# Building a Factory machine/sensor monitoring IoT pipeline. From source through to Prometheus and visualized in Grafana.

IoT sensor readings in JSON format pushed into Kafka Topic, extracted using Flink Kafka connector, flattened and into Fluss, inserted into Apache Flink tables configured with a Prometheus sink connector (BETA).

(9 June 2025)

# Overview

In a previous [blog](https://medium.com/@georgelza/a-factory-iot-instrumentation-data-stream-from-source-manipulated-using-pyflink-outputting-into-182c1ab85a7e) we explored manipulating our datastream using [Apache Flink’s](https://flink.apache.org/) Pyflink capability, with our data ending in our Fluss store.

This time round I want to build something a bit more directly related to a real-world use case.

We’re taking a “[JSON](https://www.json.org/)” document representing a reading/measurement, structured as:

{

timestamp,

metadata {

labels1

labels2

labels3

},

measurement

}

for a sensor fitted to a machine, located in a factory and storing pushing the values into a [Prometheus TSDB](https://prometheus.io/), finally to be visualized via [Grafana](https://grafana.com/).

Well, that’s the 100 000 foot view.

Now if we drill down a bit, we will first publish the readings onto 3 Apache Kafka Topics. From here we will source from the topics using Apache Flink tables configured using Apache Flink connector.

We will then flatten the nested structure of the IoT structure and store the flattened payload into a Fluss table.

Next up we will source from our [Fluss](https://alibaba.github.io/fluss-docs/) table and sink into our [Prometheus](https://prometheus.io/) TSDB using our connector.

Lastly, we will use [Grafana](https://grafana.com/) to display our metrics.

## Regarding our data.

I grouped the factories (referred to as sites) into 3 sets:

* North (siteId 101 and 104)
* South (siteId 102 and 105)
* East (siteId 103 and 106)

(simulating a sort of regional distribution, depending where in the world you are, think states, provinces etc.). Each factory (site) has multiple devices, aka machines, each machine will have multiple sensors.

## Regarding Stack:

We have a [Confluent Kafka Cluster](https://www.confluent.io/), a Apache Flink cluster and a Fluss cluster environment, all as per the previous [blog](https://medium.com/@georgelza/a-factory-iot-instrumentation-data-stream-from-source-manipulated-using-pyflink-outputting-into-182c1ab85a7e).

Regard our Fluss stack. I simply did not configure (run) the lakehouse sink to our deep storage, based [Apache Paimon](https://paimon.apache.org/) open tables format stored on HDFS layer, for these labs as we’ve shown that working in the previous blog.

Our data is again generated by our [Python](https://www.python.org/) application, the main code base is location in <*root>/app\_iot1,* which can be run via the following 3 shell scripts:

* *<root>/app\_io1/site1.sh*
* *<root>/app\_io2/site2.sh*
* *<root>/app\_io3/site3.sh*
* The first *app\_iot1* creates our simplest IoT JSON payload, for the north located sites
  + This is accomplished by setting *TSHUMAN, STRUCMOD & DEVICETYPE* = 0
* In *app\_iot2* we extend the payload to include *TSHUMAN*=1 and *STRUCMOD*=1, for the south located sites.
  + This adds a human readable date field and the location object to the metadata tag section.
* In *app\_iot3* we go one step further and add *DEVICETYPE=1* to the payload, for our east located sites.
  + This adds a text string defining the device type.

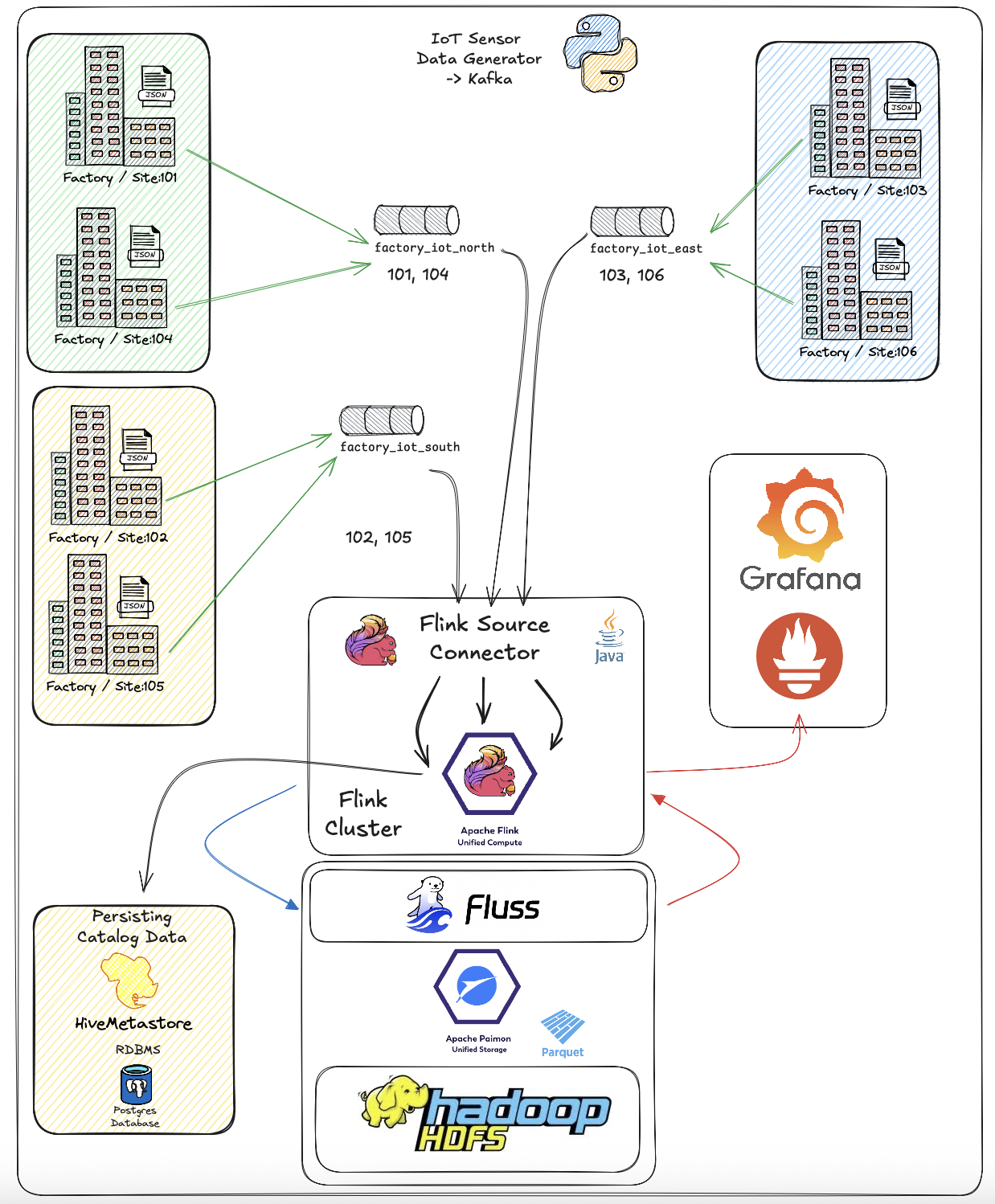
For catalog services we will be using the [Apache Hive](https://hive.apache.org/)’s and their [Metastore](https://cwiki.apache.org/confluence/display/hive/design#Design-Metastore) functionality as created in a previous blogs (but with a little version update applied recently).

For those that have been following my previous blogs, you will notice I’ve upgraded my [Confluent](https://www.confluent.io/) Kafka Cluster (now 7.9.1) and the [Apache Flink](https://flink.apache.org/) environment (now 1.20.1). The [Apache Paimon](https://paimon.apache.org/) stack has also been upgraded to 0.9.0, Fluss 0.6.0.

For this build we now also have [Prometheus](https://prometheus.io/) (v3.3.0) and [Grafana](https://grafana.com/) (11.6.1)

As always, all the code can be found in the [GIT repository](https://github.com/georgelza/DataPipeline-Kafka_Fluss_Prometheus), and yes, we’re still using a substantial amount of *Makefiles*, *Docker-compose.yml* and *Dockerfile’s*.

## Simple Overview Diagram



Once the scaffolding is up, this is it, and this is all of it. Of course you can make it allot more complicated.

To start navigate to the *<root>* folder and read the *README.md* file. This will give a similar overview as per above, that will then direct you to build the basic scaffolding using (<*root>/infrastructure/),* after which you will be directed to *<root>/devlab0/README.md* instructing you how to build and run the various examples.

## Regarding our payloads

Our first region is been naughty, they are populating the absolute minimum tags of our IoT JSON document.

Executing */<root>/app\_iot1/site1.sh>* will thus produce the below document.

{

"ts" : 123421452622,

"metadata" : {

"siteId" : 1009,

"deviceId" : 1042,

"sensorId" : 10180,

"unit" : "Psi"

},

"measurement" : 1013.3997

}

Below we have the second region, they seem to have read the documentation/expectations a bit more and populated more of the fields of our IoT JSON document.

Region South data is generated using *<root>/app\_iot2/site2.sh* application. This as you can see below adds the *ts\_human* and *location* object to the payload.

{

"ts" : 123421452622,

"metadata" : {

"siteId" : 1009,

"deviceId" : 1042,

"sensorId" : 10180,

"unit" : "Psi",

"ts\_human" : "2024-10-02T00:00:00.869Z",

"location": {

"latitude": -26.195246,

"longitude": 28.034088

}

},

"measurement" : 1013.3997

}

And lastly as we’re feeling very lucky, the East Region paid attention, and they are providing us a complete IoT Document.

To generate their payload, execute *<root>/app\_iot3/site3.sh* application which will add the *deviceType* field to the metadata tag.

{

"timestamp" : "2024-10-02T00:00:00.869Z",

"metadata" : {

"siteId" : 1009,

"deviceId" : 1042,

"sensorId" : 10180,

"unit" : "Psi",

"ts\_human" : "2024-10-02T00:00:00.869Z",

"location": {

"latitude": -26.195246,

"longitude": 28.034088

},

"deviceType" : "Oil Pump",

},

"measurement" : 1013.3997

}

So we now have an overview of the 3 payloads and how to generate them at the source end.

We will have 2 catalogs, namely:

* *hive\_catalog* with a *kafka* and *prometheus* database and then
* *fluss\_catalog* containing the *fluss* database.

The Apache Flink Cluster to Fluss cluster link is accomplished by specifying the *bootstrap.servers = ‘coordinator-server:9123’* in the catalog definition as per below:

CREATE CATALOG fluss\_catalog WITH (

'type' = 'fluss',

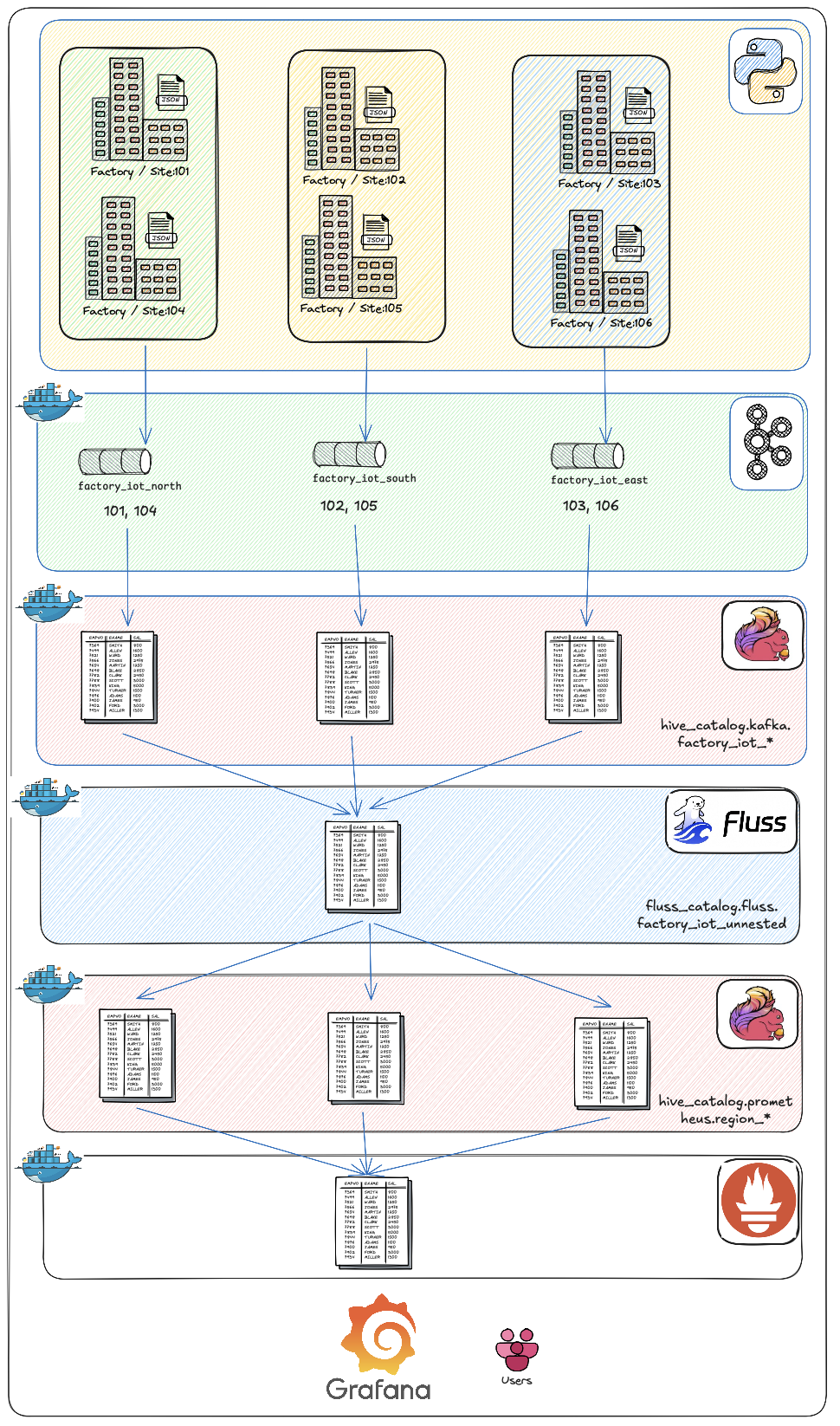
'bootstrap.servers' = 'coordinator-server:9123'

);

We have primarily 3 steps in ours pipeline;

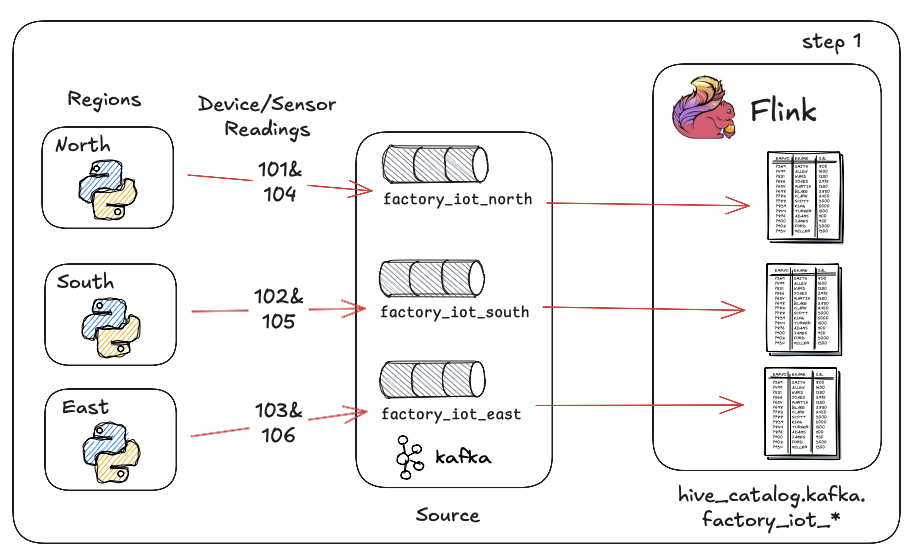
* Ingest into Apache Flink from Kafka via Kafka Connector
* Flatten into Fluss table
* Insert into Flink Table representing sink into Prometheus

## Top to Bottom.



# Kafka Source into Apache Flink (step 1):

This is our Source step.



This is accomplished using the following Flink SQL. You will find there are 3 create sections in the below SQL script. One for each region.

See: <*root>/devlab0/creFlinkFlows/2.1.creKafkaSource.sql*

CREATE OR REPLACE TABLE hive\_catalog.kafka.factory\_iot\_north (

ts BIGINT

,metadata ROW<

siteId INTEGER

,deviceId INTEGER

,sensorId INTEGER

,unit STRING

,ts\_human STRING

,location ROW<

latitude DOUBLE

,longitude DOUBLE

>

,deviceType STRING

>

,measurement DOUBLE

,ts\_wm AS TO\_TIMESTAMP\_LTZ(ts, 3)

,WATERMARK FOR ts\_wm AS ts\_wm - INTERVAL '1' MINUTE

) WITH (

'connector' = 'kafka',

'topic' = 'factory\_iot\_north',

'properties.bootstrap.servers' = 'broker:29092',

'properties.group.id' = 'devlab0',

'scan.startup.mode' = 'earliest-offset',

'format' = 'json',

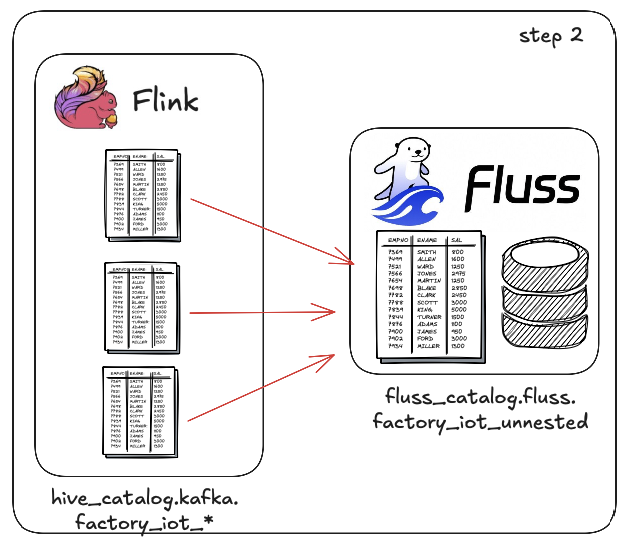
'json.fail-on-missing-field' = 'false',

'json.ignore-parse-errors' = 'true'

);

# Convert/Flatten into Fluss table (step 2):

Here we will flatten our structure into our Fluss environment. Even though we not deep storing the data into the lakehouse tier, this is an option in production, allowing for deep analytics, AI, ML etc. We can even use the Apache Flink, Pyflink capabilities as per the previous blog to do additional aggregation, or calling external API’s.



We first need to create our Fluss target table, this is accomplished using the following Flink SQL. For the Fluss tier we have a single table.

See: <*root>/devlab0/creFlinkFlows/2.2.creFlussTarget.sql*

CREATE OR REPLACE TABLE fluss\_catalog.fluss.factory\_iot\_unnested (

ts BIGINT

,siteId INTEGER

,deviceId INTEGER

,sensorId INTEGER

,unit STRING

,ts\_human STRING

,longitude DOUBLE

,latitude DOUBLE

,deviceType STRING

,measurement DOUBLE

,partition\_month STRING -- must be provided by upstream or insert logic

,ts\_wm AS TO\_TIMESTAMP\_LTZ(ts, 3)

,WATERMARK FOR ts\_wm AS ts\_wm - INTERVAL '1' MINUTE

) PARTITIONED BY (partition\_month) WITH (

'bucket.num' = '3'

,'table.datalake.enabled' = 'true'

,'table.auto-partition.time-unit' = 'MONTH'

,'table.auto-partition.num-retention'= '60'

);

We can now push from our step 1 tables into the Fluss table using the following Flink SQL. Again I’m only showing one insert, see the below script for the 3 insert statements.

See: <*root>/devlab0/creFlinkFlows/3.3.creFlussInserts.sql*

INSERT INTO fluss\_catalog.fluss.factory\_iot\_unnested

(ts, siteId, deviceId, sensorId, unit, ts\_human, longitude, latitude, deviceType, measurement, partition\_month)

SELECT

ts AS ts

,metadata.siteId AS siteId

,metadata.deviceId AS deviceId

,metadata.sensorId AS sensorId

,metadata.unit AS unit

,CAST(NULL AS STRING) AS ts\_human

,CAST(NULL AS DOUBLE) AS longitude

,CAST(NULL AS DOUBLE) AS latitude

,CAST(NULL AS STRING) AS deviceType

,measurement AS measurement

,DATE\_FORMAT(TO\_TIMESTAMP\_LTZ(ts, 3), 'yyyyMM') AS partition\_month

FROM hive\_catalog.kafka.factory\_iot\_north;

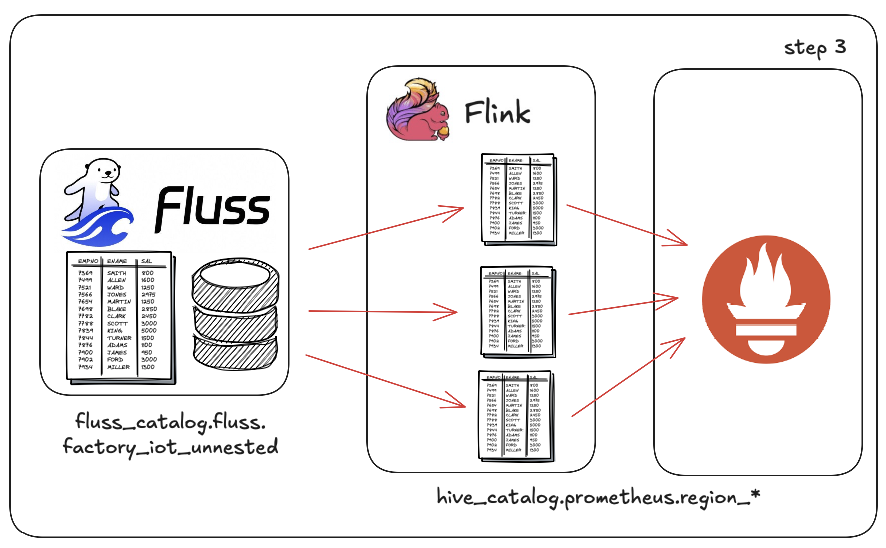
# Sink into Prometheus (step 3)

Some notes first. Our data generator can be configured to create historic data or not… Why the or not, well Prometheus don’t directly accept historic data. To disable our history data generate functionality see <*root>/app\_iot#/site#.sh*

Edit/set: *RUNHISTORIC=0*

This will disable our historic data generation loop.

If curious, with regard to loading historic data into your Prometheus environment see the <*root>/otl/README.md* file.



We first need to create our Prometheus target table/s, this is accomplished using the following Flink SQL (i.e: *region*\_*north* table), we have similar for “*region\_south*” and “*region\_east*” tables.

See: <*root>/devlab0/creFlinkFlows/2.2.crePromTarget.sql*

CREATE TABLE hive\_catalog.prometheus.region\_north (

`metric\_name` STRING

,`sensorId` INT

,`siteId` INT

,`deviceId` INT

,`measurement` DOUBLE

,`ts` TIMESTAMP(3)

) WITH (

'connector' = 'prometheus'

,'metric.endpoint-url' = 'http://prometheus:9090/api/v1/write'

,'metric.name' = 'metric\_name'

,'metric.label.keys' = '[sensorId,siteId,deviceId]'

,'metric.sample.key' = 'measurement'

,'metric.sample.timestamp' = 'ts'

);

If you look at the below SQL you will see we will have 3 separate Insert jobs, each specifying *sensor\_readings* as the metric name into the separate tables.

We do have the option to rather create a single target Prometheus table and then called the metrics:

* “*region\_north*” and changing the where to *siteId=101* and *siteId=104* and
* “*region\_south*” and change the where to *siteId=102* and *siteId=105* and
* “*region\_east*” and changed the where to *siteId=103* and *siteId=106*

or even split it further i.e.: and called the metric per site: “*sensor\_readings\_site01*”.

For now, for the blog we will run 3 Flink tables, one per region and for source use a single sites records from our *factory\_iot\_unnested* table located on our Fluss cluster.

See: <*root>/devlab0/creFlinkFlows/3.2.crePromInserts.sql*

INSERT INTO hive\_catalog.prometheus.region\_north

(metric\_name, siteId, deviceId, sensorId, measurement, ts)

SELECT

CAST('sensor\_reading' AS STRING) AS metric\_name

,siteId AS siteId

,deviceId AS deviceId

,sensorId AS sensorId

,measurement AS measurement

,TO\_TIMESTAMP(FROM\_UNIXTIME(ts / 1000)) AS ts

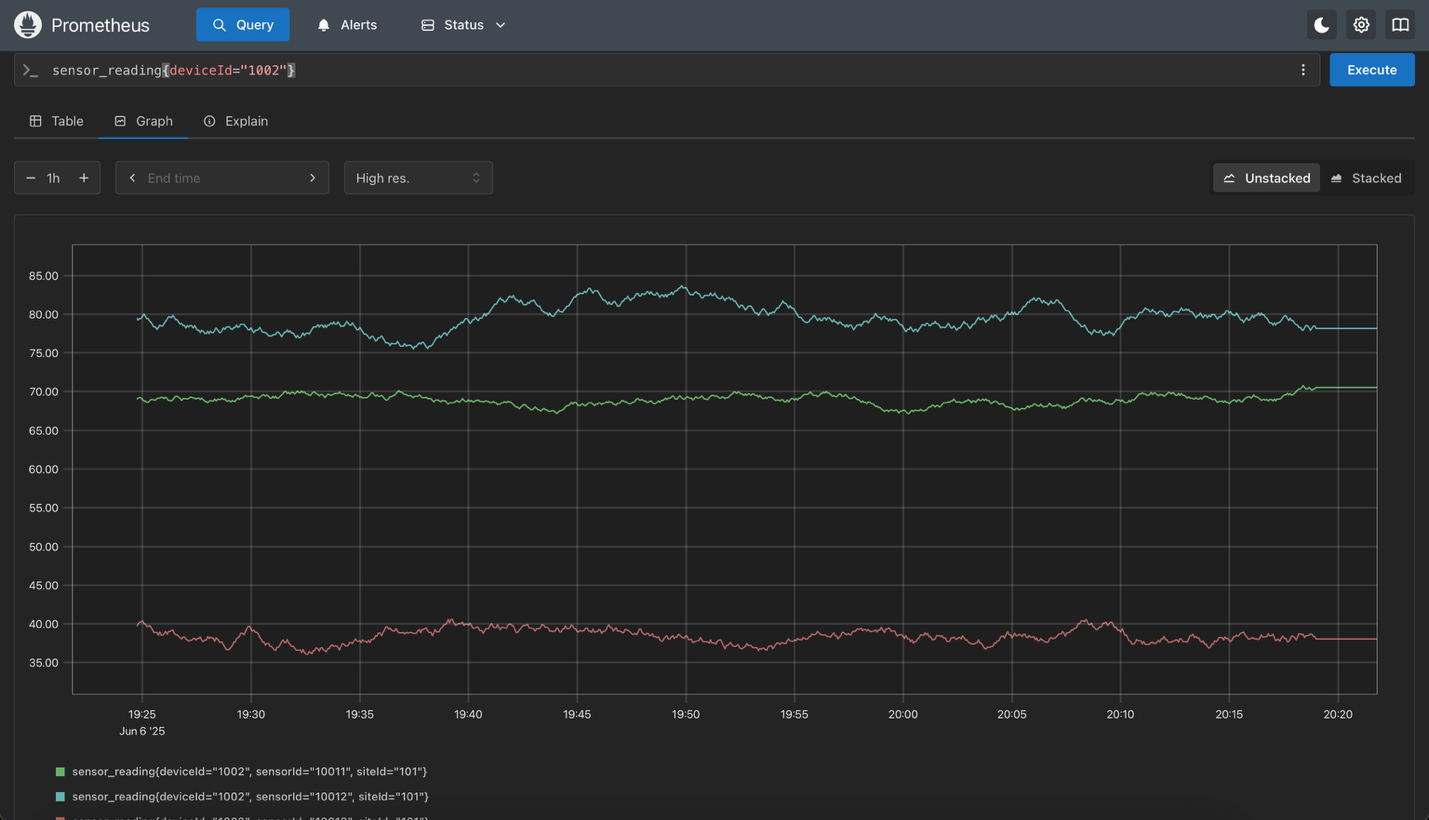
FROM fluss\_catalog.fluss.factory\_iot\_unnested

WHERE siteId=101;

Once you have the above configured you can navigate to <http://localhost:9090>. Click on graph tab and enter the text as per below and click execute.

# Prometheus

## Prometheus Graph – Device view



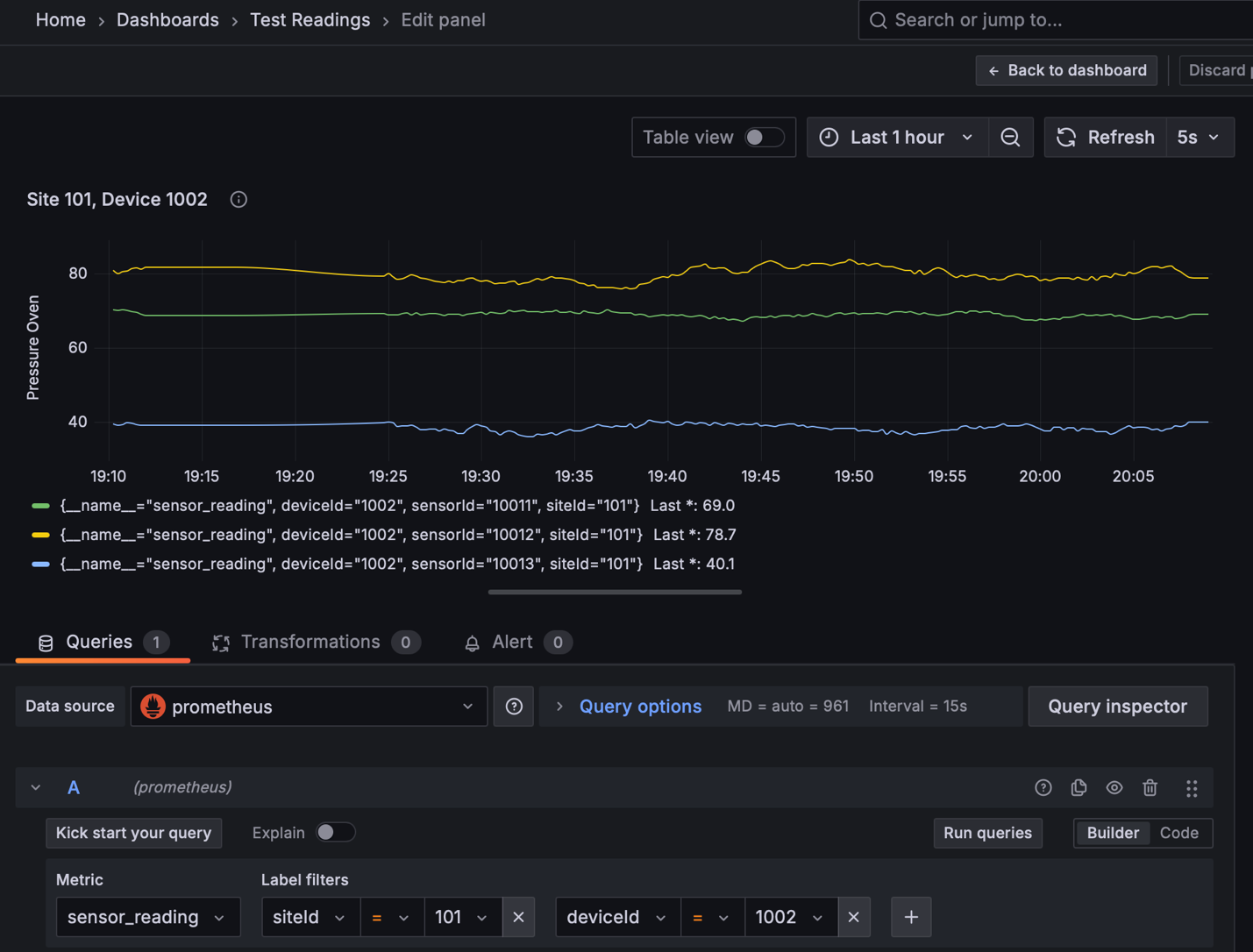
## Prometheus Graph – Sensor view

Once you have some data showing as per above you can click on one of the metric streams to drill down into that metric as show next.



# Grafana Dashboard

Lastly, I configured a Grafana dashboard (see <*root>/devlab0/Grafana/*) for the JSON file that can be imported into Grafana after you configured a “Prometheus” data source.





# And In Summary.

What can I stay, this one has been long in the making. It’s not totally completed yet. We have the tail end of the pipeline. The only missing part is the source. I still want to change the source side so that I rather push the metrics onto a Mosquito MQTT broker and have Apache Flink directly source from there.

I think this is pretty neat… Hope you enjoyed the exploration.

Good luck, this is all fraught with rabbit holes, as always, so many and you can disappear so easily… but then that’s ½ the fun.



*Note: to execute this blog start with README.md located in the root folder and work from there, it will tell you exactly what to execute in which order to download all the dependencies and build everything.*

**About Me**

I’m a techie, a technologist, always curious, love data, have for as long as I can remember always worked with data in one form or the other, Database admin, Database product lead, data platforms architect, infrastructure architect hosting databases, backing it up, optimizing performance, accessing it. Data data data… it makes the world go round.

In recent years, pivoted into a more generic Technology Architect role, capable of full stack architecture.

[George Leonard](https://www.linkedin.com/in/george-leonard-945b502/)

[georgelza@gmail.com](mailto:georgelza@gmail.com)

<https://www.linkedin.com/in/george-leonard-945b502/>

<https://medium.com/@georgelza>