Spring Boot Actuator

Actuator: Production-Ready Features

- Sub project of Spring Boot
- Adds several production grade services to your application with little or no effort
- You can manage and monitor application by using HTTP endpoints or JMX
- Auditing, health and metrics gathering can be automatically applied to your application
- Mainly used to expose different types of information about the running application – health, metrics, info, dump, env etc.

Actuator: Enabling

Add the following dependency to your Spring boot project:

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-actuator</artifactId>
    </dependency>
</dependencies>
```

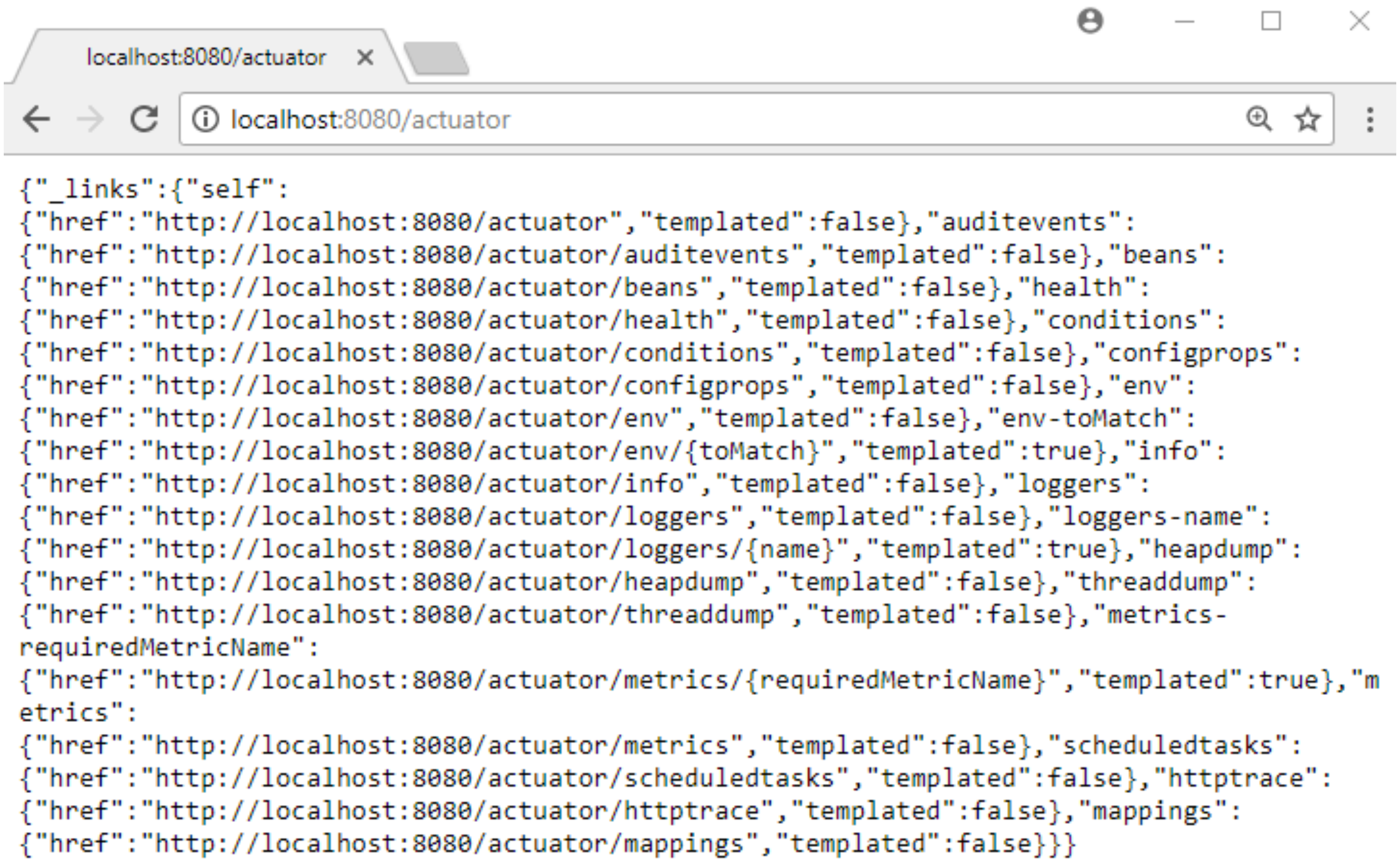
To verify, run your spring boot application and access /actuator.
Assuming 8080 is the port number on which the application is running.

http://localhost:8080/actuator

Actuator: Endpoints

- Actuator creates end points let you monitor application
- End points are exported over HTTP or JMX
- By default all endpoints except shutdown are enabled by default
- Each endpoint can individually be enabled or disabled
- Enabling and disabling of end point controls whether its bean can be created
- End points can be access from remote by exposing endpoints via JMX
- We can enable both HTTP and JMX end points
- By default end points are secured if Spring Security is present

Actuator: Endpoints discovery page


localhost:8080/actuator

localhost:8080/actuator

```
{"_links":{"self":
{"href":"http://localhost:8080/actuator","templated":false},"auditevents":
{"href":"http://localhost:8080/actuator/auditevents","templated":false},"beans":
{"href":"http://localhost:8080/actuator/beans","templated":false},"health":
{"href":"http://localhost:8080/actuator/health","templated":false},"conditions":
{"href":"http://localhost:8080/actuator/conditions","templated":false},"configprops":
{"href":"http://localhost:8080/actuator/configprops","templated":false},"env":
{"href":"http://localhost:8080/actuator/env","templated":false},"env-toMatch":
{"href":"http://localhost:8080/actuator/env/{toMatch}","templated":true},"info":
{"href":"http://localhost:8080/actuator/info","templated":false},"loggers":
{"href":"http://localhost:8080/actuator/loggers","templated":false},"loggers-name":
{"href":"http://localhost:8080/actuator/loggers/{name}","templated":true},"heapdump":
{"href":"http://localhost:8080/actuator/heapdump","templated":false},"threaddump":
{"href":"http://localhost:8080/actuator/threaddump","templated":false},"metrics-
requiredMetricName":
{"href":"http://localhost:8080/actuator/metrics/{requiredMetricName}","templated":true},"m
etrics":
{"href":"http://localhost:8080/actuator/metrics","templated":false},"scheduledtasks":
{"href":"http://localhost:8080/actuator/scheduledtasks","templated":false},"httptrace":
{"href":"http://localhost:8080/actuator/httptrace","templated":false},"mappings":
{"href":"http://localhost:8080/actuator/mappings","templated":false}}}
```

Actuator: Endpoints

End Point	Description
beans	Displays a complete list of all the Spring beans in your application.
caches	Exposes available caches.
conditions	Conditions on configuration and auto-configuration classes
configprops	Displays a collated list of all @ConfigurationProperties.
env	Exposes properties from Spring’s ConfigurableEnvironment.
flyway	Shows any Flyway database migrations that have been applied.
health	Shows application health information.
httptrace	Displays HTTP trace information (last 100 HTTP request-responses)
info	Displays arbitrary application info.
integrationgraph	Shows the Spring Integration graph.
loggers	Shows and modifies the configuration of loggers in the application.
liquibase	Shows any Liquibase database migrations that have been applied.
metrics	Shows ‘metrics’ information for the current application.
mappings	Displays a collated list of all @RequestMapping paths.
scheduledtasks	Displays the scheduled tasks in your application.
sessoins	Manage user sessions from a Spring Session-backed session store
shutdown	Lets the application be gracefully shutdown.
threaddump	Performs a thread dump.
heapdump	Returns an hprof heap dump file.
prometheus	Exposes metrics in a format understandable by a Prometheus server.
logfile	Returns the contents of the logfile

Actuator: Enabling end points

End point can enabled using property
management.endpoint.<id>.enabled = true | false

Example: To enable shutdown end point
management.endpoint.shutdown.enabled = true

You can choose to disable all end points by default and enable each end point with property

management.endpoints.enabled-by-default = false

Example: To disable all end points and enable info end point

```
management.endpoints.enabled-by-default=false
management.endpoint.info.enabled=true
```

Change how end points are exposed using properties

Actuator: Enabling Endpoints

- Change how end points are exposed over HTTP or JMX using properties

```
management.endpoints.jmx.exposure.exclude = <empty means disabled>
management.endpoints.jmx.exposure.include = *
management.endpoints.web.exposure.exclude = <empty means disabled>
management.endpoints.web.exposure.include = info, health
```

- Endpoints automatically cache responses to read operations that do not take any parameters.
To set the cache expiry use property
    management.endpoint.<endpoint>.cache.time-to-live=<duration>
    Example: management.endpoint.beans.cache.time-to-live=10s

- To change the prefix of management end point and port use properties
    management.endpoints.web.base-path = /manage
    management.server.port = 9090
    With the above setting "health" end point will be available at
    http://localhost:9090/manage/health

Actuator: Endpoint - /health

- “/health” endpoint only shows a simple UP or DOWN status
  { status: “UP" }

- To get the complete details including the status of every health indicator add property:
  management.endpoint.health.show-details=always

```
{
    status: “UP",
    details: {
        diskSpace: {
            status: "UP",
            details: {
                total: 250790436864,
                free: 75647844352,
                threshold: 10485760
            }
        }
    }
}
```

- If application has database we can see

  {"status":"UP","details":{"database":"MySQL","hello":1}}

Actuator: Endpoint - /health

- **Spring Boot Actuator comes with several predefined health indicators like**
  - **CassandraHealthIndicator**
  - **CouchbaseHealthIndicator**
  - **DiskSpaceHealthIndicator**
  - **DataSourceHealthIndicator**
  - **ElasticsearchHealthIndicator**
  - **InfluxDbHealthIndicator**
  - **JmsHealthIndicator**
  - **MailHealthIndicator**
  - **MongoHealthIndicator**
  - **Neo4jHealthIndicator**
  - **RabbitHealthIndicator**
  - **RedisHealthIndicator**
  - **SolrHealthIndicator**

Actuator: Endpoint - /health

```
# HEALTH INDICATORS configuration
management.health.defaults.enabled=true # Whether to enable default
health indicators.
management.health.db.enabled=true
management.health.couchbase.enabled=true
management.health.diskspace.enabled=true
management.health.diskspace.path= # Path used to compute the available
disk space.
management.health.diskspace.threshold=10MB # Minimum disk space that
should be available.
management.health.elasticsearch.enabled=true # Whether to enable
Elasticsearch health check.
management.health.elasticsearch.indices= # Comma-separated index names.
management.health.elasticsearch.response-timeout=100ms # Time to wait
for a response from the cluster.
management.health.jms.enabled=true # Whether to enable JMS health
check.
```

Actuator: Endpoint - /info

- Displays arbitrary information about your application
- Obtains build information from META-INF/build-info.properties
- Git information from git.properties file
- Information available in environment properties under the key info

```
info.app.name=@project.name@
info.app.description=@project.description@
info.app.version=@project.version@
info.app.encoding=@project.build.sourceEncoding@
info.app.java.version=@java.version@
```

```
<dependency>
     <groupId>pl.project13.maven</groupId>
     <artifactId>git-commit-id-plugin</artifactId> <version>2.2.5</version>
</dependency>

<plugin>
     <groupId>org.springframework.boot</groupId>
     <artifactId>spring-boot-maven-plugin</artifactId>
     <executions>
          <execution> <goals> <goal>build-info</goal> </goals> </execution>
     </executions>
</plugin>
```

Actuator: Endpoint - /loggers

- Allows you to control your logging configuration at runtime
- We can perform three operations on this endpoint
  - Retrieve the configuration of all loggers (GET /loggers)
  - Retrieve the configuration of a single logger(GET /loggers/{logger.name})
  - Set the log level of a single logger (POST /loggers/{logger.name})
    - curl 'http://localhost:9001/actuator/loggers/com.example' -i -X POST \   -H 'Content-Type: application/json' \   -d '{"configuredLevel":"debug"}'
- Endpoint configuration

```
# Maximum time that a response can be cached
management.endpoint.loggers.cache.time-to-live=0ms
# Whether to enable the loggers endpoint.
management.endpoint.loggers.enabled=true
```

Actuator: Endpoint /metrics

- Provides access to application metrics
- Publishes information about OS, JVM and Application level metrics
- Metrics include memory, heap, processors, threads, classes loaded, classes unloaded, thread pools, HTTP metrics, data source, cache, sessions, etc.
- Retrieving metrics
  - We can retrieve all metrics names
    - curl 'http://localhost:8080/actuator/metrics' -i -X GET
  - We can retrieve the metrics of a named metric
    - curl 'http://localhost:8080/actuator/metrics/jvm.memory.max' -i -X GET

Actuator: Endpoints

- /httptrace
  - Obtain basic information about the last 100 request-response exchanges
- /configprops
  - Response contains details of the application’s @ConfigurationProperties beans
- /flyway
  - Response contains details of the application’s Flyway migrations
- /liquibase
  - Response contains details of the application’s Liquibase change sets.
- /integrationgraph
  - Exposes a graph containing all Spring Integration components
- /logfile
  - The logfile endpoint provides access to the contents of the application’s log file
  - To retrieve part of the log file, use Range header
    - /curl 'http://localhost:8080/actuator/logfile' -i -X GET \   -H 'Range: bytes=0-1023’

Actuator: Endpoints

- /logfile
  - Provides access to the contents of the application's log file
  - You need to set logging.path or logging.file in application.properties
    - logging.path=/work/temp/@project.name@.log
  - To retrieve part of the log file, use Range header
    - /curl 'http://localhost:8080/actuator/logfile' -i -X GET \   -H 'Range: bytes=0-1023'
- /prometheus
  - Provides application's metrics in the format required for scraping by a Prometheus server
- /heapdump
  - Response is binary data in HPROF format
- /threaddump
  - Response contains details of the JVM's threads
- /shutdown
  - shut down the application
    - management.endpoint.shutdown.enabled=true
    - curl 'http://localhost:8080/actuator/shutdown' -i -X POST

Actuator: Custom health information

To provide custom health information we need to write a bean that implement HealthIndicator interface and provide implementation of health() method.

```
@Component
public class MyComponentHealthIndicator implements HealthIndicator {
  @Override
  public Health health() {
    int errorCode = check(); // perform some specific health check
    if (errorCode != 0) {
      return Health.down().withDetail("Error Code", errorCode).build();
    }
    return details(Health.up()).build();
  }

  private int check(){
    // custom health checks
    return 0;
  }

  private Health.Builder details(Health.Builder builder){
    builder.withDetail("someKey", "some value").withDetail("someCounter", 100);
    return builder;
  }
}
```

Actuator: Custom health information

You will see a new health indicator with name myComponent.

```
{   status: "UP",
    details: {
        myComponent: {
            status: "UP",
            details: {
                someKey: "some value",
                someCounter: 100
            }
        },
        db: {
            status: "UP",
            details: {
                database: "MySQL",
                hello: 1
            }
        },
        diskSpace: {
            status: "UP",
            details: {
                total: 250790436864,
                free: 74583785472,
                threshold: 10485760
        }
}
```

Actuator: Prometheus

- **Spring Boot uses Micrometer, an application metrics facade to integrate actuator metrics with external monitoring systems.**
- **Micrometer supports several monitoring systems like Netflix Atlas, AWS Cloudwatch, Datadog, InfluxData, SignalFx, Graphite, Wavefront, Prometheus etc.**
- **Integrate actuator with Prometheus using micrometer-registry-prometheus dependency**

- **Actuator Prometheus providers automatic scrapers for a range of common out-of-the-box technologies in a text-based data format that Prometheus understands.**

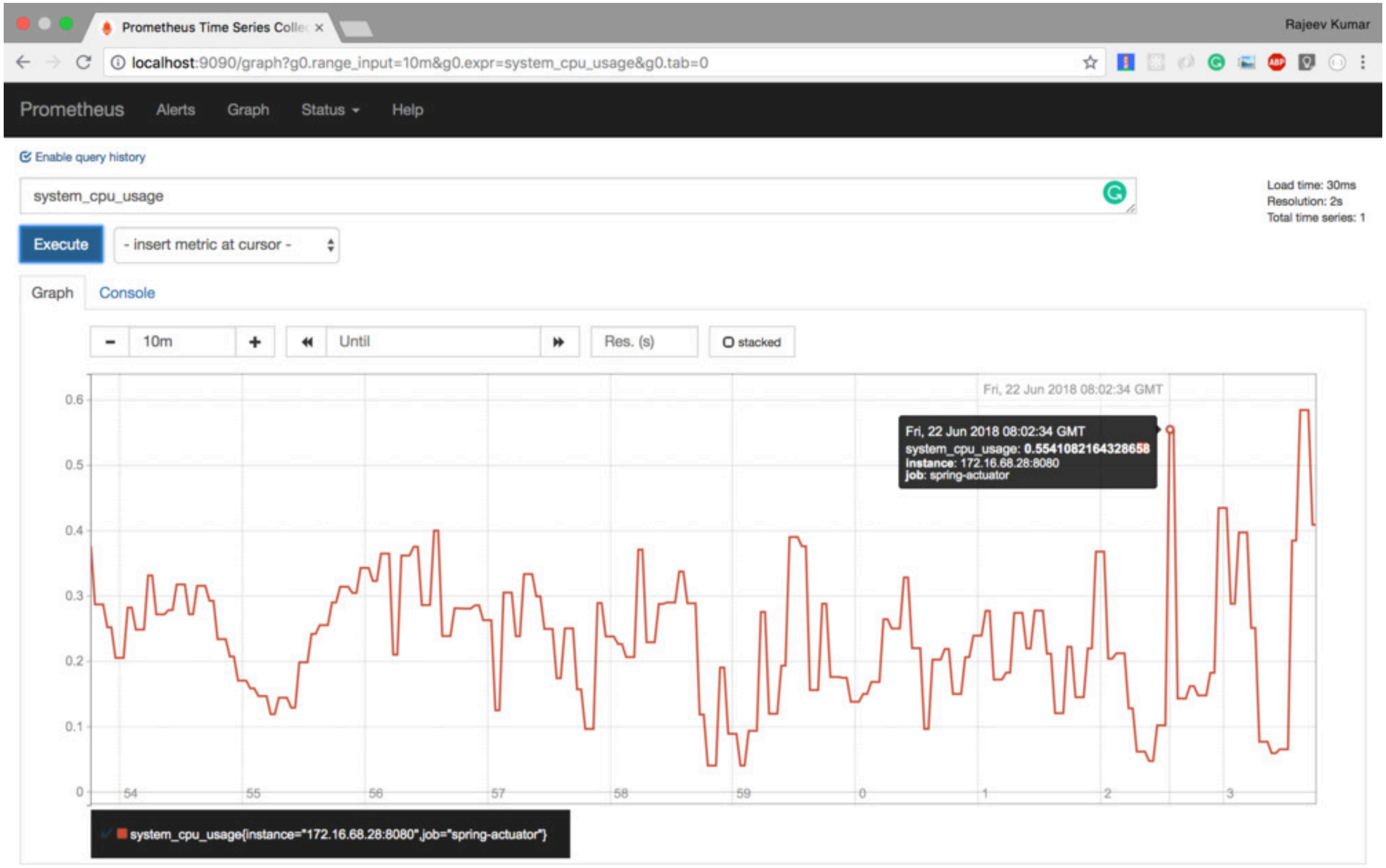
```
<dependency> <!-- Micrometer Prometheus registry  -->
        <groupId>io.micrometer</groupId>
        <artifactId>micrometer-registry-prometheus</artifactId>
</dependency>
```

  - **Spring Boot will automatically configure a PrometheusMeterRegistry and a CollectorRegistry to collect and export metrics data in a format that can be scraped by a Prometheus server.**
  - **Actuator expose /actuator/prometheus endpoint**

Actuator: Prometheus

- Prometheus is a simple, effective open-source monitoring system.
  - It provides a data store, data scrapers, an alerting mechanism and a very simple user interface.
  - A time-series database to store all the metrics data.
  - Prometheus pull metrics from individual machines and services.
  - A simple user interface where you can visualize, query, and monitor all the metrics.

Actuator: Prometheus


Prometheus Time Series Colle
Rajeev Kumar
localhost:9090/graph?g0.range_input=10m&g0.expr=system_cpu_usage&g0.tab=0
Prometheus
Alerts
Graph
Status
Help
Enable query history
Execute
- insert metric at cursor -
Graph
Console
10m
Until
Res. (s)
stacked
Fri, 22 Jun 2018 08:02:34 GMT
0.6
0.5
0.4
0.3
0.2
0.1
0
54
55
56
57
58
59
0
1
2
3

system_cpu_usage

Load time: 30ms
Resolution: 2s
Total time series: 1

Fri, 22 Jun 2018 08:02:34 GMT
system_cpu_usage: 0.5541082164328658
instance: 172.16.68.28:8080
job: spring-actuator

🟥 system_cpu_usage{instance="172.16.68.28:8080",job="spring-actuator"}

Actuator: Prometheus

- Install Prometheus using Docker
  - docker pull prom/prometheus
- Running Prometheus
  - 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

- Sample prometheus.yml

```
 # The job name is added as a label `job=<job_name>` to any timeseries scraped from this config.
# HOST_IP should be replaced with the actual IP of application exposing Prometheus metrics

scrape_configs:
  - job_name: ‘some-job-name’
       metrics_path: '/actuator/prometheus'
       scrape_interval: 5s
       static_configs:
           - targets: ['HOST_IP:8080']
```

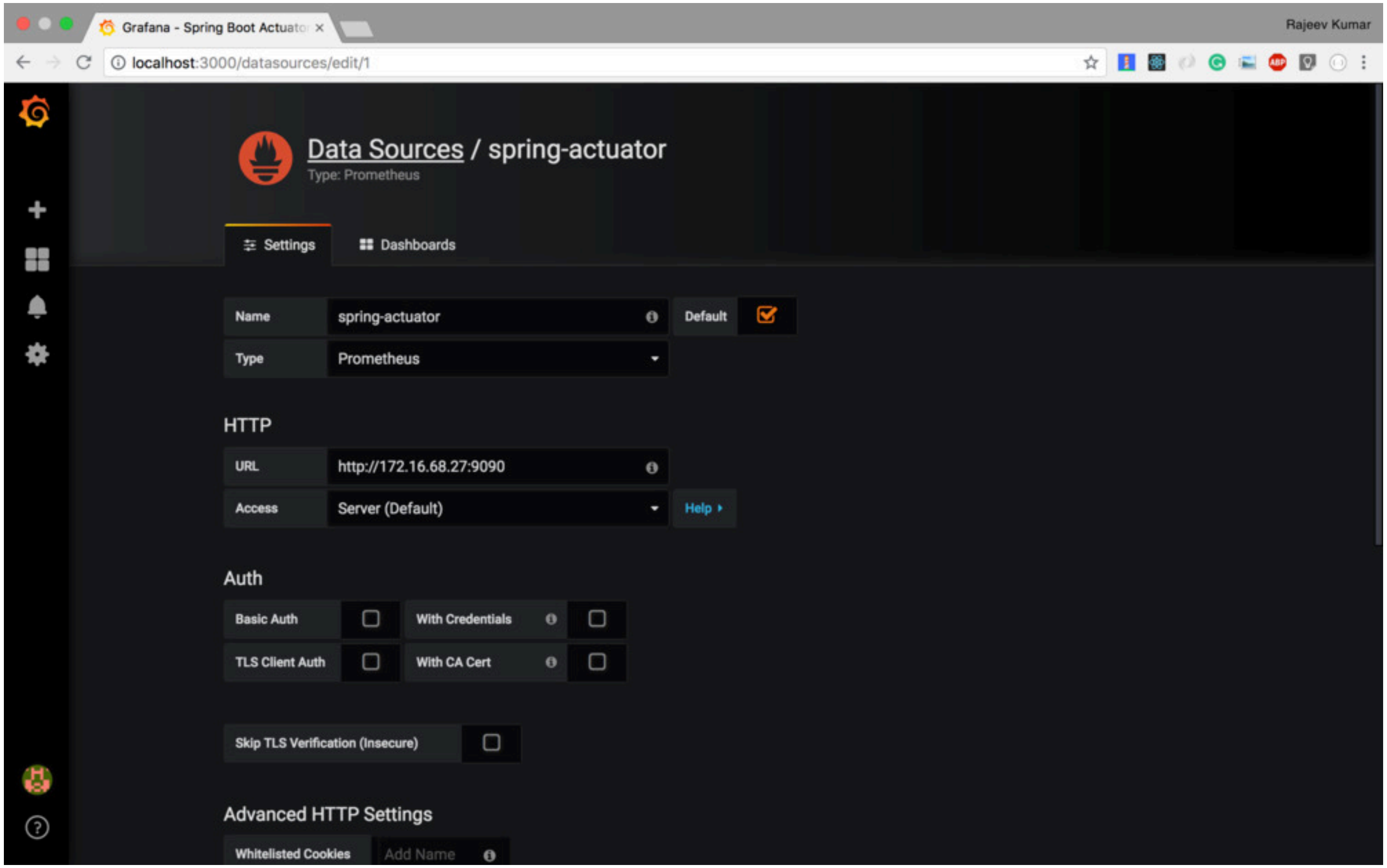
- Prometheus dashboard.
  - http://<ip of server running Prometheus docker>:9090

Actuator: Grafana

- Allows you to bring data from various data sources like Elasticsearch, Prometheus, Graphite, InfluxDB etc, and visualize them with beautiful graphs.
- You set alert rules based on your metrics data. When an alert changes state, it can notify you over email, slack, or various other channels.
- Working with Grafana
    - Install Grafana as Docker container
    - Configure Prometheus datasource for application
    - Import Grafana Spring Boot Dashboard
    - Configure the imported dashboard to use Prometheus datasource created

Actuator: Grafana

- Installing and Running Grafana as Docker container
  - docker run -d --name=grafana -p 3000:3000 grafana/grafana
- Configure Grafana to use Prometheus as data source


Grafana - Spring Boot Actuator
Rajeev Kumar
localhost:3000/datasources/edit/1

Data Sources / spring-actuator
Type: Prometheus

Settings | Dashboards

Name	spring-actuator	Default	✓
Type	Prometheus		

HTTP

URL	http://172.16.68.27:9090	
Access	Server (Default)	Help

Auth

Basic Auth	☐	With Credentials	☐
TLS Client Auth	☐	With CA Cert	☐

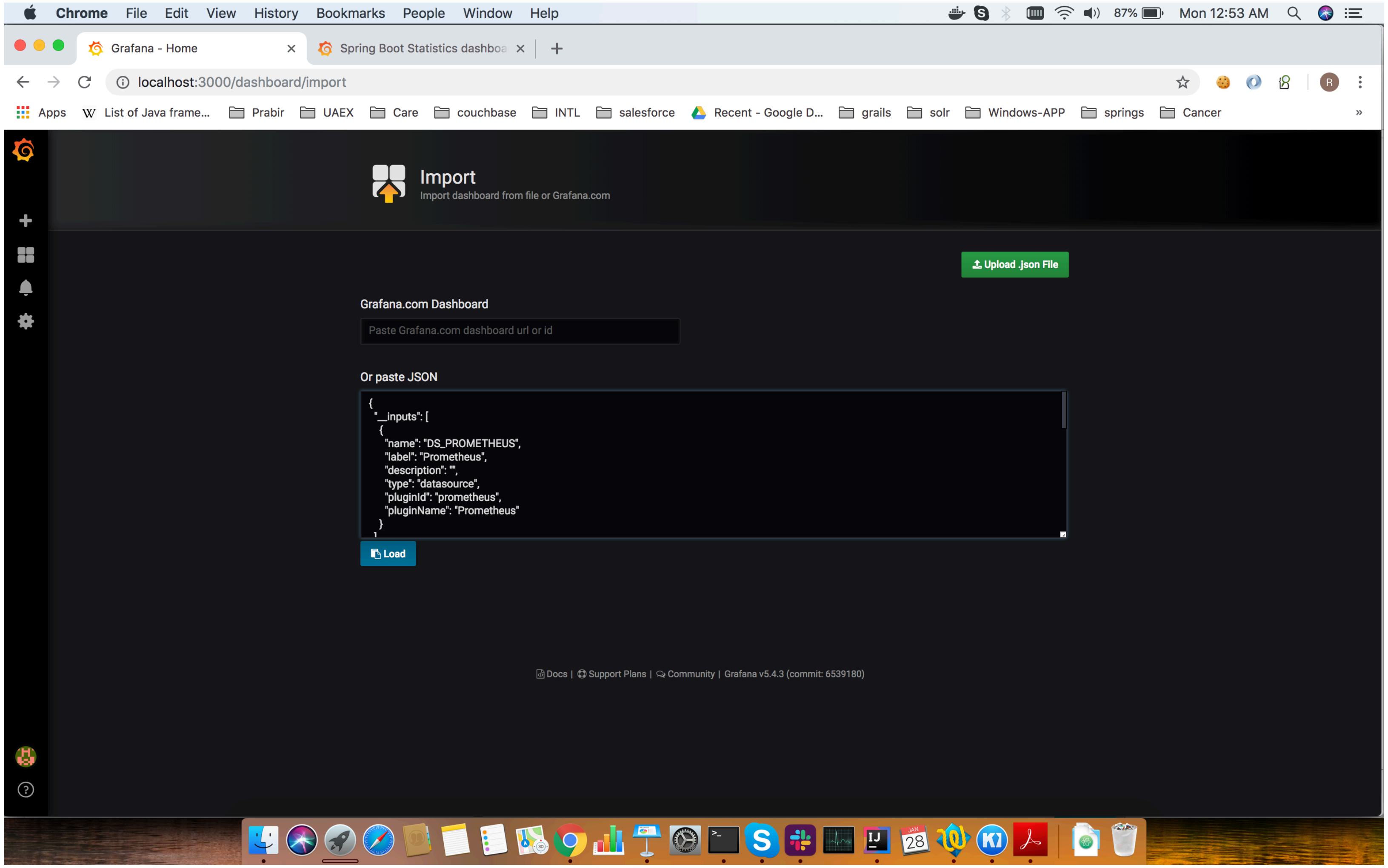
Skip TLS Verification (Insecure) ☐

Advanced HTTP Settings

Whitelisted Cookies: Add Name

Actuator: Grafana

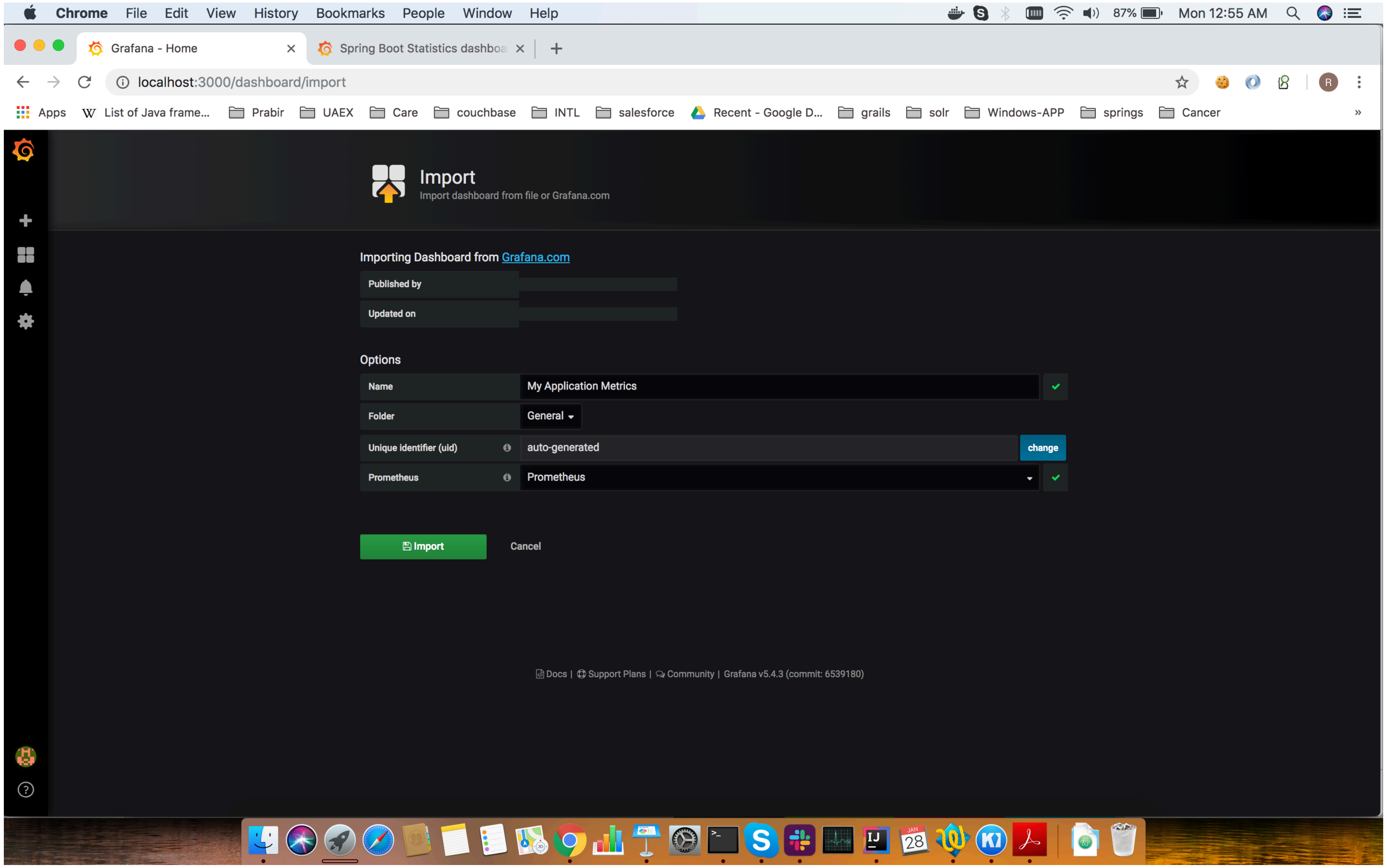
- Download JSON from https://grafana.com/api/dashboards/6756/revisions/2/download
- Got to http://localhost:3000/dashboard/import and paste the downloaded JSON and then Load


Chrome File Edit View History Bookmarks People Window Help
Mon 12:53 AM
Grafana - Home
Spring Boot Statistics dashboa
localhost:3000/dashboard/import
Apps
List of Java frame...
Prabir
UAEX
Care
couchbase
INTL
salesforce
Recent - Google D...
grails
solr
Windows-APP
springs
Cancer
Import
Import dashboard from file or Grafana.com
Upload .json File
Grafana.com Dashboard
Paste Grafana.com dashboard url or id
Or paste JSON
Load
Docs | Support Plans | Community | Grafana v5.4.3 (commit: 6539180)

```
{
  "__inputs": [
    {
      "name": "DS_PROMETHEUS",
      "label": "Prometheus",
      "description": "",
      "type": "datasource",
      "pluginId": "prometheus",
      "pluginName": "Prometheus"
    }
```

Actuator: Grafana

- **Configure the Dashboard name, Prometheus datasource to use and import.**



Import
Import dashboard from file or Grafana.com

Importing Dashboard from Grafana.com

Published by	
Updated on	

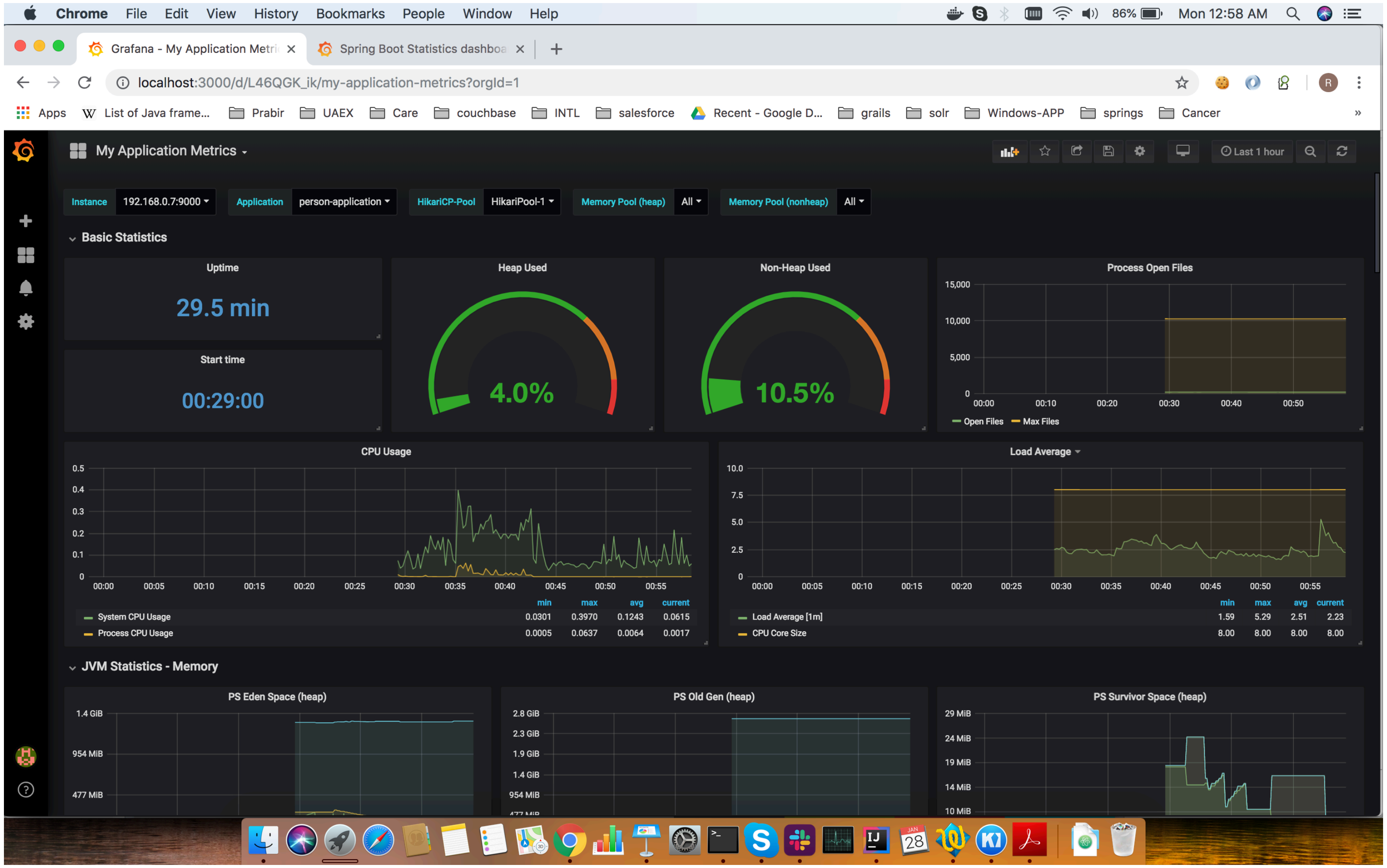
Options

Name	My Application Metrics
Folder	General
Unique identifier (uid)	auto-generated
Prometheus	Prometheus

Grafana - Home
Spring Boot Statistics dashboa
localhost:3000/dashboard/import
change
Import
Cancel
Docs | Support Plans | Community | Grafana v5.4.3 (commit: 6539180)

Actuator: Grafana

- You will directed to "My Application Metrics" dashboard



	min	max	avg	current
System CPU Usage	0.0301	0.3970	0.1243	0.0615
Process CPU Usage	0.0005	0.0637	0.0064	0.0017

	min	max	avg	current
Load Average [1m]	1.59	5.29	2.51	2.23
CPU Core Size	8.00	8.00	8.00	8.00

Actuator: Additional Information

- To see all possible spring boot actuator configuration; “ACTUATOR PROPERTIES”

https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#appendix

- To see request-response structure of actuator endpoints

https://docs.spring.io/spring-boot/docs/current/actuator-api/html/

- Git repo of Sample Actuator Project

https://gitlab.com/nbostech/springboot-codinglabs/tree/develop/actuator/spring-boot-actuator-example

- Git repo of Sample project with Actuator and Prometheus and Grafana as Docker

Follow instructions

https://gitlab.com/nbostech/springboot-codinglabs/blob/develop/actuator/person-application/Readme.md

- Prometheus https://prometheus.io/
- Grafana Dashboards https://grafana.com/dashboards