SWEN 438 *Special Topic: DevOps*

Laboratory 4: Prometheus

The purpose of this laboratory is to get some practical experience with telemetry using a tool called https://prometheus.io/.

Getting Started

The starting point is to setup the Prometheus tool. There are several approaches one can take:

- **Download** the latest package from https://prometheus.io/download/ and unpack into a suitable place.
- **Alternatively** you can create a VirtualBox Linux image (e.g. Ubuntu 21.04) and install prometheus via `sudo apt-get install prometheus`

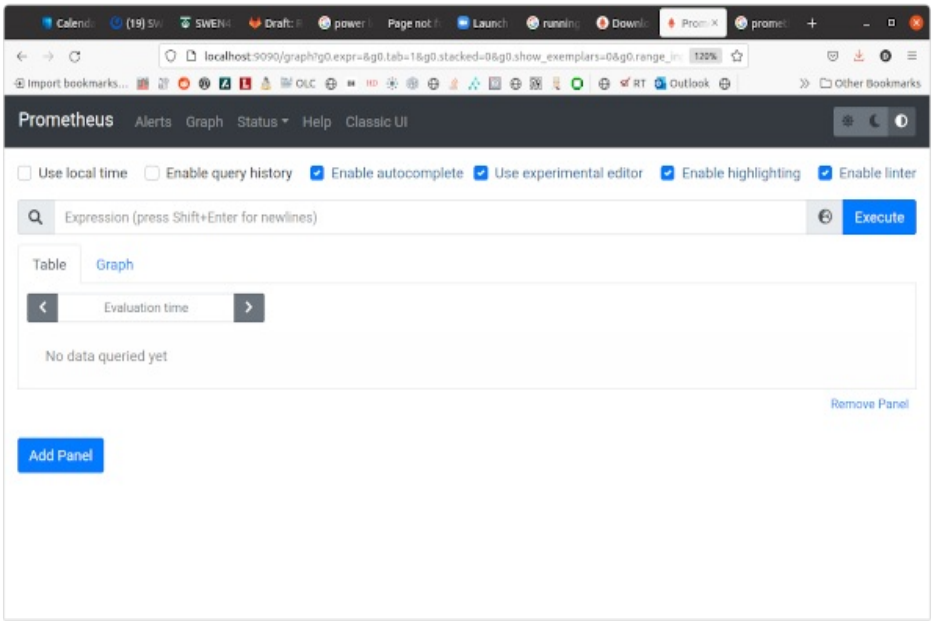
At this point, you want to modify the configration file `prometheus.yml` to set a `scrape_target` as follows:

```
scrape_configs:
  # The job name is added as a label `job=<job_name>` to any timeseries scraped from this config.
  - job_name: "Java Fiddle"

    # metrics_path defaults to '/metrics'
    # scheme defaults to 'http'.

    static_configs:
      - targets: ["localhost:8081"]
```

You should now start Prometheus running as follows:

```
./prometheus --config.file=prometheus.yml
```

To test it is working, visit `localhost:9090` and, if everything is going correctly, you should see the following:



At this point, select `Status -> Targets` and you should see Java Fiddle is listed, but marked as DOWN in red.

Part 1: Configure JavaFiddle

The next stage is to download the JavaFiddle source files from the SWEN438 homepage. We recommend you start from a fresh install to avoid accidentally breaking your assignment code, etc. Having downloaded JavaFiddle and e.g. setup a project in Eclipse or IntelliJ, you need to add the following dependency to the `pom.xml` file:

```
<dependency>
<groupId>io.prometheus</groupId>
<artifactId>simpleclient</artifactId>
<version>0.12.0</version>
</dependency>
<dependency>
```

```
<groupId>io.prometheus</groupId>
<artifactId>simpleclient_httpserver</artifactId>
<version>0.12.0</version>
</dependency>
```

Furthermore, we need to add the following code to the start of `javafiddle.Main`:

```
import io.prometheus.client.exporter.HTTPServer;

public static void main(String[] _args) throws InterruptedException, IOException {
   HTTPServer server = new HTTPServer.Builder()
          .withPort(8081)
          .build();
   ...
```

Notice the port `8081` has been selected as the point where the Prometheus metrics will be exposed. Again, its helpful to test the setup at this point. Startup the JavaFiddle application and visit `localhost:8081`. You should see an empty page for now.

Part 2: Instrument JavaFiddle with a Counter

We're now going to instrument `javafiddle.Application` using a simple `Counter`. This just records the total number of times something has happened. To do this, add the following line to the `Application` class:

```
static final Counter requests = Counter.build()
       .name("requests_total").help("Total requests.").register();
```

This creates the counter, and we now need to instrument some requests to generate data. I suggest adding the following line to the start of all the main request methods in `Application` (i.e. `create()`, `login()`, `logout()`, `save()` and `compile()`:

```
requests.inc();
```

Now, restart JavaFiddle and again visit `http://localhost:8081`. This time you should see something like this:

```
# HELP requests_total Total requests.
# TYPE requests_total counter
requests_total 0.0
# HELP requests_created Total requests.
# TYPE requests_created gauge
requests_created 1.630987984201E9
```

We can see our counter `requests_total` is current set to `0.0`. Let's increase it by going to the JavaFiddle page (`http://localhost:8080`) and running some requests (e.g. just press `Compile` once). Returning the metrics page above and you should see the counter has changed!

Now, let's see this in the Prometheus tool by visiting `http://localhost:9090`. In the "expression bar" type `requests_total` (it may autocomplete this for you) and press enter. You should now see the following line:

```
requests_total{instance="localhost:8081", job="Java Fiddle"}
```

It will also report the number of requests. You can then view this as a graph (though it won't be very interesting yet since we haven't done much). You can try generating a few more requests on JavaFiddle by hand then returning to see it in Prometheus (generally it seems to refresh you must re-execute the expression `requests_total`).

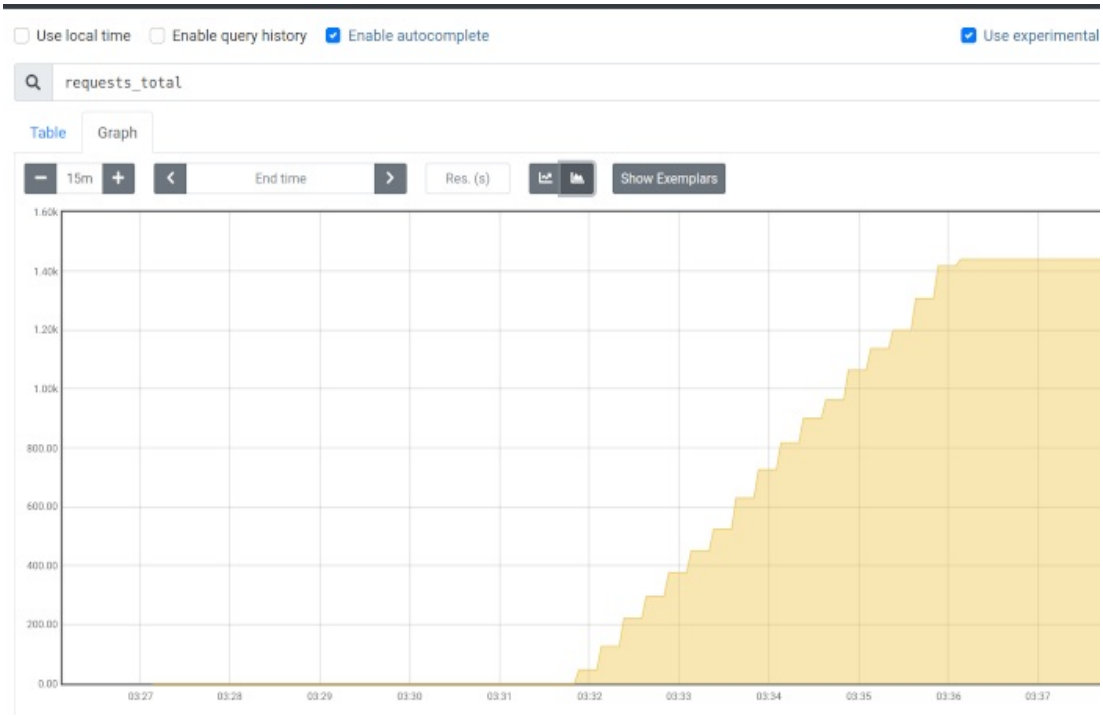
Part 3: Running a simulation

At this point, we're now going to run a JavaFiddle `Simulation` to generate some more interesting traffic. There a few ways to do this:

- **(Local Simulation)** In this mode, we let the `Simulation` start and control the JavaFiddle server. This is perhaps the easiest, but you need to copy the HTTP`Server` code from above into the `main()` method in `Simulation`. Make sure that you have stopped any other instances of JavaFiddle running (otherwise you will see it cannot connect).
- **(External Simulation)** in this mode, you control the JavaFiddle server and let the `Simulation` connect to it. I often do this by running JavaFiddle using Maven on the command-line (e.g. `mvn exec:java`). The setup is slightly more complicated as we need to pass the command-line option `-url http://localhost:8080` to the `Simulation` (e.g. as part of a run configuration in Eclipse).

All going well, you can return to the Prometheus page and reexecute the expression `requests_total` and (after a few minutes

have passed) you will see something like this:



This is showing that the number of requests is climbing over time in clear steps (which probably reflects the delay used in the simulation loop).

Part 3: Instrument JavaFiddle with a Gauge

At this point, we can play around with some other metrics. For example, suppose we want to measure the number of logged in users at any given point. To do this, we change from a `Counter` to Gauge by replacing our `requests` declaration with the following:

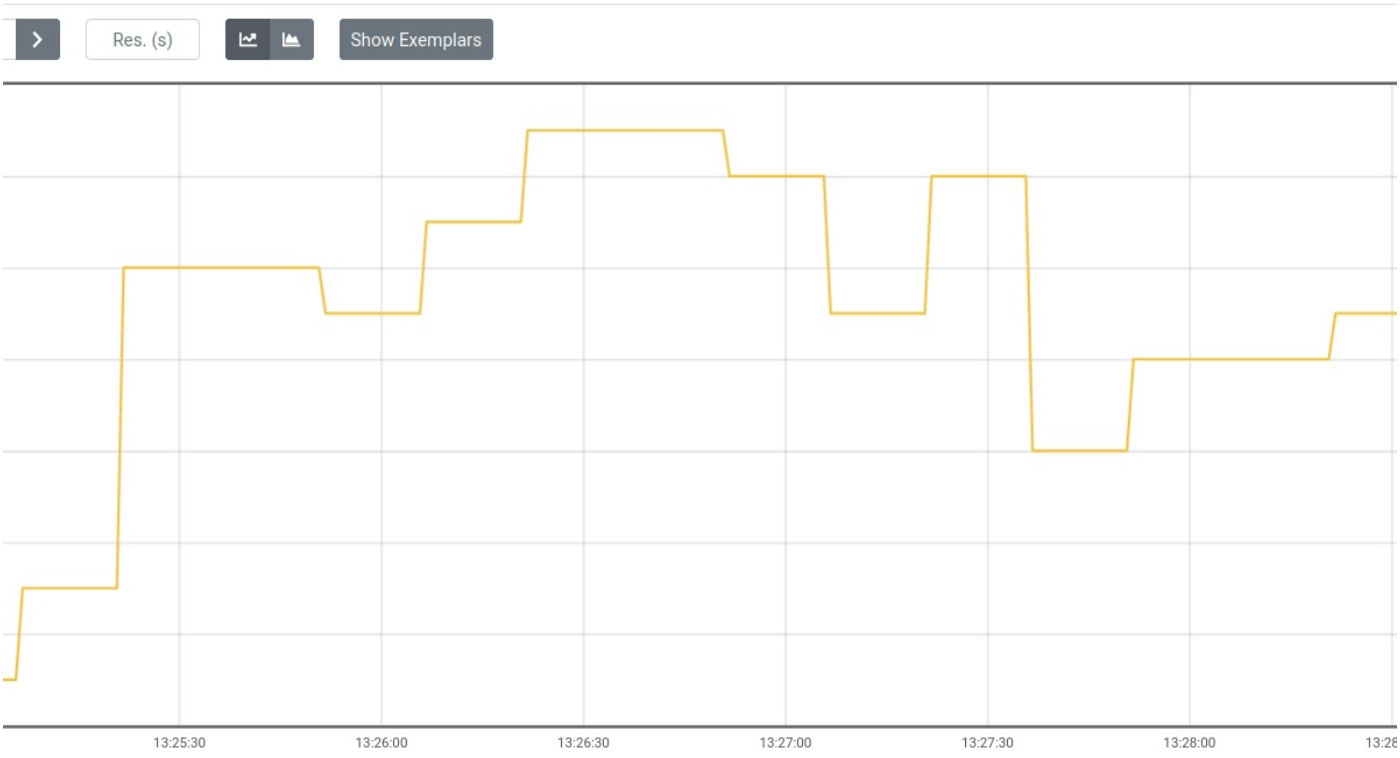
```
static final Gauge requests = Gauge.build()
       .name("requests_total").help("Total requests.").register();
```

Furthermore, we remove all of the `requests.inc()` calls from our methods, and modify `create()`, `login()` and `logout()` as follows:

```
public synchronized Login.Response create(Token token, Login.Request rq) {
   ...
   if (user == null) {
      ...
      requests.inc();
      return new Login.Response(true);
   } else {
      return new Login.Response(false);
   }
}

public synchronized Login.Response login(Token token, Login.Request rq) {
   ...
   if (user != null && user.password.equals(rq.getPassword())) {
      ...
      requests.inc();
      return new Login.Response(true);
   } else {
      return new Login.Response(false);
   }
}

public synchronized void logout(Token token, Logout.Request rq) {
   // Remove the given session
   if(sessions.containsKey(token)) {
      sessions.remove(token);
      requests.dec();
```

```
    }
}
```

Using this, I was able to generate the follwing graph of user behaviour during a simulation:



Fiddle"}

This shows a large number of users logging in (or creating accounts) at the beginning of the simulation and then, over time, logging in and loggin out.

Part 4: Exploring JVM Metrics

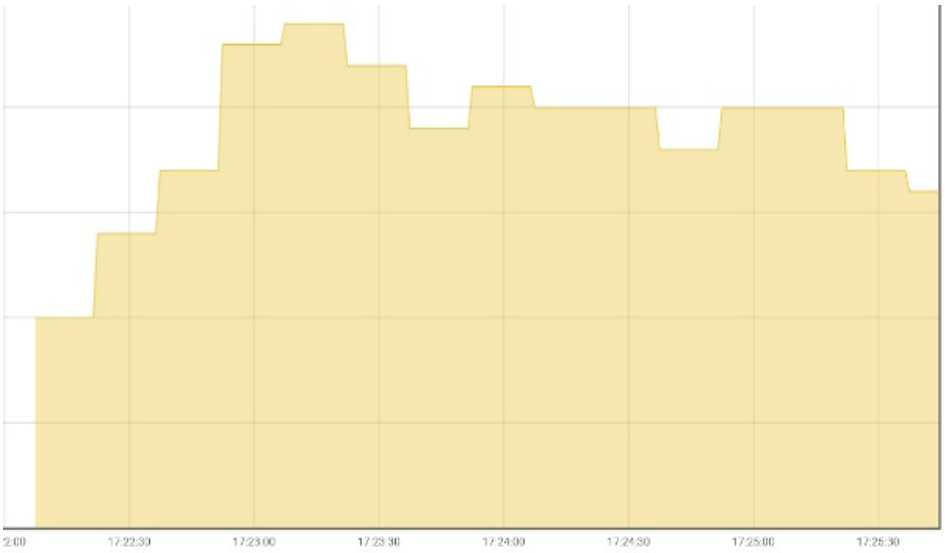
A final an interesting exercise is to generate more information from JavaFiddle about its memory usage, and other low-level metrics. This is very easy to do with the Prometheus Java Client. First, add the following dependency:

```
<dependency>
<groupId>io.prometheus</groupId>
<artifactId>simpleclient_hotspot</artifactId>
<version>0.12.0</version>
</dependency>
```

Second, add the following to the start of the `Simulation.main` method:

```
import io.prometheus.client.hotspot.DefaultExports;

...

void main(String[] args) {
   DefaultExports.initialize();
```

At this point, within Prometheus you should be able to access more metrics than before. Clicking on the “world” icon to the right of the expression bar brings up a range of metrics, and scrolling down one kind find e.g. `jvm_threads_current`. The following graph was generated for a simulation run showing how the number of threads evolves over time:



Submission and Marking

For this lab, please submit the file `Application.java` to illustrate changes you have made, along with one or more screenshots of graphs generated via the ECS Submission system https://apps.ecs.vuw.ac.nz/submit/SWEN438/Laboratory_4.

Full marks will be earned for meeting the following criteria:

1. Evidence that Prometheus has been used to conduct experiments.
2. Evidence that different aspects of Prometheus have been explored (e.g. gauges or JVM Metrics)
