A CLOUD GURU

Lesson Description - Using the Prometheus Java Client Library

The Prometheus client libraries can greatly simplify the process of collecting and exposing metrics in your custom code. In this lesson, we will demonstrate how to use one of these client libraries: the Prometheus client library for Java. We will add instrumentation using this client library to some simple Java code and show how Prometheus can be configured to scrape the resulting metrics.

Relevant Documentation

- Java Client Library
- Client Libraries

Lesson Reference

Log in to the Prometheus server.

Note: You can clone the Java code to your local machine and work with it in the IDE or text editor of your choice, but if you want to pull metrics into Prometheus, you will likely need to copy the project to the Prometheus server for that step.

Clone the example project from GitHub:

```
git clone https://github.com/linuxacademy/content-prometheusdd-java-client-lib-example.git
```

Change directory into the project:

```
cd content-prometheusdd-java-client-lib-example
```

Install Java:

```
sudo apt-get update
```

```
sudo apt-get install -y openjdk-8-jdk
```

Run the project. You should see it begin counting, printing the current count to the console:

```
./gradlew run
```

You can stop the application with **Ctrl**+**C**.

Edit the project's `build.gradle`:

```
vi build.gradle
```

Add the Prometheus Java client library dependencies:

```
dependencies {
    implementation 'io.prometheus:simpleclient:0.8.1'
    implementation 'io.prometheus:simpleclient_httpserver:0.8.1'

    ...
}
```

Edit the project's `Main` class:

```
vi src/main/java/com/linuxacademy/prometheusdd/clientlibexample/
Main.java
```

Add a counter to expose the current count as a Prometheus metric called `current_count`. Also, create an `HTTPServer` to provide an endpoint Prometheus can scrape metrics from:

```
package com.linuxacademy.prometheusdd.clientlibexample;

import io.prometheus.client.Counter;
import io.prometheus.client.exporter.HTTPServer;
import java.io.IOException;

public class Main {

    static final Counter currentCount = Counter.build()
        .name("current_count").help("Current count.").register();

    public static void main(String[] args) throws
```

```
  InterruptedException {
          try {
              HTTPServer server = new HTTPServer(8081);
          } catch (IOException e) {
              System.out.println("Failed to start metrics endpoint.");
              e.printStackTrace();
          }

          System.out.println("Counting to 1000...");
          for (int i = 0; i <= 1000; i++) {
              System.out.println(i);
              currentCount.inc();
              Thread.sleep(1000);
          }
          System.out.println("Done counting!");
      }

  }
```

Run your code:

```
./gradlew run
```

Leave the code running. Check your metrics endpoint in a browser at `http://<PROMETHEUS_SERVER_PUBLIC_IP>:8081`. You should see your `current_count` metric.

Edit your Prometheus configuration file to add your application as a new scrape target:

```
sudo vi /etc/prometheus/prometheus.yml
```

Add a scrape config to scrape metrics for your app:

```
scrape_configs:

  ...

  - job_name: 'Java Counter Example'
    static_configs:
    - targets: ['localhost:8081']
```

Restart Prometheus to reload the config:

```
sudo systemctl restart prometheus
```

Note: If your Java app stops (e.g., because it finished counting), you will need to run it again for Prometheus to scrape metrics from it.

Access Prometheus in a browser at `http://<PROMETHEUS_SERVER_PUBLIC_IP>:9090`. Run a query to see your `current_count` metric:

```
current_count
```