

Lab 13

Part A: Logging

Modify the Bank application so that logging is done properly to a log file instead of `System.out.println()`.

Apply the different levels of logging for the bank and check that these levels work correctly.

Part B: Actuators

Add the actuators dependency to the Bank application

Call and study the output of the following actuators:

`/health`

`/env`

`/beans`

`/configprops`

`/mappings`

`/scheduledtask`

Also shut the application down using the actuator.

Part C: Prometheus and Grifana

Write a simple spring boot web application with the following REST controller:

```
@RestController
public class GreetingController {

    @RequestMapping(value="/greeting")
    public String greeting() {
        return "Hello World";
    }
}
```

Run the application and check if it works.

Then add the following dependency in POM.xml.

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

Reload the maven POM dependencies

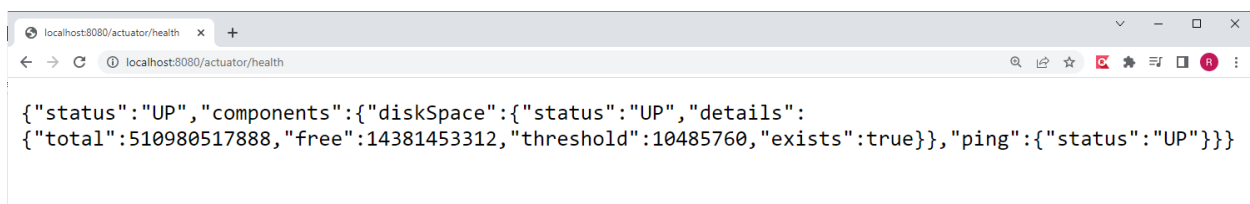
Then add the following configuration in application.properties:

```
management.endpoints.web.exposure.include=*
management.endpoint.health.show-details=always
```

Rerun the application again and check if you can call some endpoints.



```
{ "_links": { "self": { "href": "http://localhost:8080/actuator", "templated": false }, "beans": { "href": "http://localhost:8080/actuator/beans", "templated": false }, "caches": { "href": "http://localhost:8080/actuator/caches", "templated": false }, "caches-cache": { "href": "http://localhost:8080/actuator/caches/{cache}", "templated": true }, "health": { "href": "http://localhost:8080/actuator/health", "templated": false }, "health-path": { "href": "http://localhost:8080/actuator/health/{*path}", "templated": true }, "info": { "href": "http://localhost:8080/actuator/info", "templated": false }, "conditions": { "href": "http://localhost:8080/actuator/conditions", "templated": false }, "configprops": { "href": "http://localhost:8080/actuator/configprops", "templated": false }, "configprops-prefix": { "href": "http://localhost:8080/actuator/configprops/{prefix}", "templated": true }, "env-toMatch": { "href": "http://localhost:8080/actuator/env/{toMatch}", "templated": true }, "env": { "href": "http://localhost:8080/actuator/env", "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 }, "metrics": { "href": "http://localhost:8080/actuator/metrics", "templated": false }, "scheduledtasks": { "href": "http://localhost:8080/actuator/scheduledtasks", "templated": false }, "mappings": { "href": "http://localhost:8080/actuator/mappings", "templated": false } }
```



```
{ "status": "UP", "components": { "diskSpace": { "status": "UP", "details": { "total": 510980517888, "free": 14381453312, "threshold": 10485760, "exists": true } }, "ping": { "status": "UP" } } }
```

Now add the following micrometer-prometheus dependency in the POM file.

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

Reload the maven POM dependencies

Rerun the application again and check if you can call the Prometheus actuator at

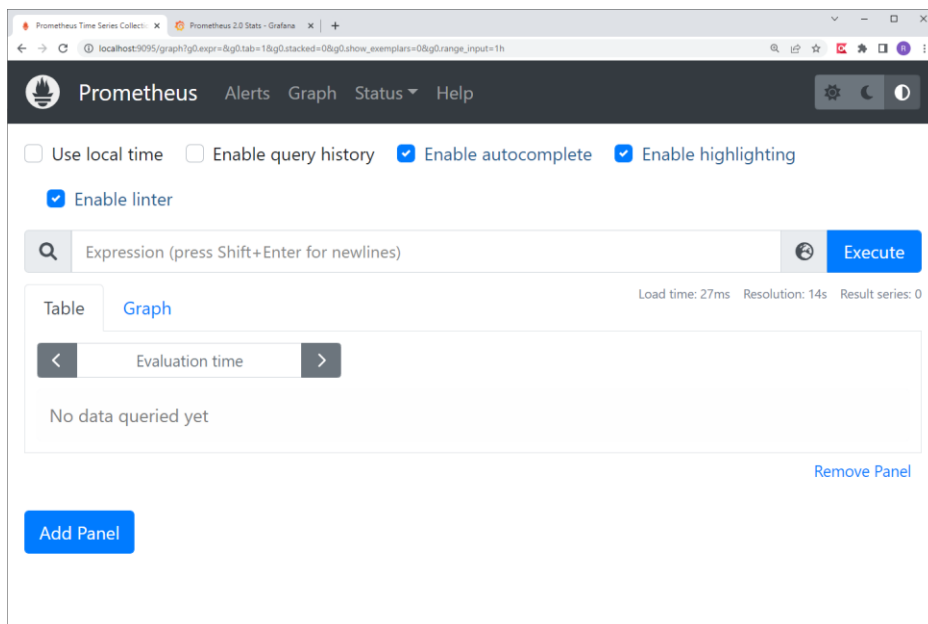
<http://localhost:8080/actuator/prometheus>

```
localhost8080/actuator/prometheus
localhost8080/actuator/prometheus

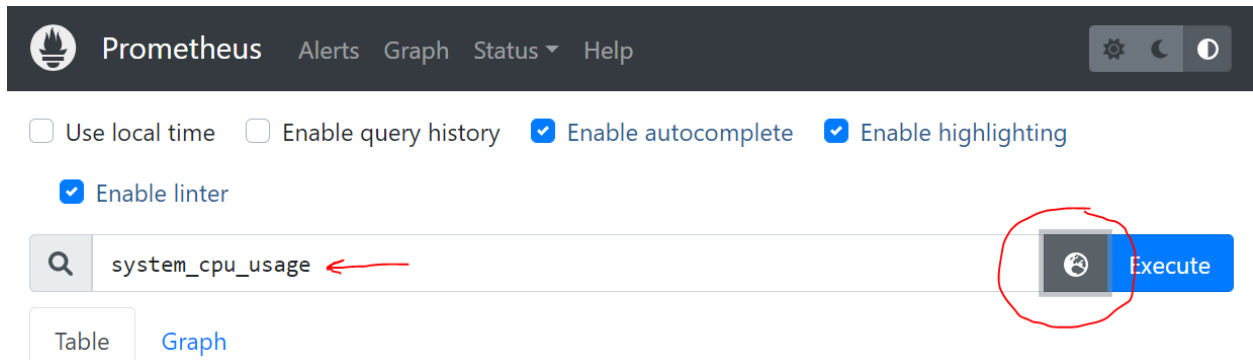
# HELP jvm_classes_unloaded_classes_total The total number of classes unloaded since the Java virtual
machine has started execution
# TYPE jvm_classes_unloaded_classes_total counter
jvm_classes_unloaded_classes_total 0.0
# HELP tomcat_sessions_created_sessions_total
# TYPE tomcat_sessions_created_sessions_total counter
tomcat_sessions_created_sessions_total 0.0
# HELP application_ready_time_seconds Time taken (ms) for the application to be ready to service
requests
# TYPE application_ready_time_seconds gauge
application_ready_time_seconds{main_application_class="mvc.SpringBootMVCAApplication",} 3.577
# HELP jvm_gc_memory_allocated_bytes_total Incremented for an increase in the size of the (young)
heap memory pool after one GC to before the next
# TYPE jvm_gc_memory_allocated_bytes_total counter
jvm_gc_memory_allocated_bytes_total 6.1865984E7
# HELP process_uptime_seconds The uptime of the Java virtual machine
# TYPE process_uptime_seconds gauge
process_uptime_seconds 1.0
```

Then double click the file **C: \CS544\Tools\prometheus \startPrometheus.bat**. You can find a shortcut of this file in the Desktop inside of the folder called “**Tool Shortcuts**”.

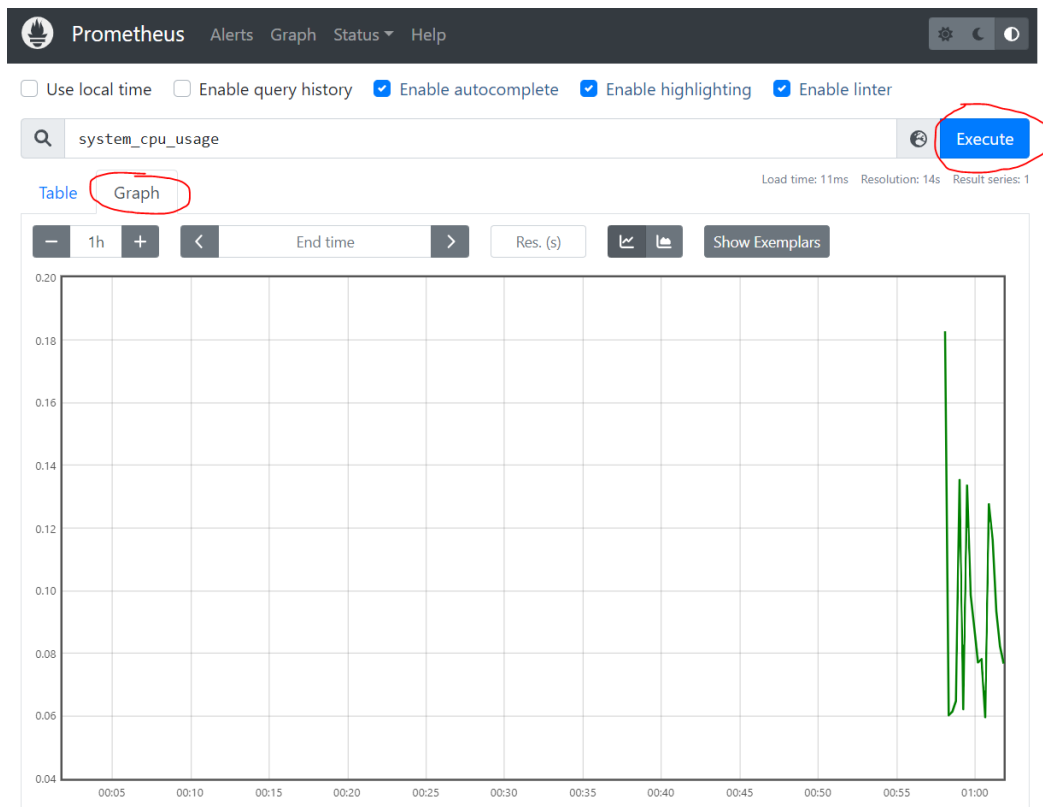
Then in the browser go to **http://localhost:9095/** and see if Prometheus is running.



Now click the Open Metrics Explorer button left from the Execute button and select **system_cpu-usage**.

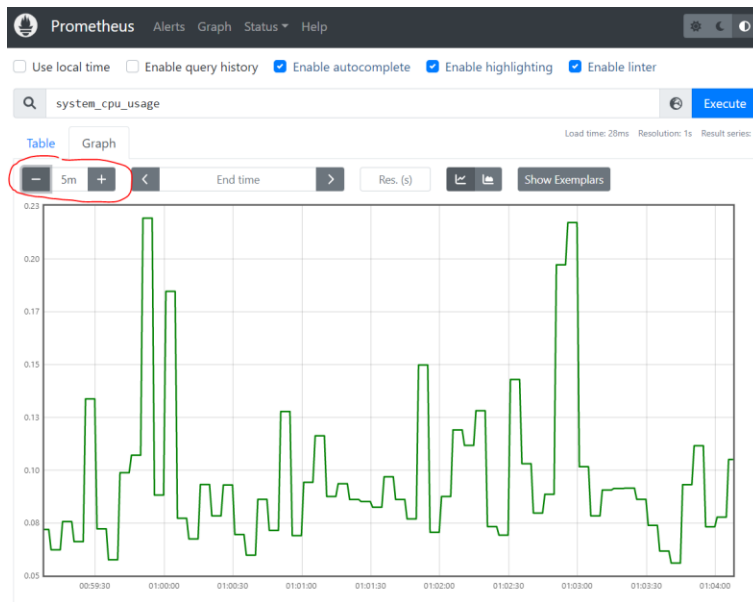


Then click the Execute button and then select the Graph tab:



You now see the cpu usage of your machine.

You can zoom in to 5 minutes:



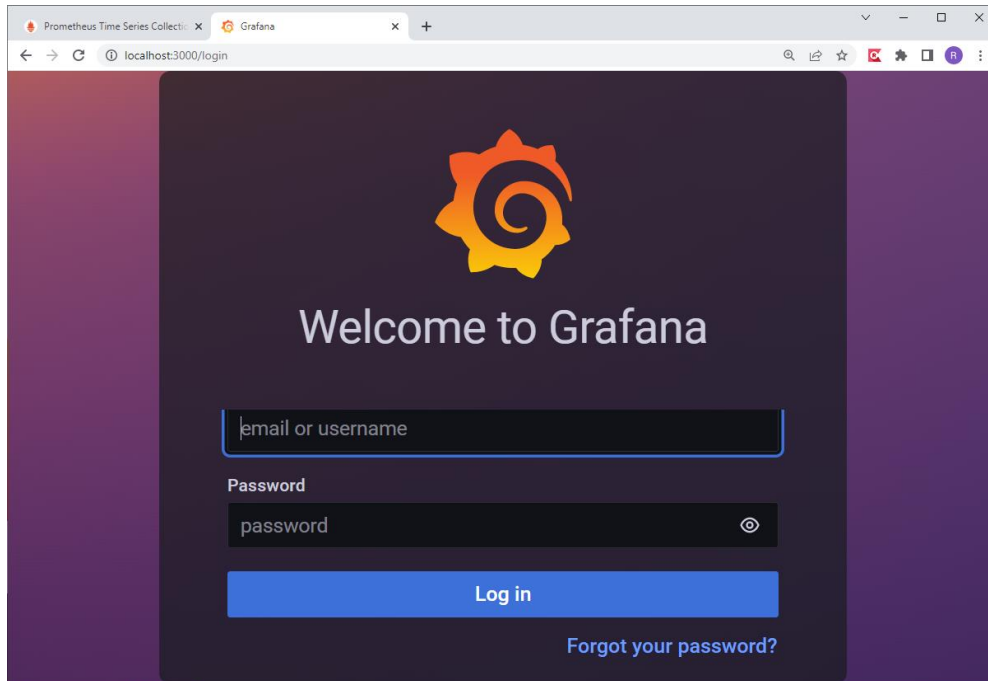
Now try the following metrics:

- disk_free_bytes
- process_cpu_usage
- jvm_classes_loaded_classes

Now double click the file **C:\CS544\Tools\grafana\bin\grafana-server.exe**. You can find a shortcut of this file in the Desktop inside of the folder called “**Tool Shortcuts**”.

This should start grafana.

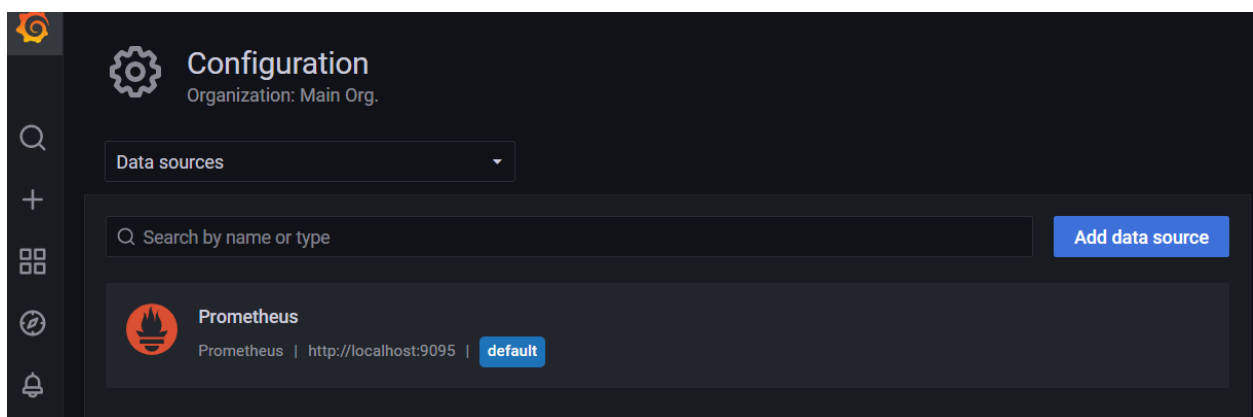
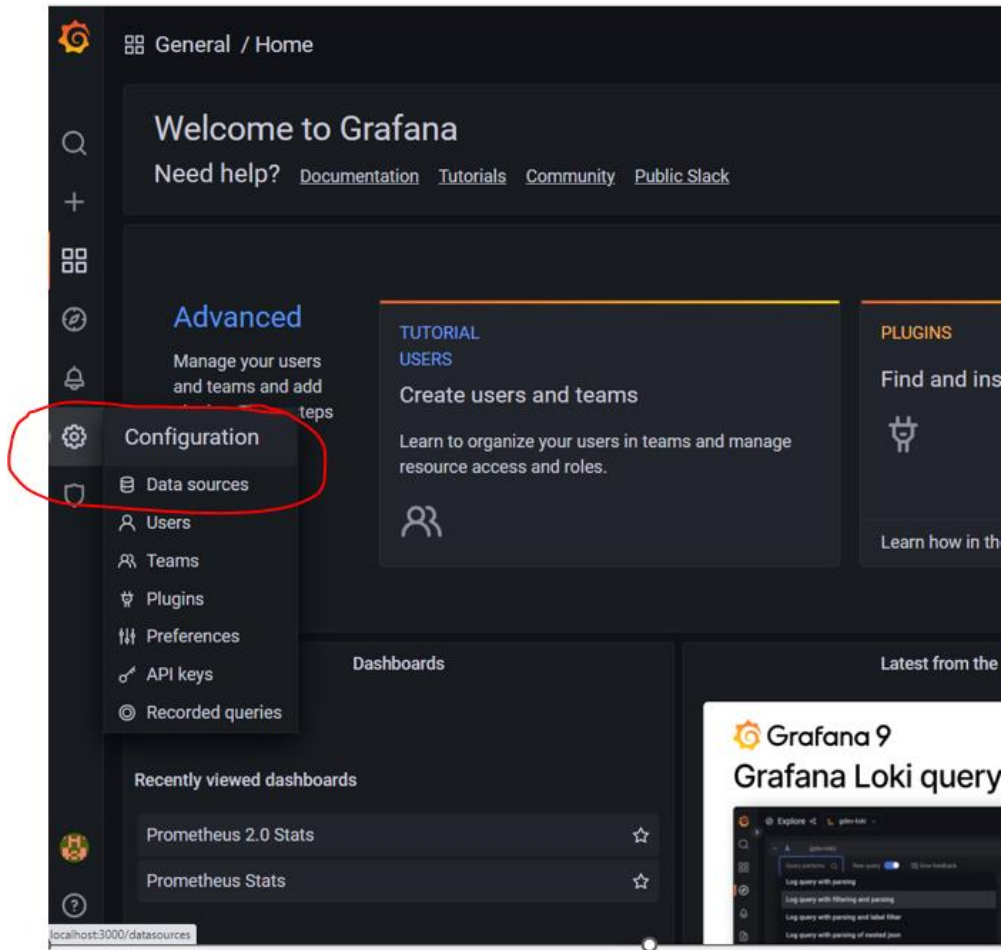
In the browser go to **http://localhost:3000/**



Login with username **admin** and password **admin**.

When Grafana asks for a new password, just type **admin** again.

Select **Administration** (in the picture you will read “Configuration” due to it is an old picture).
→ **Data sources** and click **Prometheus**.



Change the URL to <http://localhost:9095/> if this is not done yet.

Then scroll down to the bottom of the page and select Save and Test

The screenshot shows the Prometheus Settings page in Grafana. The page is titled "Data Sources / Prometheus" and has a "Type: Prometheus" label. The "Settings" dropdown is selected. The "Name" field is "Prometheus" and the "Default" toggle is turned on. The "HTTP" section is expanded, showing the "URL" field set to "http://localhost:9095". The "Access" dropdown is set to "Server (default)" with a "Help" link. The "Allowed cookies" field is set to "New tag (enter key to add)". The "Timeout" field is set to "Timeout in seconds". The "HTTP Method" dropdown is set to "POST". The "Misc" section is expanded, showing the "Disable metrics lookup" toggle turned off and the "Custom query parameters" field set to "Example: max_source_resolution=5m&timeout=10". The "Exemplars" section has a "+ Add" button. At the bottom, there are buttons for "Back", "Explore", "Delete", and "Save & test". The footer contains links for "Documentation", "Support", "Community", and "Enterprise (Free & unlicensed)", along with the version "v8.4.6 (c53173ff6)".

Prometheus Time Series Collect... Prometheus Settings - Grafana

localhost:3000/datasources/edit/6gwD6xUnk

Data Sources / Prometheus

Type: Prometheus

Settings

Name Prometheus Default ☒

HTTP

URL

Access Server (default) Help >

Allowed cookies

Timeout

HTTP Method POST

Misc

Disable metrics lookup ☐

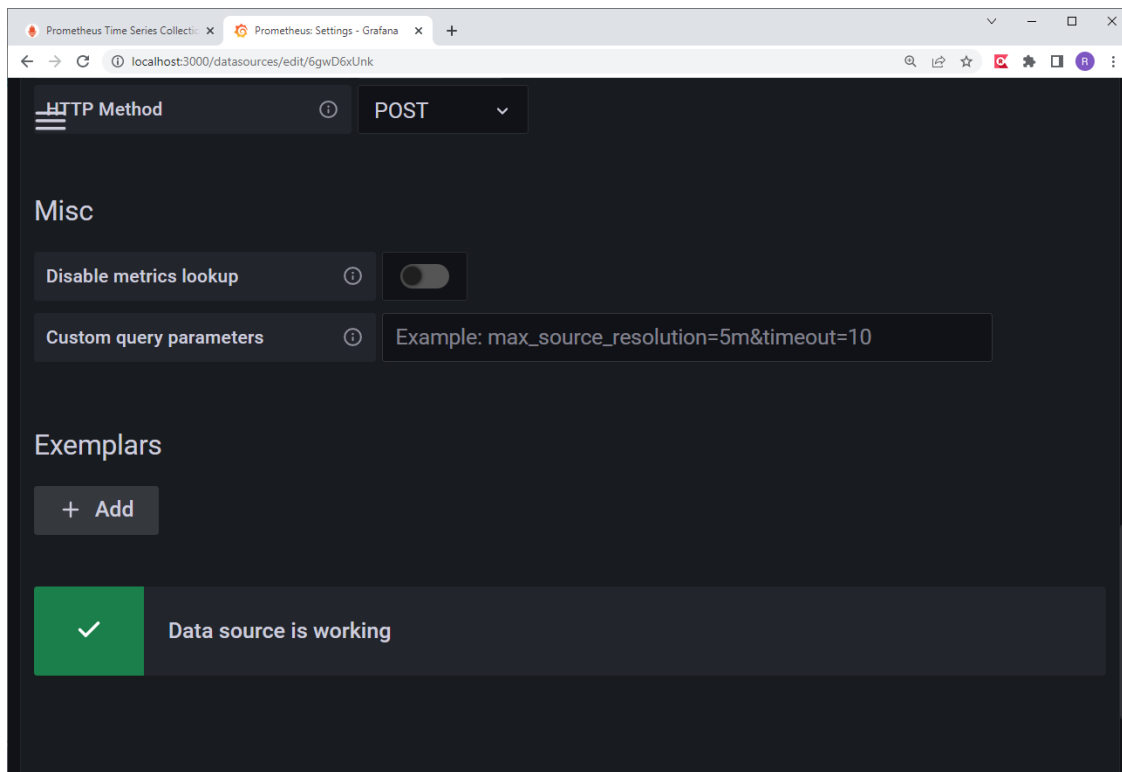
Custom query parameters

Exemplars

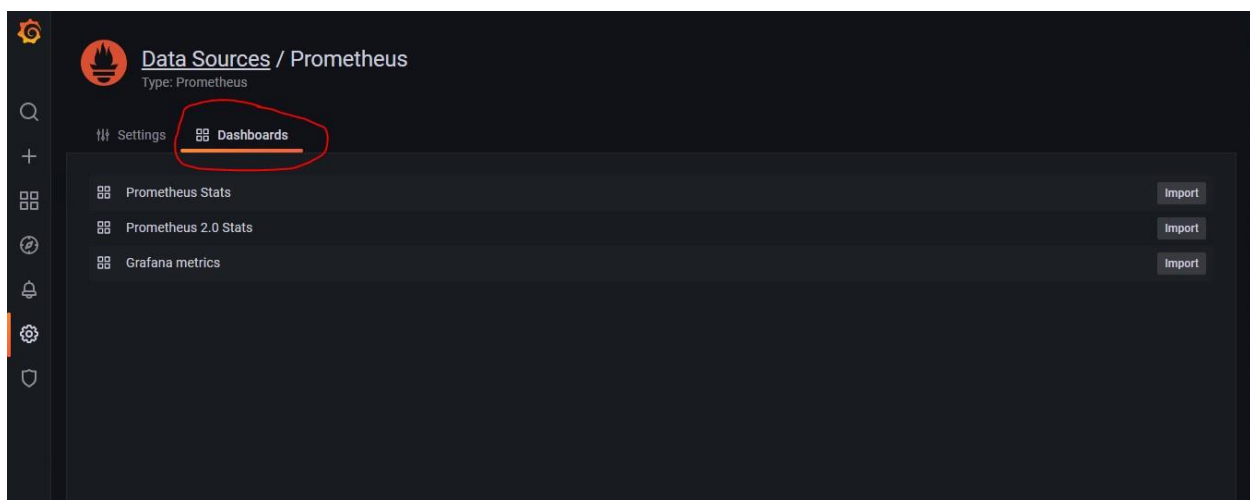
+ Add

Back Explore Delete Save & test

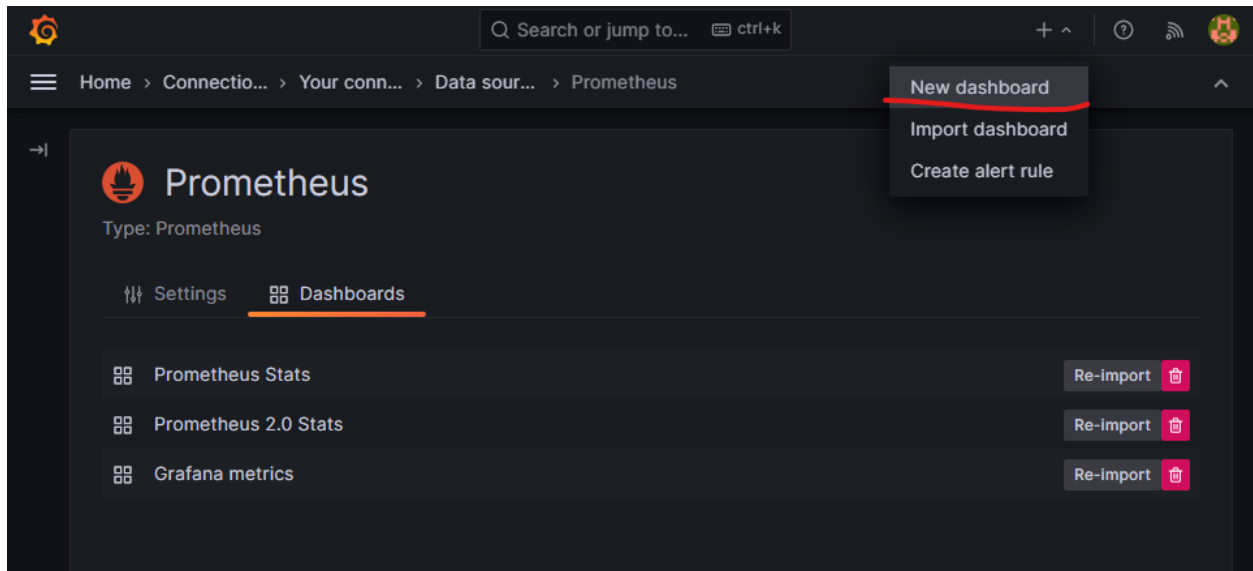
Documentation | Support | Community | Enterprise (Free & unlicensed) | v8.4.6 (c53173ff6)



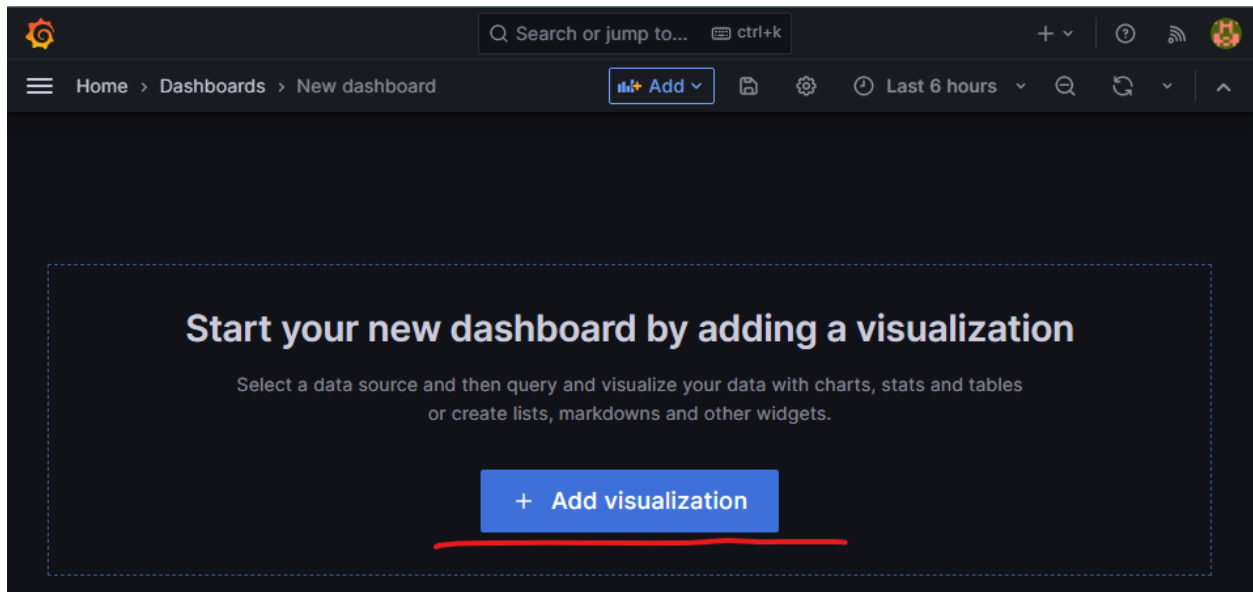
Then go the top of the page and select **Dashboards** instead of Settings

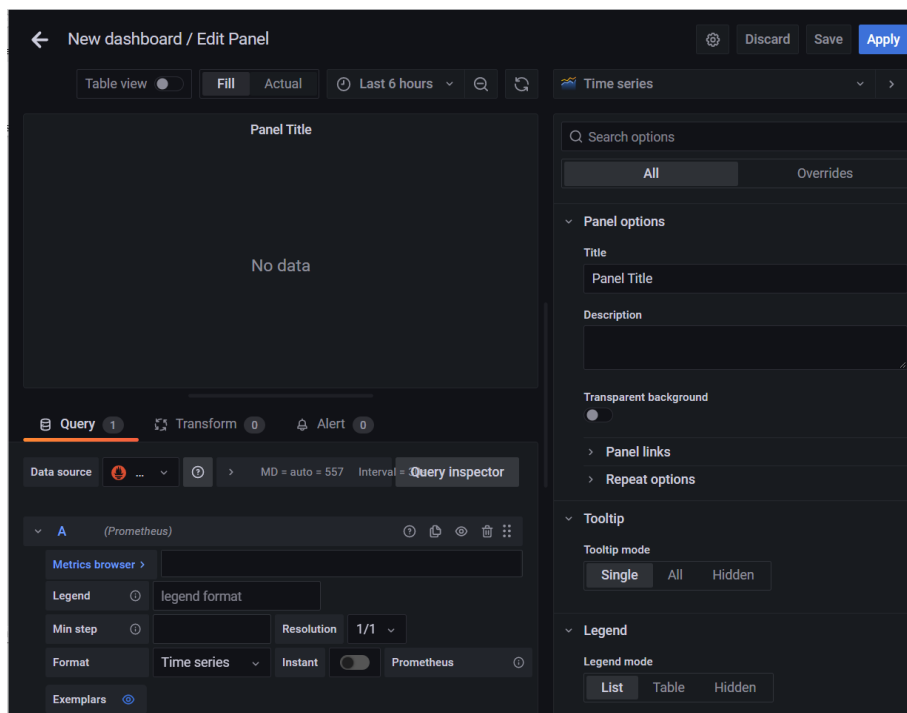


Create a new “**Dashboard**”

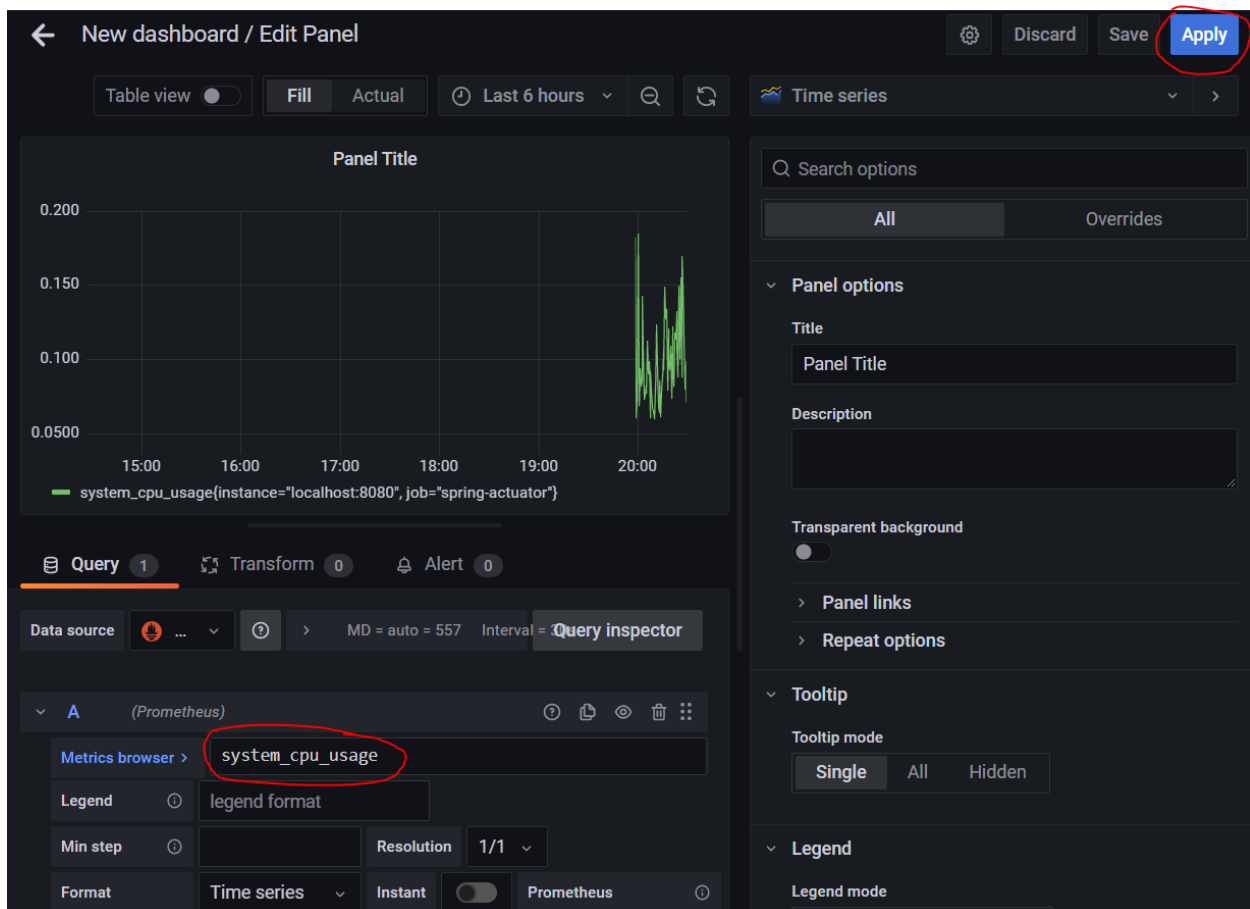


Then click **Add visualization**.



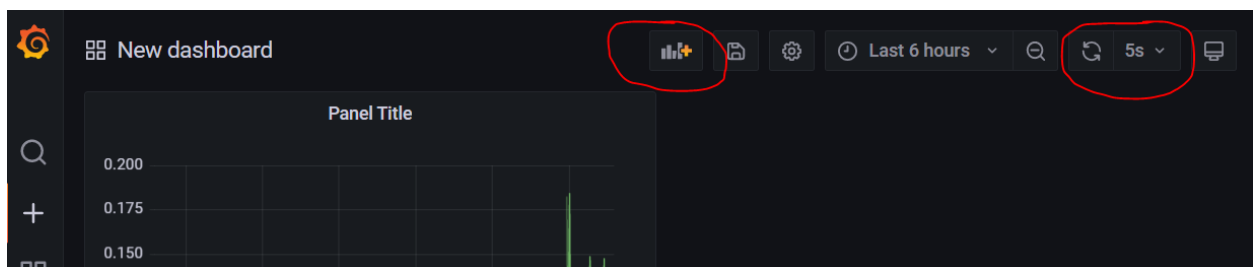
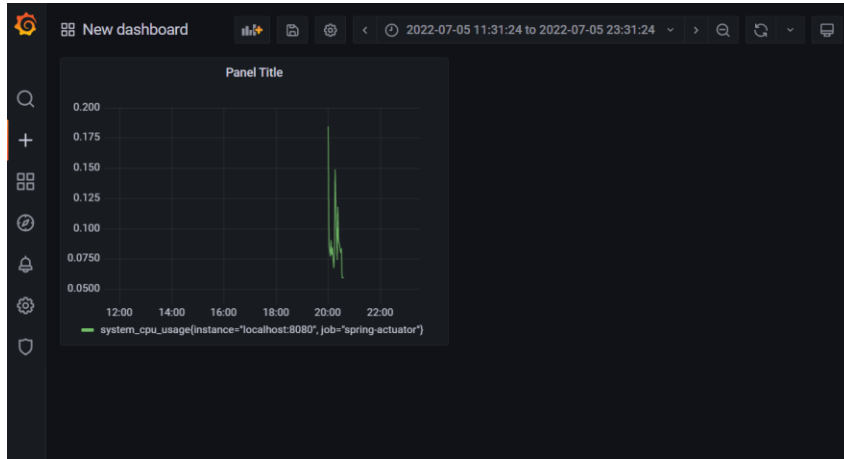


Now we can copy metric names that we used in Prometheus and type it in the metric box.



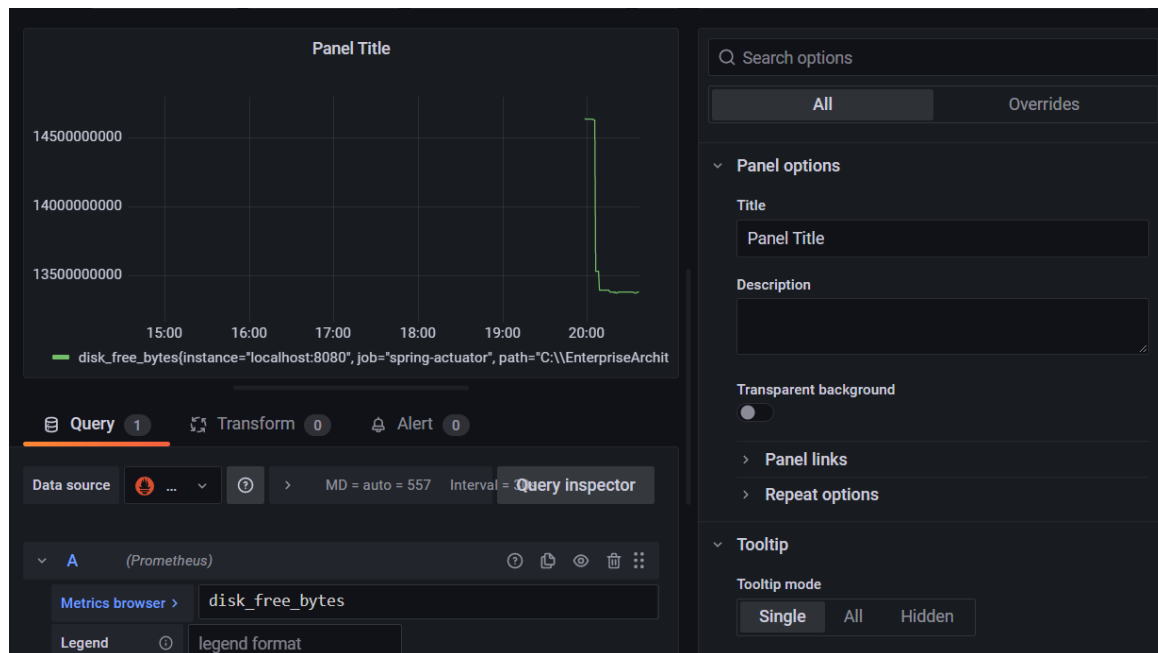
When you add system_cpu_usage in the Metrics field and select the Query inspector button, the graph will be shown.

When you click the Apply button, this graph is added to our dashboard.

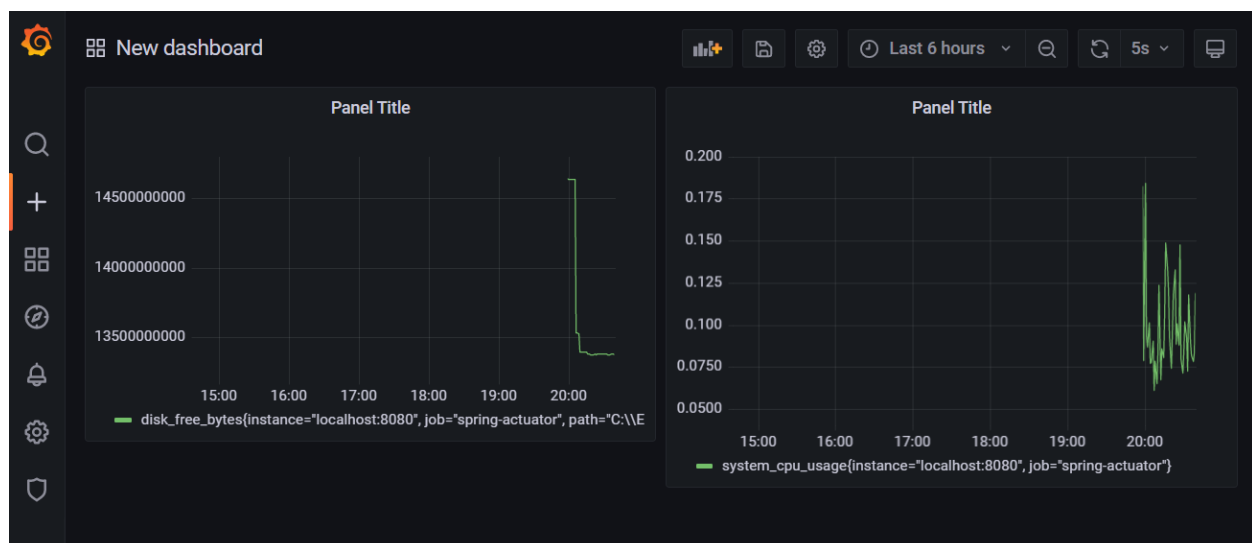


Let the dashboard update itself every 5 seconds

Then click the add new panel button and click on **Visualization**.

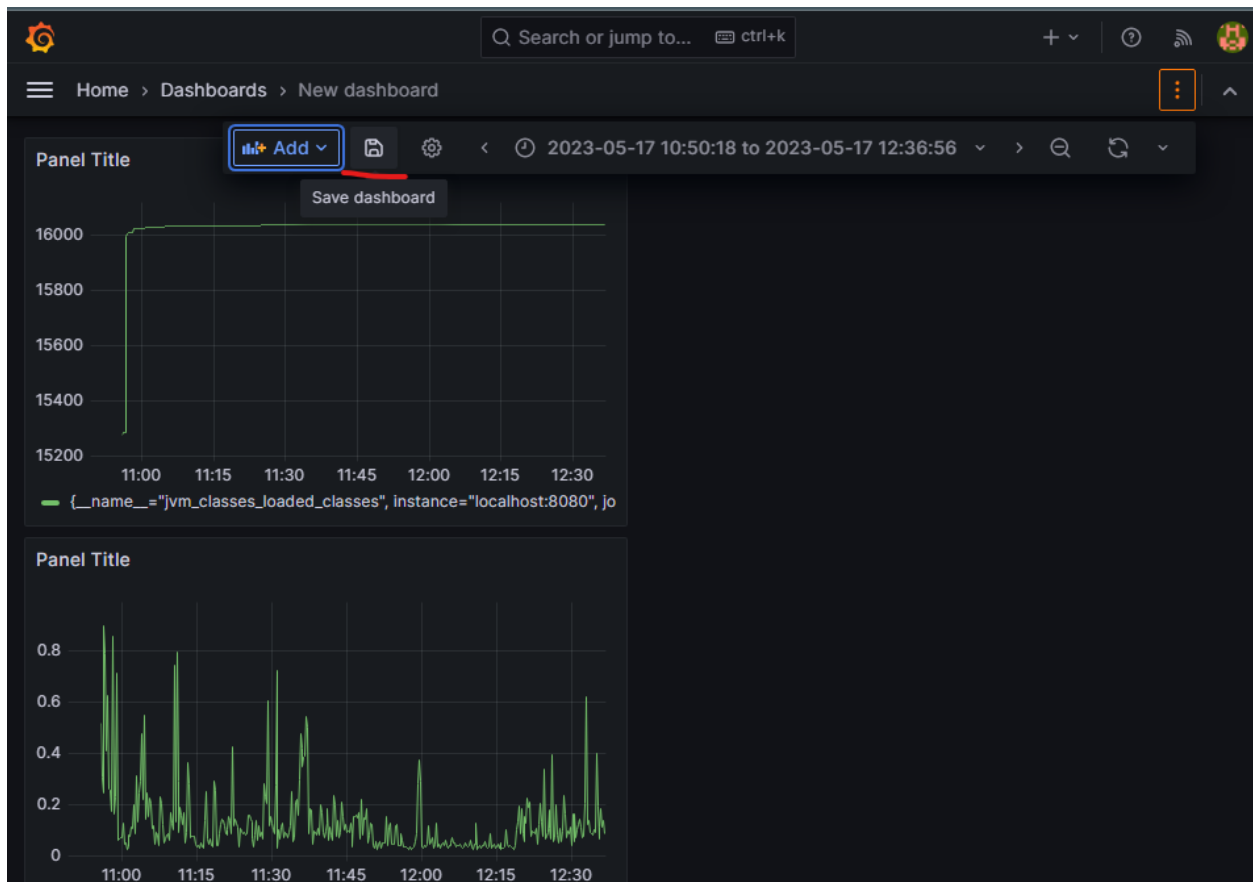


Create the **disk_free_bytes** graph and add it to the dashboard.





Do this also with “jvm_classes_loaded_classes” metric.

Save your Dashboard:



The screenshot shows the 'Save dashboard' dialog box in Grafana. The dialog has a title bar with a close button. The main content area is titled 'Save dashboard' and 'New dashboard'. It contains a 'Details' section with two input fields: 'Dashboard name' and 'Folder'. The 'Dashboard name' field contains the text 'New dashboard', which is underlined in red. The 'Folder' field is a dropdown menu currently set to 'General'. At the bottom of the dialog are two buttons: 'Cancel' and 'Save'. The 'Save' button is highlighted with a red underline.

Now in the browser go to <https://grafana.com/grafana/dashboards/12900>




[← All dashboards](#)

SpringBoot APM Dashboard

Dashboard for Spring Boot Metrics

[Overview](#) [Revisions](#) [Reviews](#)



A dashboard for Java Springboot Applications deployed on a Kubernetes Cluster.

The dashboard supports ALL Nodes in the Node selector.

Get this dashboard

Data source:

[Grafana 7.0.3](#) [Prometheus 1.0.0](#)

Click the **Download JSON** link:

Import the dashboard template:

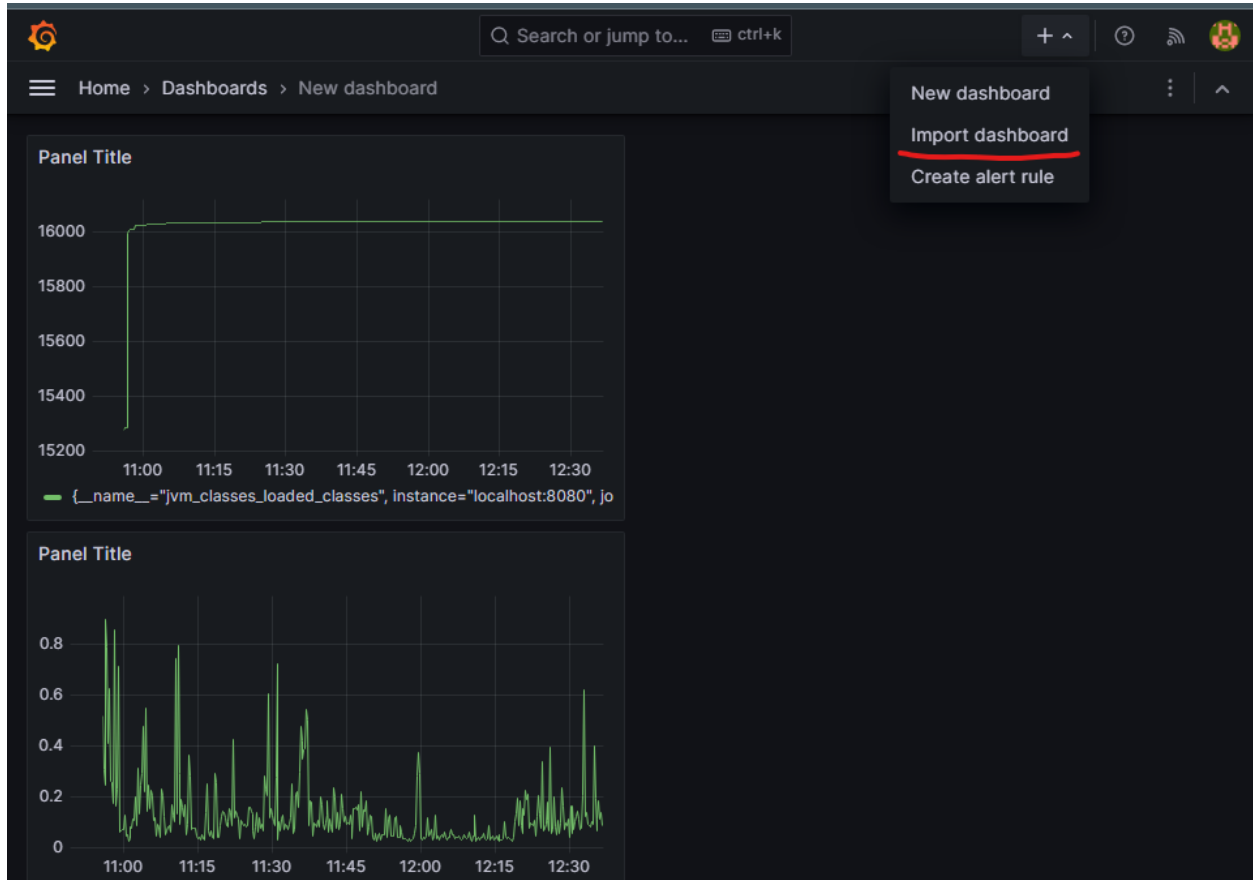
Copy ID to clipboard

or

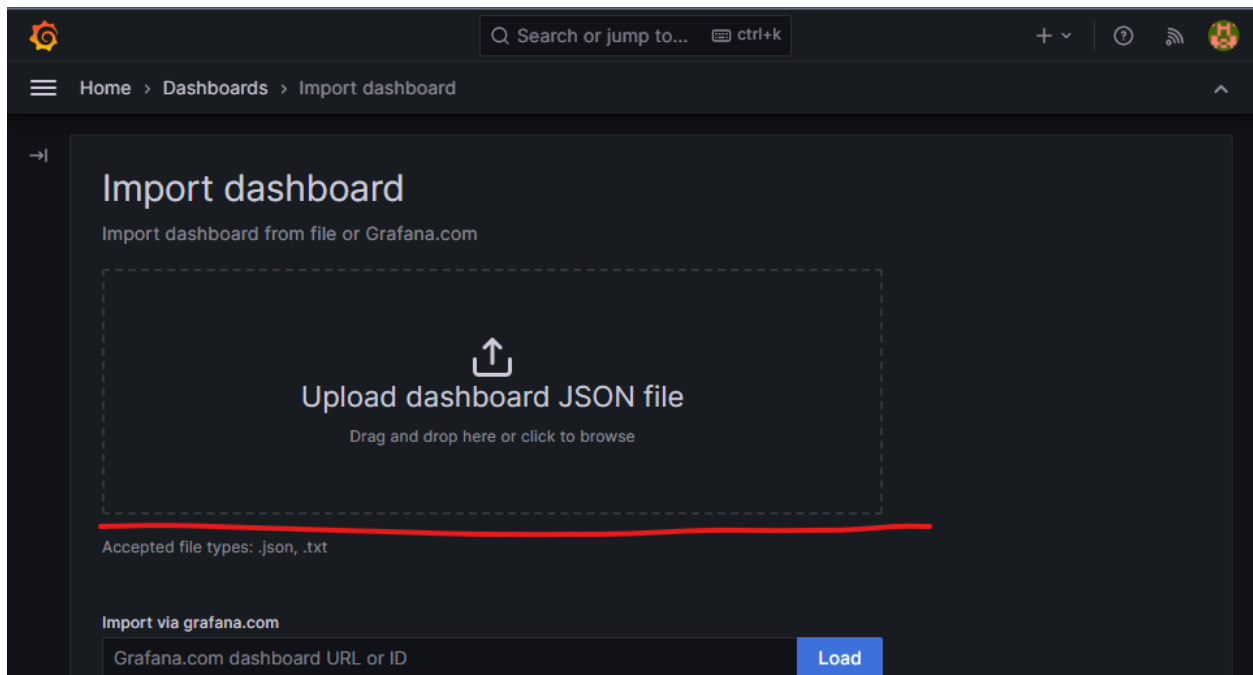
Download JSON

[Docs: Importing dashboards](#)

In Grafana, click in the + symbol and click on “**Import dashboard**”.



Click the **Upload dashboard JSON file** button and select the file you downloaded.



Select the just downloaded JSON file and select Prometheus as data source. Then click **Import**.

Options

Name

Folder

General

Unique identifier (UID)

The unique identifier (UID) of a dashboard can be used for uniquely identify a dashboard between multiple Grafana installs. The UID allows having consistent URLs for accessing dashboards so changing the title of a dashboard will not break any bookmarked links to that dashboard.

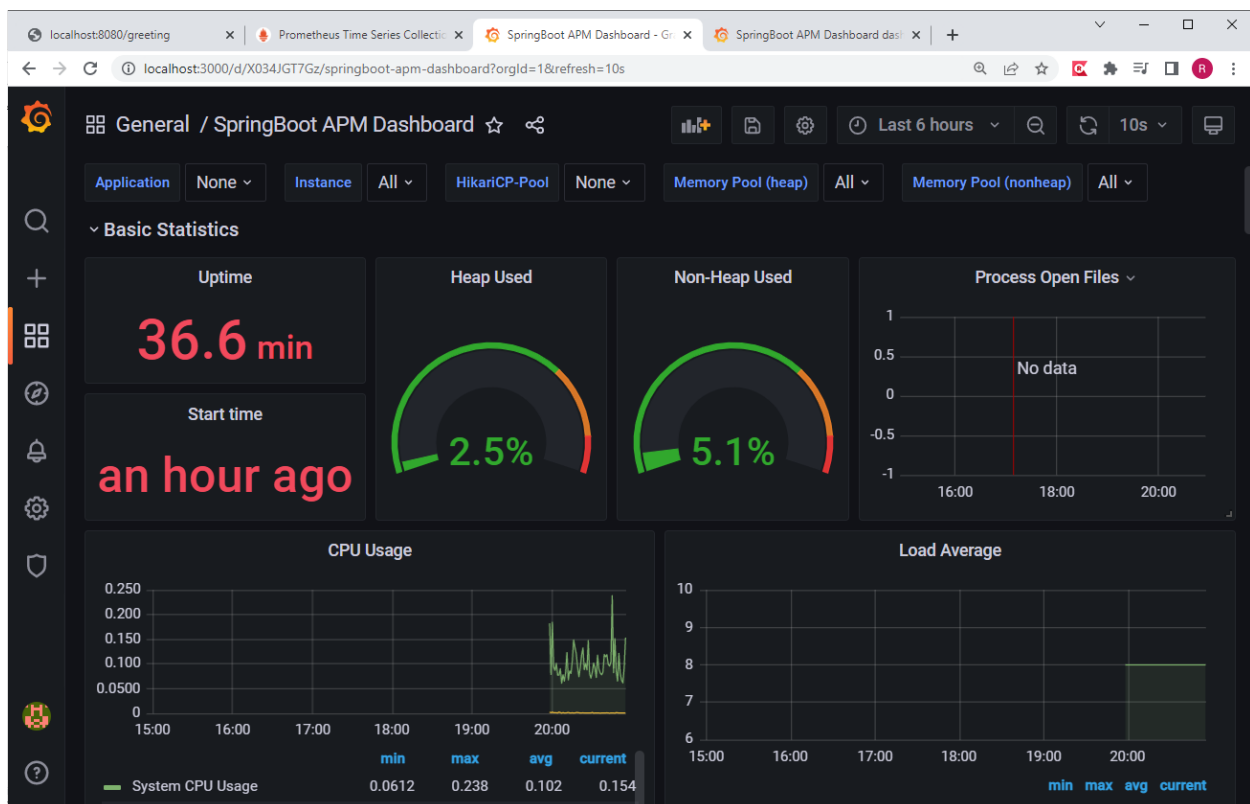
X034JGT7Gz [Change uid](#)

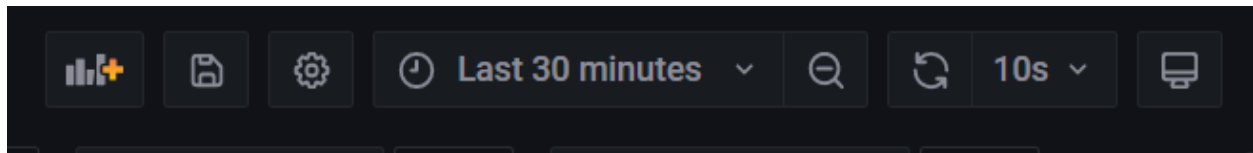
Prometheus

Prometheus

[Import](#) [Cancel](#)

We have now a nice dashboard for our application. Take a screenshot of this dashboard





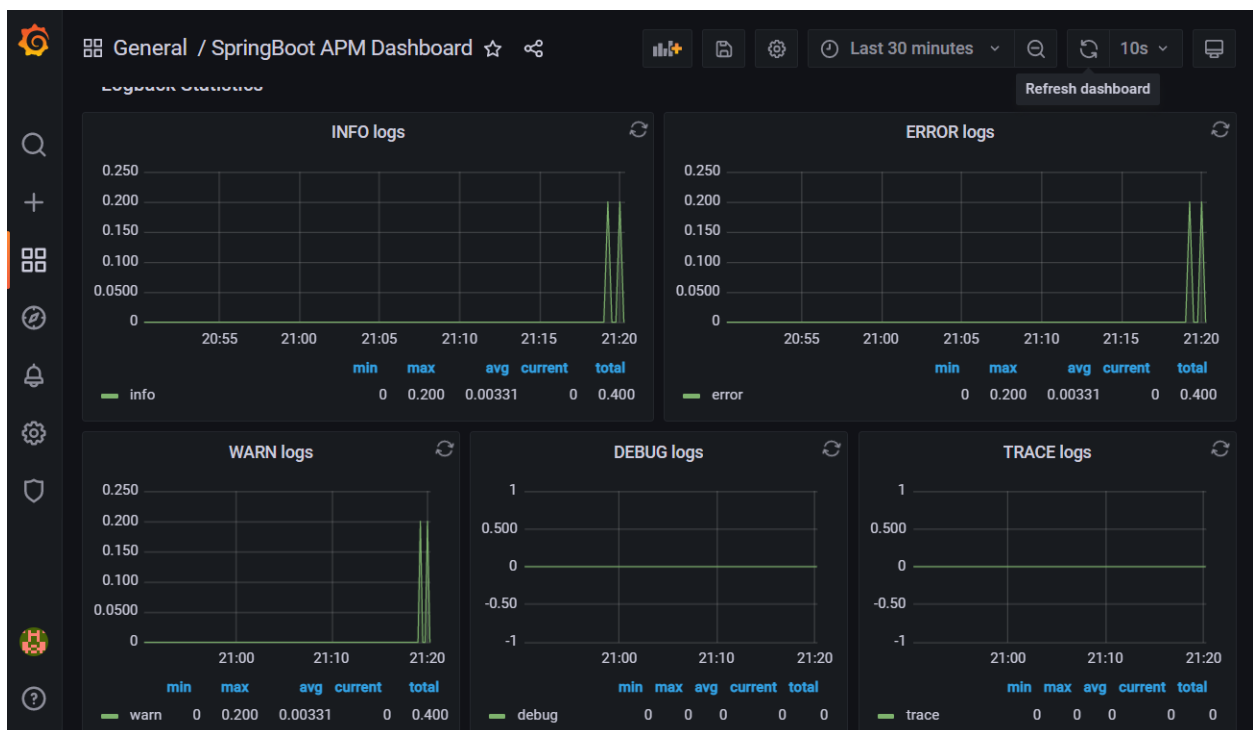
You can now play a little bit with the dashboard.

Now modify the application as follows:

```
@RestController
public class GreetingController {
    Logger logger = LoggerFactory.getLogger(GreetingController.class);

    @RequestMapping(value="/greeting")
    public String greeting() {
        logger.info("An INFO Message");
        logger.warn("A WARN Message");
        logger.error("An ERROR Message");
        return "Hello World";
    }
}
```

Restart the application and call the endpoint a few times.



Now you see the graphs of the logs change.

Part D

Suppose you need to test the current Bank application. Elaborate what elements of the application you need to test. In other words, how would you test this application?

Part E

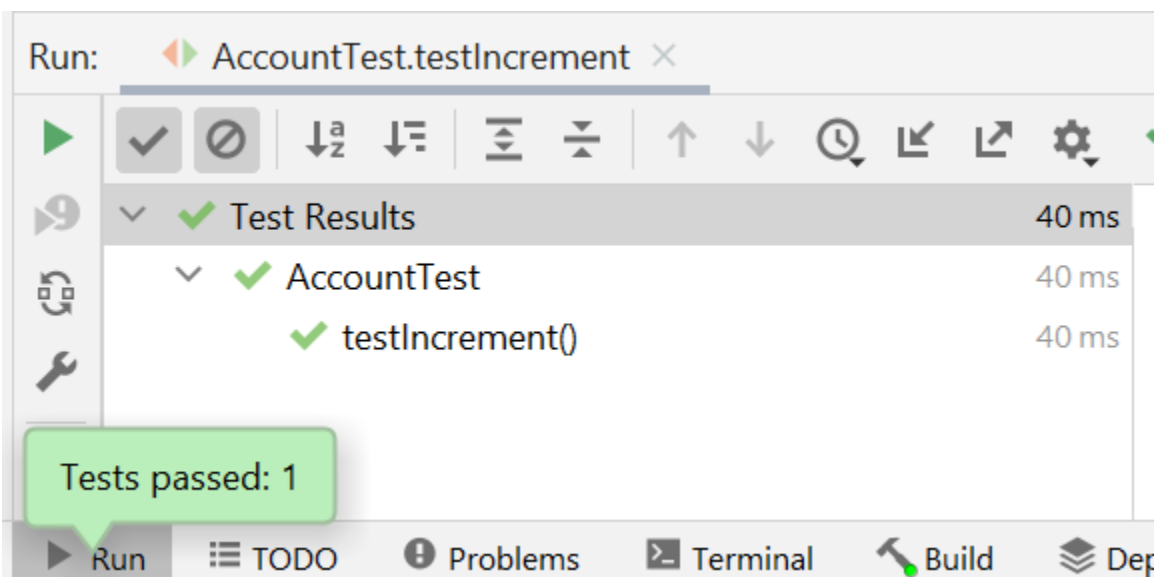
In the Bank application create the Class AccountTest in the package bank in src/test/java

```
package bank;

import bank.domain.Account;
import org.junit.jupiter.api.Test;
import static org.hamcrest.MatcherAssert.assertThat;
import static org.hamcrest.CoreMatchers.*;
import static org.hamcrest.Matchers.*;

public class AccountTest {
    @Test
    public void testIncrement() {
        Account account = new Account();
        account.deposit(100.0);
        assertThat( account.getBalance(), closeTo (100.0, 0.01));
    }
}
```

Run the test.



Write more unit tests for the Account class.

What to hand in?

- A zip file for part A
- Make screenshots of the actuator output for part B and put them in a word file.
- Make a screenshot of the Grafana dashboard of part C
- A document with the answer of part D.
- A zip file for part E