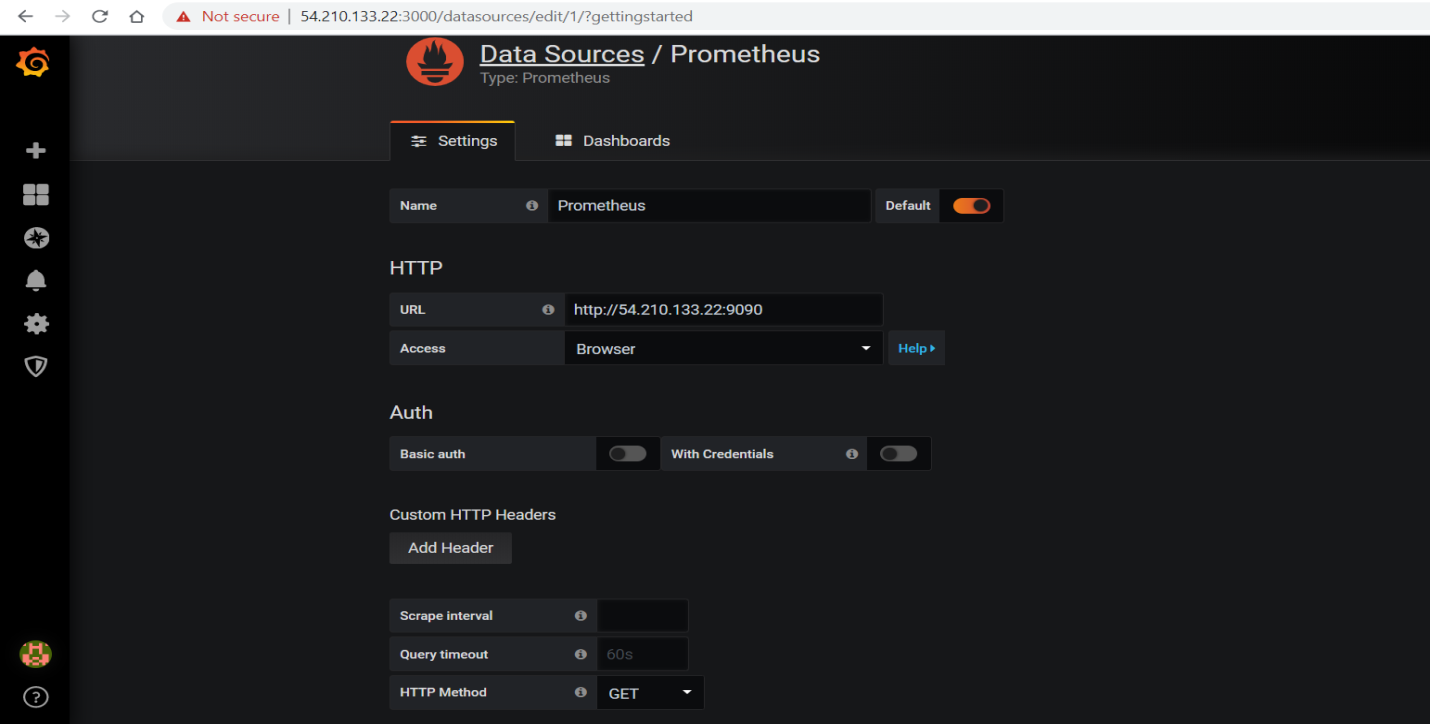
Give above details and Click on **Log In**



Click on Add data source for adding Prometheus:

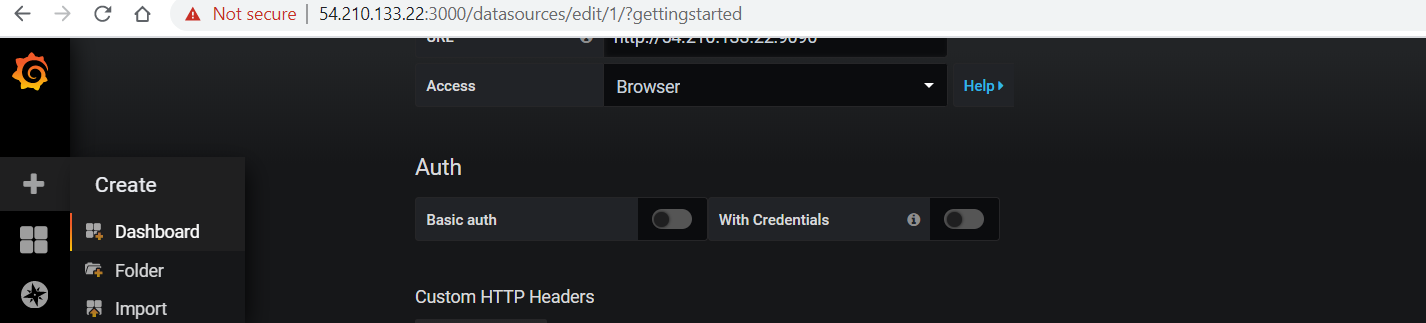


Click on **Select**

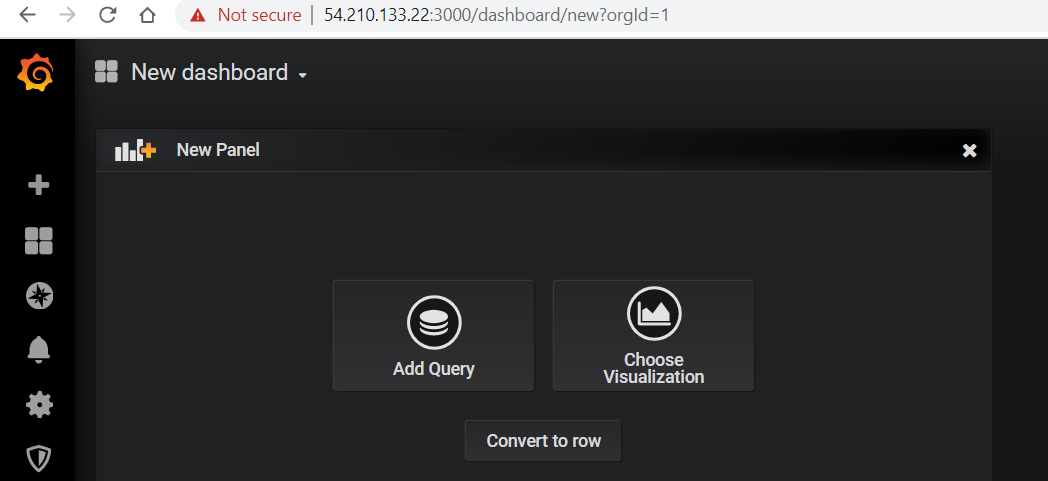


Give details as shown above and and click on **Save & Test**

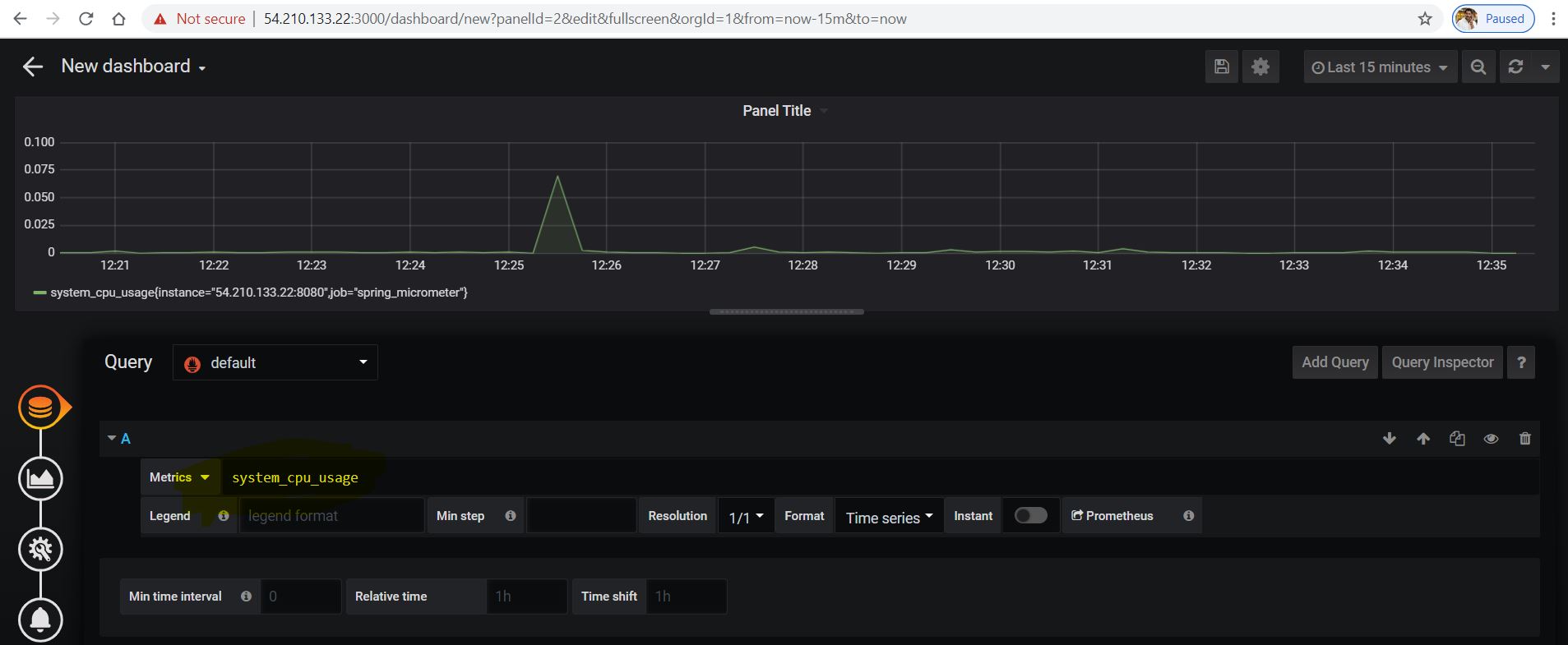
**Create DashBoard:**



Click on ‘**+**’ sign **Dashboard**

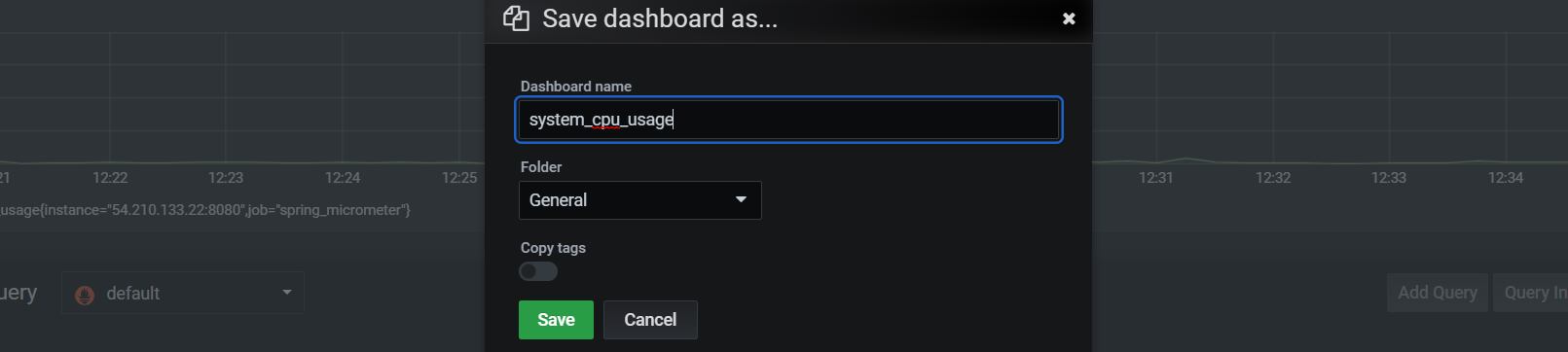


Click on **Add Query**



Here I am going to check **system cpu usage**

Give as shown above and click on Save button



Give a name for Dashboard & Click on **Save**

