**Configure Prometheus and Grafana in Dockers**

P[rometheus](https://prometheus.io/) is an open-source software application used for event monitoring and alerting. It records real-time metrics in a [time series database](https://en.wikipedia.org/wiki/Time_series_database) with flexible queries and real-time alerting. A simple user interface where you can visualize, query, and monitor all the metrics.

G[rafana](https://grafana.com/) is an open-source platform for data visualization, monitoring, and analysis. In Grafana, users can create dashboards with panels, each representing specific metrics over a set time-frame. Grafana supports graph, table, heatmap, and free text panels as well as integration with official and community-built plugins and apps that could be visualized too.

Why Prometheus and Grafana?

*Prometheus dashboard also has simple graphs. But Grafana’s graphs are way better it can be customized according to the user requirements. That’s why, in this post, we’ll integrate Grafana with Prometheus to import and visualize our metrics data.*

**Spring Boot Actuator’s Prometheus endpoint**

As you know Spring Boot has different actuator endpoints. Prometheus also one of the endpoint.

* **/prometheus — returns metrics like the previous one, but formatted to work with a Prometheus server**

The Prometheus endpoint exposes metrics data in a format that can be retrieved by a Prometheus server. You can see the exposed metrics data by navigating to the Prometheus endpoint (<http://localhost:8080/actuator/prometheus>). Here are the details you can see in your browser.

# HELP jvm\_buffer\_memory\_used\_bytes An estimate of the memory that the Java virtual machine is using for this buffer pool  
# TYPE jvm\_buffer\_memory\_used\_bytes gauge  
jvm\_buffer\_memory\_used\_bytes{id="direct",} 81920.0  
jvm\_buffer\_memory\_used\_bytes{id="mapped",} 0.0  
# HELP jvm\_threads\_live The current number of live threads including both daemon and non-daemon threads  
# TYPE jvm\_threads\_live gauge  
jvm\_threads\_live 23.0  
# HELP tomcat\_global\_received\_bytes\_total   
# TYPE tomcat\_global\_received\_bytes\_total counter  
tomcat\_global\_received\_bytes\_total{name="http-nio-8080",} 0.0  
# HELP jvm\_gc\_pause\_seconds Time spent in GC pause  
# TYPE jvm\_gc\_pause\_seconds summary  
jvm\_gc\_pause\_seconds\_count{action="end of minor GC",cause="Allocation Failure",} 7.0  
jvm\_gc\_pause\_seconds\_sum{action="end of minor GC",cause="Allocation Failure",} 0.232  
jvm\_gc\_pause\_seconds\_count{action="end of minor GC",cause="Metadata GC Threshold",} 1.0  
jvm\_gc\_pause\_seconds\_sum{action="end of minor GC",cause="Metadata GC Threshold",} 0.01  
jvm\_gc\_pause\_seconds\_count{action="end of major GC",cause="Metadata GC Threshold",} 1.0  
jvm\_gc\_pause\_seconds\_sum{action="end of major GC",cause="Metadata GC Threshold",} 0.302  
# HELP jvm\_gc\_pause\_seconds\_max Time spent in GC pause  
# TYPE jvm\_gc\_pause\_seconds\_max gauge  
jvm\_gc\_pause\_seconds\_max{action="end of minor GC",cause="Allocation Failure",} 0.0

So how we monitor these metrics in a perfect manner? For that, we are going to use Prometheus docker image and visualize the data in Prometheus graph and Grafana. Let’s begin.

# ****Configure Prometheus in Docker****

1. **Download Prometheus Docker image**

First, download the Prometheus docker image using docker pull command.

$ docker pull prom/prometheus

Once it downloads you can see the prom/prometheus image using $ docker image ls

**2. Configure Prometheus endpoint**

Next, we need to configure the Prometheus to retrieve metrics data from Spring Boot Actuator /prometheus endpoint. For that, we need to create a **prometheus.yml** file and add the following configurations to the .yml file.

# my global config

global:

scrape\_interval: 15s # Set the scrape interval to every 15 seconds. Default is every 1 minute.

evaluation\_interval: 15s # Evaluate rules every 15 seconds. The default is every 1 minute.

# scrape\_timeout is set to the global default (10s).

# Load rules once and periodically evaluate them according to the global 'evaluation\_interval'.

rule\_files:

# - "first\_rules.yml"

# - "second\_rules.yml"

# A scrape configuration containing exactly one endpoint to scrape:

# Here it's Prometheus itself.

scrape\_configs:

# The job name is added as a label `job=<job\_name>` to any timeseries scraped from this config.

- job\_name: 'prometheus'

# metrics\_path defaults to '/metrics'

# scheme defaults to 'http'.

static\_configs:

- targets: ['127.0.0.1:9090']

- job\_name: 'spring-actuator'

metrics\_path: '/actuator/prometheus'

scrape\_interval: 5s

static\_configs:

- targets: ['HOST\_IP:8080']

*spring-actuator job inside scrape\_configs section.*

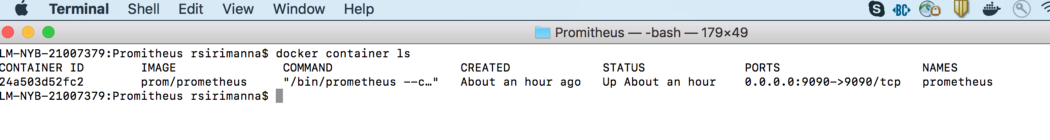
*The metrics\_path is the path of the Actuator’s prometheus endpoint. The targets section and make sure to replace the HOST\_IP with your machine Ip. localhost won’t work here because we’ll be connecting to the HOST machine from the docker container.*

**3. Run the Prometheus docker container in the background**

$ docker run -d --name prometheus -p 9090:9090 -v <PATH\_TO\_prometheus.yml\_FILE>:/etc/prometheus/prometheus.yml prom/prometheus --config.file=/etc/prometheus/prometheus.yml

*Here make sure to replace the <PATH\_TO\_prometheus.yml\_FILE> with the PATH where you have stored the Prometheus configuration file.*

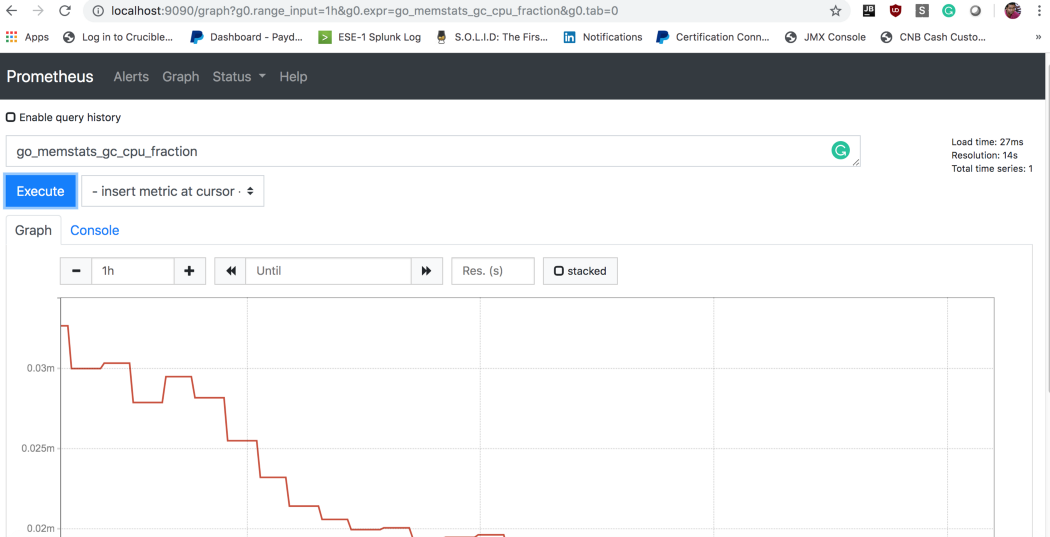
you can see prom/prometheushas been deployed in the container using docker container lscommand.



prom/prometheus Image

**4. Visualizing Spring Boot Metrics from Prometheus dashboard**

That’s it! You can now navigate to [http://localhost:9090](http://localhost:9090/) explore the Prometheus dashboard. Here I have selected ***go\_memstats\_gc\_cpu\_fraction*** metric for demo purpose. You can define any metric here



*go\_memstats\_gc\_cpu\_fraction Prometheus Graph*

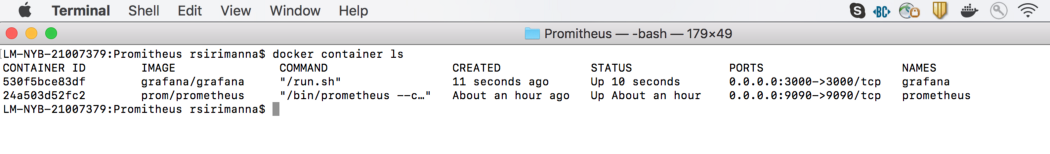
You can check out the official Prometheus documentation to learn more about [Prometheus Query Expressions](https://prometheus.io/docs/introduction/first_steps/#using-the-expression-browser).

**Setup Grafana in Docker**

Now, we are going to integrate Prometheus metrics to Grafana dashboard. To do that, first, pull the Grafana docker image.

$ docker run -d --name grafana -p 3000:3000 grafana/grafana

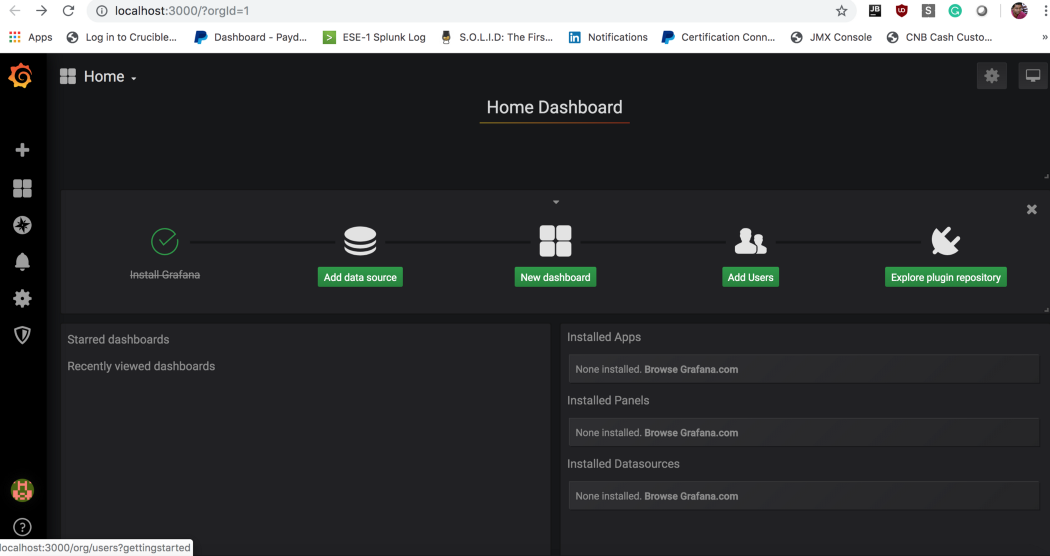
You can type docker container ls to see the list of Docker containers -



Prometheus and Grafana container list

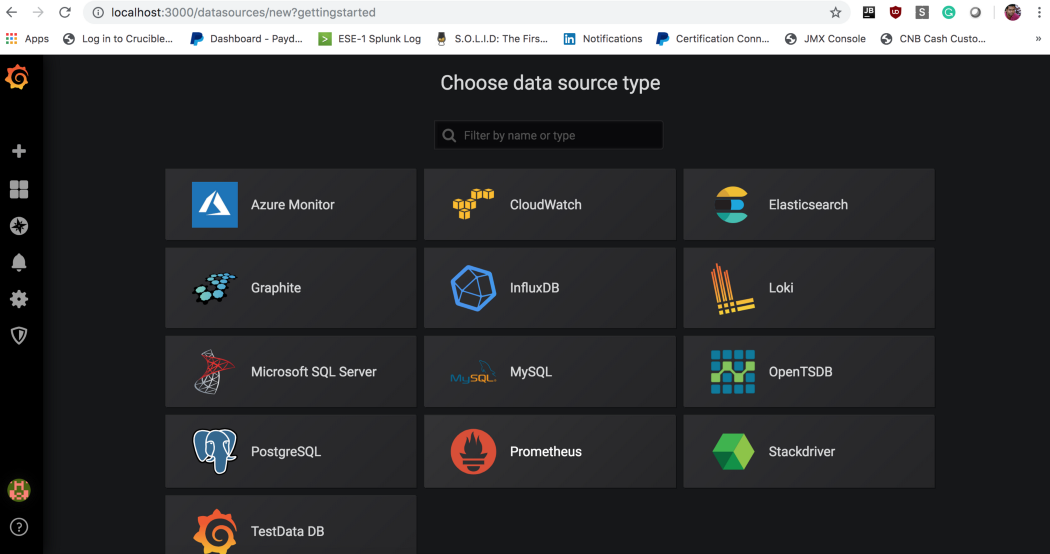
**Integrate Grafana with Prometheus metrics**

You can now navigate to [http://localhost:3000](http://localhost:3000/) and log in to Grafana with the default username admin and password admin. You can see the home page as follows



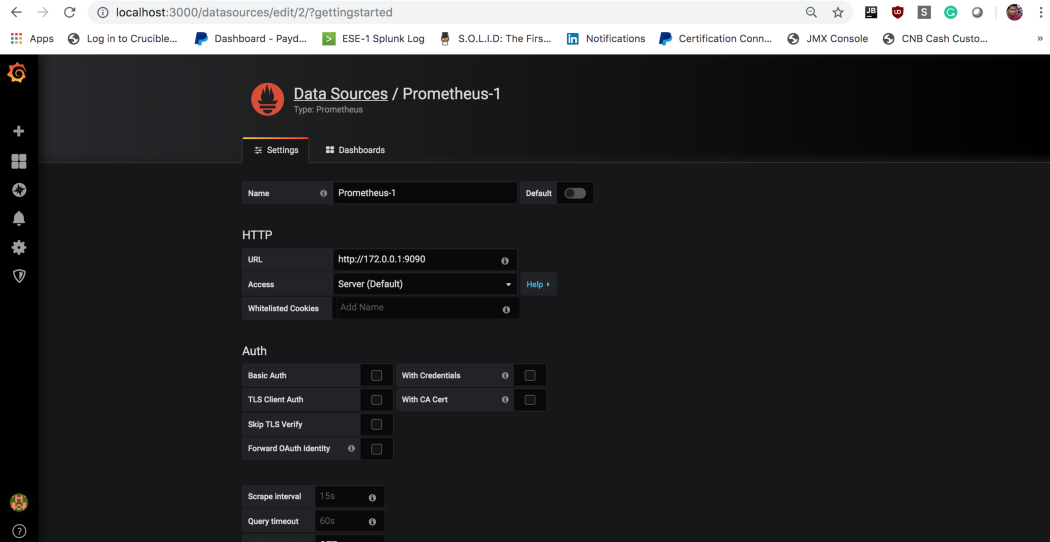
Grafana Home Page

After that click on **Add Data Source** and select **Prometheus**



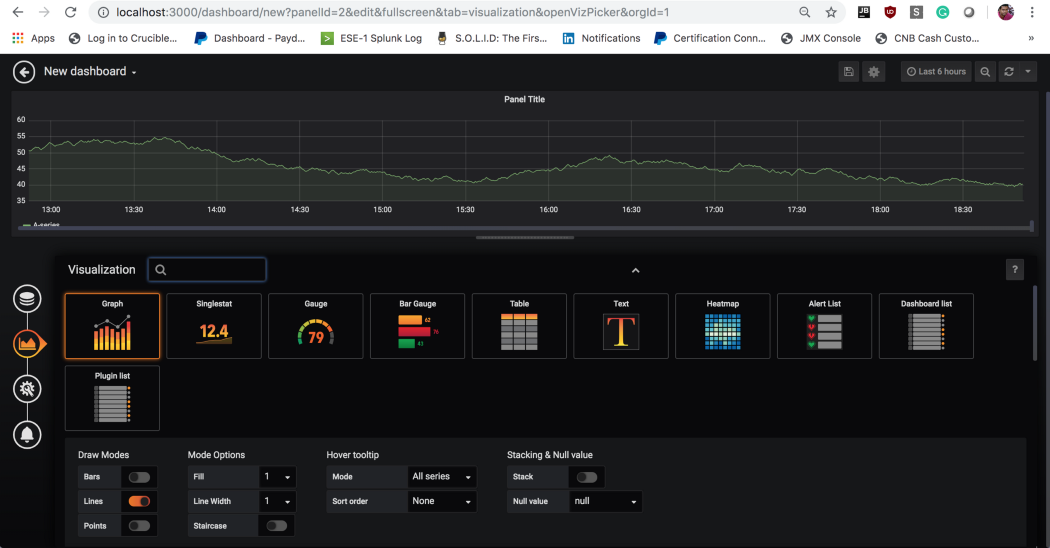
Grafana Data source List

Add HTTP URL as you defined in **prometheus.yml**

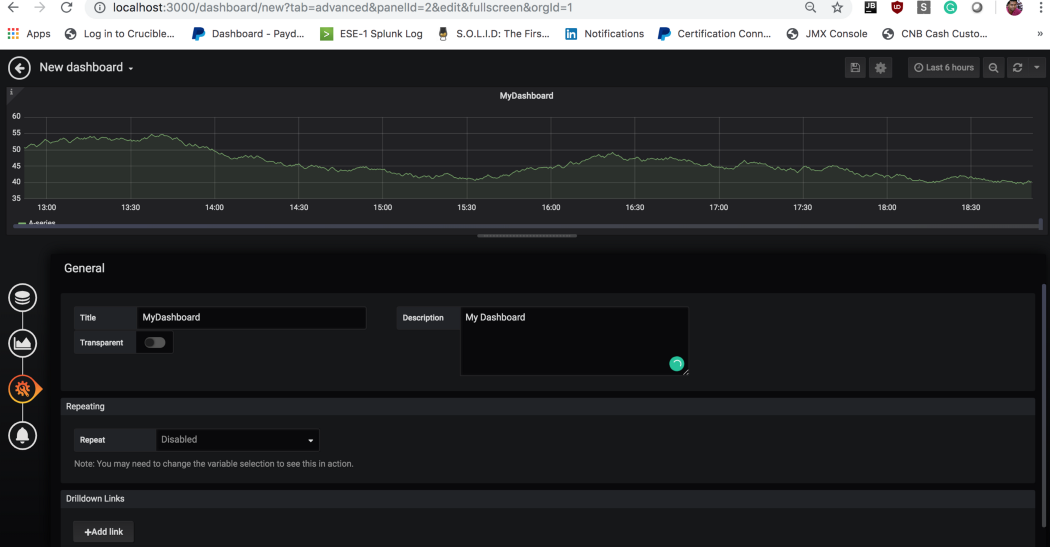


Prometheus data source Configuration- Replace “URL” field with your IP and defined port number

After configuring the data source, Its time to create a dashboard to visualize Prometheus metrics. For that click on **New Dashboard** in home page and click on **Choose Visualization**. Here you can select visualization (Graph, Singlestat, Guage, Table, Text, etc.). I used Graph mode to visualize metrics data.

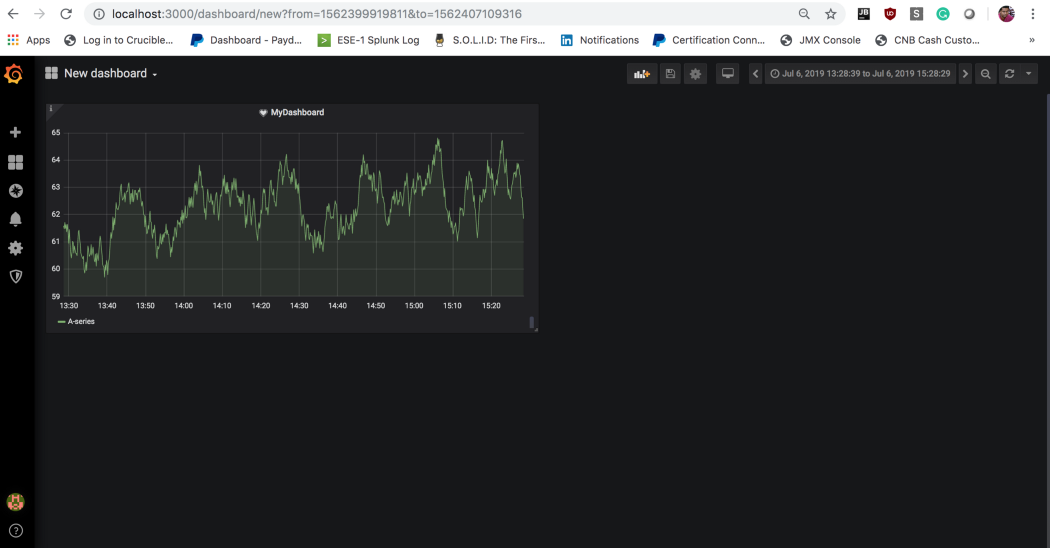


Setup Grafana Dashboard — Select Visualization Type



Setup Dashboard

After that, you have to provide Prometheus Query expression in Grafana’s query editor. For demo purpose, I have given ***go\_memstats\_gc\_cpu\_fraction***visualize metrics in Grafana and finally, you can see the graph as follows.



Prometheus metrics visualized in Grafana

**Conclusion**

Grafana is a powerful tool for visualization and that can be integrated with many data sources like Elasticsearch, AWS Cloudwatch, Azure Monitor, MySQL server, etc. Prometheus endpoint is widely used in Spring Boot to gather application details. This combination would help the developer to monitor application in real-time.