**PROMETHEUS WORK**

**PROMETHEUS:**

Prometheus is a open source systems monitoring and alerting toolkit. It collects metrics from configured targets(endpoint) at the specified interval. It can display the results as a graph. It can trigger alerts based on some condition observed to be true.

**Why We Need Prometheus:**

What if we need to know how many request hit or how to know what is the time interval taken by processor or system to handle the request. If we need know how many request came in particular interval of time Ex. Last 5hours or etc… Here Prometheus comes into picture. By using Prometheus we can monitor the system performance, CPU usage. Also we can monitor request duration time, request processed time, response size, any error or exception during processing the request and so on. We can visualize above things in a graph in time series format.

What is time series here? Time series is a collection or series of data points indexed in time order.

Prometheus shows data(metrics) collected from endpoint or targets in a time series format.

Prometheus also used to give alert based on condition what we specify. Take a example we have a website that contains components like database, authentication page, user profile, transaction history etc…

In some time database get crashed due to some reason or database capacity become full. In this situation component which need access to database will get down or some unnecessary malfunctioning takes place like while login it shows error where entered password is even though correct. In this situation website maintenance team need to know database get crashed or no enough space to push user data into database. It is achieved by writing query to give alert based on condition such as, give alert while database capacity reaches 80% or at a given time, processor only handle certain amount of request or any exception occurs. Prometheus provides a functional query language called PromQL (Prometheus Query Language) to write alert or lets the user select and aggregate time series data in real time. The result of an expression can either be shown as a graph, viewed as tabular data in Prometheus's expression browser.

**Metrics in Prometheus:**

Prometheus client library gives four different types of metrics.

1. Counter
2. Gauge
3. Histogram
4. Summary

**Counter:**

Counter metric type is used to any value that increase, such as request hit count, error count, task completed. It cannot used for value which will decrease Ex. CPU usage. Counter metric value only increase or reset to zero.

**Gauge:**

Gauge metric type can be used for values that arbitrarily goes up and down, such as current CPU usage or number of concurrent request or number of jobs in a queue.

**Histogram:**

Histogram metric type measures the frequency of value observations that fall into specific predefined buckets. You could measure request duration for a specific HTTP request call using histograms. Rather than storing every duration for every request, Prometheus will make an approximation by storing the frequency of requests that fall into particular buckets.

By default, these buckets are: .005, .01, .025, .05, .075, .1, .25, .5, .75, 1, 2.5, 5, 7.5, 10.

We can use histogram for request duration, response size etc…

**Summary:**

Summaries and histogram more over similar. With histograms, quantiles are calculated on the Prometheus server. With summaries, they are calculated on the application server. For more detailed comparison visit **[summary vs histogram](https://prometheus.io/docs/practices/histograms/" \l "quantiles)** comparison given in Prometheus documentation.

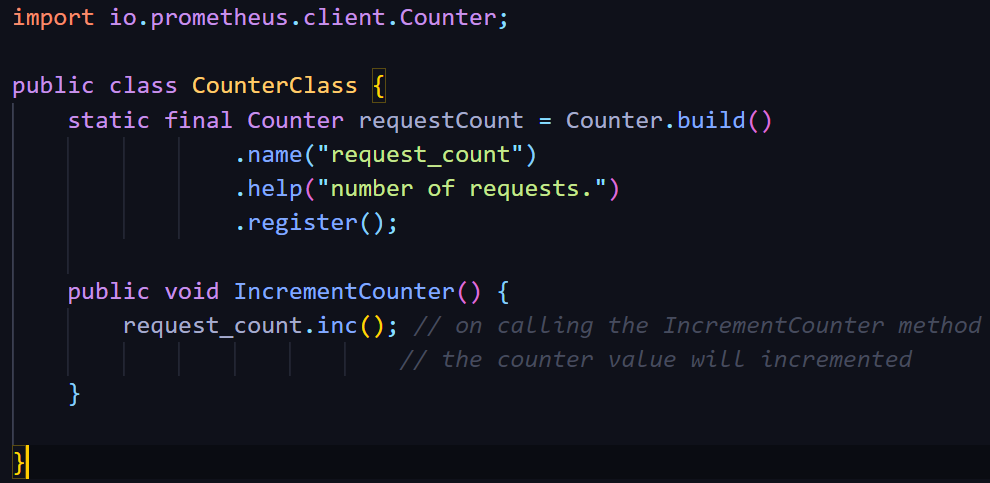
**Prometheus metrics deep dive:**

**Counter:**

Java client for counters.

1. **Counter.build()** --> builder method
2. **public void inc()** --> to increment counter by one.
3. **public void inc(double amt)** --> increment the counter whatever double value given.

Code example:

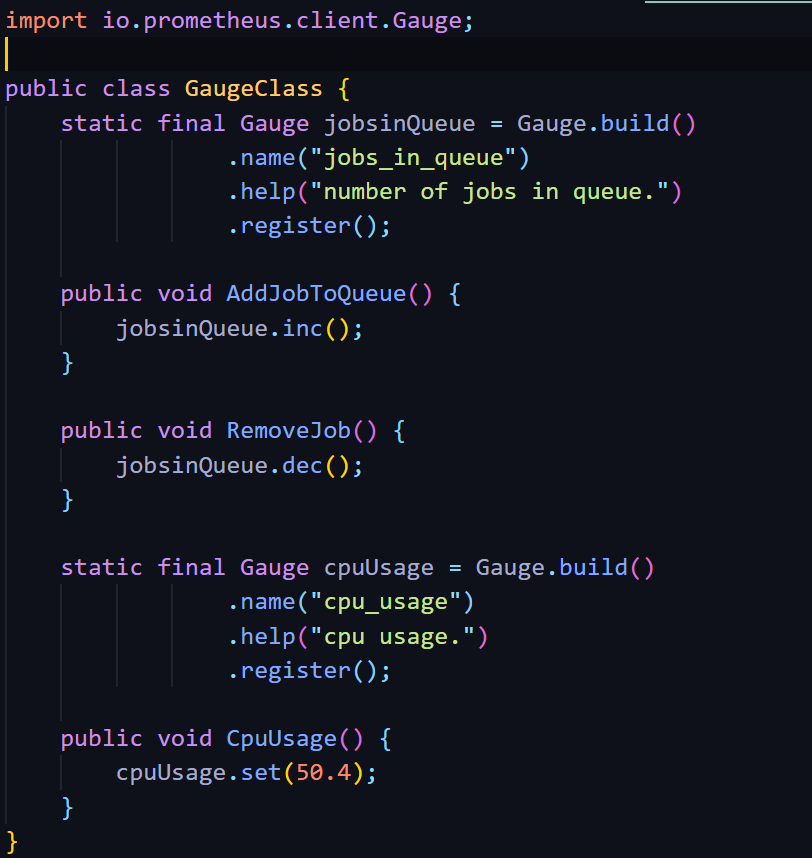


**Gauge:**

Java client for Gauges.

1. **Gauge.build()** --> builder method.
2. **public void inc()** --> to increment gauge metric by 1.
3. **public void inc**(double amt) --> increment the gauge metric whatever double value given.
4. **public void dec()** --> to decrement gauge metric by 1.
5. **public void dec(double amt)** --> decrement the gauge metric whatever double value given.
6. **public void set(double amt)** --> set the gauge metric whatever double value specify

Code example:

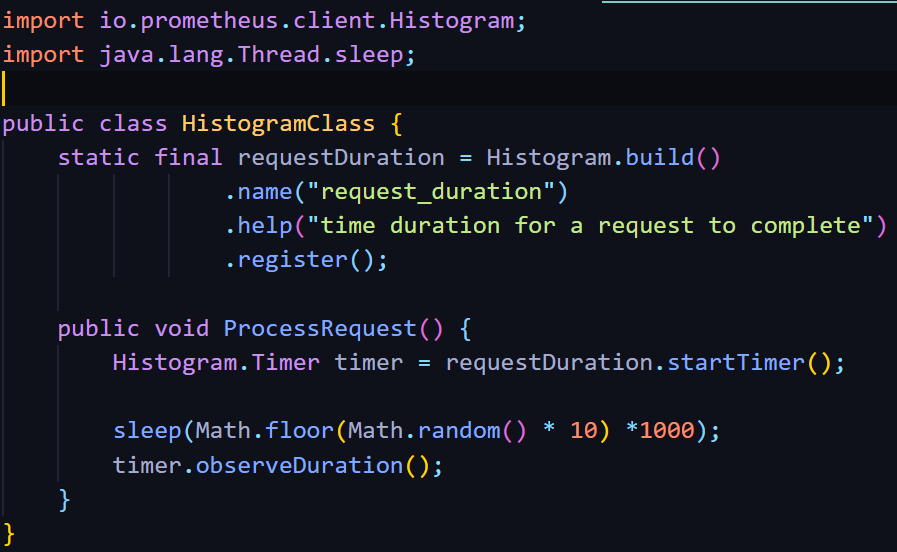


**Histogram:**

Java client for histogram.

1. **Histogram.build()** --> builder method.
2. **public Timer startTimer()** --> which returns a Histogram.Timer object.
3. **public void observe(double amt)** -->which will record whatever double value you specify
4. **public double time(Runnable timeable)** --> executes the Runnable and measures how long it took to execute. The same definition also exists for Callable.

Code example:

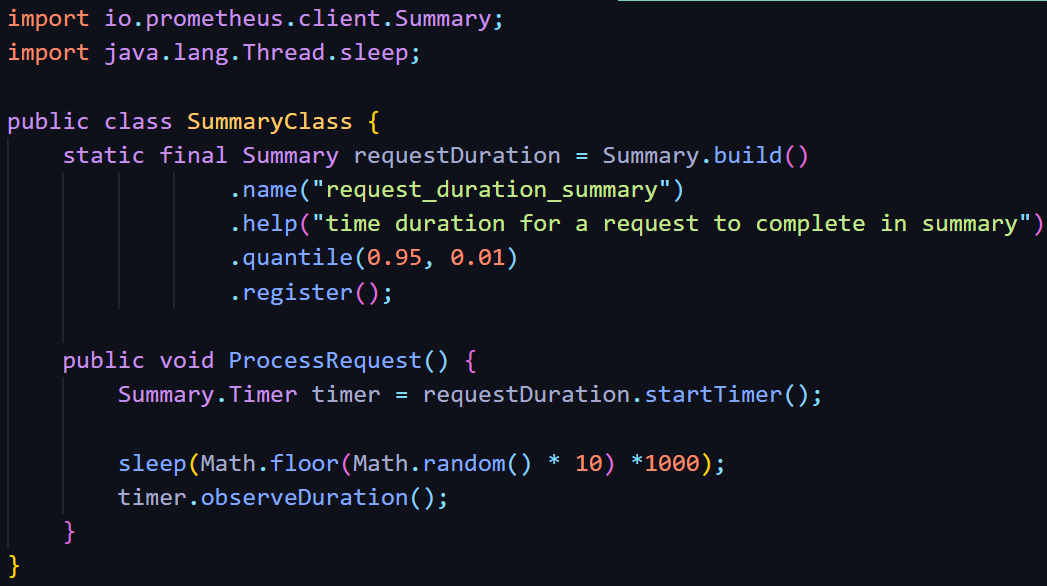


**Summary:**

Java client for summary.

1. **Summary.build()** --> builder method.
2. **public Timer startTimer()** --> which returns a Summary.Timer object.
3. **public void observe(double amt)** -->which will record whatever double value you specify
4. **public double time(Runnable timeable)** --> executes the Runnable and measures how long it took to execute. The same definition also exists for Callable.

Code example:



**Prometheus installation:**

1. Download Prometheus from [prometheus.io](https://prometheus.io/download/) website.
2. Extract the ZIP file in preferred directory.

**Running the Prometheus**.

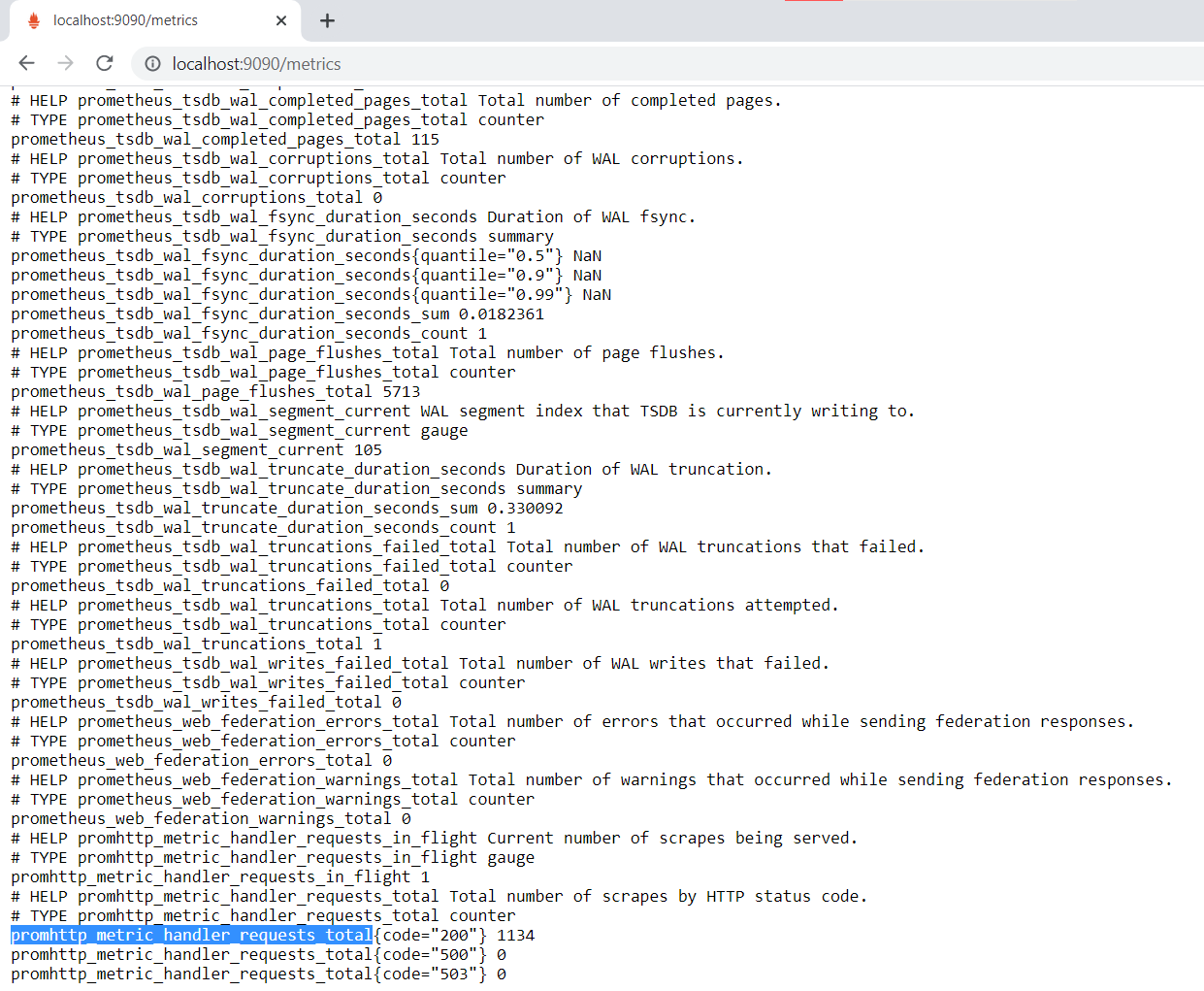
1. In Command Prompt move to the location where Prometheus file present.
2. Run the following command **prometheus --config.file=prometheus.yml**



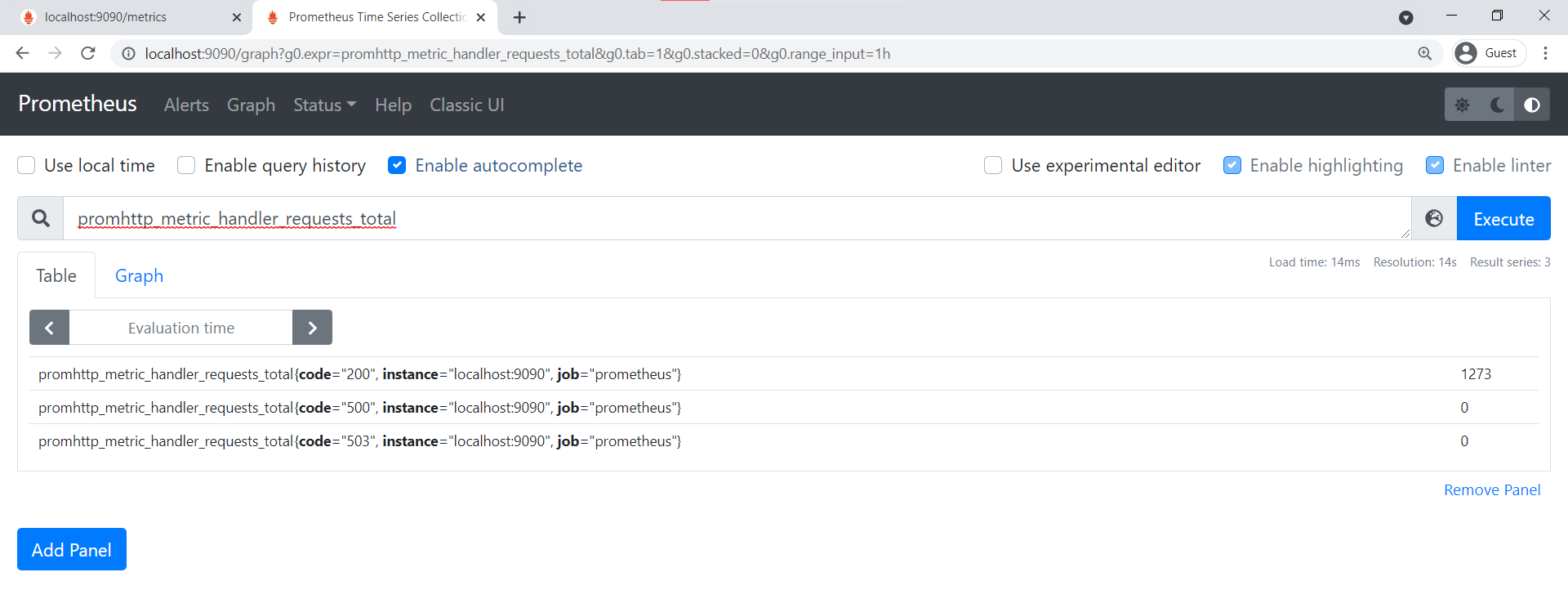
1. Now open localhost:9090/metrics and localhost:9090 in different tabs.

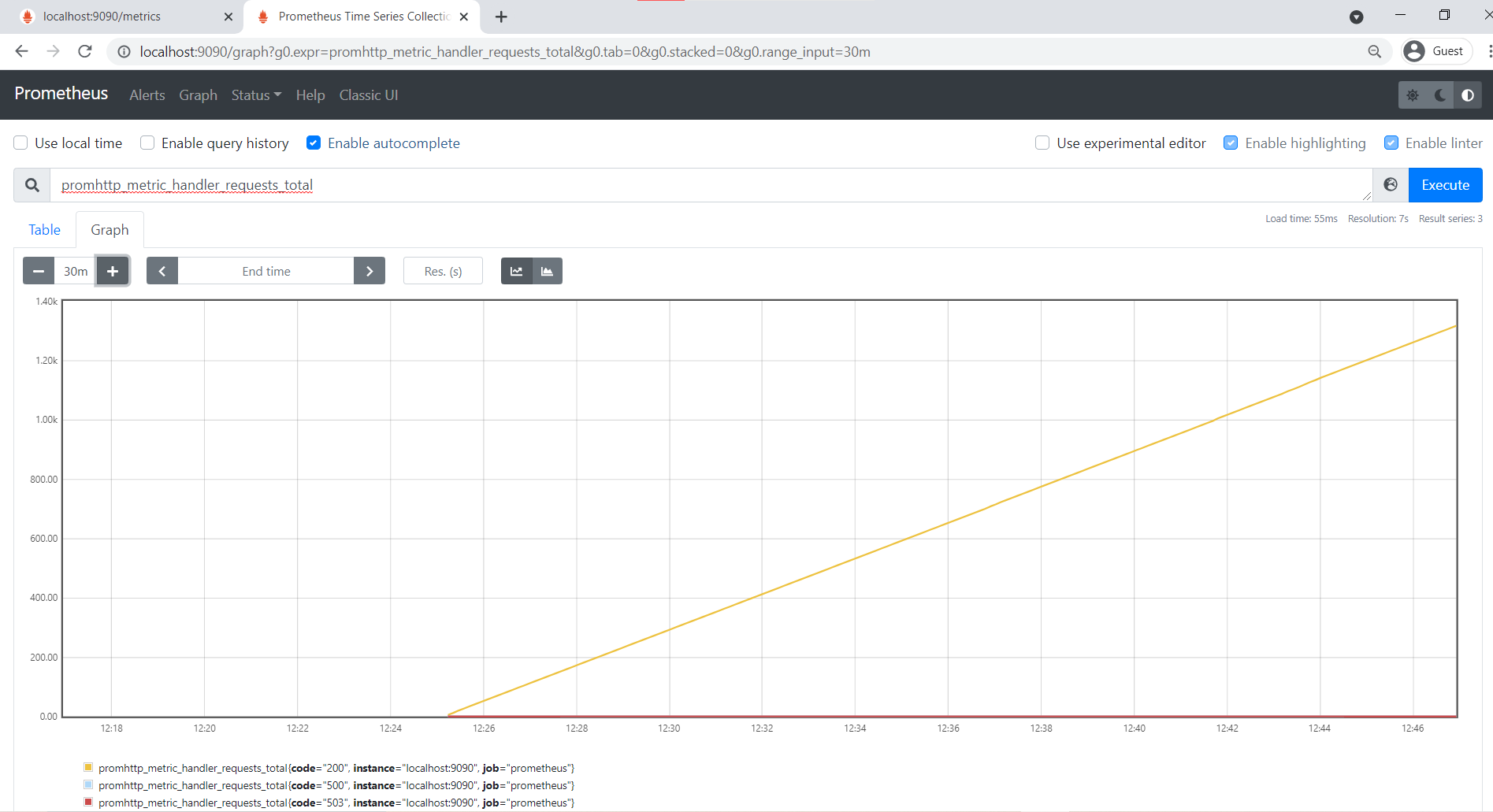
**Prometheus metrics first look:**

* Prometheus scraps metrics from own end point(localhost:9090).
* In localhost:9090/metrics you can see bunch of lines, which is wide variety of host-level metrics about memory usage, garbage collection, and more,. scraped from endpoint.
* Prometheus written in Go language hence what you seeing in 9090/metrics endpoint is represent in Go language.
* You can find **promhttp\_metric\_handler\_requests\_total** inbottom of the page. Copy this and move to localhost:9090/graph



* Paste the expression **promhttp\_metric\_handler\_requests\_total** in Expression input box and click execute button.
* Now move to graph tab.
* You can visualize the metrics scraped by Prometheus from its endpoint in graph as time series format.





**My work in Prometheus: Populate time series data in Prometheus using java.**

To Populate time series data, weather report for four cities over the year 1995 to 2019 is taken.

Jar files Required.

1. simpleclient\_hotspot-0.11.0.jar (Used to initialize default Hotspot collectors with the default registry)
2. simpleclient\_httpserver-0.11.0.jar (Used to create server for specified PORT)
3. simpleclient-0.11.0.jar (Here all prometheus client java metrics library present)
4. opencsv-5.4.jar (Used to read CSV file in java)

Weather is defined in a simple way as changing in temperature over the time period.

For visualizing weather data in Prometheus, I use Gauge metrics to populate weather data in prometheus. Why Gauge metrics, temperature value goes up and down it not only increase or decrease. Counter cannot be used because Counter only get incremented or reset to zero. Hence temperature value is set to Gauge metric. Prometheus scrap the metric at given scarp interval.

Java code example:

**package** com.zc.prometheus;

**import** java.io.FileReader;

**import** java.io.IOException;

**import** io.prometheus.client.Gauge;

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

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

**import** java.util.\*;

**import** java.util.concurrent.TimeUnit;

**import** com.opencsv.CSVReader;

**import** com.opencsv.CSVReaderBuilder;

**public** **class** CitiesTemperatureVisualization {

**static** **final** Gauge ***temparatureGauge*** = Gauge.*build*()

.name("cities\_temparature\_info").help("Temparature visualization for four city"

+ "[\"Chennai\", \"Kolkata\", \"Delhi\", \"Mumbai\"] from 1995 to 2019")

.labelNames("City", "Date", "Month", "Year")

.register();

**public** **static** String[] *cities* = {"Chennai", "Kolkata", "Delhi", "Mumbai"};

**public** **static** **void** main(String[] args) {

DefaultExports.*initialize*();

HTTPServer server = **null**;

**try** {

server = **new** HTTPServer(1230);

}

**catch** (IOException e) {

e.printStackTrace();

}

String[][] darr = **new** String[9200][7];

**int** i=-1, j=-1;

**float** df;

**try** {

String file = "F:\\temperature\\Cities.csv";

FileReader filereader = **new** FileReader(file );

CSVReader csvReader = **new** CSVReaderBuilder(filereader)

.withSkipLines(1)

.build();

List<String[]> allData = csvReader.readAll();

**for** (String[] row : allData) {

++i;

j =-1;

**for** (String cell : row) {

**if**(cell !=**null**)

darr[i][++j] = cell.trim();

}

}

}

**catch** (Exception e) {

e.printStackTrace();

}

**for**(i=0; i<darr.length; ++i) {

**for**(j=0; j<7; j++) {

**if**(j > 2 && darr[i][j] != **null**) {

df = Float.*parseFloat*(darr[i][j]);

System.***out***.println(df);

*Cities*(darr[i][0], darr[i][1], darr[i][2], *cities*[j-3], df);

}

}

**try** {

Thread.*sleep*(1);

}

**catch** (InterruptedException e) {

e.printStackTrace();

}

}

System.***out***.println("Completed");

**try** {

TimeUnit.***DAYS***.sleep(1);

} **catch** (Exception e) {

e.printStackTrace();

}

server.stop();

}

**public** **static** **void** Cities(String month, String date, String year, String city, **float** temparature) {

***temparatureGauge***.labels(city, date, month, year).set(temparature);

}

}

What I done in the code?

* Read all four city temperature data from CSV file and stored it in two dimensional string array.
* Created a server in PORT 1230. Prometheus scraps the metrics from this end point.
* For this what server we creating, will be registered in prometheus.yml file.
* labelnames are used to identify specific value in prometheus graph. In this example Date, Month, Year and City label are used.
* After this temperature value is set into gauge metric by giving different label values for each date, month, year and city. In our case we have data from 1995 to 2019, hence for each year 365 days temperature for four cities is set into gauge metric.

Expression or query to visualize the temperature in prometheus.

1. **cities\_temparature\_info --->** to visualize all the temperature.
2. **cities\_temparature\_info{City=”Chennai”, Date=”21” Month = “1”, Year = ”2000”} --->** to visualize temperature for specific date, month and year.

Prometheus.yml file

global:

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

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

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

alerting:

alertmanagers:

- static\_configs:

- targets:

# - alertmanager:9093

rule\_files:

# - "first\_rules.yml"

# - "second\_rules.yml"

scrape\_configs:

- job\_name: 'prometheus'

static\_configs:

- targets: ['localhost:9090']

- job\_name: 'cityTemperature'

static\_configs:

- targets: ['localhost:1230']

**NOW WE GOING TO FETCH TEMPERATURE DATA FROM PROMETHEUS FOR PARTICULAR TIME RANGE IN JAVA**

Time range Ex: Chennai: 21/5/2010 to 16/9/2017

To fetch data from prometheus first need to create HTTP client connection tool. In url we cannot give special characters directly hence need to build the url for specific parameter value. After carry out the request, the response is in string format hence need to parse the string to JSONObject. After parsing the JSON object. Data is easily accessible. Json object contains data in the form of key value pair. All the metrics fetched from request is in data key part and values as array representation. So these values are stored in JSONArray to access easily.

Jar files Required.

1. **httpclient-4.5.13.jar** (Used to create HTTP connection)
2. **httpcore-4.4.14.jar** (Used to build custom client and server side HTTP service)
3. **fastjson-1.1.15.jar** (Used to access JSON related libraries)
4. **commons-logging-1.2.jar** (Used for different logging implementations)

Java code example:

**package** com.zc.prometheus;

**import** java.io.IOException;

**import** java.net.URISyntaxException;

**import** java.util.Scanner;

**import** org.apache.http.ParseException;

**import** org.apache.http.client.methods.CloseableHttpResponse;

**import** org.apache.http.client.methods.HttpGet;

**import** org.apache.http.client.utils.URIBuilder;

**import** org.apache.http.impl.client.CloseableHttpClient;

**import** org.apache.http.impl.client.HttpClients;

**import** org.apache.http.util.EntityUtils;

**import** com.alibaba.fastjson.JSONArray;

**import** com.alibaba.fastjson.JSONObject;

**public** **class** ListTemperatureGivenRange {

**static** String *parameterValue*="cities\_temparature\_info";

**static** CloseableHttpClient *httpClient*=HttpClients.*createDefault*();

**static** URIBuilder *uri*=**null**;

**static** HttpGet *get* =**null**;

**static** CloseableHttpResponse *response*=**null**;

**static** JSONObject *jsonObject*=**null**;

**static** JSONObject *metric* = **null**;

**static** JSONArray *value* = **null**;

**static** JSONArray *result* = **null**;

**static** **int**[] *monthTotalDate* = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

**static** **int** *startYear*, *startMonth*, *startDate*, *endYear*, *endMonth*, *endDate*, *eM*=12, *tempEndDate*;

**static** String *City*;

**public** **static** **void** alert() {

System.***out***.println("<------------------\*------------------>");

System.***out***.println("Give any one of the City \"Name\" from the below list");

System.***out***.println("1. Chennai\n2. Kolkata\n3. Delhi\n4. Mumbai\n");

System.***out***.println("NOTE: You only given range of year from:");

System.***out***.println("1995 to 2019");

System.***out***.println("NOTE: Dont give month as Jan Dec etc.. Month are only represented by\n1,2,3,....12");

System.***out***.println("Leap year also included given range according to that\n");

System.***out***.println("<----------------###--------------------->\n");

}

**public** **static** **void** reset() {

**int**[] tempMonthTotalDate = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

*monthTotalDate* = tempMonthTotalDate;

*eM*=12;

}

**public** **static** **void** isLeapYear(**int** year) {

**if**( ((year % 4 == 0) && (year % 100 != 0)) || (year % 400 == 0) ){

**if**(*tempEndDate* != *monthTotalDate*[1])

*monthTotalDate*[1] = 29;

}**else** {

**if**(*tempEndDate* != *monthTotalDate*[1])

*monthTotalDate*[1] = 28;

}

}

**public** **static** **void** ListTemperature() {

**try** {

**do** {

*isLeapYear*(*startYear*);

**for**(**int** j = *startMonth*; j<=*eM*; ++j) { // month loop

**for**(**int** k=*startDate*; k<=*monthTotalDate*[j-1]; ++k) { // date loop

*parameterValue* = "cities\_temparature\_info{City='" + *City* + "',Date='"+k

+ "',Month='"+j+ "',Year='"+*startYear*+"'}";

*uri*=**new** URIBuilder("http://127.0.0.1:9090//api/v1/query");

*uri*.addParameter("query",*parameterValue*);

*get*=**new** HttpGet(*uri*.build());

*response* = *httpClient*.execute(*get*);

String resStr= EntityUtils.*toString*(*response*.getEntity(),"UTF-8");

*jsonObject*=JSONObject.*parseObject*(resStr);

*result*=*jsonObject*.getJSONObject("data").getJSONArray("result");

**for**(**int** l=0;l<*result*.size();l++){

JSONObject metric = *result*.getJSONObject(l).getJSONObject("metric");

*value* = *result*.getJSONObject(l).getJSONArray("value");

System.***out***.println(metric.get("Date")+"\t"+metric.get("Month")+"\t"

+metric.get("Year")+"\t"+metric.get("City")+"\t"+*value*.getString(1));

}

}

*startDate* = 1;

}

*startMonth* = 1;

**int** t = ++*startYear*;

**if**(*startYear* < *endYear*)

*startYear* = t;

**else**

*startYear* = 0;

*tempEndDate* = 0;

} **while** (*startYear* != 0);

}

**catch** (URISyntaxException | ParseException | IOException e) {

e.printStackTrace();

}

}

@SuppressWarnings("resource")

**public** **static** **void** main(String[] args) **throws** InterruptedException {

String[] errorArr = {"Year must in the range 1995 to 2019 and end year not less than start year",

"Month must in the range 1 to 12",

"Date must be in the range of 1 to 31 or 1 to 29 for leap year"};

**boolean** flag = **false**;

String repeat = "N";

**int** tempStartYear = 0;

Scanner scan = **new** Scanner(System.***in***);

*alert*();

**do** {

flag = **false**;

System.***out***.println("Remember \"City name\" is \"Case Sensitive\" "

+ "Must start with \"UpperCase\" followded by \"LowerCase\" letters\n");

System.***out***.println("Enter City Name:");

*City* = scan.next();

**if**(*City*.equals("Chennai") || *City*.equals("Kolkata")|| *City*.equals("Mumbai") || *City*.equals("Delhi")) {

System.***out***.println("Enter Starting year: ");

*startYear* = scan.nextInt();

**if**(*startYear* < 1995 || *startYear*> 2019) {

System.***out***.println(errorArr[0]);

}

**else** {

*isLeapYear*(*startYear*);

tempStartYear = *startYear*;

System.***out***.println("Enter starting month");

*startMonth* = scan.nextInt();

**if**(*startMonth* < 1 || *startMonth* > 12)

System.***out***.println(errorArr[1]);

**else** {

System.***out***.println("Enter starting date");

*startDate* = scan.nextInt();

**if**( *startDate* < 1 || *startDate* > *monthTotalDate*[*startMonth*-1])

System.***out***.println(errorArr[2]);

**else** {

System.***out***.println("Enter End Year");

*endYear* = scan.nextInt();

**if**(*endYear* < 1995 || *endYear*> 2019 || *endYear* < *startYear*)

System.***out***.println(errorArr[0]);

**else** {

System.***out***.println("Enter End Month");

*isLeapYear*(*endYear*);

*endMonth* = scan.nextInt();

**if**(*endMonth* < 1 || *endMonth* > 12)

System.***out***.println(errorArr[1]);

**else** {

System.***out***.println("Enter End date");

*endDate* = scan.nextInt();

**if**(*endDate* < 1 || *endDate* > *monthTotalDate*[*endMonth*-1] )

System.***out***.println(errorArr[2]);

**else** {

**if**(*startYear* == *endYear* && *startMonth* == *endMonth* && *endDate* < *startDate*)

System.***out***.println("\nYou entered same month and

year\nThen end date cannot be less than start date\n");

**else**

flag = **true**;

}

}

}

}

}

}

**if**(flag) {

System.***out***.println(*City*+": "+*startDate*+"/"+*startMonth*+"/"+*startYear*+" to "

+*endDate*+"/"+*endMonth*+"/"+*endYear*);

Thread.*sleep*(1000);

System.***out***.println("\nDate"+"\t"+"Month"+"\t"+"Year"+"\t"

+"City"+"\t"+"Temperature");

**if**(*endYear* == *startYear*) {

*eM* = *endMonth*;

*monthTotalDate*[*eM*-1] = *endDate*;

*tempEndDate* = *endDate*;

}

*ListTemperature*();

*reset*();

**if**(*endYear* != tempStartYear) {

*startMonth* = 1;

*startDate* = 1;

*eM* = *endMonth*;

*monthTotalDate*[*eM*-1] = *endDate*;

*tempEndDate* = *endDate*;

*startYear* = *endYear*;

*ListTemperature*();

}

*reset*();

}

}

**else**

System.***out***.println("Entered City, not in the list. Also check, \"City Name\" is Case Sensitive");

System.***out***.println("<--------------------\*---------------->");

System.***out***.println("Enter 'Y' to Continue or 'N' to Terminate");

repeat = scan.next();

} **while** (repeat.toUpperCase().equals("Y"));

}

}

What I done in the code?

* Getting range input from user for different cities.
* Checking the range must match the given condition. If not error will be given and again get the value from user.
* After checking the condition, for the given range, metrics is fetched from prometheus and displayed in console.
* Additionally leap year condition, date must not exceed from given month range and etc…, validated in the code while getting input from user.
* Over 30 test cases tested and temperature value cross checked with loaded temperature value from CSV file.

**All code, Temperature data (in CSV file) used in this project and output screen record available in GitHub repository.**

Link to repository: **[TemparatureVisualizationInPrometheus](https://github.com/RAGHUPRIYANTHRAMASAMY/TemparatureVisualizationInPrometheus)**

Click above link to see the code in repository.