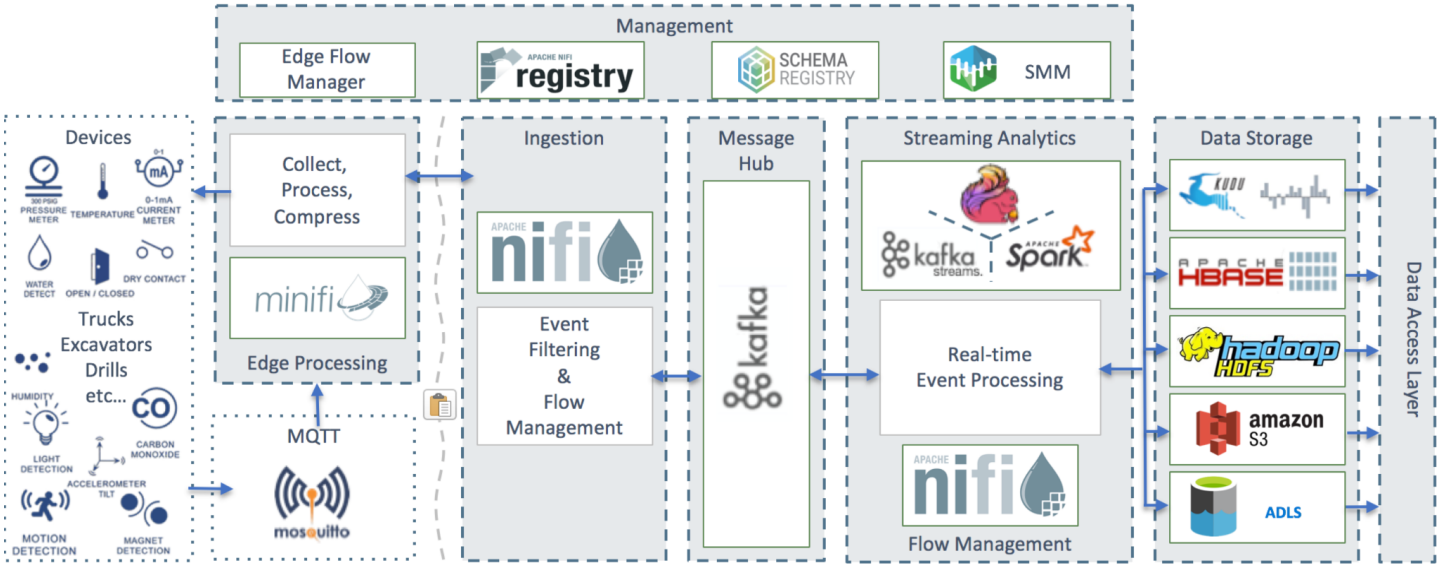
# CDF Workshop Student Guide

# Introduction

In this hands-on workshop, you will build a full OT to IT workflow for an IoT Predictive Maintenance use case. Below is the architecture diagram, showing all the components you will set up over the lab exercises. While the diagram divides the components according to their location (factory, regional or datacenter level) in this workshop all such components will reside in one single host.



# Labs summary

| S.No | Description |
|---|---|
| **Lab 1** | On the Apache NiFi, run a simulator to send IoT sensor data to the MQTT broker. |
| **Lab 2** | Create the MiNiFi flow on the Edge Flow Manager and publish it for the MiNiFi agent to start sending data to the NiFi cluster. |
| **LAB 3** | Configuring the NiFi flow and pushing data to Kafka |
| **LAB 4** | Streams Messaging Manager (SMM) to check and monitor Kafka |
| **LAB 5** | From Kafka to Kudu |
| **LAB 6** | Fast analytics on fast data with Kudu and Impala |

# Lab 0 - Initial setup

## Pre-requisites

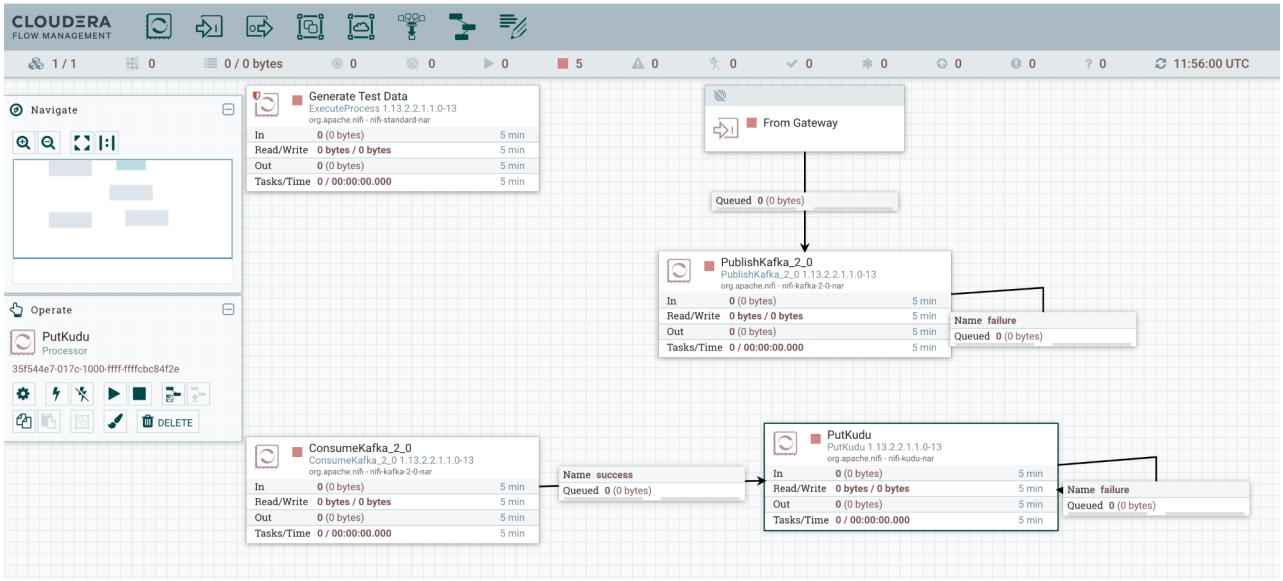
1. Laptop with a supported OS (Windows 7 not supported).
2. A modern browser like Google Chrome (IE not supported).

## Cloudera Manager and other services

Login into Cloudera Manager with username/password `admin/supersecret1`, and familiarize yourself with the services installed. The URLs to access the services are listed below

| Service | URL |
|---|---|
| Cloudera Manager | http://<your vm-public-IP>:7180 |
| Edge Flow Manager | http://<your vm-public-IP>:10088/efm/ui |
| NiFi | http://<your vm-public-IP>:8080/nifi/ |
| NiFi Registry | http://<your vm-public-IP>:18080/nifi-registry |
| Hue | http://<your vm-public-IP>:8888 |
| SMM | http://<your vm-public-IP>:9991/#/ |

Below is a screenshot of flow which we are going to define in this workshop.



Let's Begin...

# Lab 1 - Apache NiFi: setup machine sensors simulator

In this lab you will run a simple Python script that simulates IoT sensor data from some hypothetical machines, and send the data to a MQTT broker (mosquitto). The MQTT broker plays the role of a gateway that is connected to many and different types of sensors through the "mqtt" protocol. Your cluster comes with an embedded MQTT broker that the simulation script publishes to. For convenience, we will use NiFi to run the script rather than Shell commands.

The VM provided in this lab is already having mosquitto installed and configured.

Below are the outputs from one of such VM's.

```
$ systemctl status mosquitto
```

```
[root@cdp ~]# systemctl status mosquitto
● mosquitto.service - Mosquitto MQTT v3.1/v3.1.1 Broker
   Loaded: loaded (/usr/lib/systemd/system/mosquitto.service; enabled; vendor preset: disabled)
   Active: active (running) since Wed 2021-09-29 12:18:33 UTC; 6min ago
     Docs: man:mosquitto.conf(5)
           man:mosquitto(8)
 Main PID: 5317 (mosquitto)
   CGroup: /system.slice/mosquitto.service
           └─5317 /usr/sbin/mosquitto -c /etc/mosquitto/mosquitto.conf
```

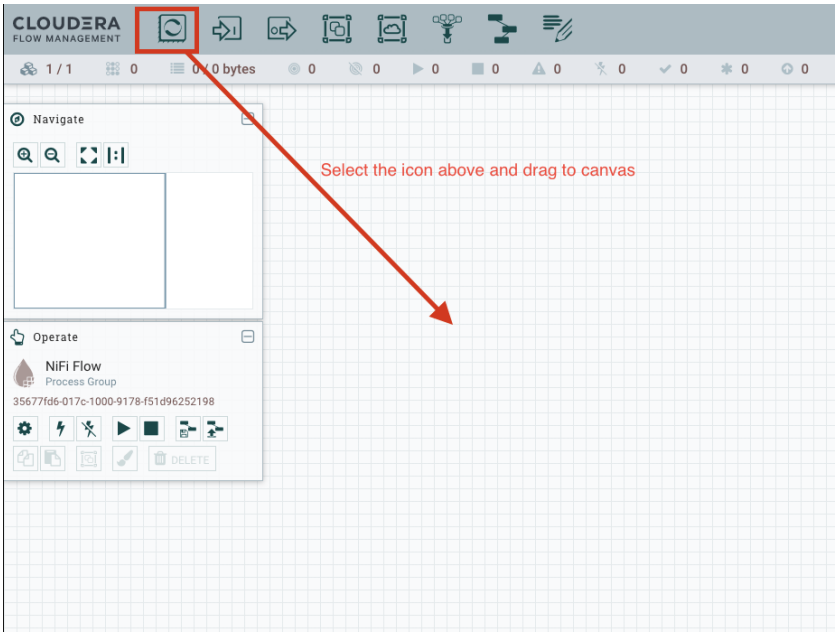
Below is the output of the sample simulator python programme.

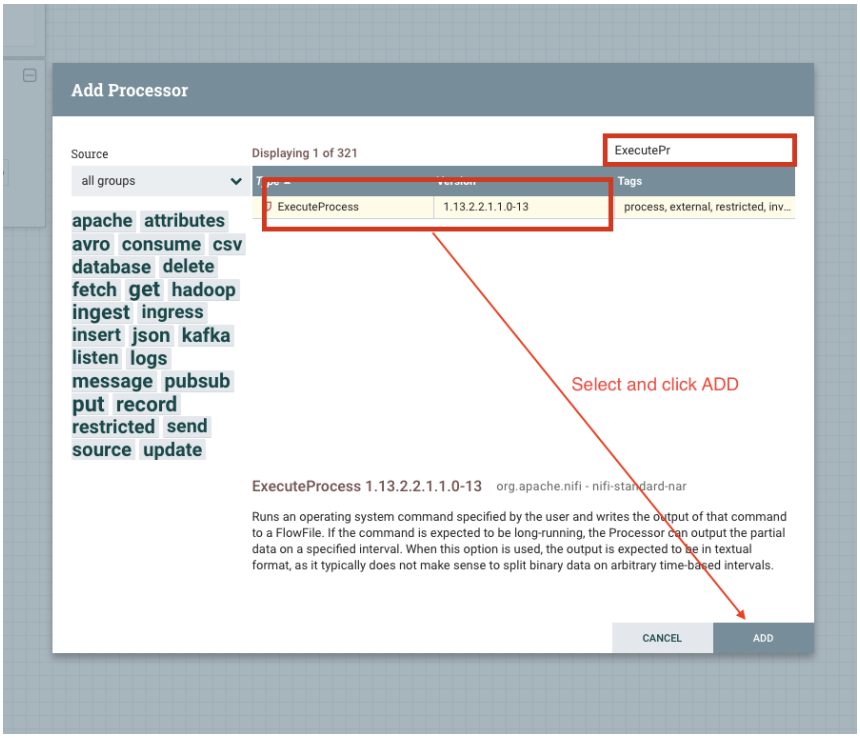
```
$ cd /opt/demo
$ python3 simulate.py
```

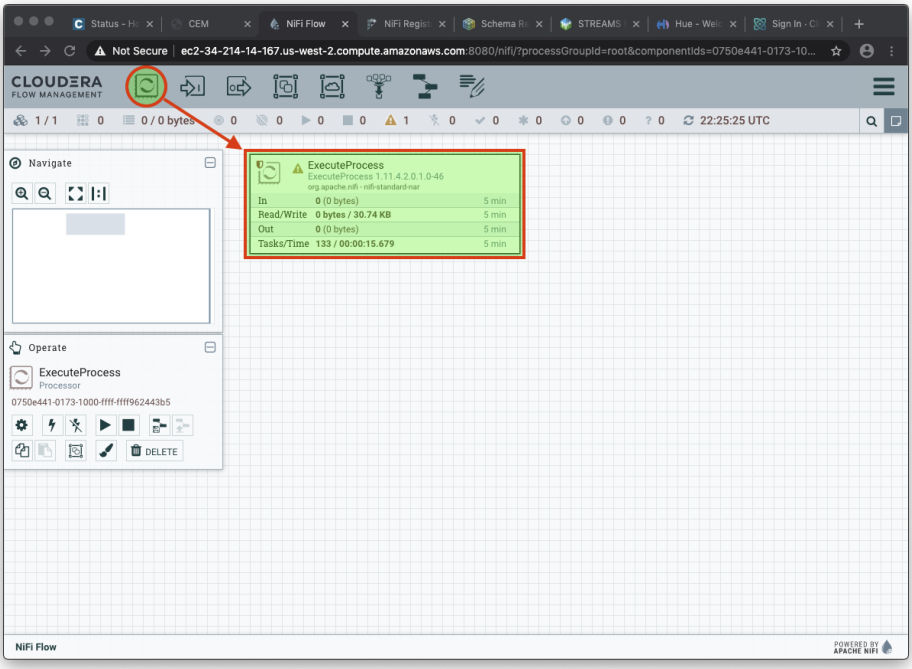
```
[centos@cdp demo]$ cd /opt/demo
[[centos@cdp demo]$ python3 simulate.py
[{"sensor_id": 98, "sensor_ts": 1632918949154217, "sensor_0": 2, "sensor_1": 5, "sensor_2": 2, "sensor
_3": 40, "sensor_4": 47, "sensor_5": 62, "sensor_6": 25, "sensor_7": 109, "sensor_8": 36, "sensor_9":
 7, "sensor_10": 5, "sensor_11": 10}
```

## Step 1: Setup the processors on NiFi

a. Go to Apache NiFi (http://<public-ip:8080/nifi) and add a Processor (ExecuteProcess) to the canvas.



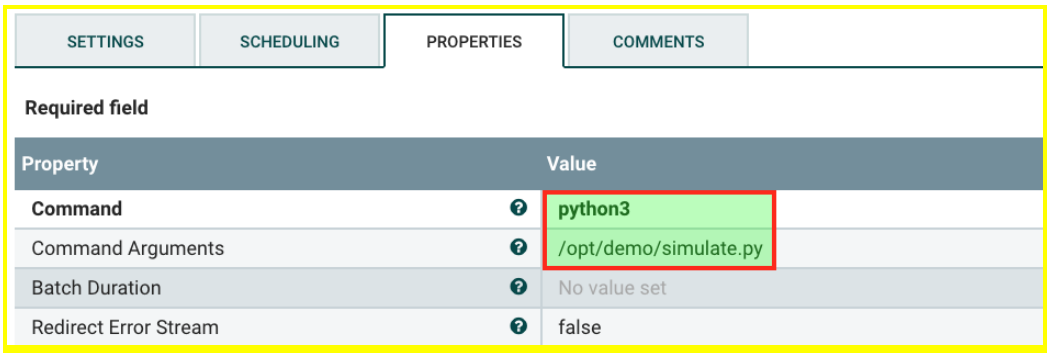



## Step 2: Configure the processor

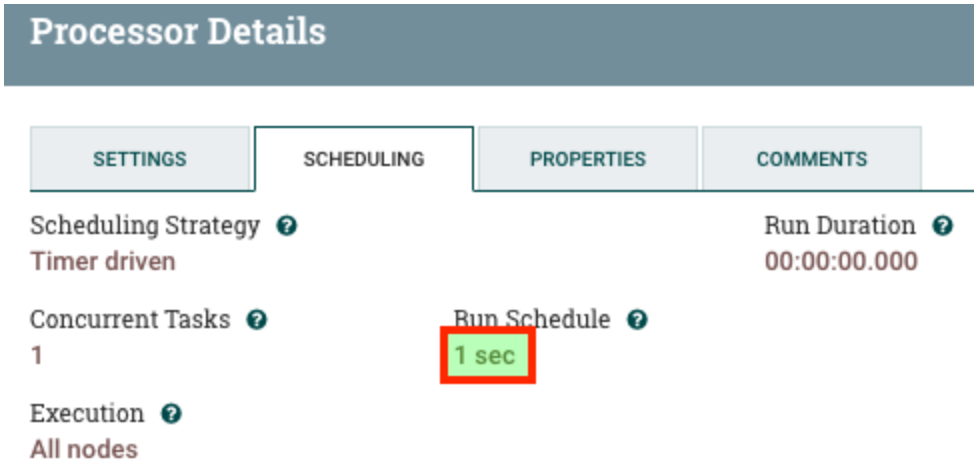
a. Right-click the processor, select Configure (or, alternatively, just double-click the processor). On the PROPERTIES tab, set the properties shown below to run our Python simulate script.

| Property | Value |
|---|---|
| Command | python3 |
| Command Argument | /opt/demo/simulate.py |

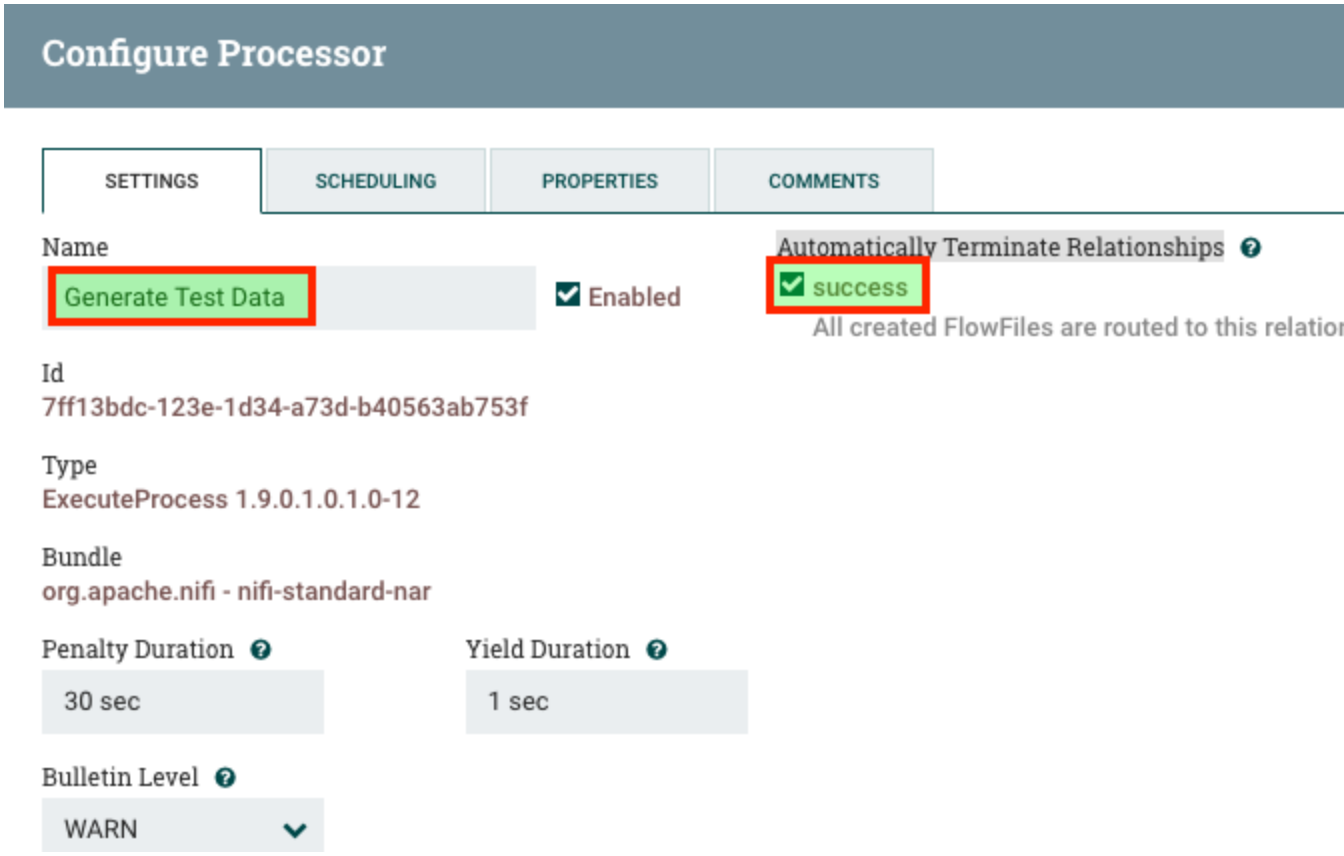



b. In the **SCHEDULING** tab, set to Run Schedule: 1 sec

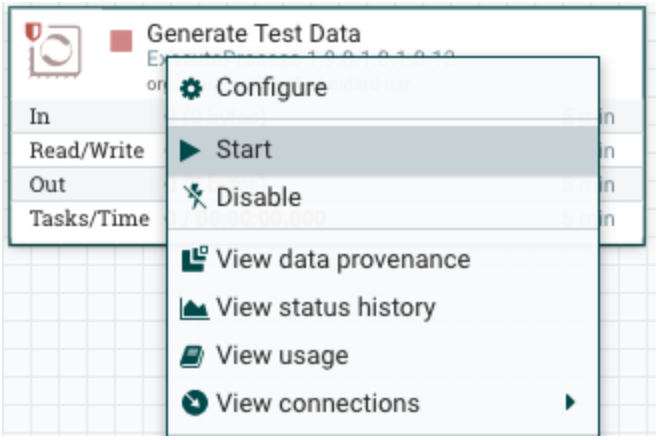
Alternatively, you could set that to other time intervals: 1 sec, 30 sec, 1 min, etc…




c. In the SETTINGS tab:
   i. Check the "success" relationship in the AUTOMATICALLY TERMINATED RELATIONSHIPS section.
   ii. Set the processor name to "Generate Test Data"
   iii. Click Apply

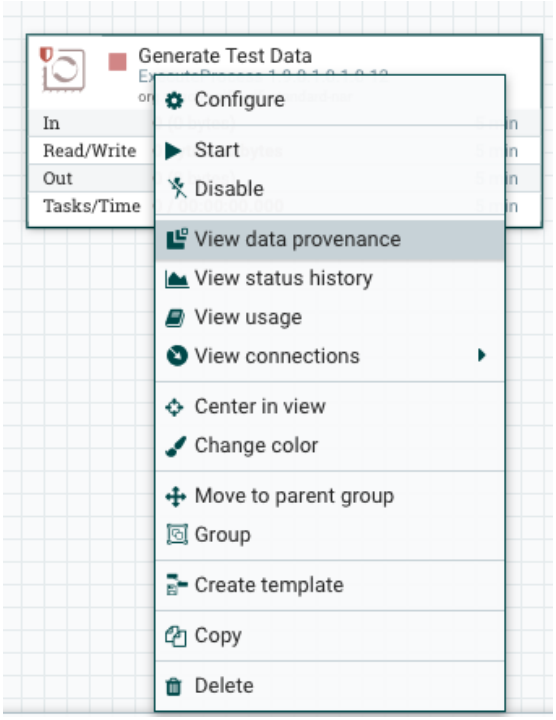



d. You can then right-click to Start this simulator runner.



## Step 3: Validate the processor

a. Right-click and select Stop after a few seconds and look at the provenance. You'll see that it has run a number of times and produced results.



NiFi Data Provenance

Displaying 1,000 of 1,000
Oldest event available: 10/29/2019 02:44:35 UTC

Showing 1,000 of 1,000+ events that match the specified query, please refine the search. Clear search

Filter | by component name

| Date/Time | Type | FlowFile Uuid | Size | Component Name | Component Type | Node |
|---|---|---|---|---|---|---|
| 10/29/2019 14:15:01.707... | DROP | f8cb0afc-ef58-4128-af1d-... | 238 bytes | Generate Test Data | ExecuteProcess | ip-10-0-1-192.us-west-2.c... |
| 10/29/2019 14:15:01.707... | CREATE | f8cb0afc-ef58-4128-af1d-... | 238 bytes | Generate Test Data | ExecuteProcess | ip-10-0-1-192.us-west-2.c... |
| 10/29/2019 14:15:00.648... | DROP | a8593d6e-7014-4276-8b... | 239 bytes | Generate Test Data | ExecuteProcess | ip-10-0-1-192.us-west-2.c... |
| 10/29/2019 14:15:00.648... | CREATE | a8593d6e-7014-4276-8b... | 239 bytes | Generate Test Data | ExecuteProcess | ip-10-0-1-192.us-west-2.c... |
| 10/29/2019 14:14:59.588... | DROP | 7bae8a48-24e9-4706-93... | 239 bytes | Generate Test Data | ExecuteProcess | ip-10-0-1-192.us-west-2.c... |
| 10/29/2019 14:14:59.588... | CREATE | 7bae8a48-24e9-4706-93... | 239 bytes | Generate Test Data | ExecuteProcess | ip-10-0-1-192.us-west-2.c... |
| 10/29/2019 14:14:58.529... | DROP | b727f9d0-b4c7-40c1-b8d... | 238 bytes | Generate Test Data | ExecuteProcess | ip-10-0-1-192.us-west-2.c... |
| 10/29/2019 14:14:58.529... | CREATE | b727f9d0-b4c7-40c1-b8d... | 238 bytes | Generate Test Data | ExecuteProcess | ip-10-0-1-192.us-west-2.c... |
| 10/29/2019 14:14:57.471... | DROP | 013119a1-2f59-42ea-a88... | 241 bytes | Generate Test Data | ExecuteProcess | ip-10-0-1-192.us-west-2.c... |
| 10/29/2019 14:14:57.471... | CREATE | 013119a1-2f59-42ea-a88... | 241 bytes | Generate Test Data | ExecuteProcess | ip-10-0-1-192.us-west-2.c... |
| 10/29/2019 14:14:56.413... | DROP | c1ce6530-8eb1-4a9e-816... | 238 bytes | Generate Test Data | ExecuteProcess | ip-10-0-1-192.us-west-2.c... |
| 10/29/2019 14:14:56.413... | CREATE | c1ce6530-8eb1-4a9e-816... | 238 bytes | Generate Test Data | ExecuteProcess | ip-10-0-1-192.us-west-2.c... |
| 10/29/2019 14:14:55.357... | DROP | 09d42dde-272f-4b0c-807... | 237 bytes | Generate Test Data | ExecuteProcess | ip-10-0-1-192.us-west-2.c... |
| 10/29/2019 14:14:55.357... | CREATE | 09d42dde-272f-4b0c-807... | 237 bytes | Generate Test Data | ExecuteProcess | ip-10-0-1-192.us-west-2.c... |
| 10/29/2019 14:14:54.291... | DROP | 6edbadee-2182-487a-96... | 238 bytes | Generate Test Data | ExecuteProcess | ip-10-0-1-192.us-west-2.c... |

# Lab 1 Completed

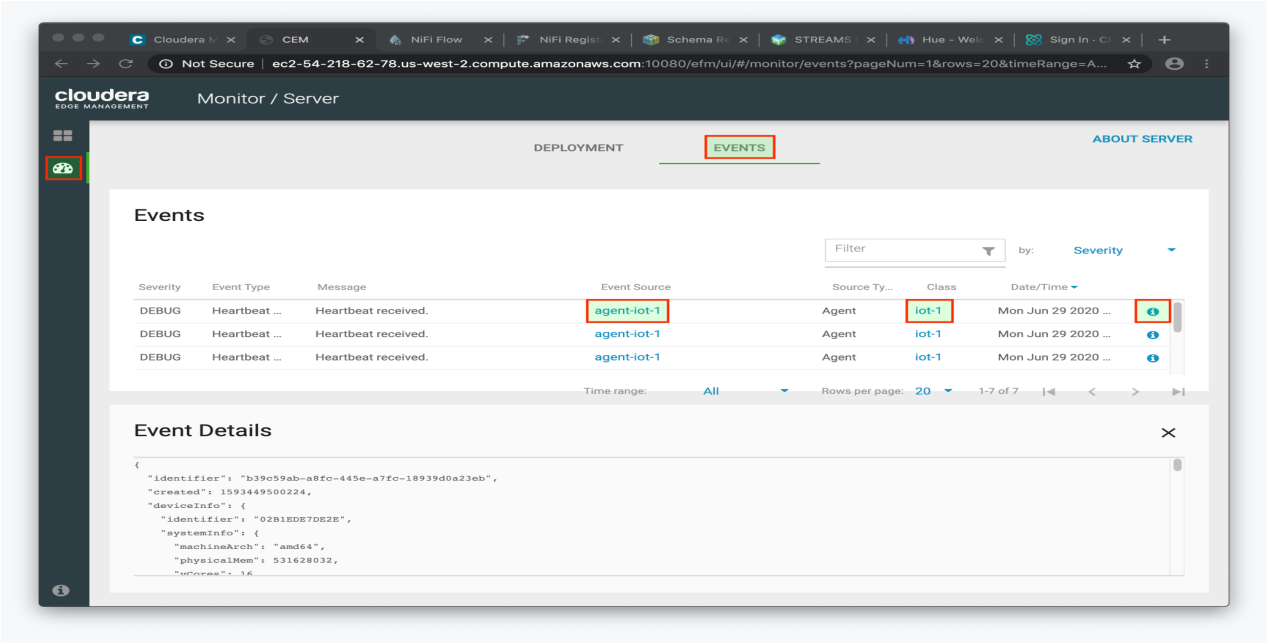
# Lab 2 - Configuring Edge Flow Management

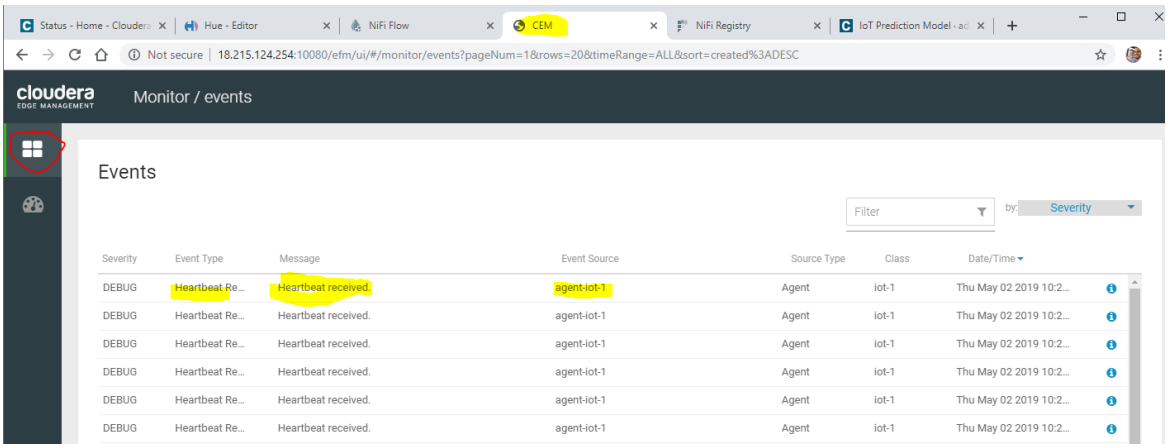
Cloudera Edge Flow Management (EFM) gives you a visual overview of all MiNiFi agents in your environment, and allows you to update the flow configuration for each one, with versioning control thanks to the NiFi Registry integration. In this lab, you will create the MiNiFi flow and publish it for the MiNiFi agent to pick it up.

## Step1: Verify heartbeats from MiNiFi agents

a. Open the EFM Web UI at *http://<public_dns>:10088/efm/ui/* and select the Dashboard tab

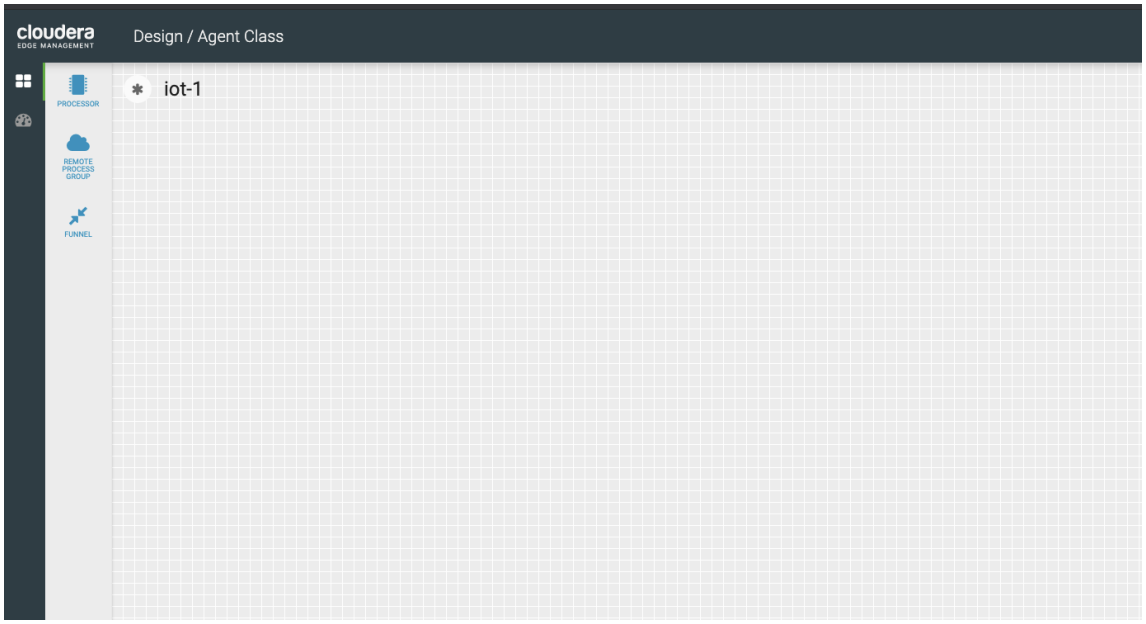
Click on the EVENTS header and verify that your EFM server is receiving heartbeats from the MiNiFi agent. Click on the info icon on a heartbeat record to see the details of the heartbeat.



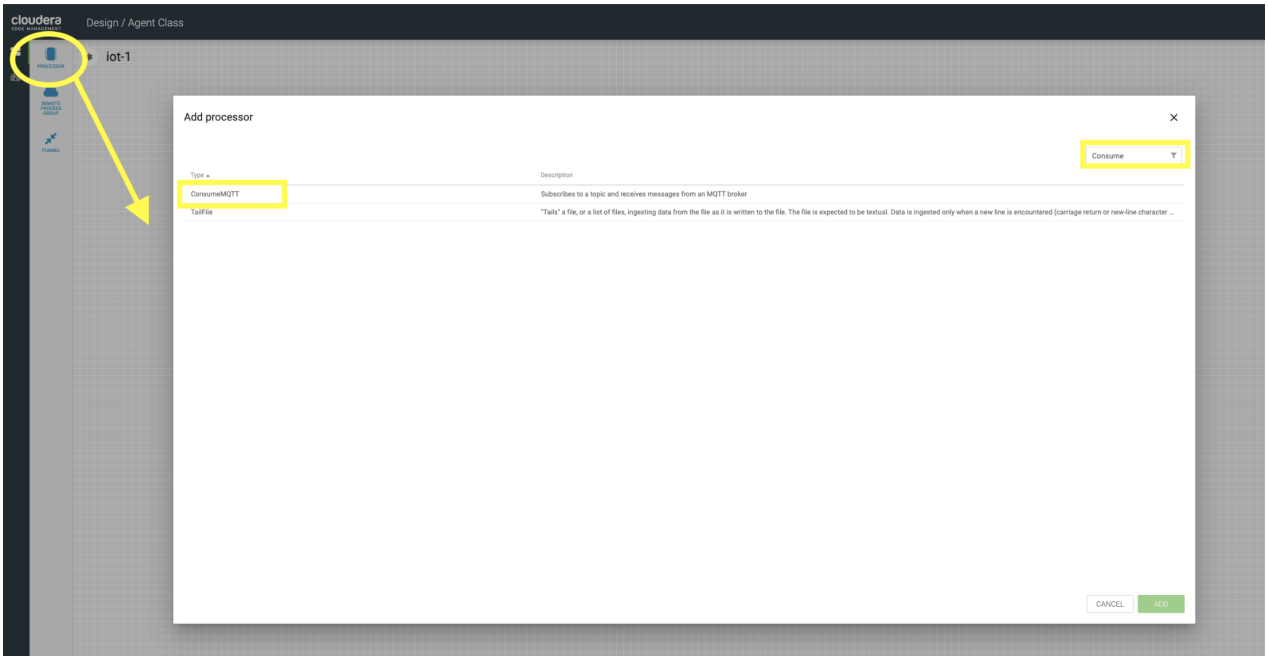



Step 2 : Designing the EFM flow

a. Select the Flow Designer tab ( ). To build a dataflow, select the desired class (`iot-1`) from the table and click OPEN. Alternatively, you can double-click on the desired class.

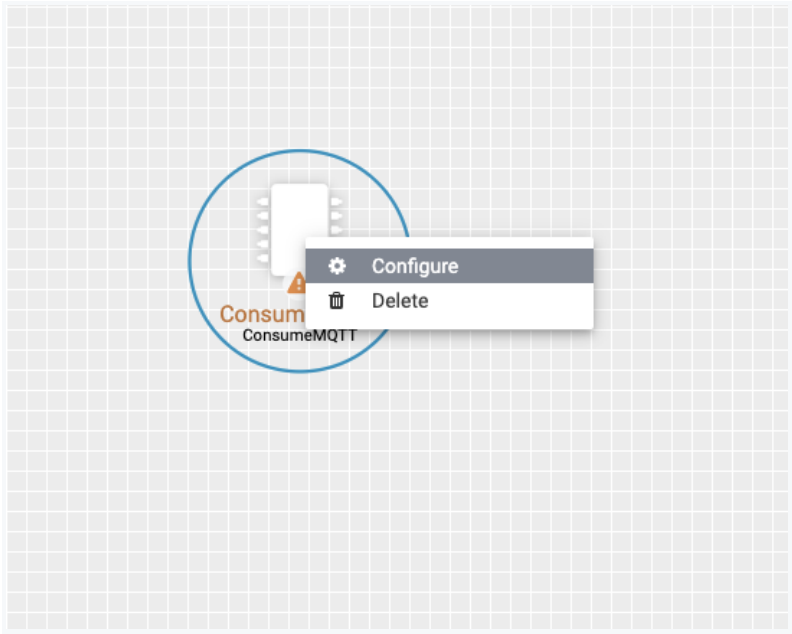



b. Add a *ConsumeMQTT* Processor to the canvas, by dragging the processor icon to the canvas, selecting the *ConsumeMQTT* processor type and clicking on the Add button. Once the processor is on the canvas, double-click it and configure it with below settings:

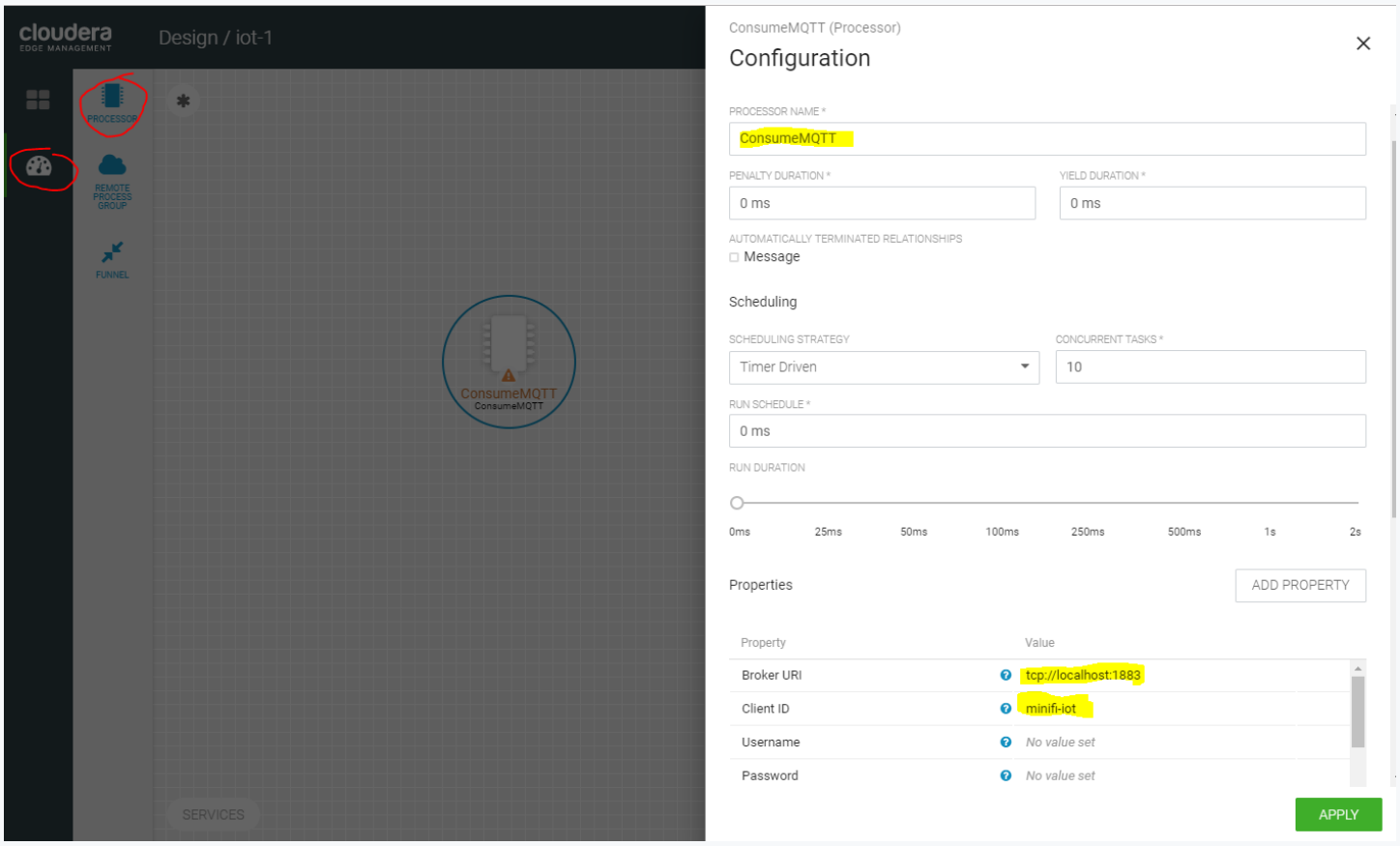



Select the processor and click ADD.

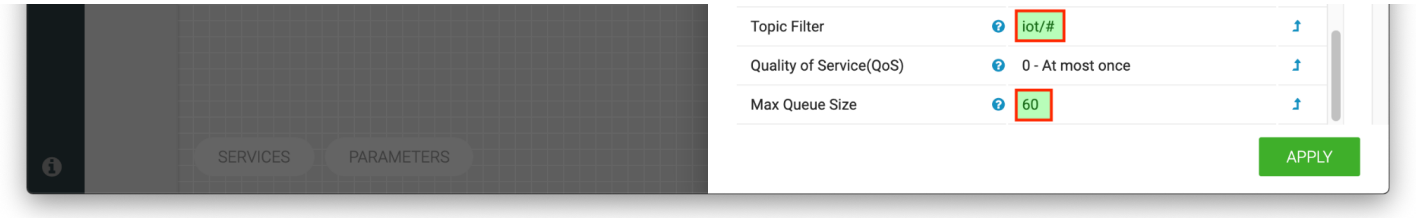
c. Right Click and configure the *ConsumeMQTT* Processor with the below settings:



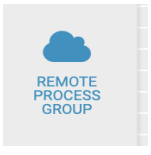

```
Broker URI: tcp://localhost:1883 OR tcp://<internal-hostname>:1883
Client ID: minifi-iot
Topic Filter: iot/#
Max Queue Size = 60
```




And ensure you scroll down on the properties page to set the Topic Filter and Max Queue Size:

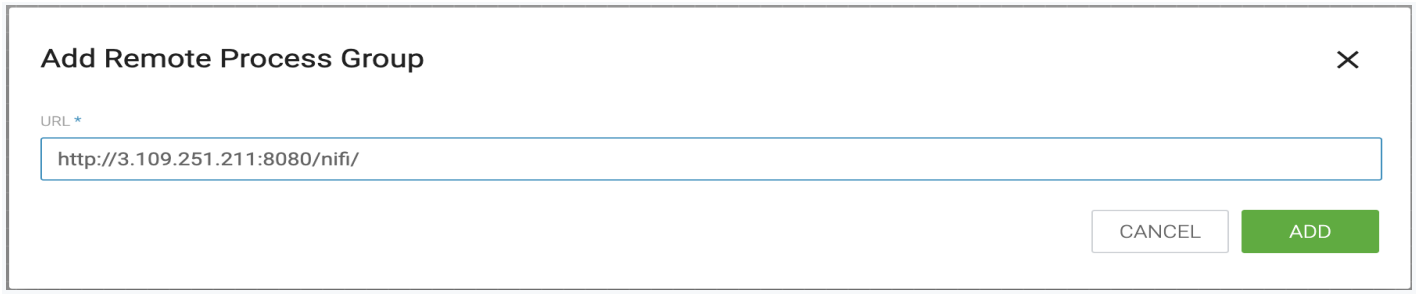



Click on APPLY.

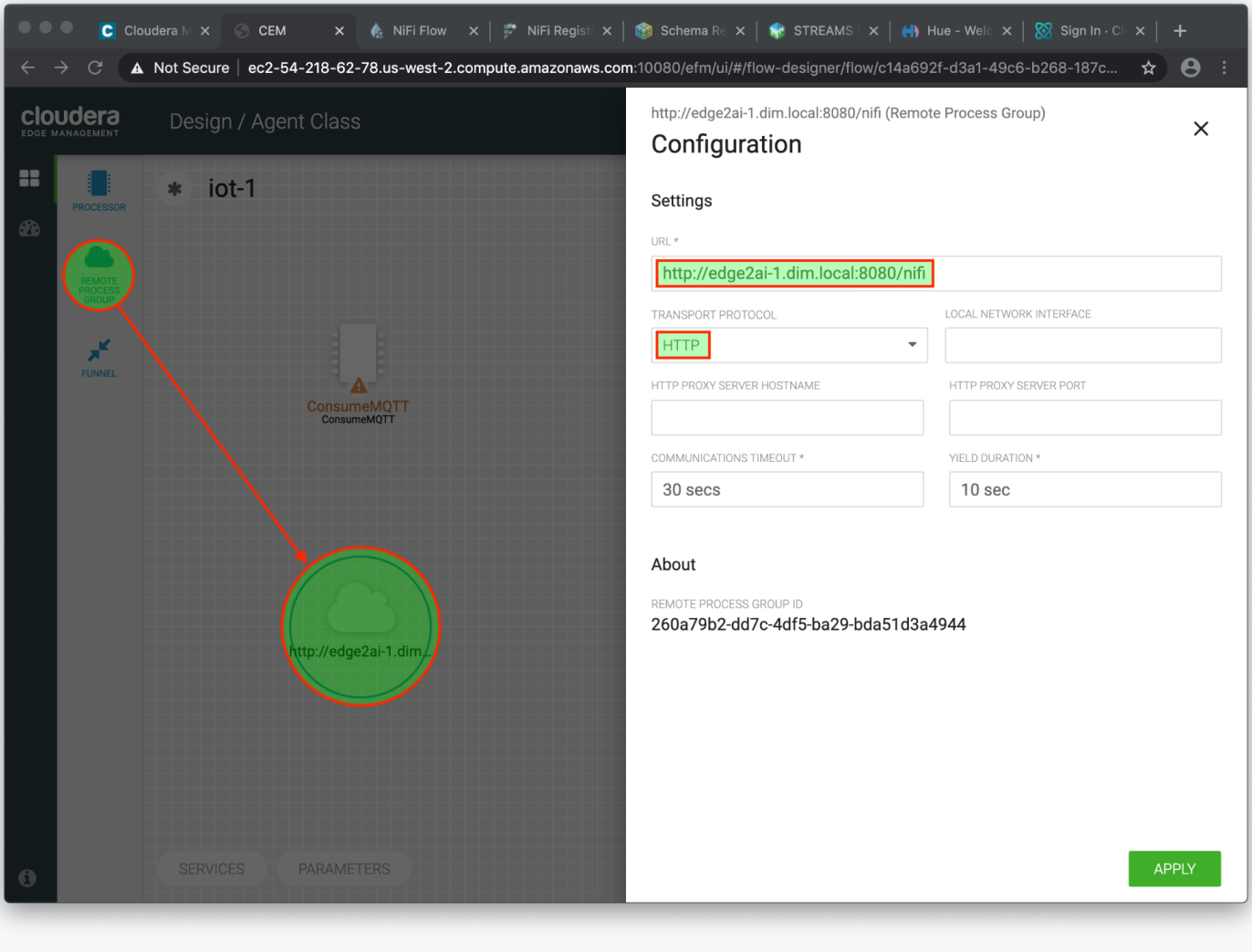
d. Add a *Remote Process Group* 

to the canvas and configure it as follows and click on ADD.

```
URL = http://<public-IP>:8080/nifi
```



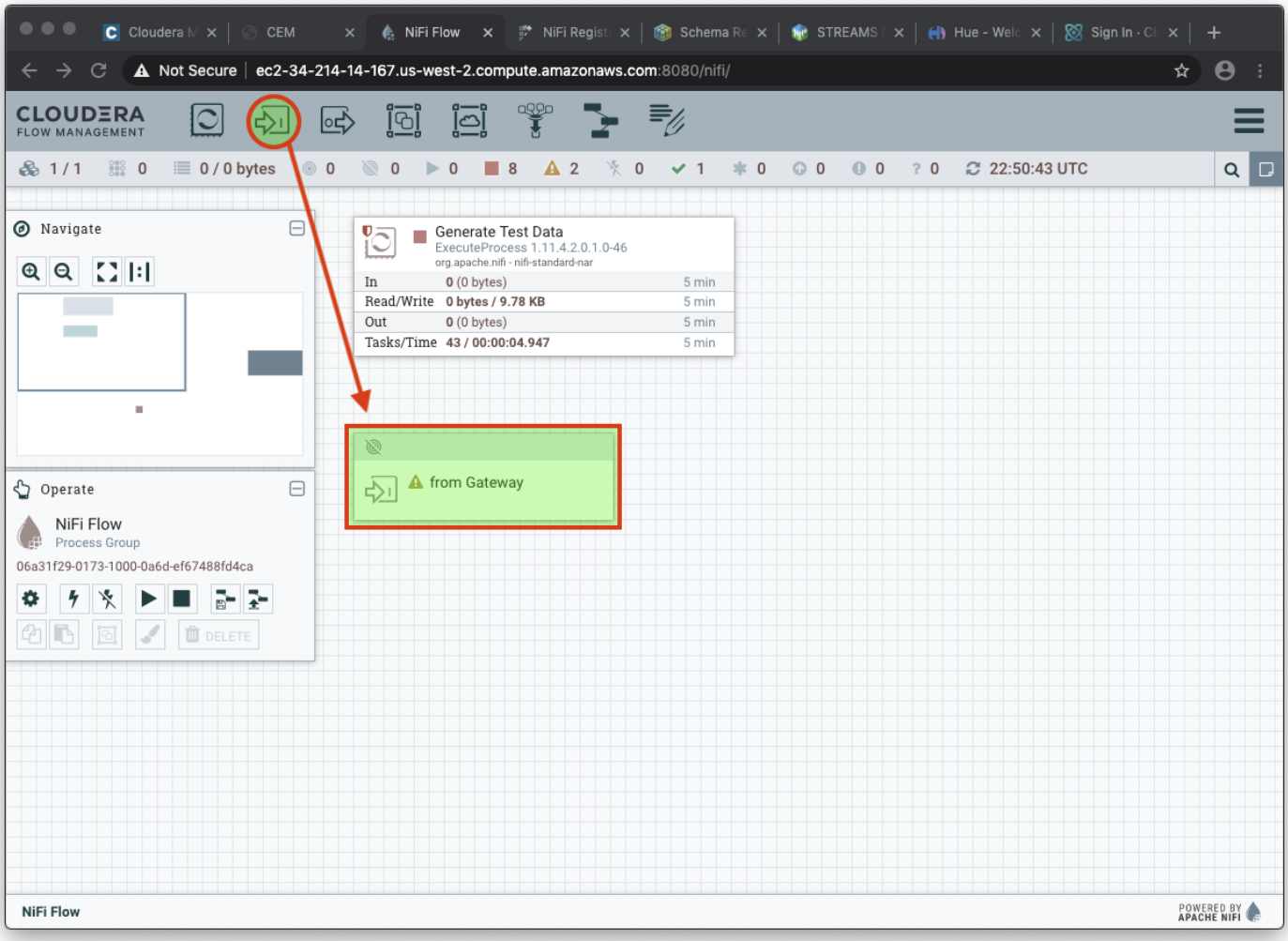

e. Right click on the newly created Remote Process Group and click on CONFIGURE



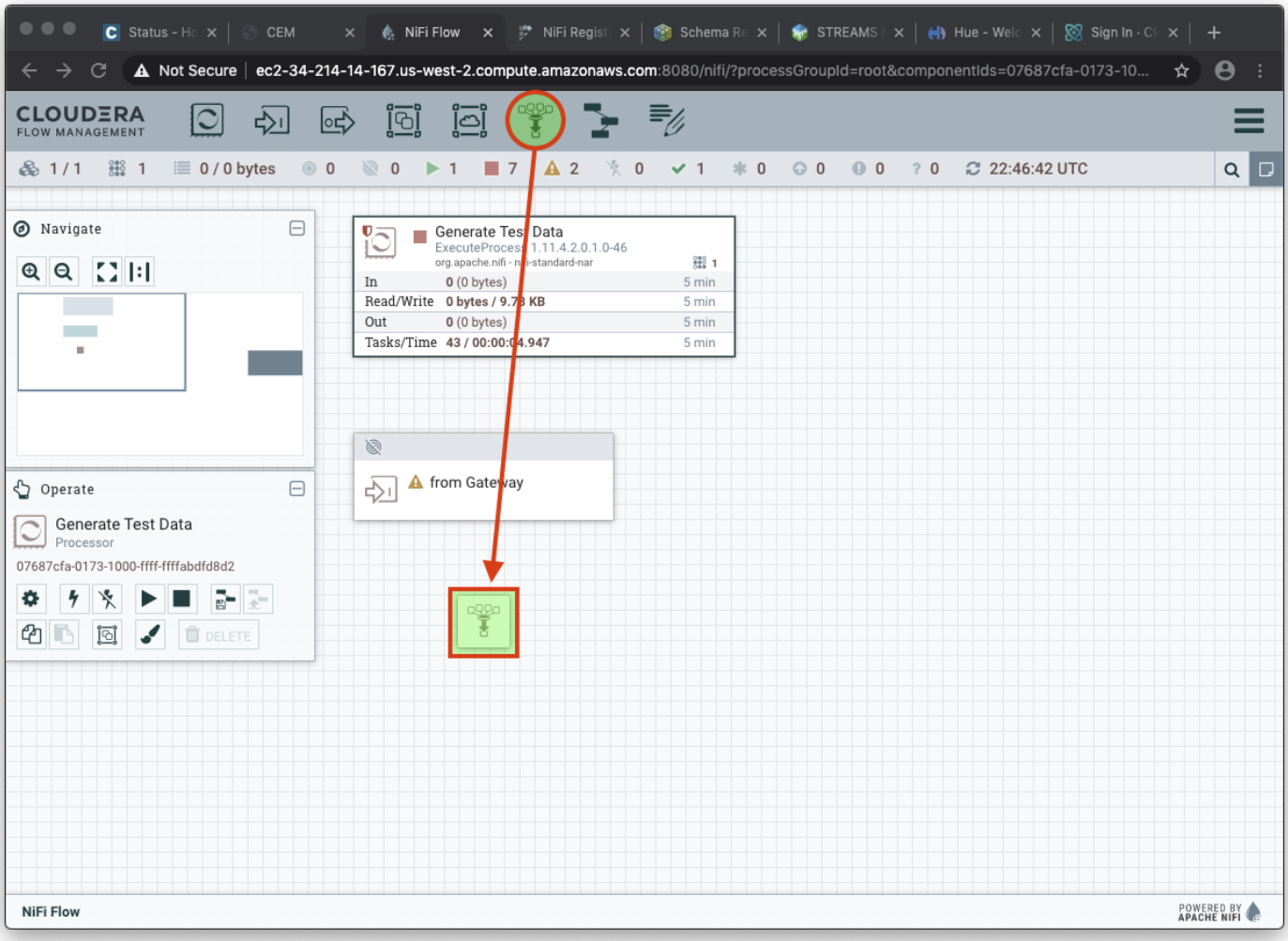
## Step 3 : Designing the NiFi flow

At this point you need to connect the ConsumerMQTT processor to the RPG, however, you first need the ID of the NiFi entry port.

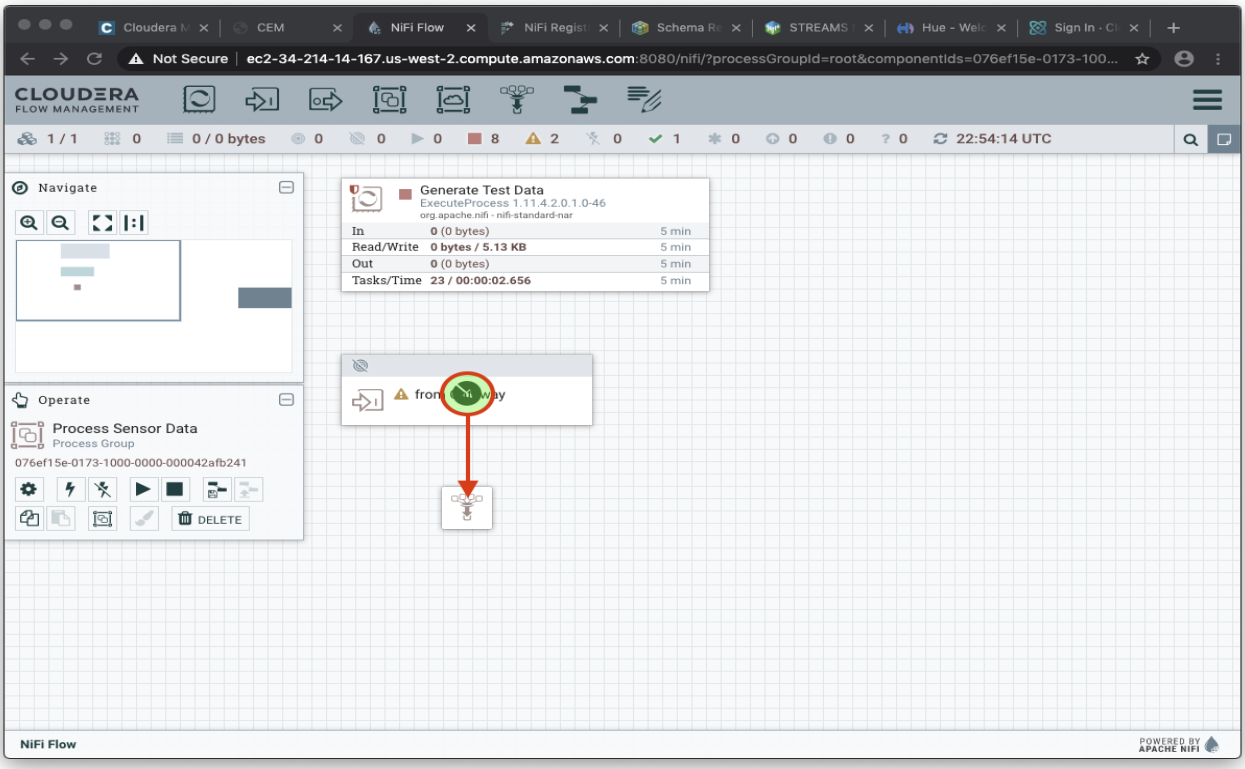
Open NiFi Web UI at http://<public-IP>:8080/nifi/ and add an *Input Port* to the canvas. Call it something like "from Gateway" and copy the ID of the input port, as you will soon need it.



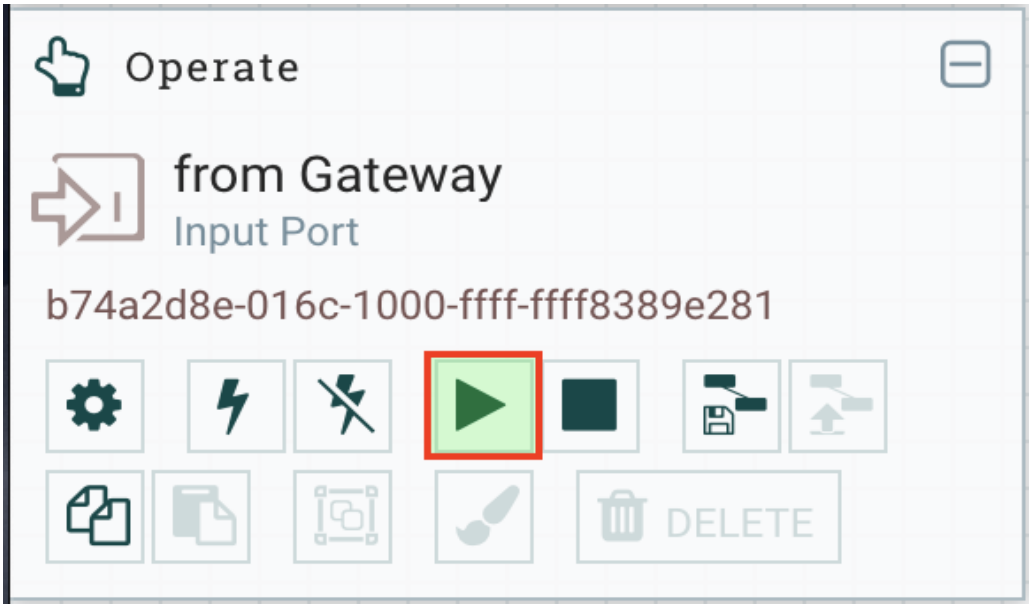

To terminate the NiFI *Input Port* let's, for now, add a *Funnel* to the canvas…



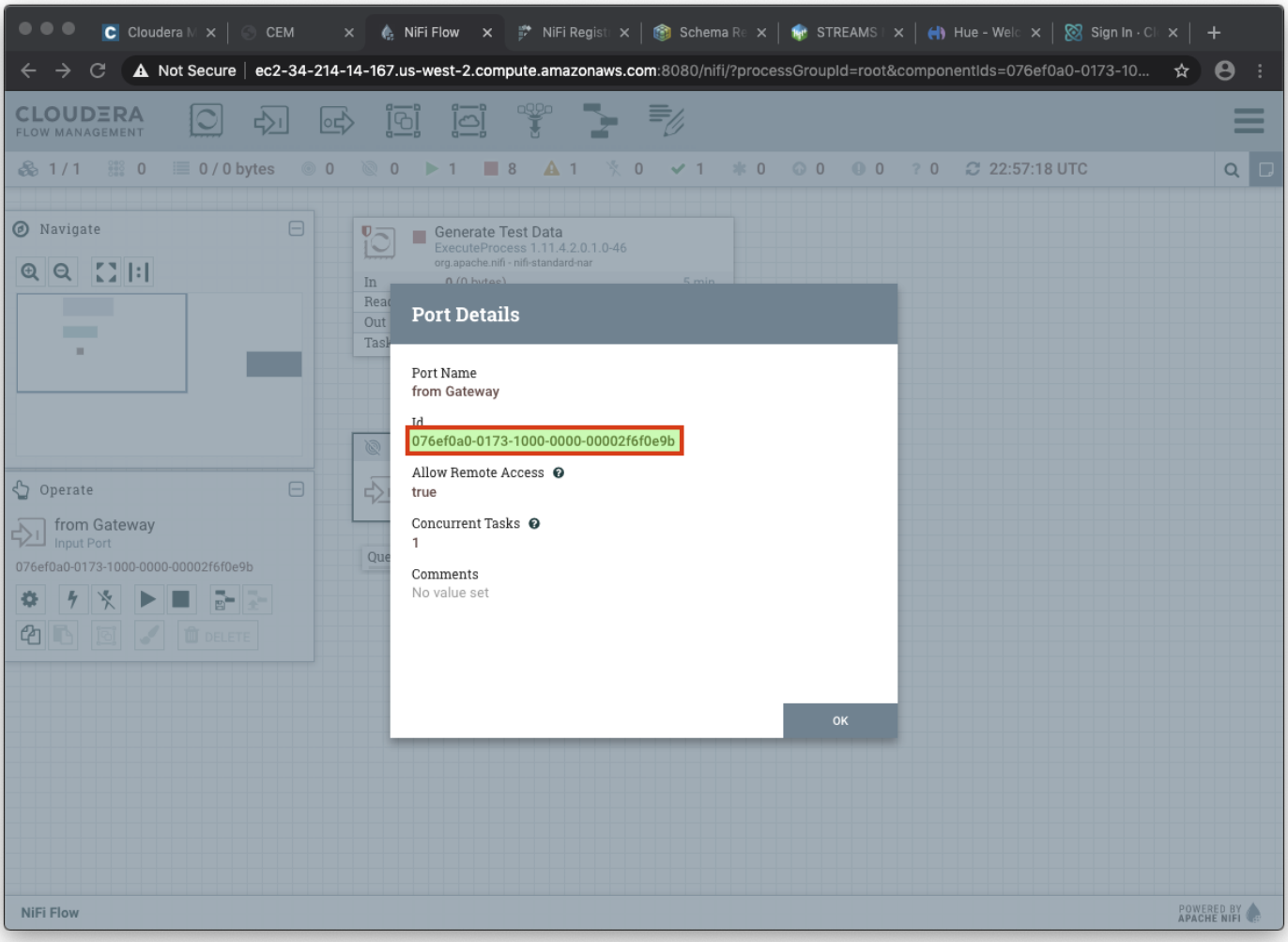

set up a connection from the Input Port to it. To set up a connection, hover the mouse over the Input Port until an arrow symbol is shown in the center. Click on the arrow, drag it and drop it on the Funnel to connect the two elements.




Right-click on the Input Port and start it. Alternatively, click on the Input Port to select it and then press the start ("play") button on the Operate panel:

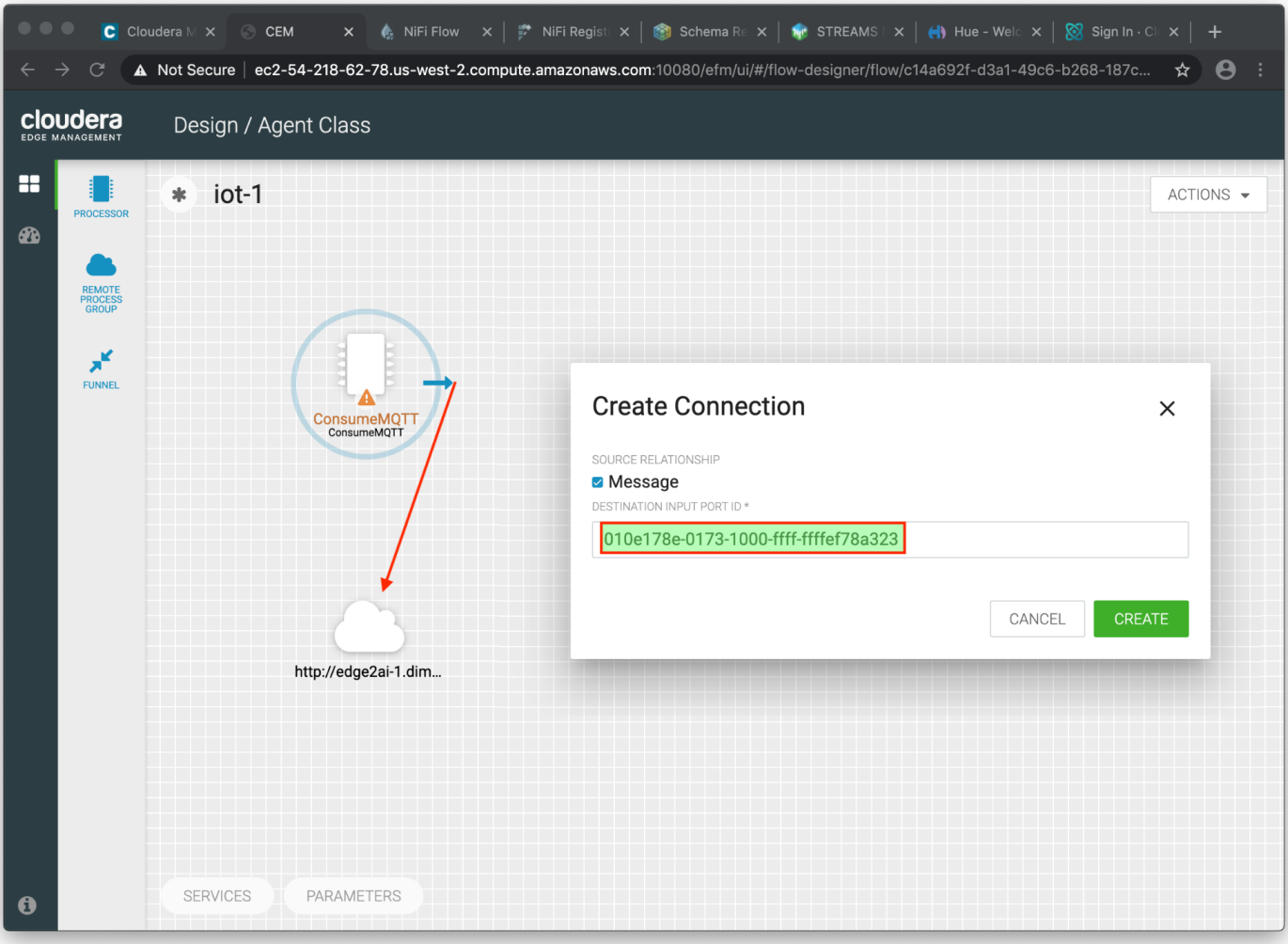


You will need the ID of the *Input Port* to complete the connection of the *ConsumeMQTT* processor to the RPG (NiFi). Double-click on the *Input Port* and copy its ID.



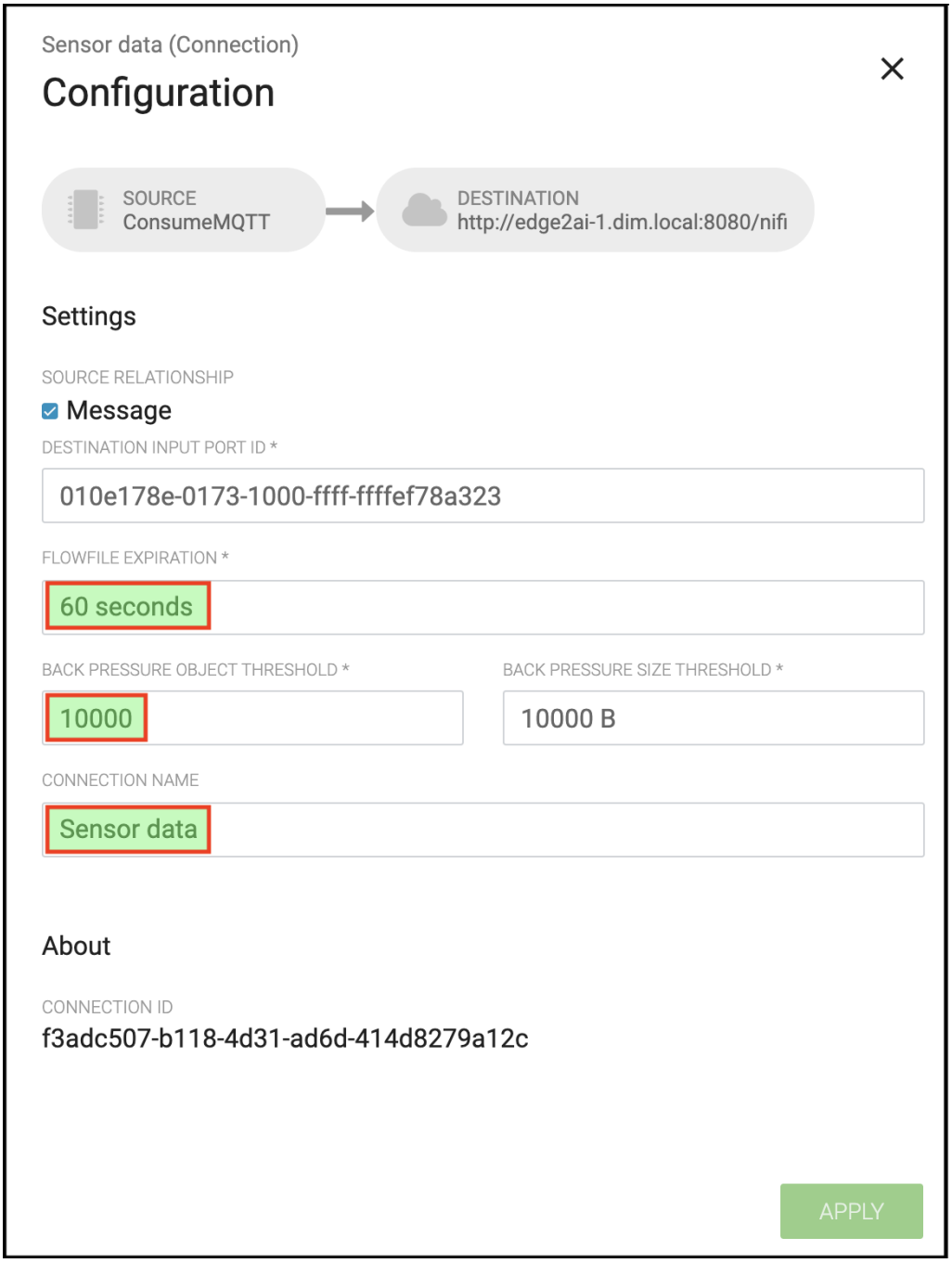

## Step 4 : Integration between EFM and NiFi

Back to the Flow Designer, connect the ConsumeMQTT processor to the RPG. The connection requires an ID and you can paste here the ID you copied from the Input Port. Make sure that there are NO SPACES!



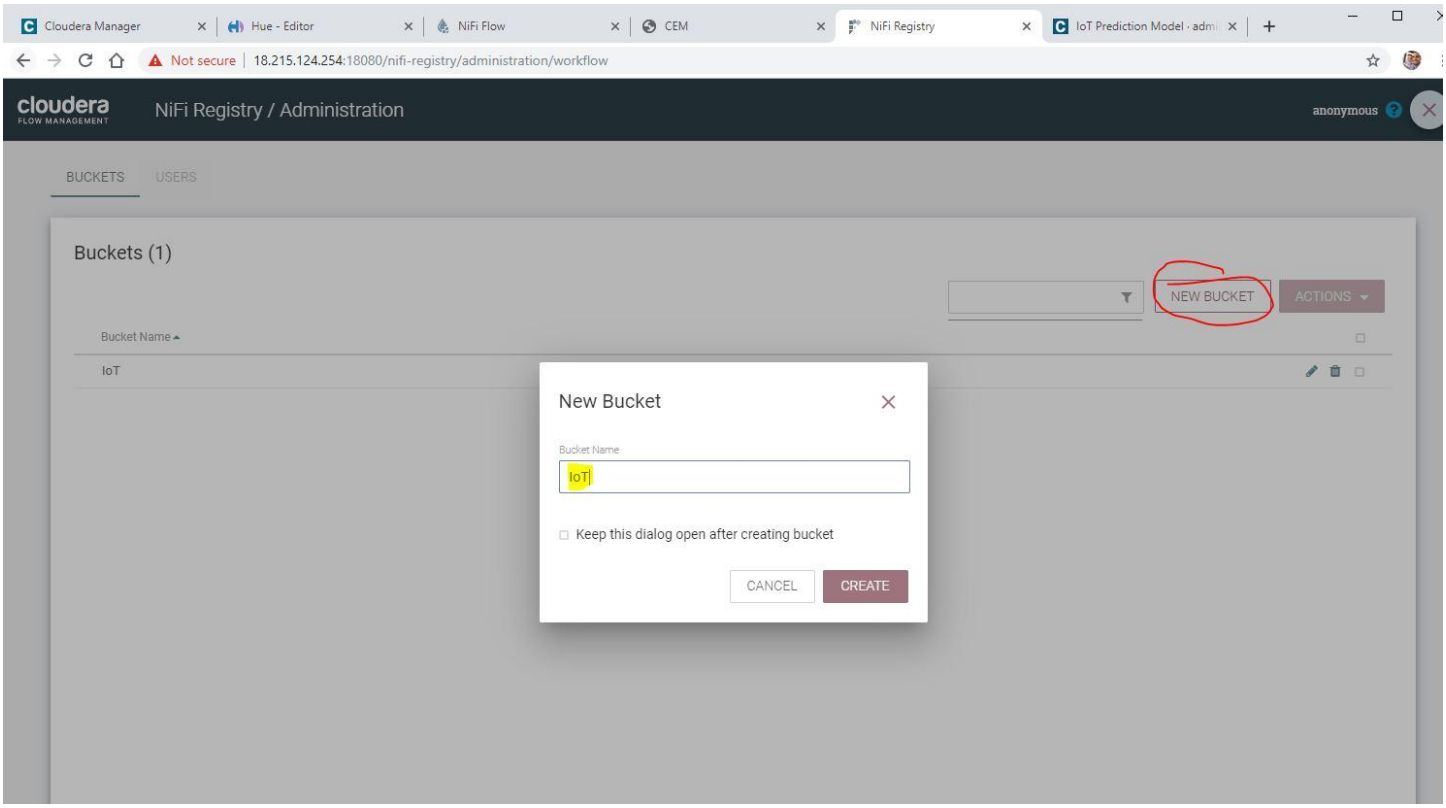

Double-click the connection and update the following configuration:

| Property | Value |
|---|---|
| Flowfile Expiration | 60 seconds |
| Back Pressure Object Threshold | 10000 |
| Connection Name | Sensor Data |

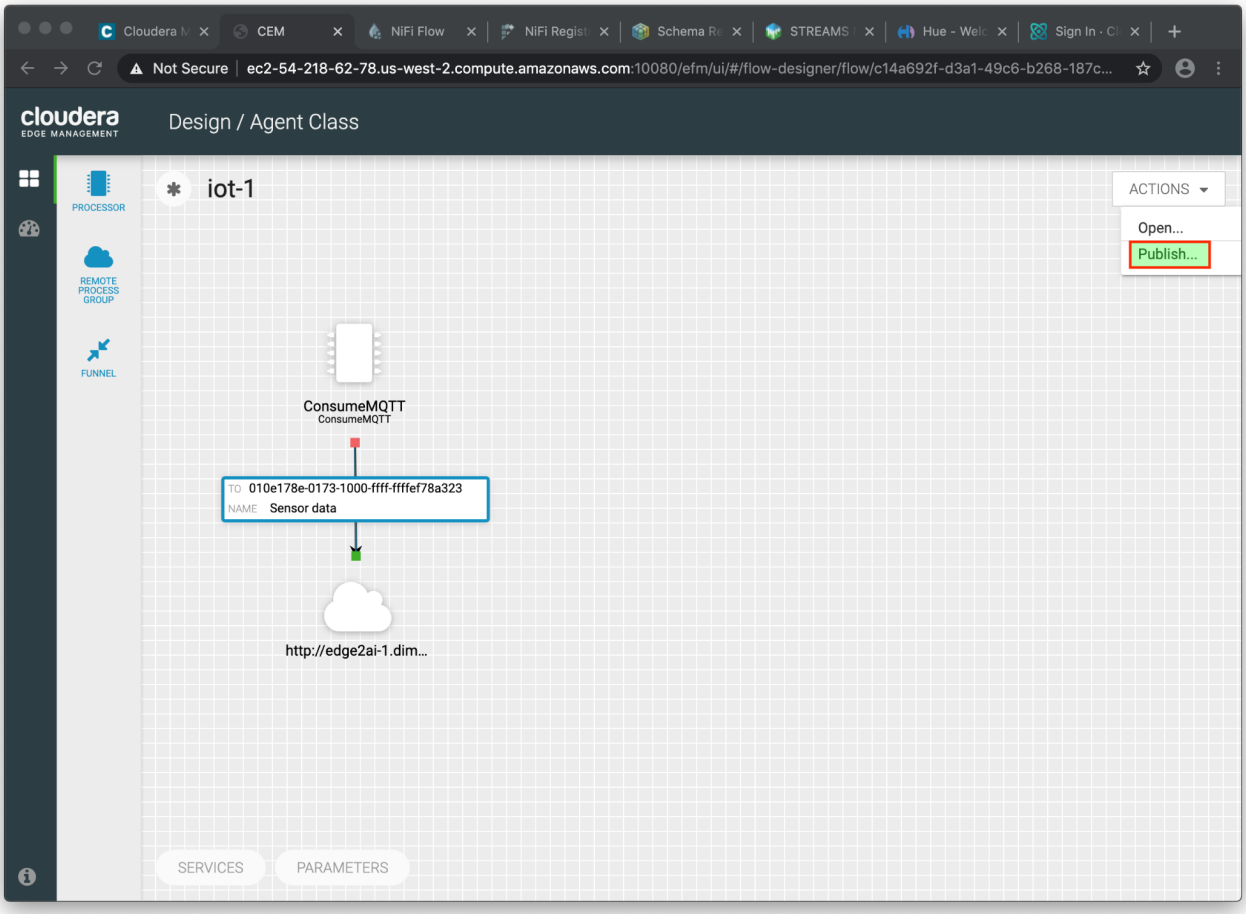


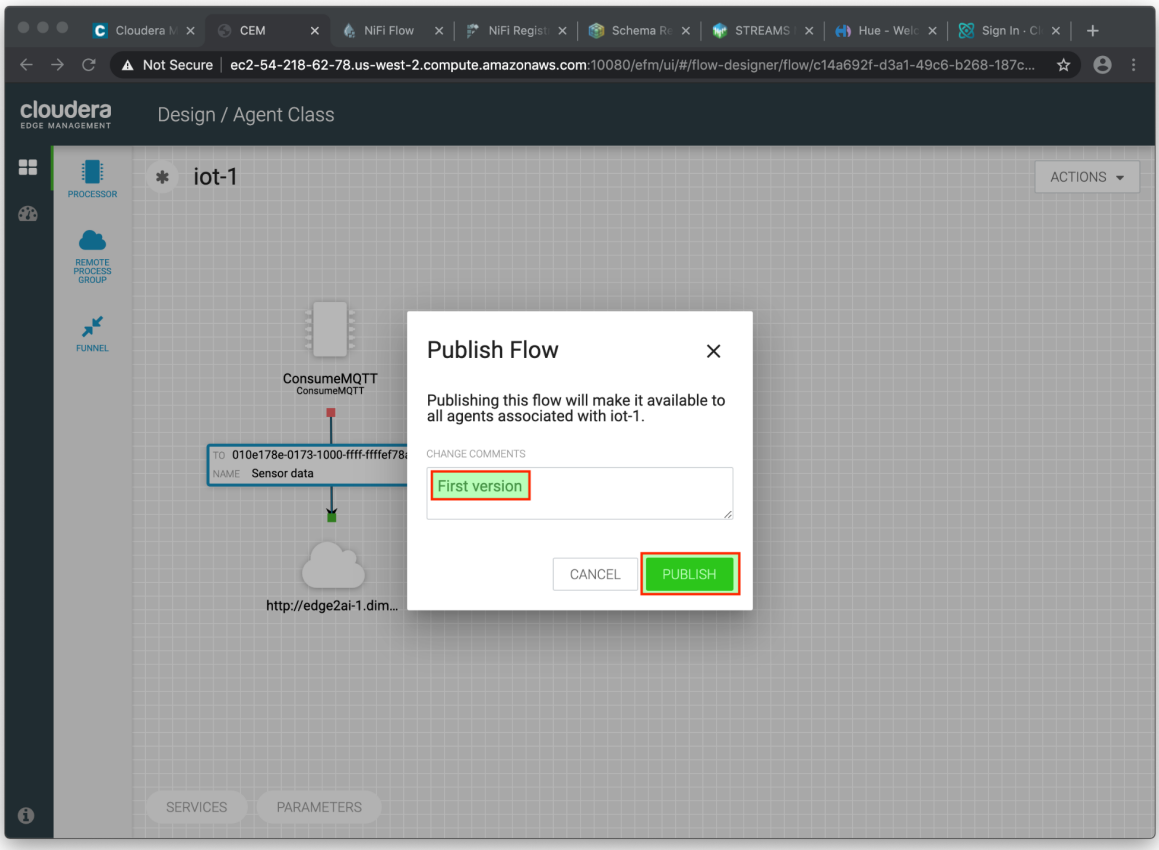

The Flow is now complete, but before publishing it, create the Bucket in the *NiFi Registry* so that all versions of your flows are stored for review and audit. Open the NiFi Registry at

`http://<public_dns>:18080/nifi-registry`, click on the wrench/spanner icon (  ) on the top-right corner and create a bucket called **`IoT`** (ATTENTION: the bucket name is CASE-SENSITIVE).

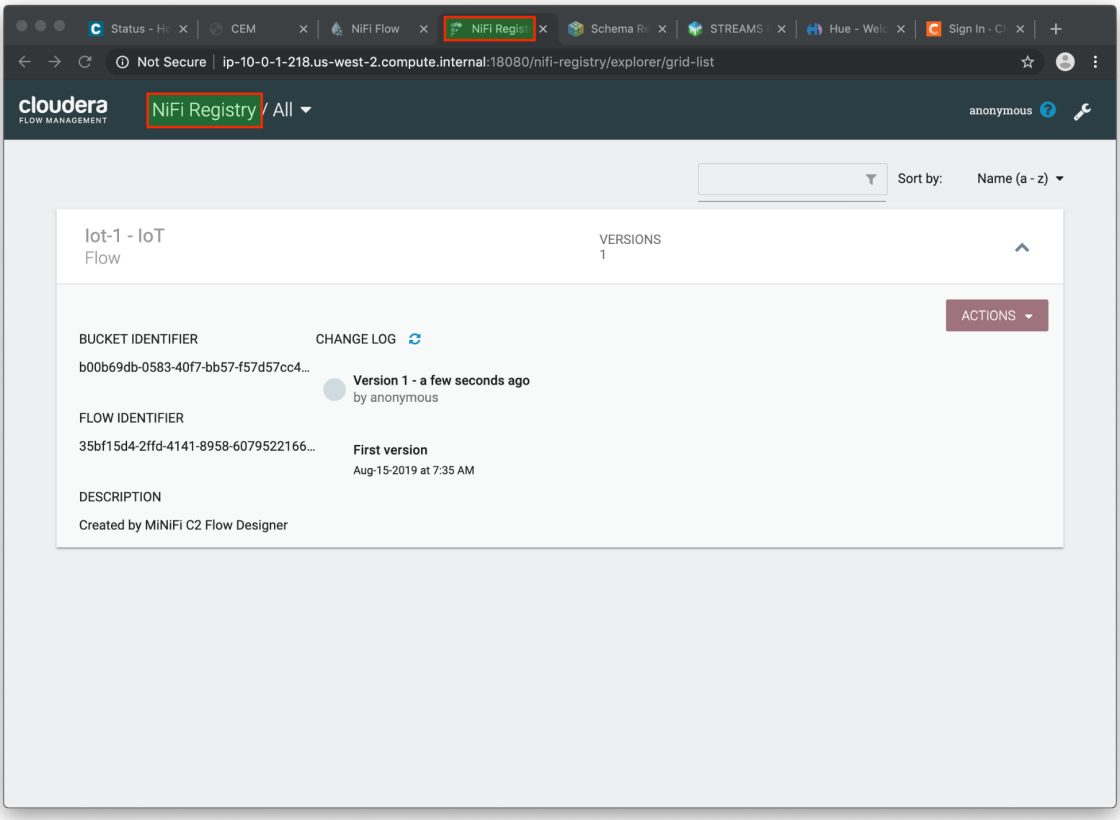



You can now publish the flow for the MiNiFi agent to automatically pick up. Click Publish, add a descriptive comment for your changes and click Apply.

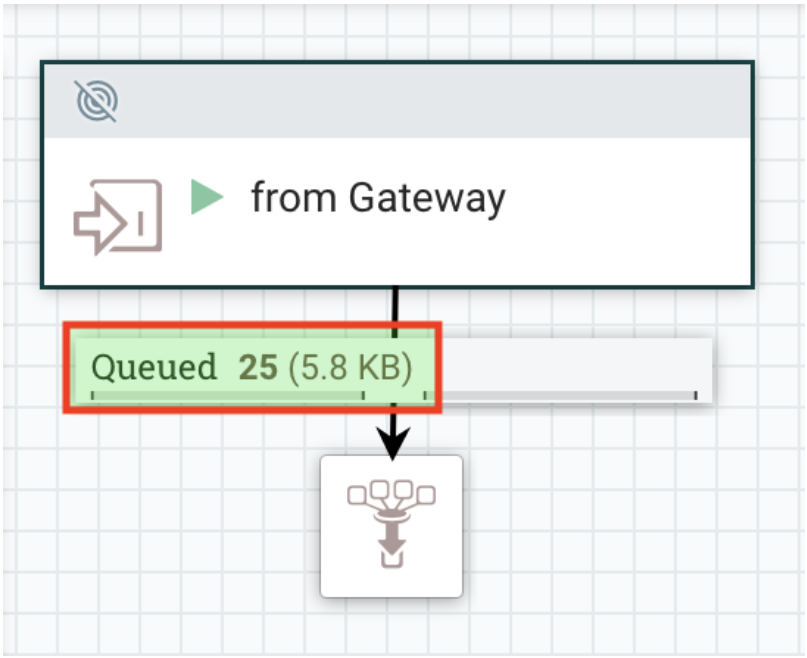

Not Secure
ec2-54-218-62-78.us-west-2.compute.amazonaws.com:10080/efm/ui/#/flow-designer/flow/c14a692f-d3a1-49c6-b268-187c...
cloudera
EDGE MANAGEMENT
Design / Agent Class
PROCESSOR
REMOTE PROCESS GROUP
FUNNEL
iot-1
ACTIONS
Open...
Publish...
ConsumeMQTT
ConsumeMQTT
TO 010e178e-0173-1000-ffff-ffffef78a323
NAME Sensor data
http://edge2ai-1.dim...
SERVICES
PARAMETERS




Go back to the NiFi Registry Web UI and click on the NiFi Registry name, next to the Cloudera logo. If the flow publishing was successful, you should see the flow's version details in the NiFi Registry.




At this point, you can test the edge flow up until NiFi. Start the NiFi simulator (ExecuteProcess processor) again and confirm you can see the messages queued in NiFi.



You can stop the simulator (Stop the NiFi processor) once you confirm that the flow is working correctly.

Before we move on to the next lab we will have to stop all the processors currently on the canvas and clear the connection queue which has flow files which we saw in the previous step.

To clear the queue, right click on the connection and click on “Clear Queue”

Once the queue is cleared, we can delete the connection from the INPUT PORT to the FUNNEL.

Once the connection is deleted we can now also delete the FUNNEL.

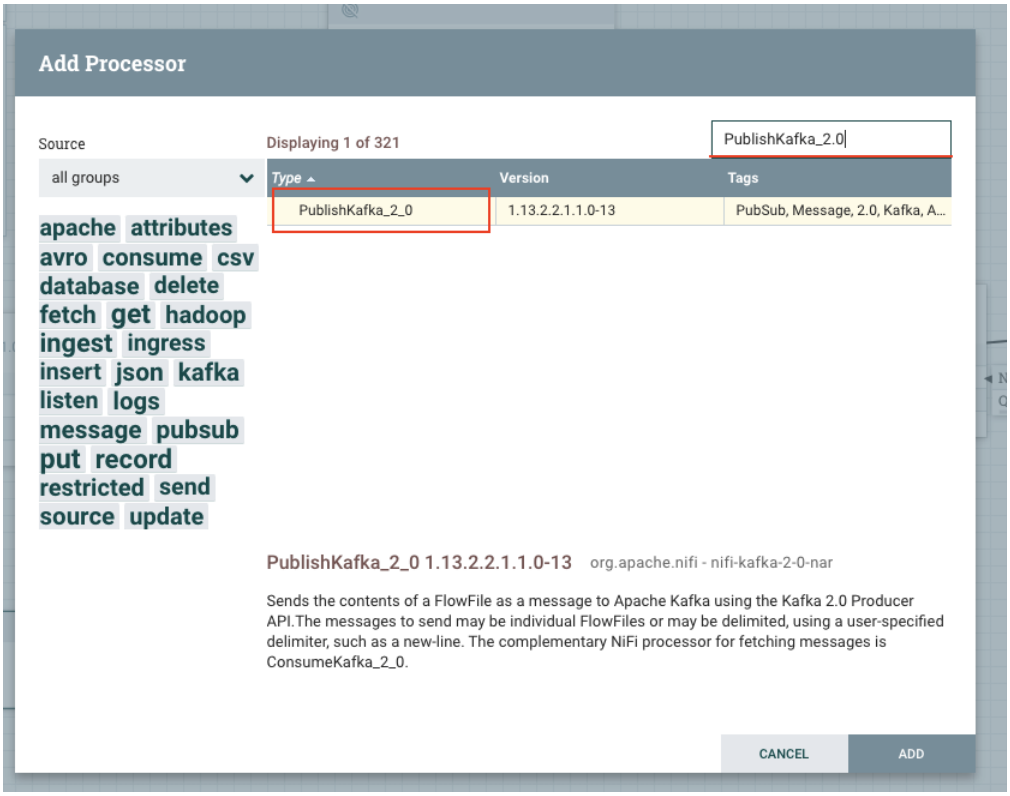
**LAB 2 completed**

# Lab 3 - Configuring the NiFi flow and pushing data to Kafka

In this lab, you will create a NiFi flow to receive the data from all gateways and push it to Kafka.

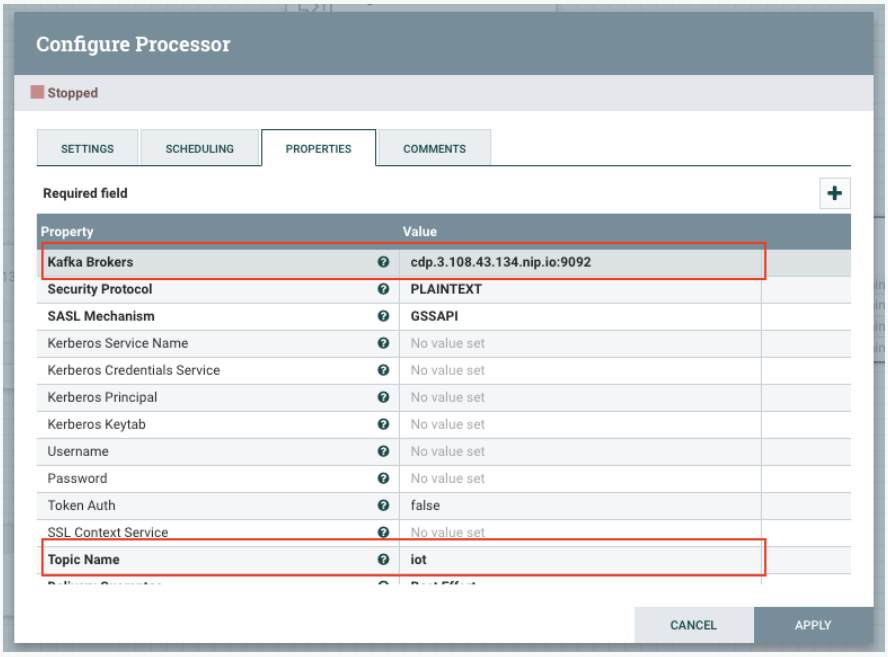
Step 1 : Setup the NiFi flow

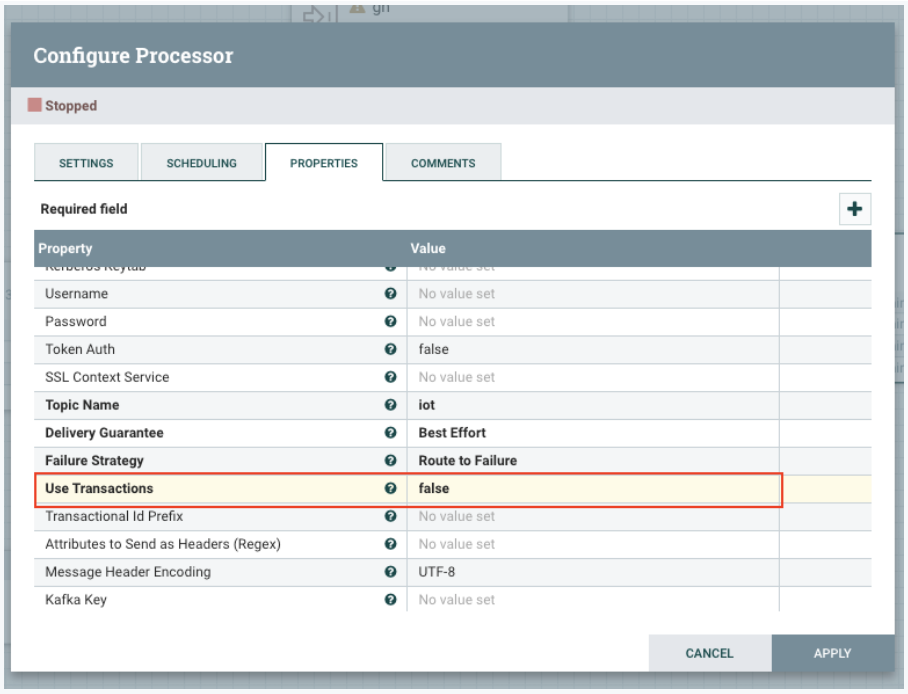
Open the NiFi web UI and add a *PublishKafka_2.0* processor and configure it as follows.



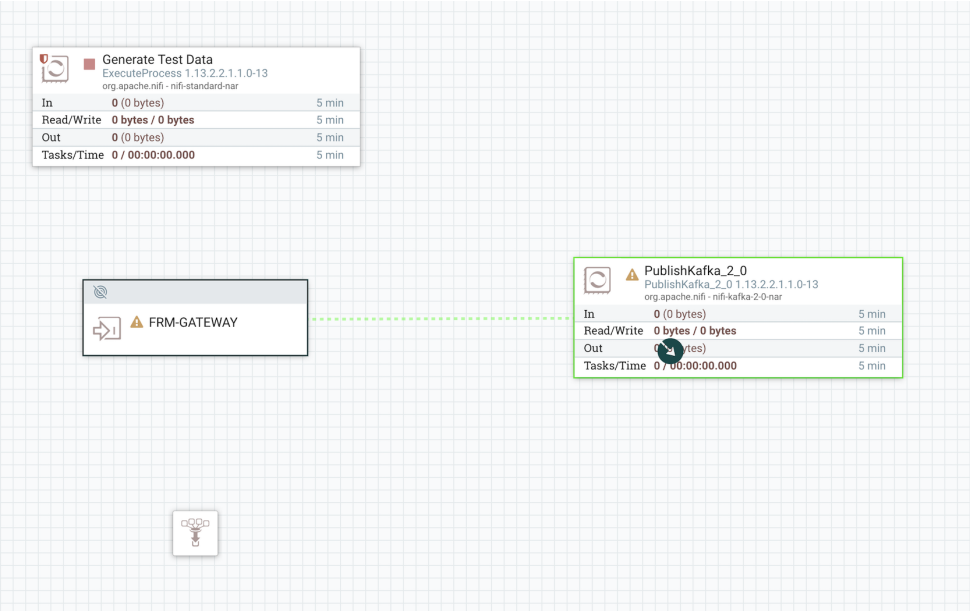

Make sure to use the ‘hostname of the VM assigned to you.(Refer excel sheet shared by the instructor)

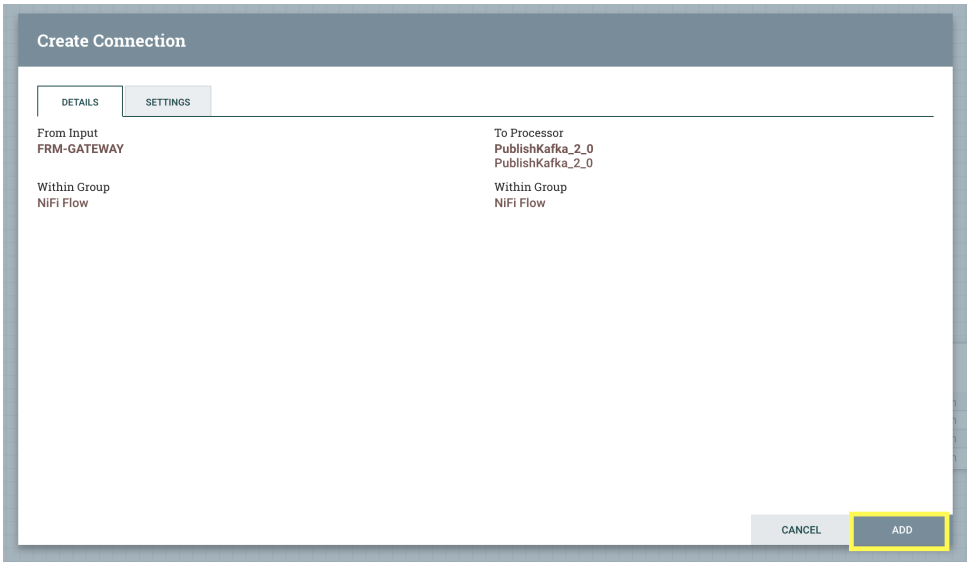
```
Kafka Brokers: <hostname>:9092
Topic Name: iot
Use Transactions: False
```

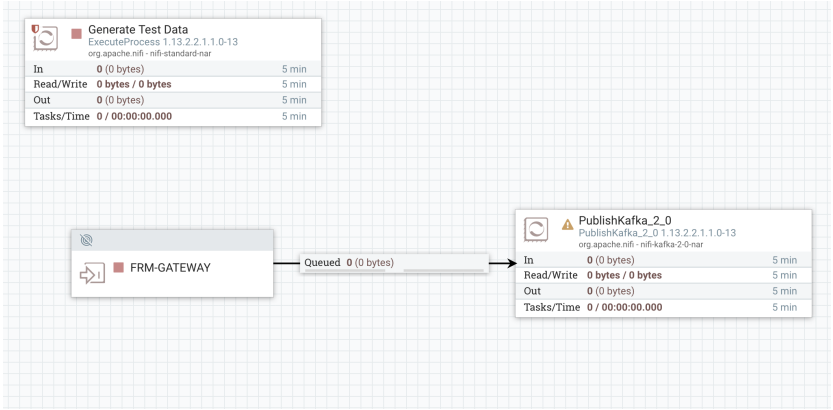


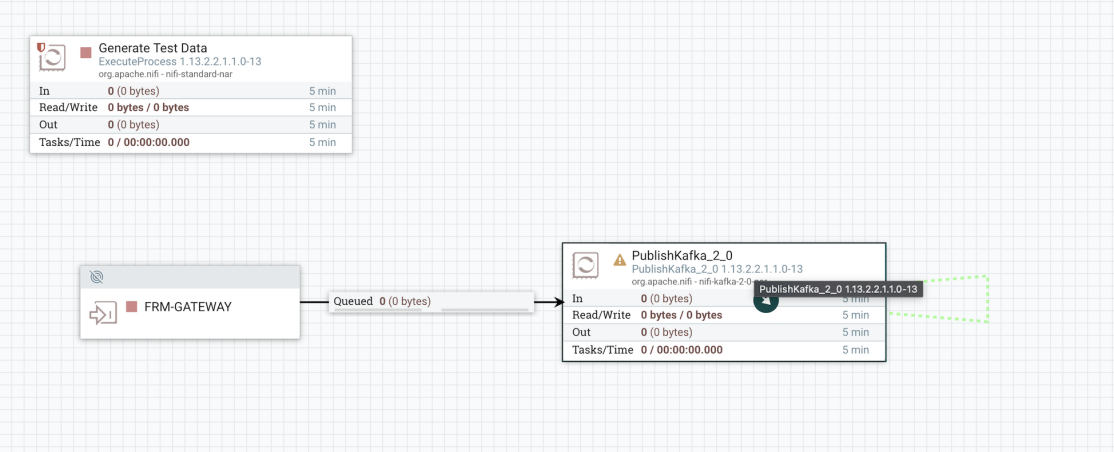

Connect the Input Port(From Gateway) to the PublishKafka processor by dragging the destination of the current connection from the Funnel  to the PublishKafka.

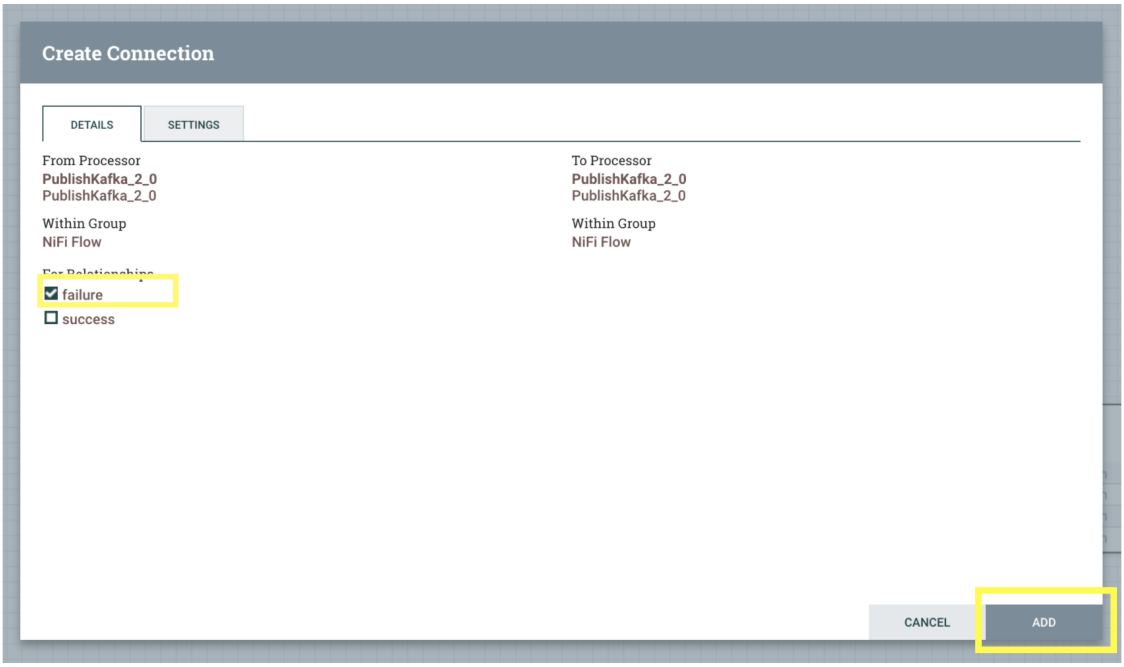

Generate Test Data
ExecuteProcess 1.13.2.2.1.1.0-13
org.apache.nifi - nifi-standard-nar
In 0 (0 bytes) 5 min
Read/Write 0 bytes / 0 bytes 5 min
Out 0 (0 bytes) 5 min
Tasks/Time 0 / 00:00:00.000 5 min
FRM-GATEWAY
PublishKafka_2_0
PublishKafka_2_0 1.13.2.2.1.1.0-13
org.apache.nifi - nifi-kafka-2-0-nar
In 0 (0 bytes) 5 min
Read/Write 0 bytes / 0 bytes 5 min
Out 0 (0 bytes) 5 min
Tasks/Time 0 / 00:00:00.000 5 min

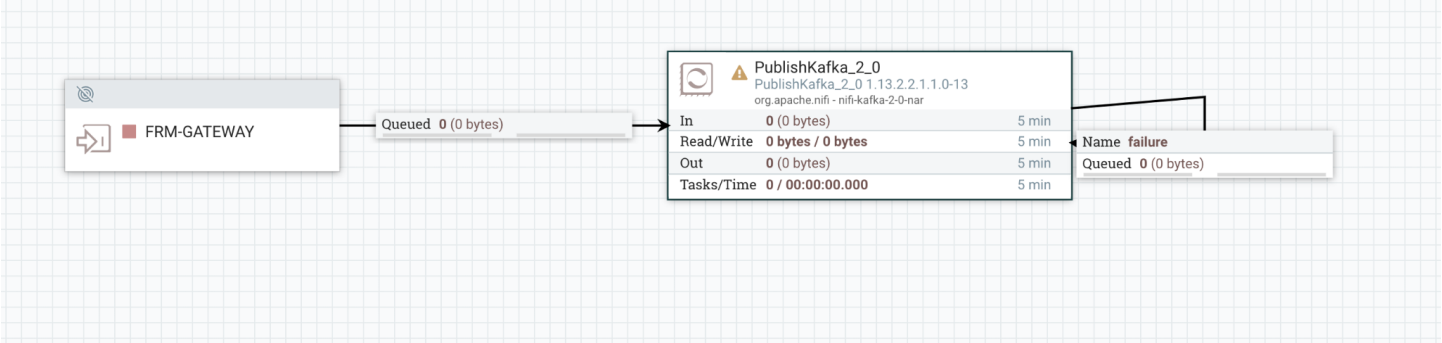

Create Connection
DETAILS
SETTINGS
From Input
FRM-GATEWAY
Within Group
NiFi Flow
To Processor
PublishKafka_2_0
PublishKafka_2_0
Within Group
NiFi Flow
CANCEL
ADD


Generate Test Data
ExecuteProcess 1.13.2.2.1.1.0-13
org.apache.nifi - nifi-standard-nar
In 0 (0 bytes) 5 min
Read/Write 0 bytes / 0 bytes 5 min
Out 0 (0 bytes) 5 min
Tasks/Time 0 / 00:00:00.000 5 min
FRM-GATEWAY
Queued 0 (0 bytes)
PublishKafka_2_0
PublishKafka_2_0 1.13.2.2.1.1.0-13
org.apache.nifi - nifi-kafka-2-0-nar
In 0 (0 bytes) 5 min
Read/Write 0 bytes / 0 bytes 5 min
Out 0 (0 bytes) 5 min
Tasks/Time 0 / 00:00:00.000 5 min

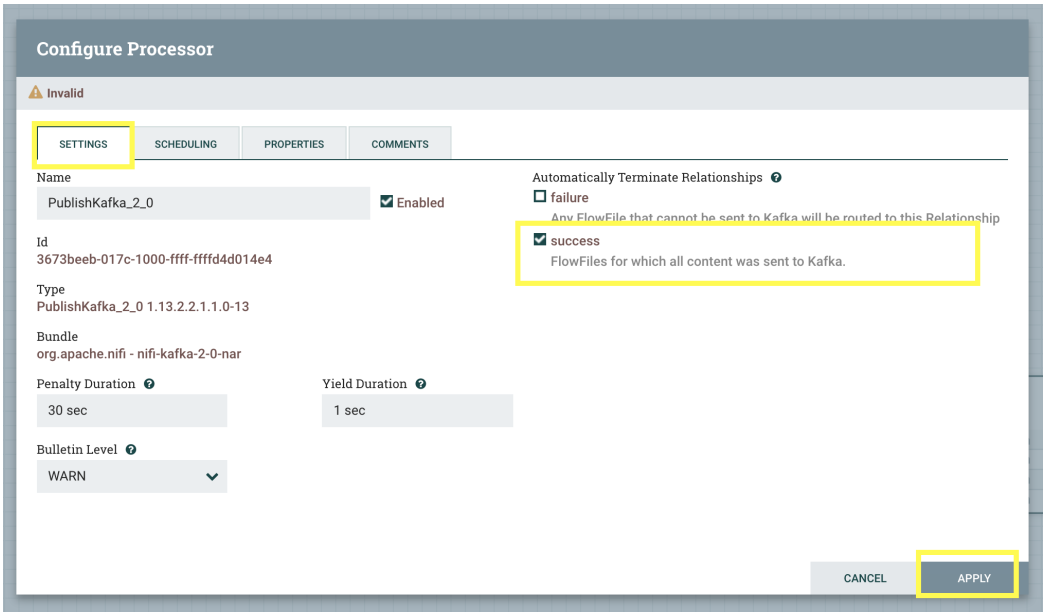
Create a connection from the PublishKafka to itself for failures (successes go to Kafka). Then you can start the Kafka processor.



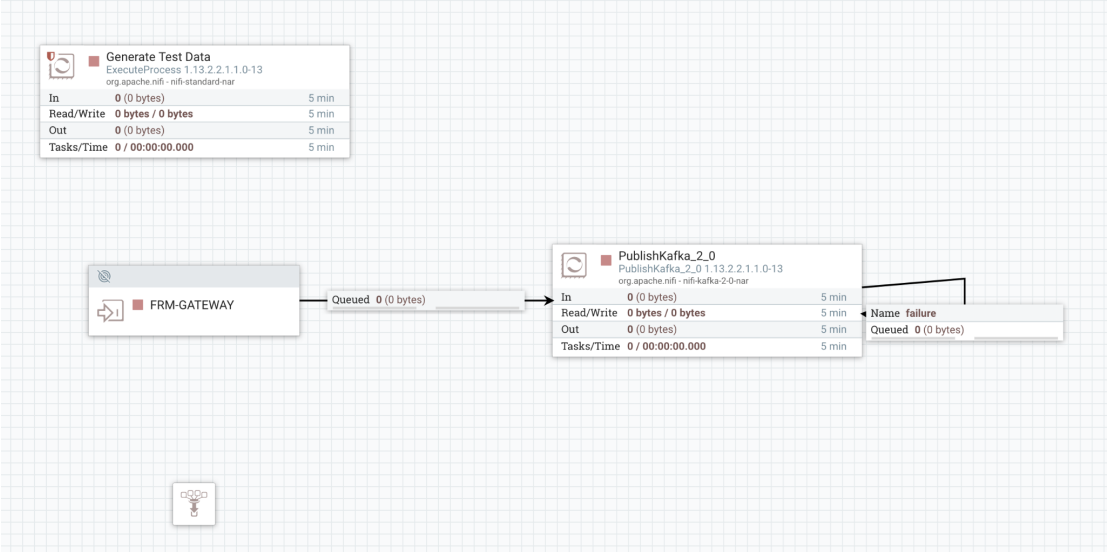






As the PublishKafka_2.0 processor still shows and invalid mark  we need to auto-terminate the processor on Success. Right click the process and click Configure, navigate to the Settings tab and check the box next to Success and click APPLY




The final flow on NiFi should look something like this




You can add more processors as needed to process, split, duplicate or re-route your FlowFiles to all other destinations and processors.

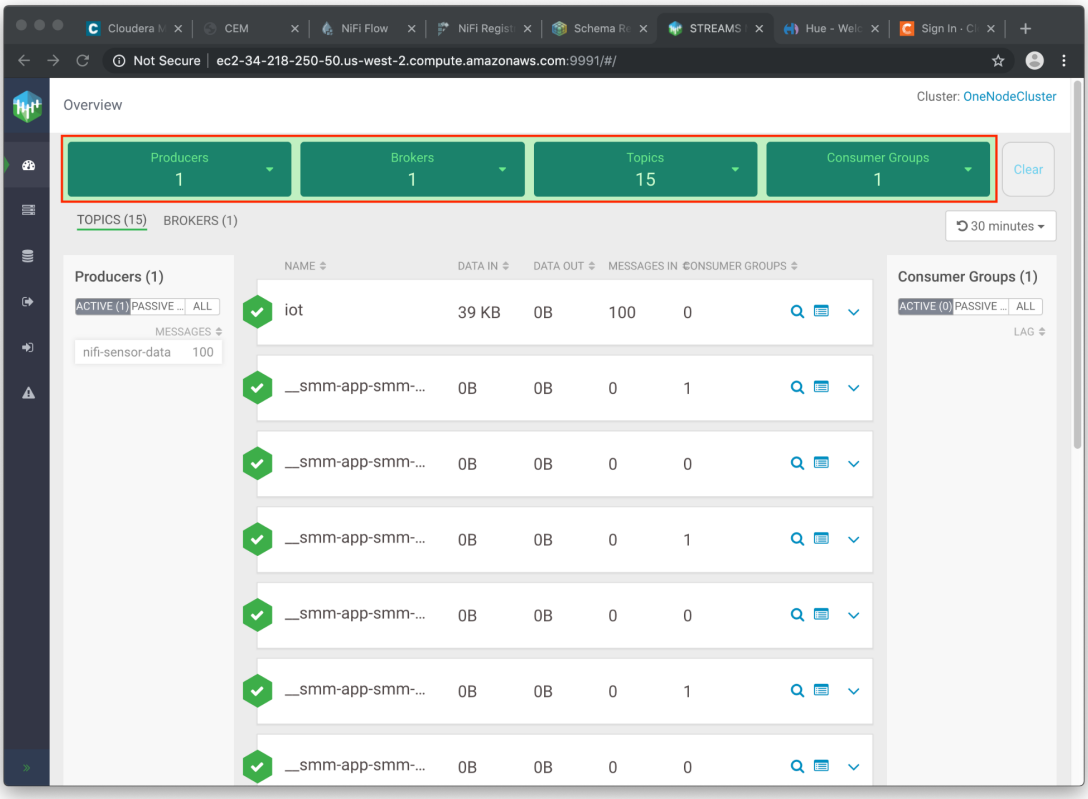
# Lab 4 - Streams Messaging Manager (SMM) to check and monitor Kafka

Now that our NiFi flow is pushing data to Kafka, it would be good to have a confirmation that everything is running as expected. In this lab you will use Streams Messaging Manager (SMM) to check and monitor Kafka.

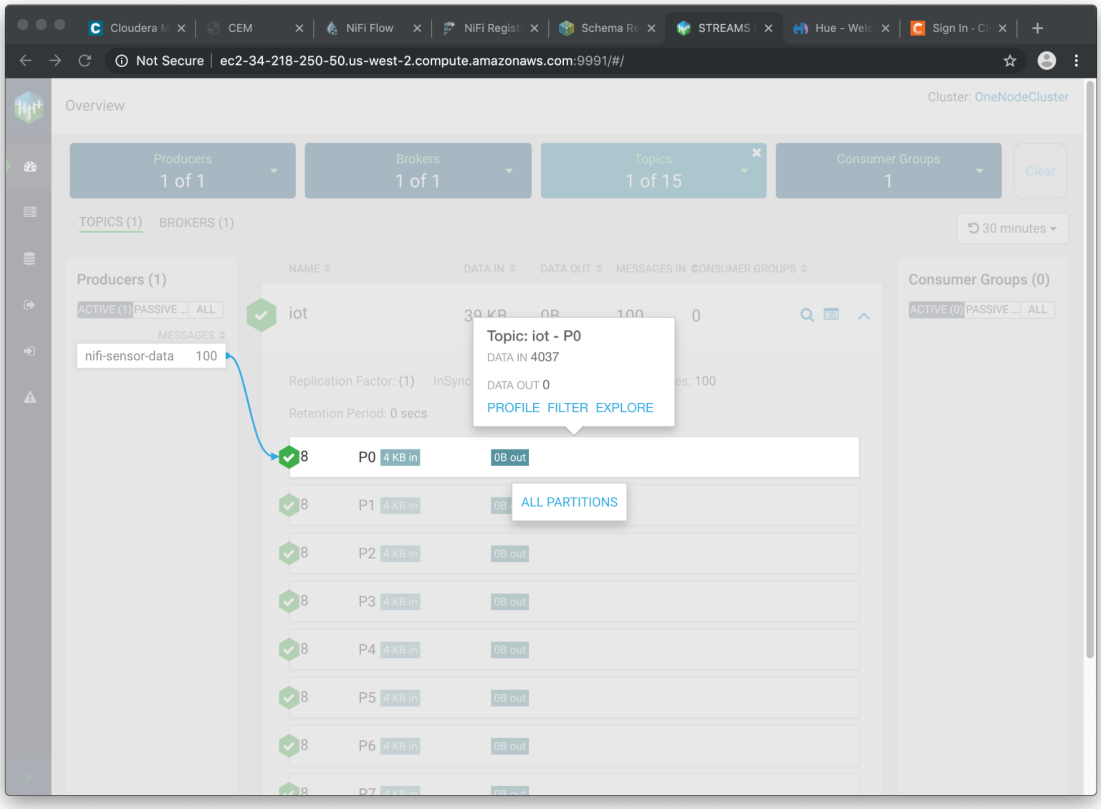
1. Start the NiFi ExecuteProcess simulator again and confirm you can see the messages queued in NiFi. Leave it running.
2. To access the SMM Web UI use the following URL

```
http://<PUBLIC-IP>:9991/#/
```

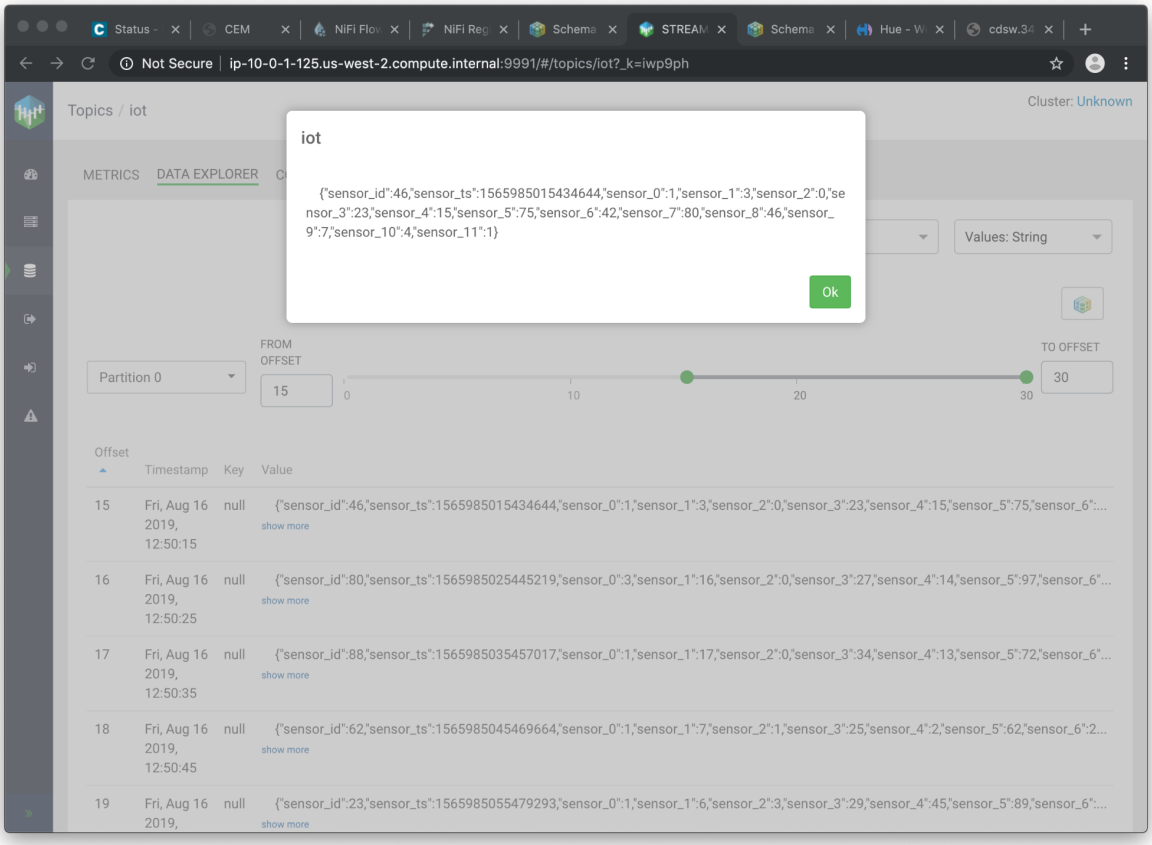
3. Go to the Stream Messaging Manager (SMM) Web UI and familiarize yourself with the options there. Notice the filters (blue boxes) at the top of the screen.



- Filter by Topic and select the `iot` topic, you'll be able to see all the producers and consumers that are writing to and reading from it, respectively. Since we haven't implemented any consumers yet, the consumer list should be empty.
- Click on the topic to explore its details. You can see more details, metrics and the breakdown per partition. Click on one of the partitions and you'll see additional information and which producers and consumers interact with that partition.




Click on the EXPLORE link to visualize the data in a particular partition. Confirm that there's data in the Kafka topic and it looks like the JSON produced by the sensor simulator.

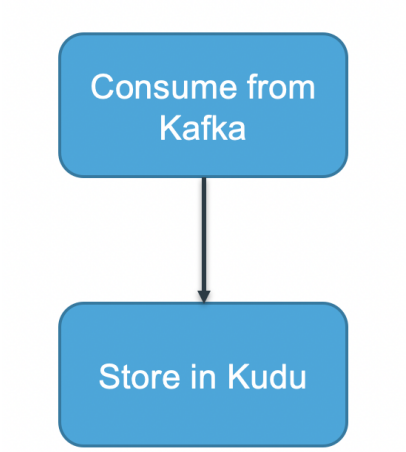

Not Secure ip-10-0-1-125.us-west-2.compute.internal:9991/#/topics/iot?_k=iwp9ph
Topics / iot
Cluster: Unknown
METRICS
DATA EXPLORER
iot
{"sensor_id":46,"sensor_ts":1565985015434644,"sensor_0":1,"sensor_1":3,"sensor_2":0,"sensor_3":23,"sensor_4":15,"sensor_5":75,"sensor_6":42,"sensor_7":80,"sensor_8":46,"sensor_9":7,"sensor_10":4,"sensor_11":1}
Ok
Values: String
Partition 0
FROM OFFSET
15
TO OFFSET
30
Offset
Timestamp
Key
Value
15 Fri, Aug 16 2019, 12:50:15 null {"sensor_id":46,"sensor_ts":1565985015434644,"sensor_0":1,"sensor_1":3,"sensor_2":0,"sensor_3":23,"sensor_4":15,"sensor_5":75,"sensor_6":...
show more
16 Fri, Aug 16 2019, 12:50:25 null {"sensor_id":80,"sensor_ts":1565985025445219,"sensor_0":3,"sensor_1":16,"sensor_2":0,"sensor_3":27,"sensor_4":14,"sensor_5":97,"sensor_6"...
show more
17 Fri, Aug 16 2019, 12:50:35 null {"sensor_id":88,"sensor_ts":1565985035457017,"sensor_0":1,"sensor_1":17,"sensor_2":0,"sensor_3":34,"sensor_4":13,"sensor_5":72,"sensor_6"...
show more
18 Fri, Aug 16 2019, 12:50:45 null {"sensor_id":62,"sensor_ts":1565985045469664,"sensor_0":1,"sensor_1":7,"sensor_2":1,"sensor_3":25,"sensor_4":2,"sensor_5":62,"sensor_6":2...
show more
19 Fri, Aug 16 2019, null {"sensor_id":23,"sensor_ts":1565985055479293,"sensor_0":1,"sensor_1":6,"sensor_2":3,"sensor_3":29,"sensor_4":45,"sensor_5":89,"sensor_6":...
show more

# Lab 5 - From Kafka to Kudu

1. Open the browser and navigate to the HUE UI using the URL
   http://<your vm-public-IP>:8888

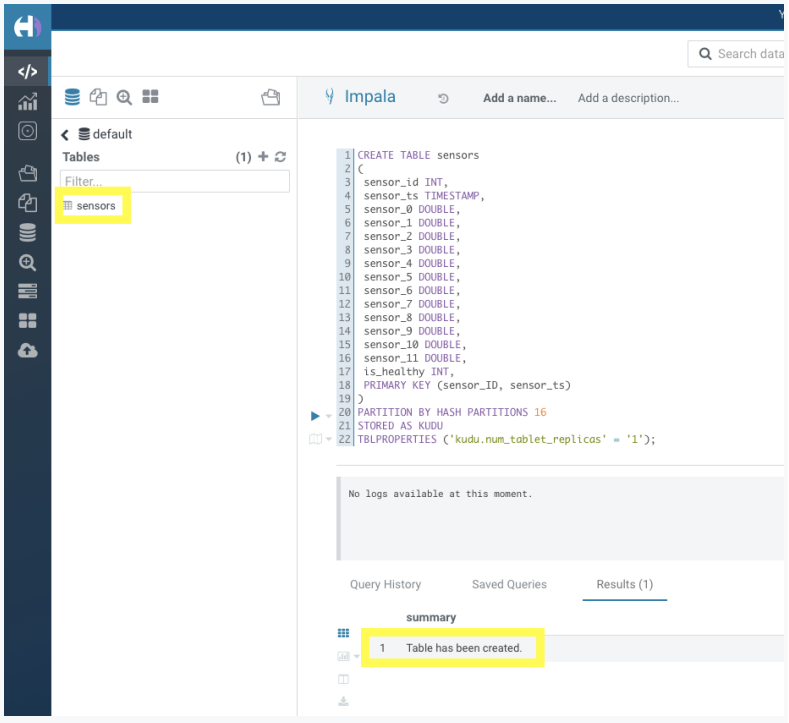
   Create your account using the following credentials:
   UserName : hueuser
   Password: hueuser

2. Until lab 4 we have built our first flow. The next flow will read the Kafka records and store them in Kudu. The high level view of this flow looks like the following:

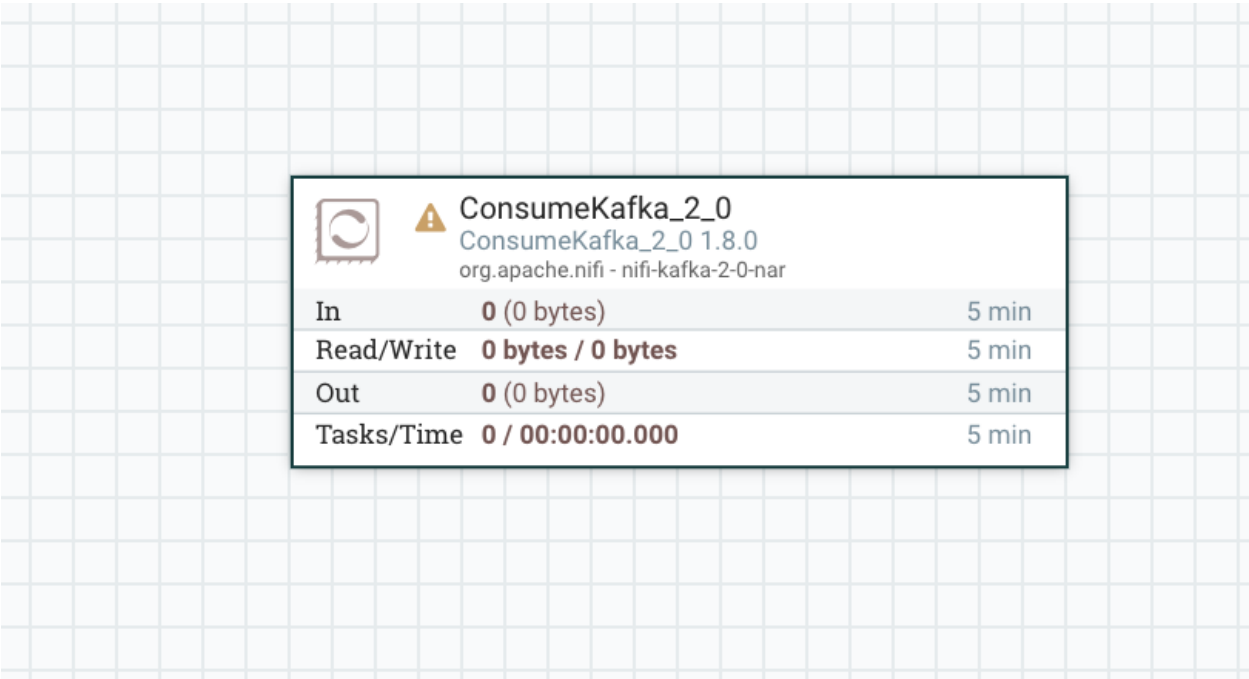



3. First, create the Kudu table. Login into Hue, and in the Impala Query, run this statement:

```
CREATE TABLE sensors
(
 sensor_id INT,
 sensor_ts TIMESTAMP,
 sensor_0 DOUBLE,
 sensor_1 DOUBLE,
 sensor_2 DOUBLE,
 sensor_3 DOUBLE,
 sensor_4 DOUBLE,
 sensor_5 DOUBLE,
 sensor_6 DOUBLE,
 sensor_7 DOUBLE,
 sensor_8 DOUBLE,
 sensor_9 DOUBLE,
 sensor_10 DOUBLE,
 sensor_11 DOUBLE,
 is_healthy INT,
 PRIMARY KEY (sensor_ID, sensor_ts)
)
PARTITION BY HASH PARTITIONS 16
STORED AS KUDU
TBLPROPERTIES ('kudu.num_tablet_replicas' = '1');
```




Let's start! Now you should already be familiar with how to add a new processor to the NiFi canvas. Add the **ConsumeKafka_2_0** processor:



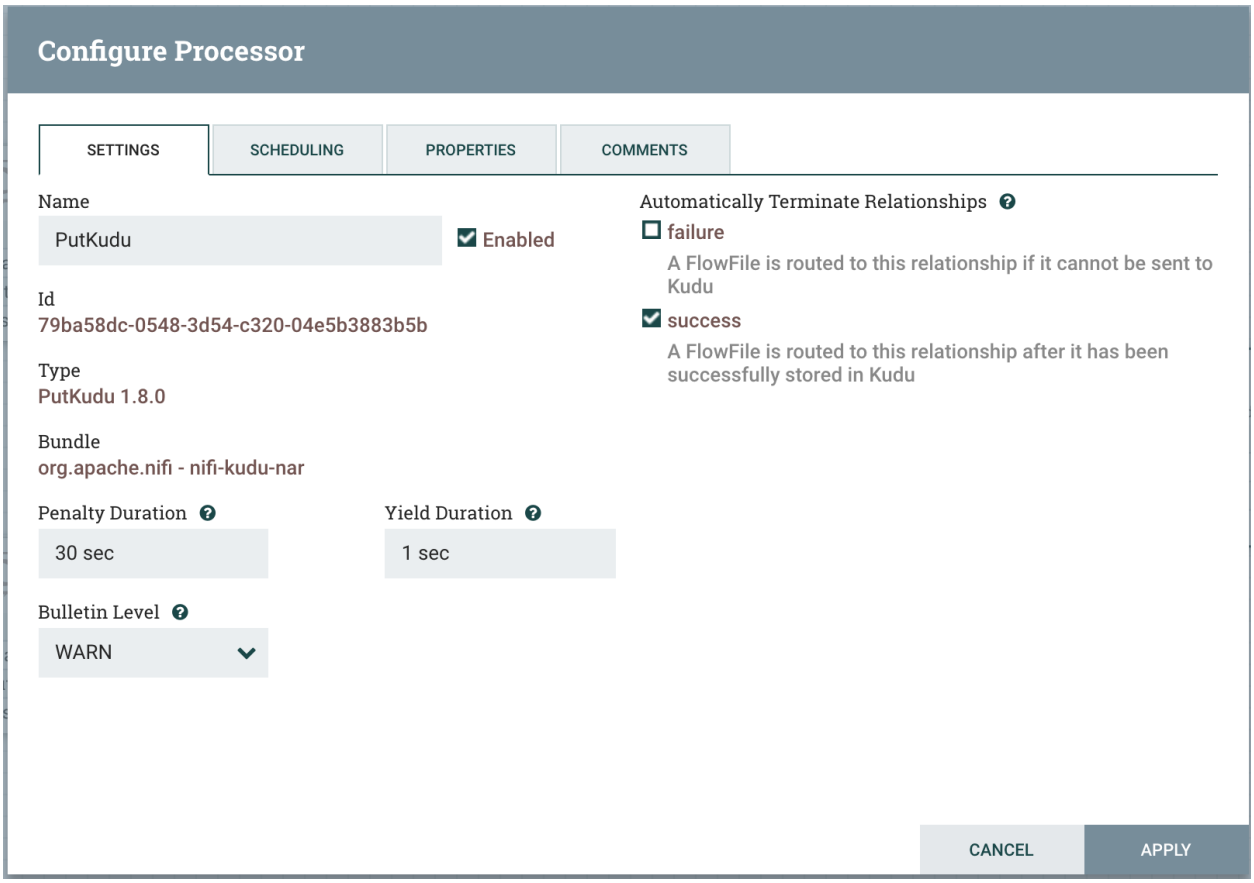

Double click on the **ConsumeKafka_2_0** and go to the **PROPERTIES** tab

And add the value for the following properties:

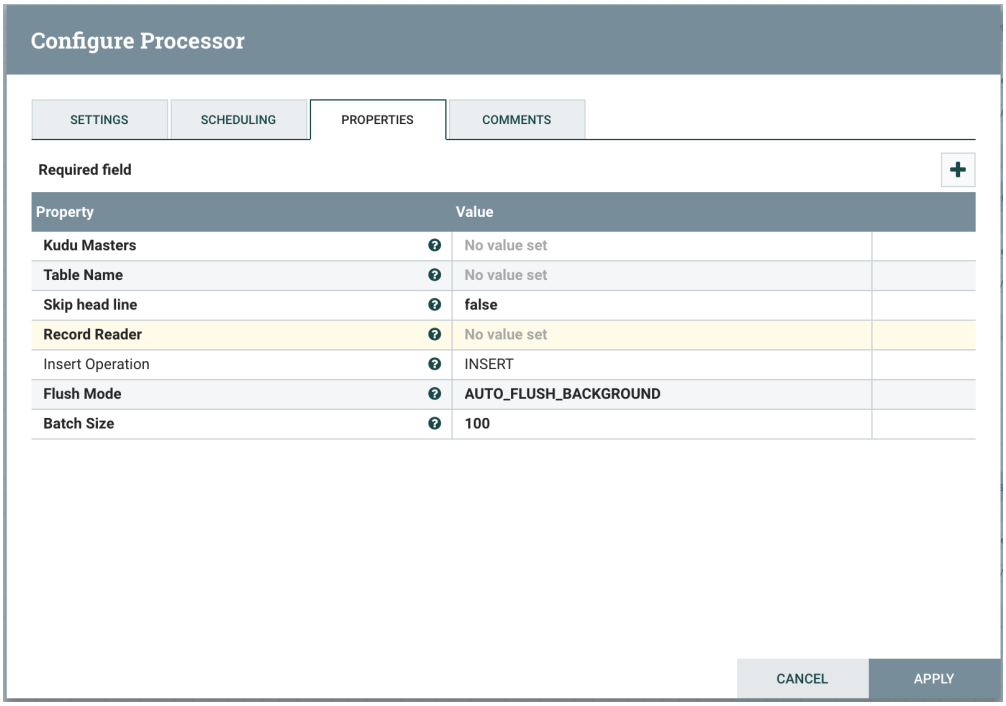
- **Kafka Brokers**: <internal-hostname>:9092
- **Topic Name(s)**: iot
- **Group ID**: nifi_consumer

Once done, click **APPLY**

Now add a **PutKudu** processor to the canvas and connect the **ConsumeKafka** to the **PutKudu** component, then double click on **PutKudu** and go to the **SETTINGS** tab.



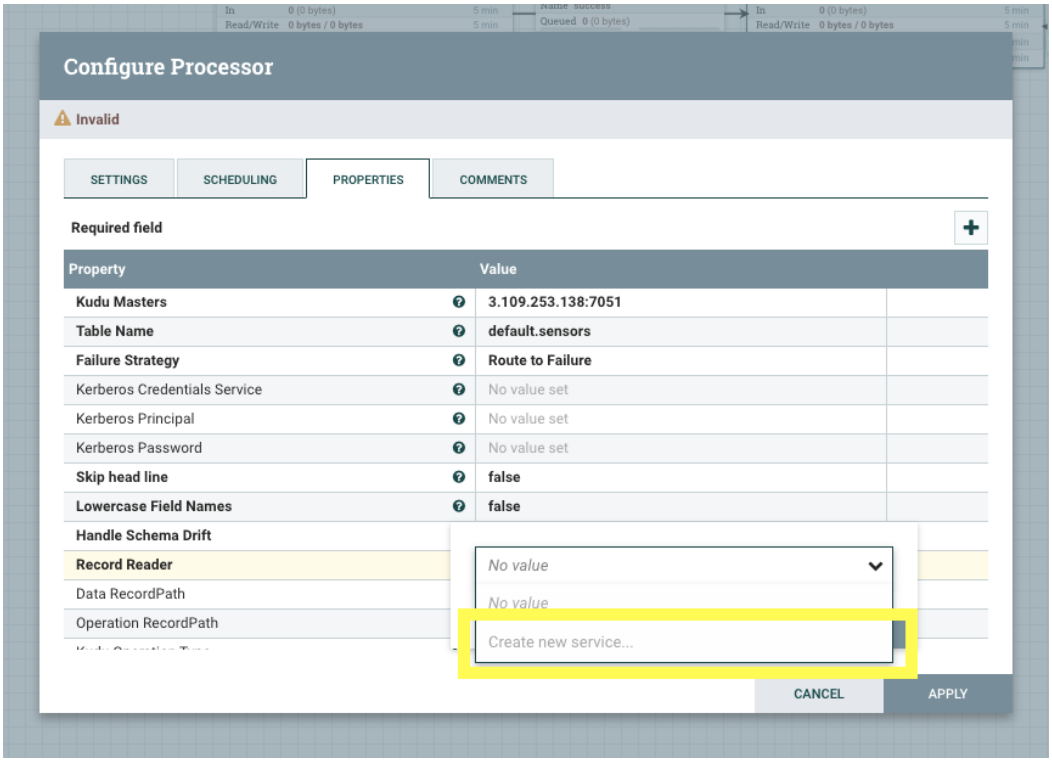

Check the **success** box and go to the **PROPERTIES** tab

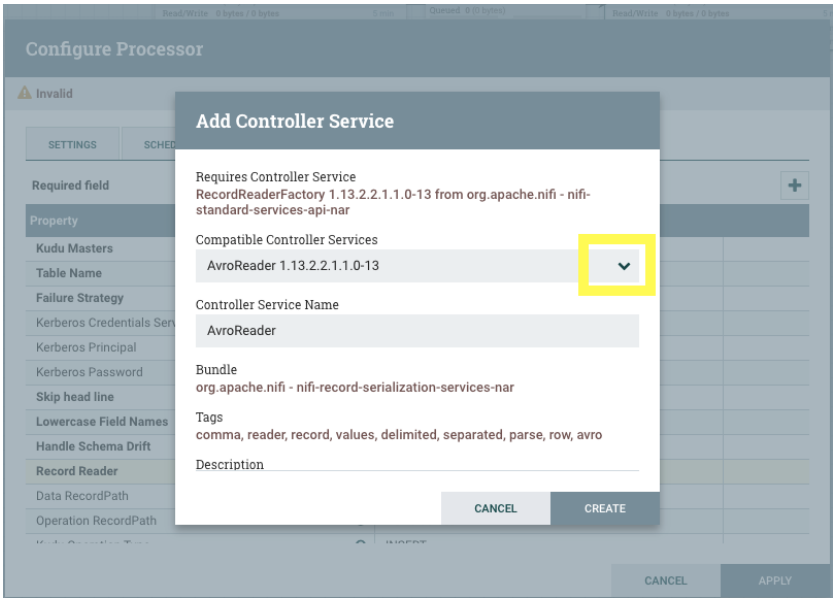



Set:

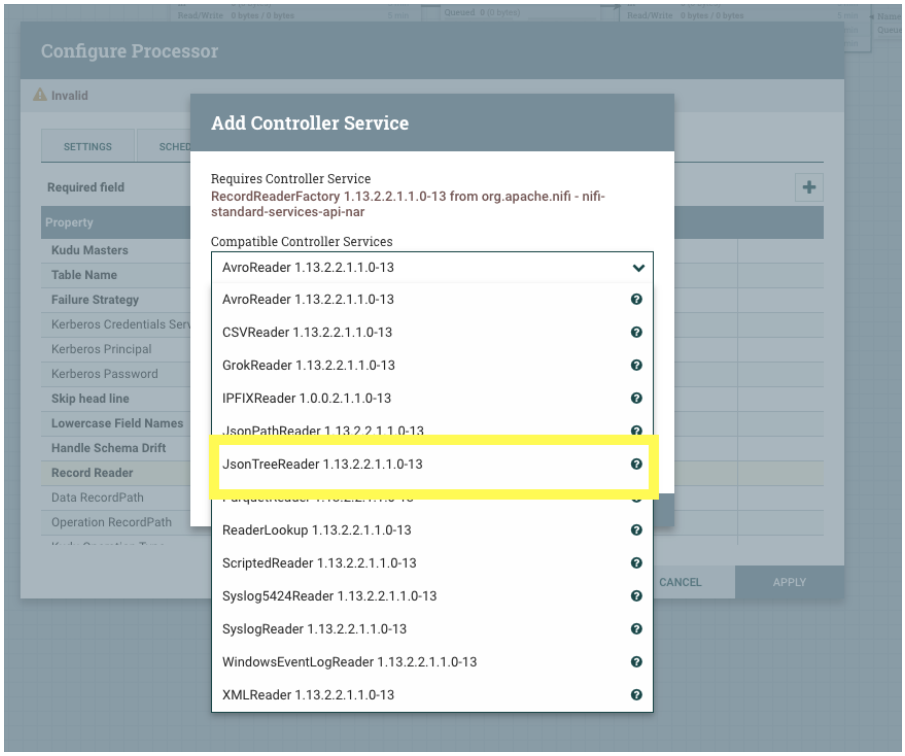
- **Kudu Masters**: <internal-hostname>:7051
- **Table Name**: default.sensors
- **Insert Operation**: UPSERT

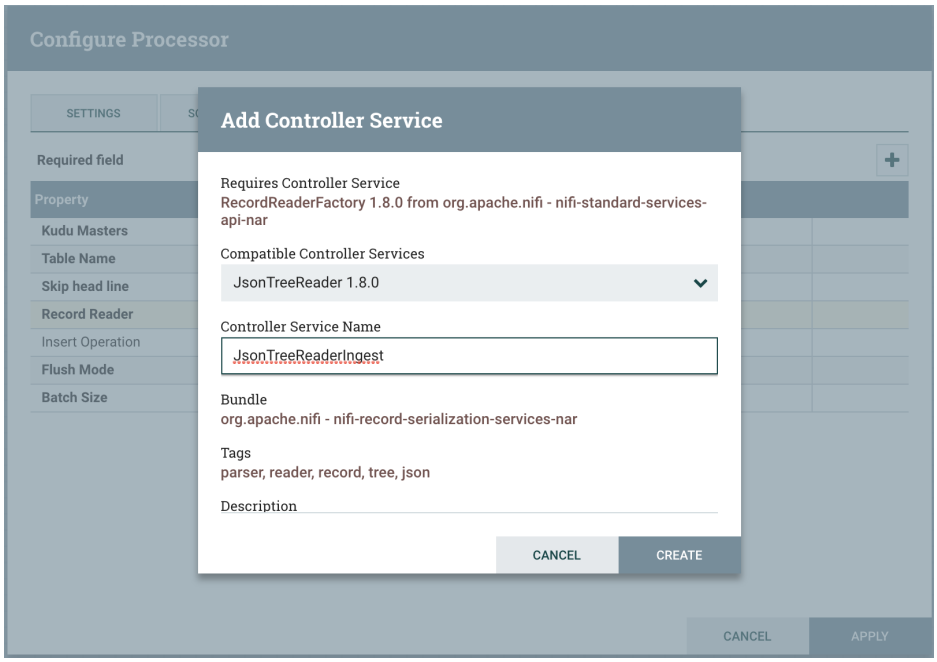
Then for the **Record Reader**, click on **Create new service…**



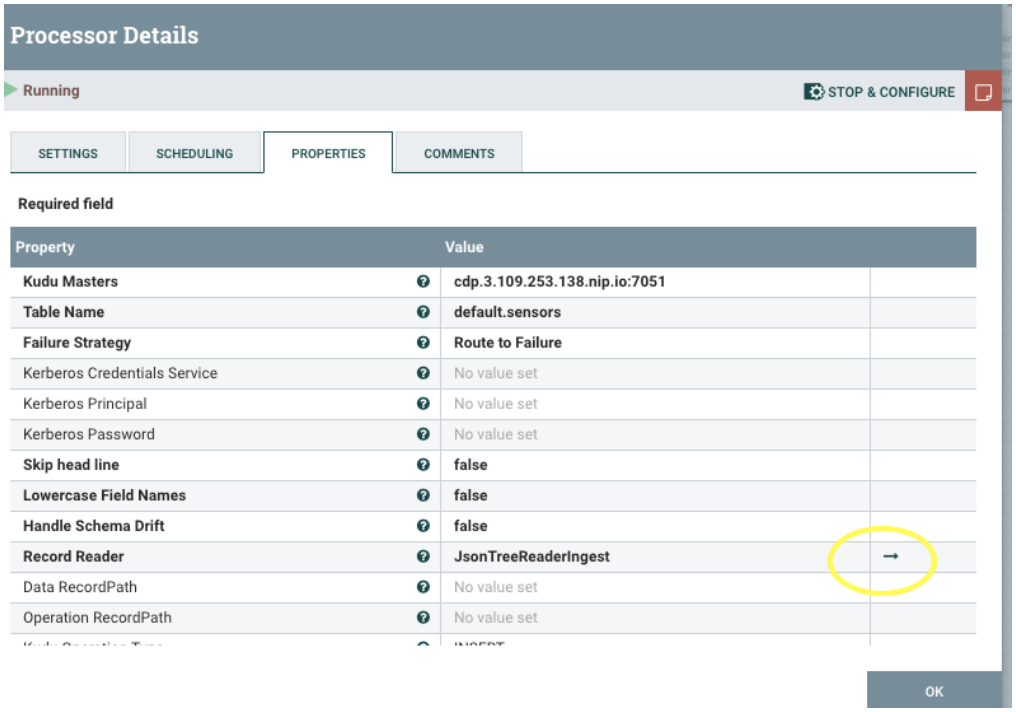



In the **Compatible Controller Services** select **JsonTreeReader** and give it a name (JsonTreeReaderIngest)

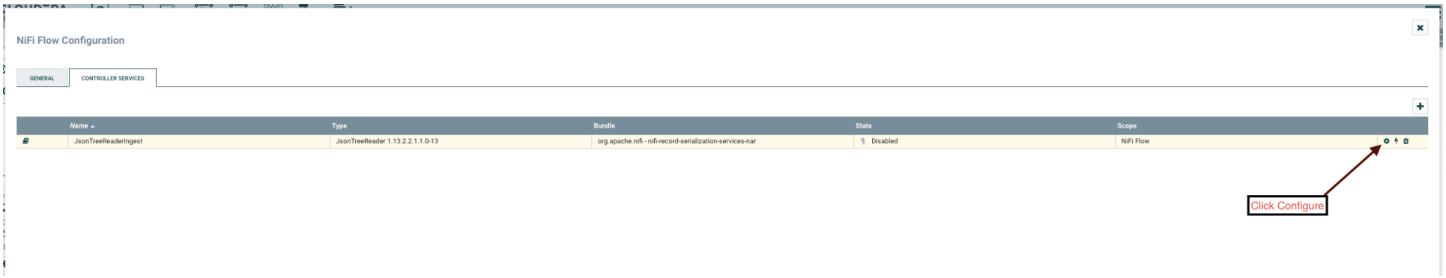





Then click **CREATE,** and click on the small arrow to bring you to the **Controller Services**.



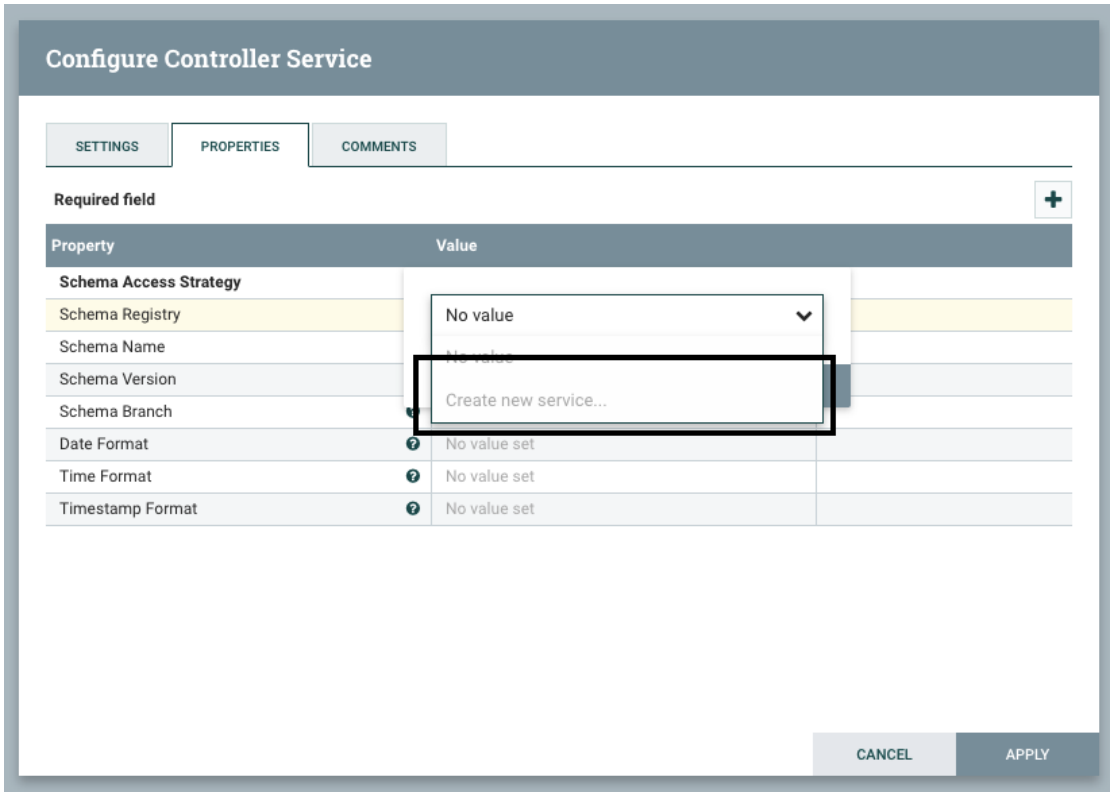
Click on the **configure** icon on the right end of the **JsonTreeReaderIngest** row and go to the **PROPERTIES** tab. Set



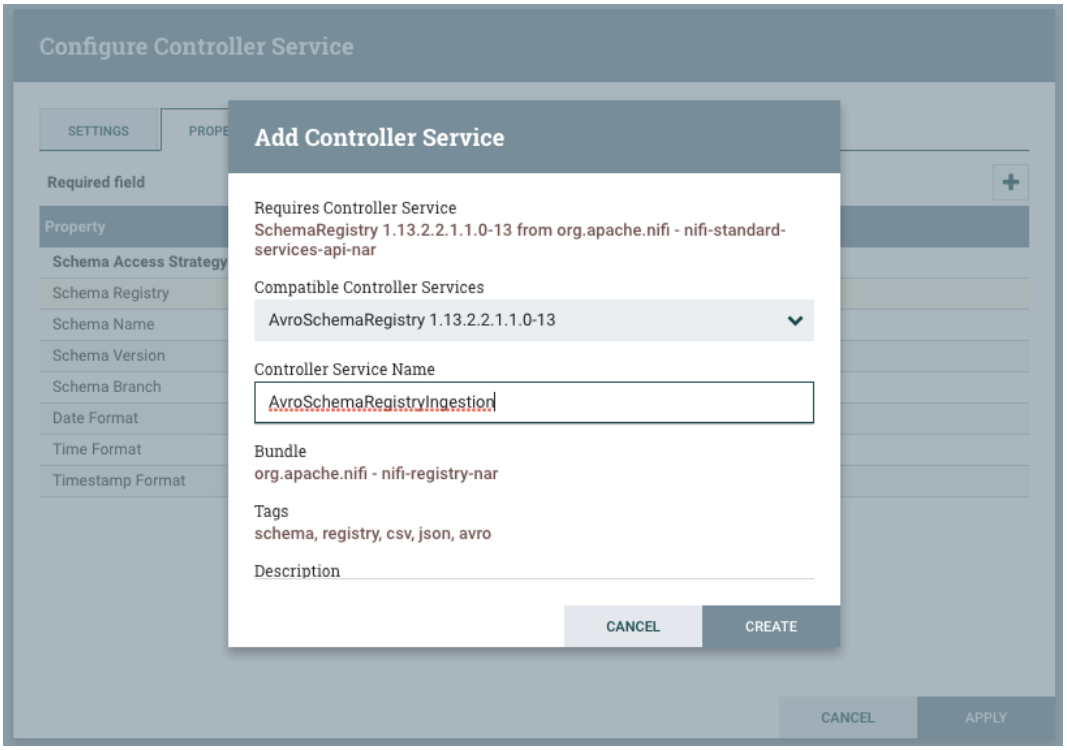

- Schema Access Strategy = Use ‘Schema Name’ Property
- Schema Name = demo-schema

## Create Schema Registry

The JsonReader needs to know how to interpret the json string it will receive, that is, it needs to know the schema. We will now create another **controller service** that will store the schema.

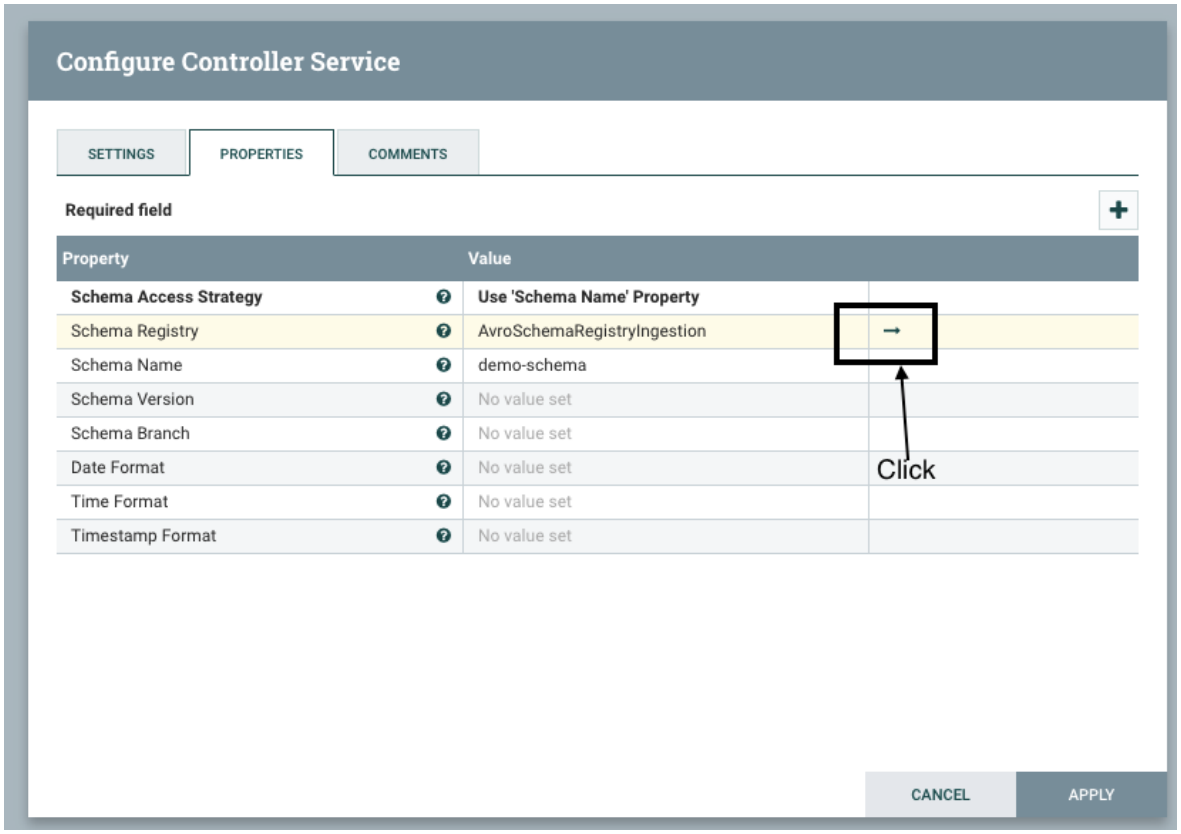



In the **Schema Registry** field click to the value field, click **Create new service** and select **AvroSchemaRegistry** Call it *AvroSchemaRegistryIngestion*.



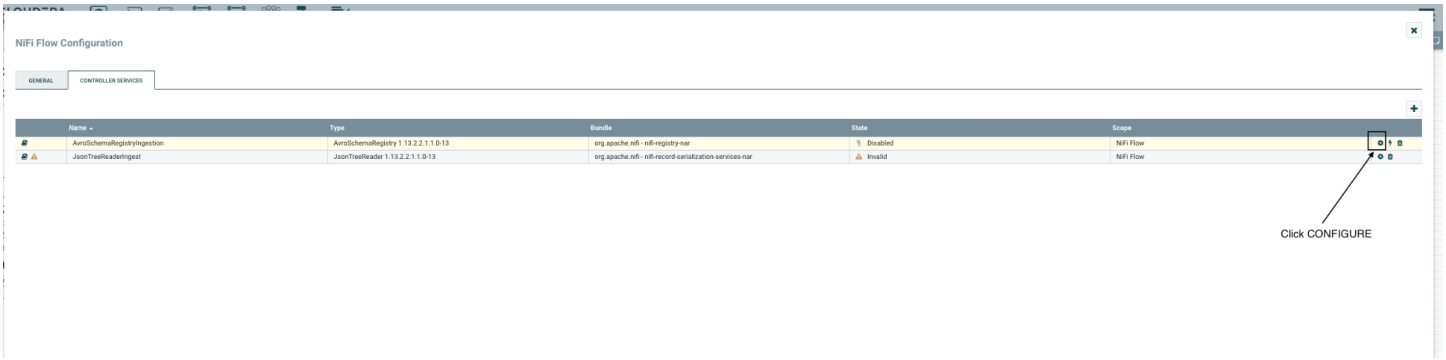

Then click **CREATE**.

Now click on the arrow icon in the **Schema Registry** row

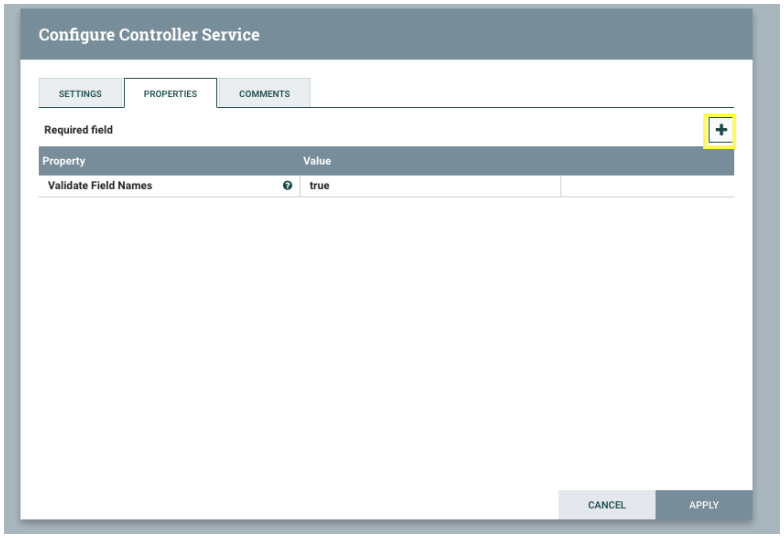


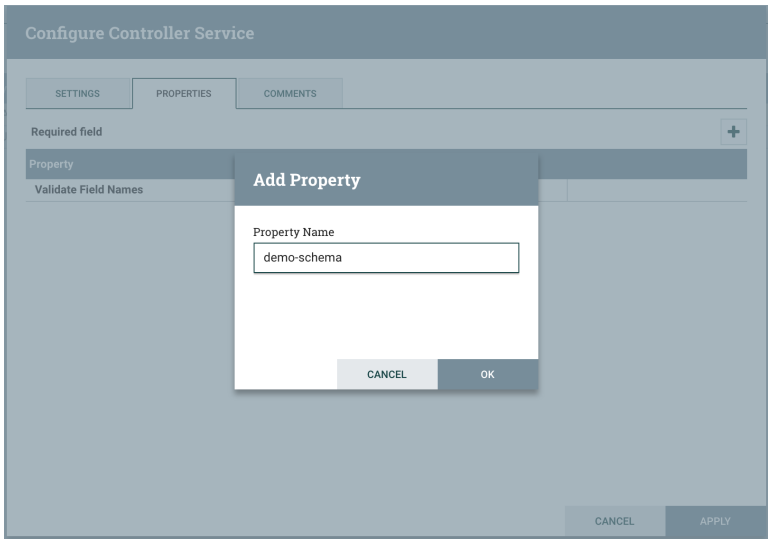
Click YES to save the changes.

Click on the **Configure** icon for the **AvroSchemaRegistryIngestion**




then click on the plus sign and add value *demo-schema*



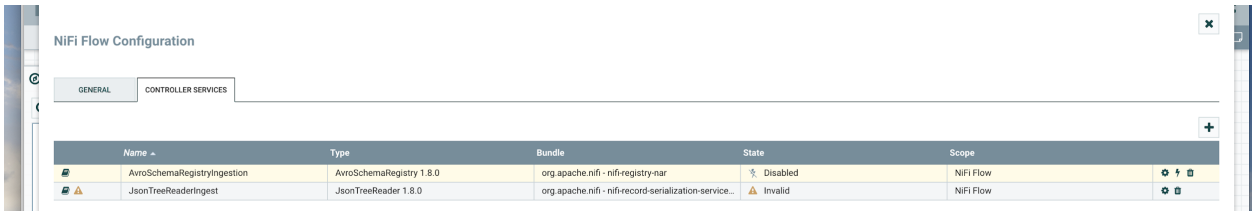



## Insert the below schema:

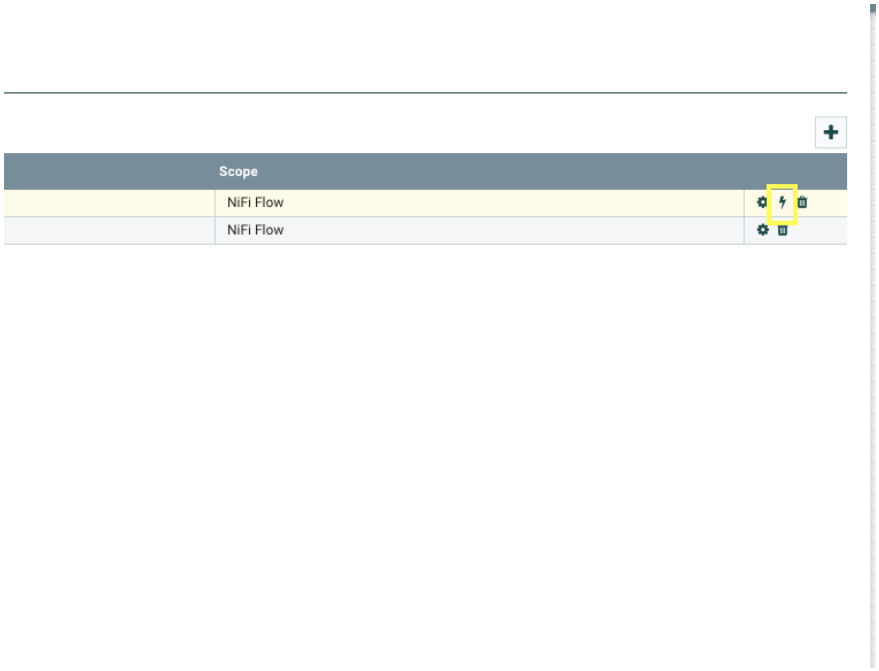
```
{
 "type": "record",
 "name": "SensorReading",
 "namespace": "com.cloudera.example",
 "doc": "This is a sample sensor reading",
 "fields": [
  {
   "name": "sensor_id",
   "doc": "Sensor identification number.",
   "type": "int"
  },
  {
   "name": "sensor_ts",
   "doc": "Timestamp of the collected readings.",
   "type": "long"
  },
  {
   "name": "is_healthy",
   "doc": "Flag indicating health (healthy == 1)",
   "type": "int"
  },
  {
   "name": "response",
   "doc": "response record",
   "type": {
    "type": "record",
    "name": "CDSWResponse",
    "namespace": "com.cloudera.example",
    "doc": "This is a CDSW Model response",
    "fields": [
     {
      "name": "result",
      "doc": "Result",
      "type": "int"
     }
    ]
   }
  },
  {
  "name": "sensor_0",
  "doc": "Reading #0.",
  "type": "int"
  },
  {
  "name": "sensor_1",
  "doc": "Reading #1.",
  "type": "int"
  },
  {
  "name": "sensor_2",
  "doc": "Reading #2.",
  "type": "int"
  },
  {
  "name": "sensor_3",
  "doc": "Reading #3.",
  "type": "int"
  },
  {
  "name": "sensor_4",
  "doc": "Reading #4.",
  "type": "int"
  },
  {
  "name": "sensor_5",
  "doc": "Reading #5.",
  "type": "int"
  },
  {
  "name": "sensor_6",
  "doc": "Reading #6.",
  "type": "int"
  },
  {
  "name": "sensor_7",
  "doc": "Reading #7.",
  "type": "int"
  },
  {
  "name": "sensor_8",
  "doc": "Reading #8.",
  "type": "int"
  },
  {
  "name": "sensor_9",
```

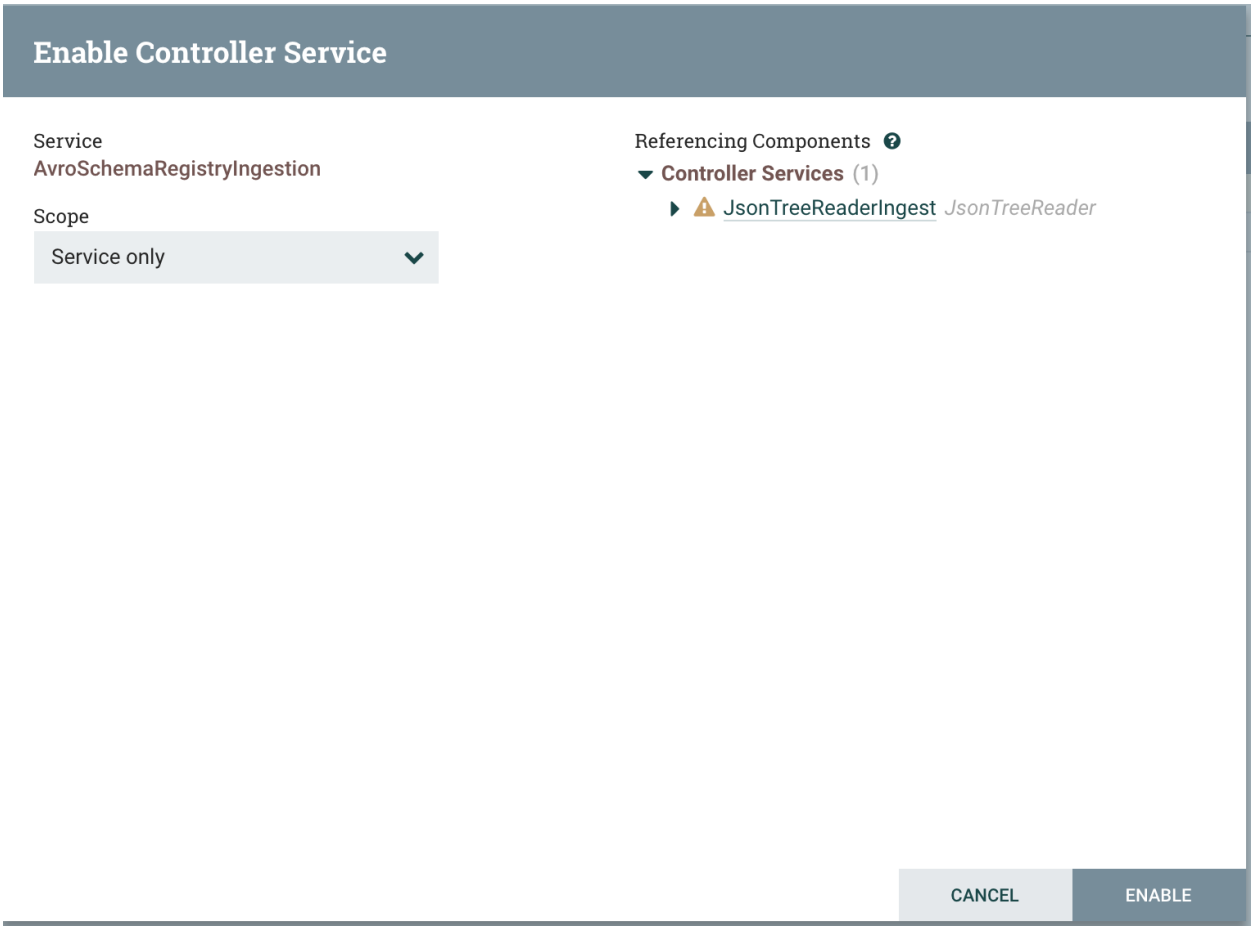
```
  "doc": "Reading #9.",
  "type": "int"
  },
  {
  "name": "sensor_10",
  "doc": "Reading #10.",
  "type": "int"
  },
  {
  "name": "sensor_11",
  "doc": "Reading #11.",
  "type": "int"
  }
 ]
}
```

And click **OK**, then click **APPLY.**

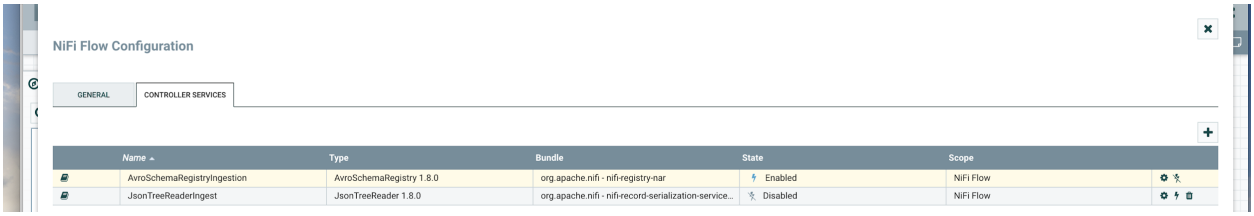



On the right, click the **Enable** icon for the **AvroSchemaRegistryIngestion**

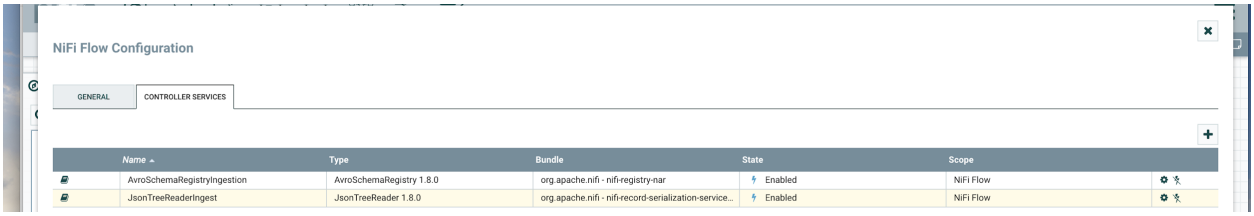





Click **ENABLE** and then **CLOSE**, and the warning icon should disappear

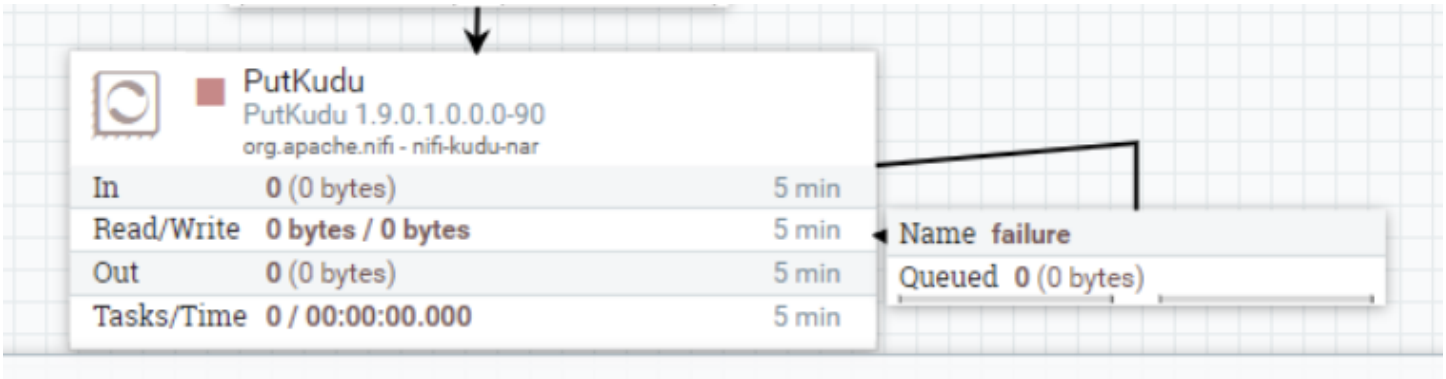



Then enable also the **JsonTreeReaderIngest.**

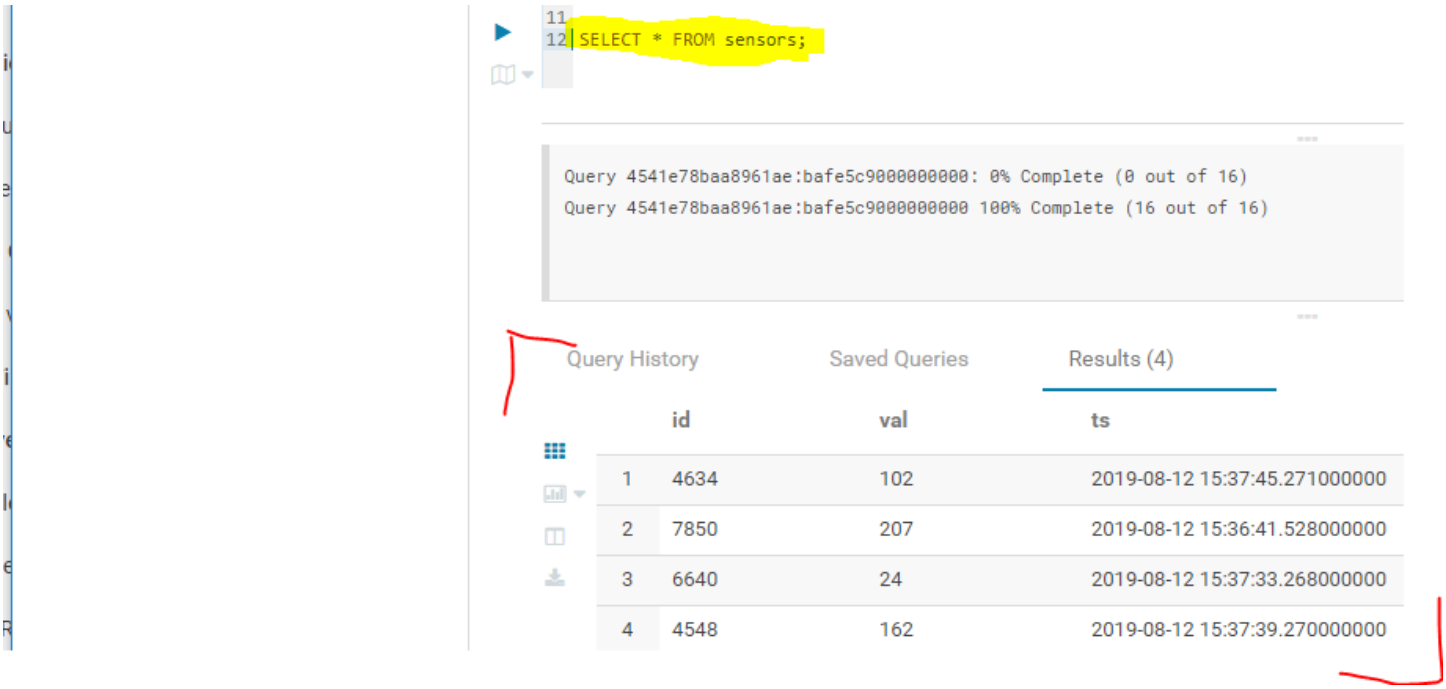



You can now close the **NiFi Flow Configuration** and go back to the main NiFi canvas

Now drag a connection from the **PutKudu** to itself




Now we are ready to test the end2end flows. Start each component and verify that data is stored in Kudu:




| | id | val | ts |
|---|---|---|---|
| 1 | 4634 | 102 | 2019-08-12 15:37:45.271000000 |
| 2 | 7850 | 207 | 2019-08-12 15:36:41.528000000 |
| 3 | 6640 | 24 | 2019-08-12 15:37:33.268000000 |
| 4 | 4548 | 162 | 2019-08-12 15:37:39.270000000 |

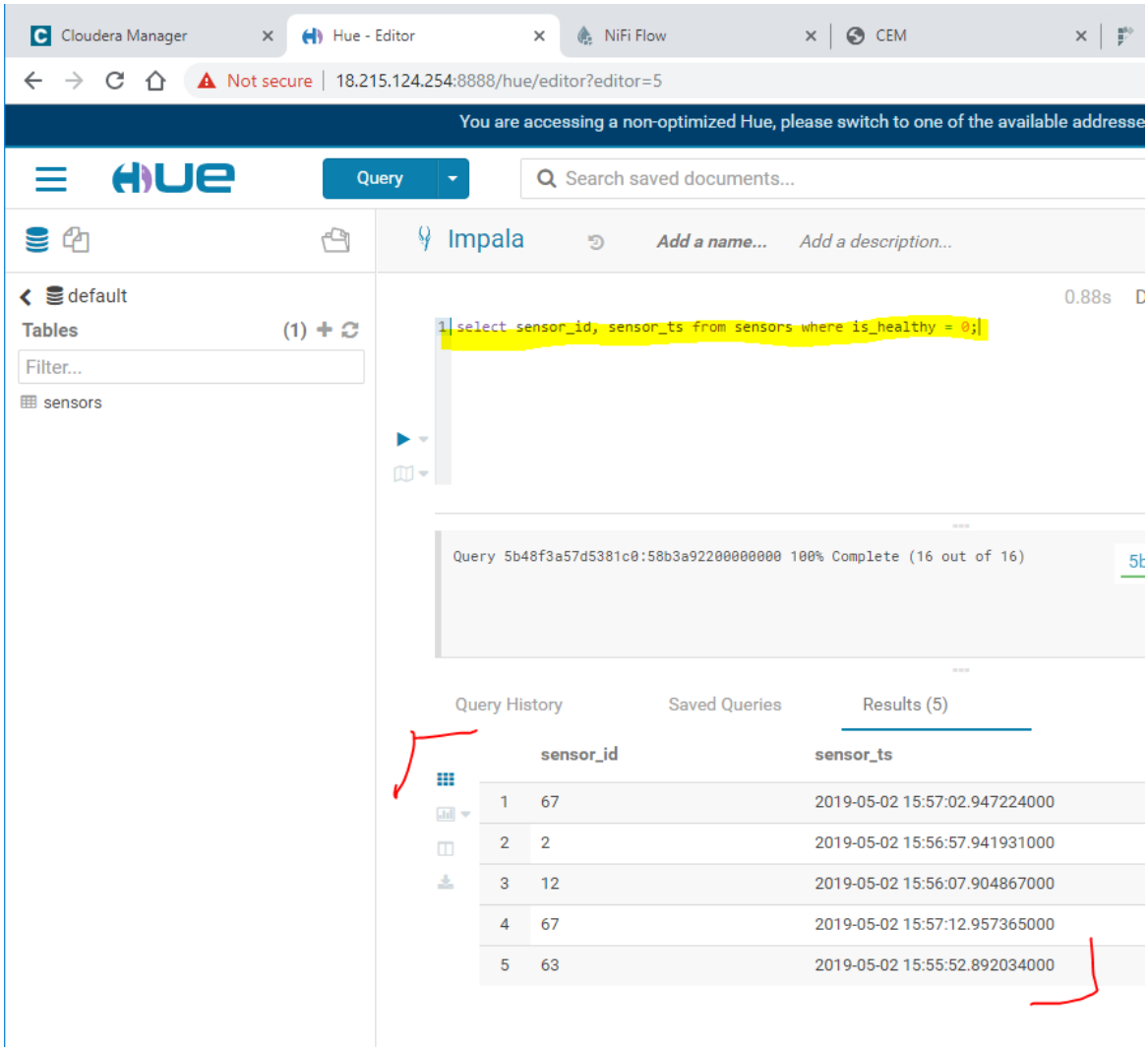
# Lab 6 - Fast analytics on fast data with Kudu and Impala

In this lab, you will run some SQL queries using the Impala engine. You can run a report to inform you which machines are likely to break in the near future.

Login into Hue, and run the following statement in the Impala Query

```
select sensor_id, sensor_ts from sensors where is_healthy = 0;
```

Run a few times a SQL statement to count all rows in the table to confirm the latest inserts are always picked up by Impala. This allows you to build real-time reports for fast action.



# This concludes Cloudera Data Flow Workshop

## Thanks for your participation..