Rapport d'Architectures et programmations distribuées

Ait Ettajer Haytham - Théophile Diez - Eric Sombroek - Alexandre Olives - Matthieu Taieb

Analyze-iot-weather-station

Lors du module d’architecture nous avions le choix entre plusieurs projets à faire. Nous avons choisi le projet d’IOT. Le but du projet est d’utiliser l’internet des objets pour récupérer les informations qui viennent des différentes “Weather station” à travers la Raspberry Pi. Nous vous montrons le déroulement du projet.

Context:

Over the past two years, San Jose has experienced a shift in weather conditions from having the hottest temperature back in 2016 to having multiple floods occur just within 2017. You have been hired by the City of San Jose as a Data Scientist to build Internet of Things (IoT) and Big Data project, which involves analyzing the data coming in from several weather stations using a data-in-motion framework and data-at-rest framework to improve monitoring the weather.

For this purpose, we used:

- Hortonworks Connected Data Architecture (Hortonworks Data Flow (HDF)
- Hortonworks Data Platform (HDP))
- MiNiFi subproject of Apache NiFi to build this product.
- Raspberry Pi
- Sense HAT to replicate the weather station data
- HDF Sandbox and HDP Sandbox on Docker to analyze the weather data

The goal was to show meaningful insights on temperature, humidity and pressure reading and to learn how to use these technology.

Step 1: Setup

1. connect the Sense HAT hardware to the Raspberry Pi through the pins
2. Plug a microUSB to provide power and internet access
3. Access the Raspberry Pi by SSH
4. Setup the environment on Raspberry Pi
    a. install Sense Hat soft

```
sudo apt-get update
sudo apt-get install sense hat
sudo pip3 install pillow
```

    b. MiNiFi agent

```
sudo apt-get update && sudo apt-get install oracle-java8-jdk
upload MiNiFi agent
unzip minifi-[version]-bin.zip
```

5. Setup the environment on our computer
    a. MiNiFi toolkit
    b. Calibrate the Raspberry Pi

Step 2: Start VirtualBox
Start the VB manager and start a 32Go RAM
For the 2 first month we got problem on the installation with azure .

Step 3: Map IP to the Hostname
In \drivers\etc, open hosts and map the VB IP

Step 4: Setup account for HDP/HDF Ambari

Step 5: Configure NiFi

1. Login http://sandbox-hdf.hortonworks.com:8080:
2. Update NiFi configuration and restart

Here start the problem when we try to launch NiFi with the azure horton we got a problem on the launch and when we try to launch it with the VM on the ISEP session we cannot find the module of NiFi .

So we can't do the next step (Adding GeoLite2 database
    a. Access webshell client
    b. Create directory and grant permissions NiFi needs
    c. download GeoLite2

The table is accessible in
/sandbox/tutorial-files/820/nifi/input/GeoFile/GeoLite2-City_20180605/GeoLite2-City.mmdb)

The environment is now ready to use. We will now focus on the preprocessing and storing of datas in a NoSQL database and analyse it in real-time

After seeing the IOT weather station deployed and the data architecture connected we're going to collect SAT HAT weather data via CDA.

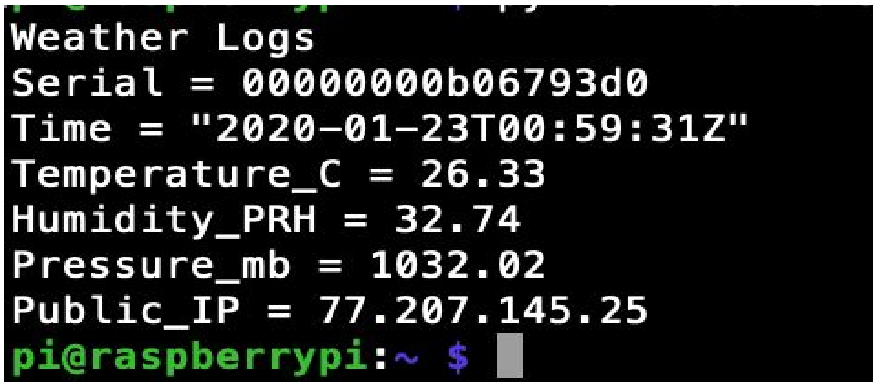
Step 1: create a Python script to record the SAT HAT weather data

To do this, we chose to implement a python script on the Raspberry Pi.
First we create a python file, then we retrieve the serial number of the Raspberry Pi with the command "get_serial ()". Then, we get the time of the sensor reading via the "get_time ()" command. And finally we collect the weather readings from Sense HAT.

Then we calibrated the SAT HAT temperature sensor readings. Indeed, the temperature sensor readings are skewed due to the Raspberry Pi's processor which emits heat close to the sensor. So in order to recover the real value of the measurements we have to calibrate the sensor. For this, the function "get_cpu_temp_c ()" allows to recover the CPU temperature. This value is then used to calibrate the sensor thanks to the function "calibrate_temp_c (cpu_temp_c, temp_c)".

Then we have to recover the public IP address of the Raspberry Pi. A python function allows us to retrieve it in a JSON file. Finally we just have to display the weather values on the screen.

All we have to do is to execute the previous functions correctly in order in a python script, save it as a file and run it.



```
Weather Logs
Serial = 00000000b06793d0
Time = "2020-01-23T00:59:31Z"
Temperature_C = 26.33
Humidity_PRH = 32.74
Pressure_mb = 1032.02
Public_IP = 77.207.145.25
pi@raspberrypi:~ $
```

Step 2: Create a NiFi stream to store MiNiFi data on HDFS

We started by configuring NiFi on HDFS, to do this we accessed the NiFi user interface at http://sandbox-hdf.hortonworks.com:9090/nifi/, added an input port we named “from_MiNiFi”, and a processor we named PutHDFS and changed its properties. Then all that was left to do was to set up PutHDFS.

Step 3: create a MiNiFi stream to transfer data to NiFi

We first created a MiNiFi stream using NiFi and then saved it as a NiFi model and converted it to a MiNiFi model. We then checked the status of PutHDFS with the source of the data and verified that the data is stored in HDFS via HDP Files View.

In this part we install the module in the raspberry but on the Azure we got problem when we try to install the NiFi module we got an error and on the isep account we don't got the opportunity to install NiFi because the hdp version is too old maybe.

After checking that we verified that the raw sensor data move from MiNiFi to HDF NiFi to HDP HDFS. We gonna add geographic location attributes to the dataset.

1. Creation of the HBase Table

For the first step of this category we add the sense_hat_logs to the table. For that we access HDP Sandbox shell with the Web Shell Client at http://sandbox-hdp.hortonworks.com:4200. Then we Open HBase Shell: "hbase shell" and put "create 'sense_hat_logs','weather'". At this point NiFi will have a place to store the sensor data.

2. Enhance NiFi Flow to Store Geo Data to HBase

For the next step we download and import a prebuilt NiFi DataFlow template to NiFi.

After downloading WeatherDataMiNiFiToHbase.xml template file onto the computer. We head to NiFi UI at http://sandbox-hdf.hortonworks.com:9090/nifi. Then we use the template icon located in the Operate Palette. From the Components Toolbar, we drag the template icon onto the graph and select the WeatherDataMiNiFiToHBase template file. We remove the queue between Copy of From_MiNiFi and PreProcessDataForHBaseAndHadoop by right clicking on the queue. After that we remove Copy of From_MiNiFi input. Next, we connect From_MiNiFi input port to PreProcessDataForHBaseAndHadoop Process Group. When the Create Connection window appears, we select ADD. We enter into PreProcessDataForHBaseAndHadoop Process Group . After that we re-configure the GeoEnrichIP processor. We take the full pathname from GeoLite DB acquired in Deploy IoT Weather Station via Connected Data Architecture tutorial section We add GeoLite2 database to HDF Sandbox CentOS. Then we update MaxMind Database File. We click on the NiFi Flow breadcrumb in the bottom left corner to go to the root. For the configuration of the HBase Client Service for PutHBaseJSON. We configure PutHBaseJSON. We Enable the HBase Client Service.

We enable the **Enable Controller Service**. Then we start the portion that just connected to the input port on GeoEnriched NiFi flow. We enhanced GeoEnriched NiFi flow . Then we highlighted three components: PreProcessDataForHBaseAndHadoop, Process Group, PutHBaseJSON ,PutHDFS . Press the start button to activate this section of the flow.

3. Verify HBase Table Populated

We go back to HDP Web Shell Client and use the HBase **scan** (scan 'sense_hat_logs') command to see check that the table has data Weather Data .

After this 3 step we just enhanced the previous NiFi flow to pull in geographic insights using GeoEnrichIP processor based on the Raspberry Pi's public IP address, so we can tell what the city and state the weather readings were drawn from. Additionally, you created an HBase table that allows NiFi to store this data into HDP.

Finally we had to visualize those rows of data. The method chosen was using Zeppelin notebook with jdbc(phoenix). Basically we only need to type classic SQL commands while adding '%jdbc(phoenix)' in header position.

Here we only have one raspberry pi connected so we can only display 3 graphs: Pressure, Humidity and Temperature. but with the location information brought by the IP analysis we can easily imagine making weather maps.

SuperSet in trucking-iot

NiFi wouldn’t work, so we decided to change project for superset in which NiFi is not required. We also decided to use a weather dataset already in our hands from another ISEP project.

thus we installed a Virtual Machine, executing the following commands.

First of all: dependencies.

```
sudo apt-get install build-essential libssl-dev libffi-dev python-dev python-pip libsasl2-dev libldap2-dev
```

Python :

```
pip install --upgrade setuptools pip
```

Then superset:

```
pip install superset
```

defining admin credentials:

```
fabmanager create-admin --app superset
```

There was an issue with fablab and we spent about 2 hours fixing it.
Next we initialized the database

```
superset db upgrade
```

We loaded our dataset.

```
superset load_examples
```

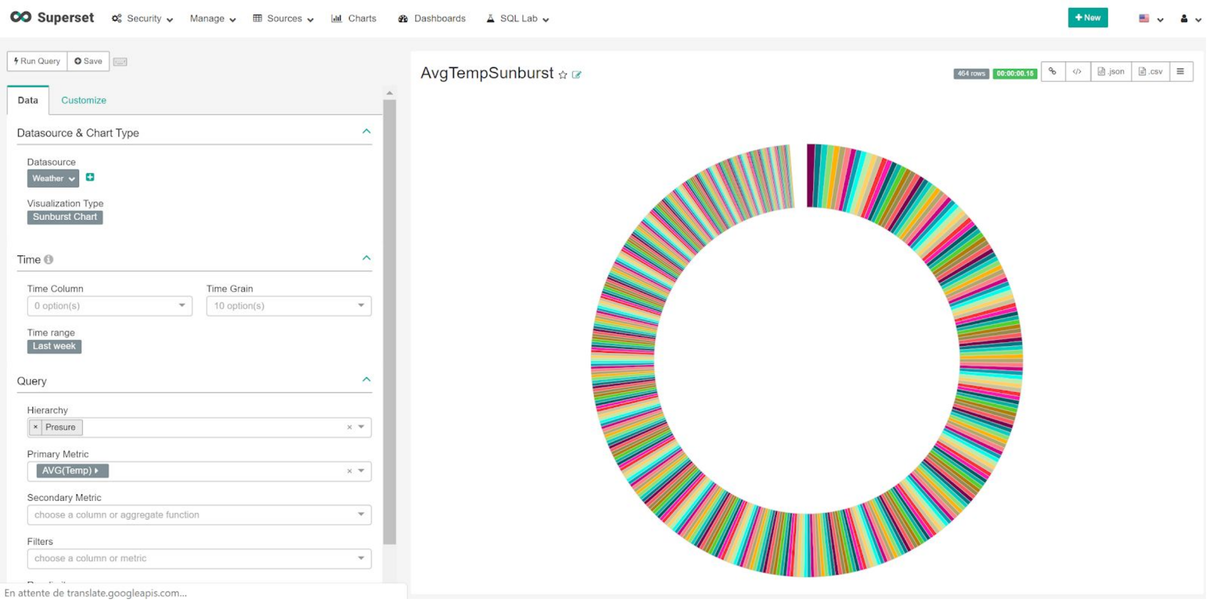
rights initialization :

```
superset init
```

finally we run the server.

```
superset runserver -d
```

A ce moment-là Superset était fonctionnelle, nous avons importé les données que nous avons récoltés sur la météo nous les avons importés en tant que data source puis nous avons imagé ça comme dans les screens suivants :



AvgTempSunburst

464 rows 00:00:00.15

Datasource & Chart Type

Datasource: Weather

Visualization Type: Sunburst Chart

Time

Time Column: 0 option(s)

Time Grain: 10 option(s)

Time range: Last week

Query

Hierarchy: Presure

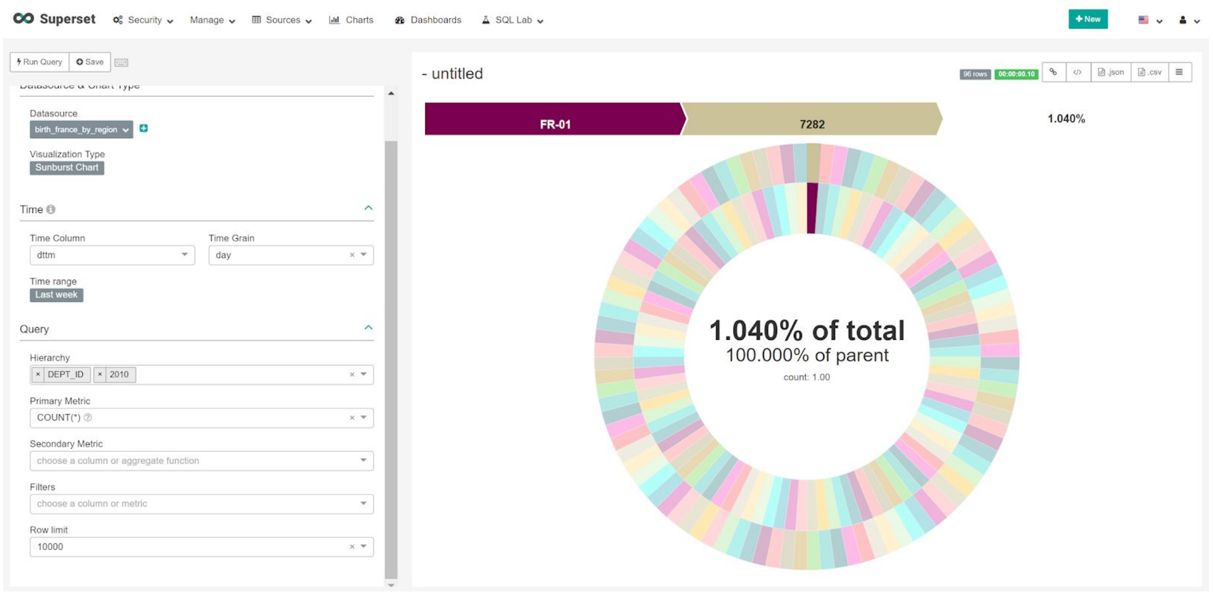
Primary Metric: AVG(Temp)

Secondary Metric: choose a column or aggregate function

Filters: choose a column or metric

En attente de translate.googleapis.com...

Figure 1 : Nous avons représenté la pression en fonction de l'average de la température ce qui nous permet de voir pendant les 3 mois de données enregistrés l'average de la température pour chaque valeur de la pression .



Datasource: birth_france_by_region

Visualization Type: Sunburst Chart

Time

Time Column: dttm

Time Grain: day

Time range: Last week

Query

Hierarchy: DEPT_ID, 2010

Primary Metric: COUNT(*)

Secondary Metric: choose a column or aggregate function

Filters: choose a column or metric

Row limit: 10000

- untitled

96 rows 00:00:00.10

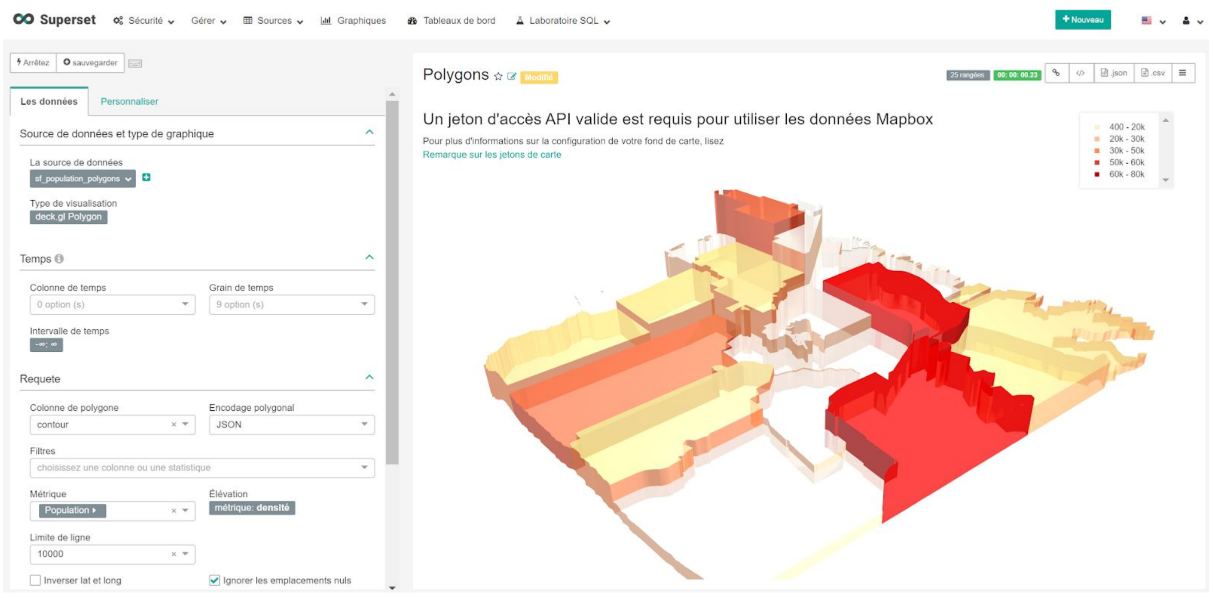
FR-01 7282 1.040%

1.040% of total

100.000% of parent

count: 1.00

Figure 2 : Nous avons voulu essayer de mettre 2 paramètre dans la hiérarchie ce qui permet de créer 2 cercle un sur l'autre . Dans ce cas actuelle nous avons vu les naissance en fonction du département ainsi que de l'année 2010 .



Source de données et type de graphique

La source de données: sf_population_polygons

Type de visualisation: deck.gl Polygon

Temps

Colonne de temps: 0 option (s)

Grain de temps: 9 option (s)

Intervalle de temps: -∞ : ∞

Requete

Colonne de polygone: contour

Encodage polygonal: JSON

Filtres: choisissez une colonne ou une statistique

Métrique: Population

Élévation: métrique: densité

Limite de ligne: 10000

☐ Inverser lat et long

☑ Ignorer les emplacements nuls

Polygons Modifié

25 rangées 00: 00: 00.23

Un jeton d'accès API valide est requis pour utiliser les données Mapbox

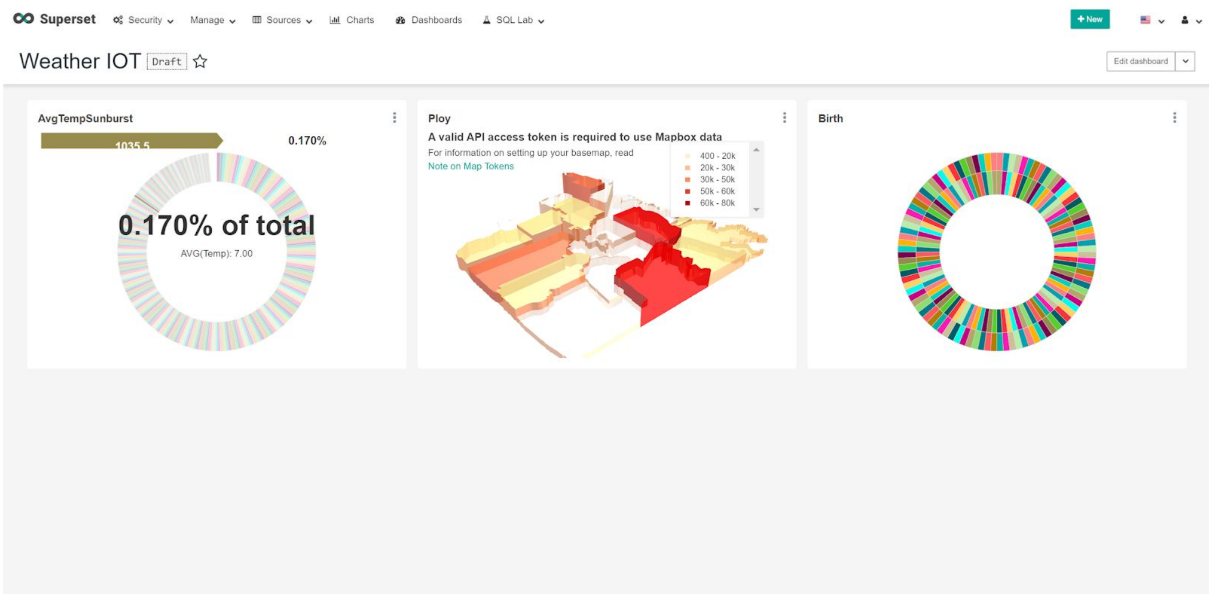
Pour plus d'informations sur la configuration de votre fond de carte, lisez

Remarque sur les jetons de carte

■	400 - 20k
■	20k - 30k
■	30k - 50k
■	50k - 60k
■	60k - 80k

Figure 3 : Ici nous avons voulu tester d'autre fonctionnalité de Superset en regardant la visualisation en polygone , ça nous permet d'avoir une vision

sur la population en fonction de leur localisation . Ce type de visualisation sont très utilisé pour représenter des statistiques sociales en fonction de la localisation.



A valid API access token is required to use Mapbox data
For information on setting up your basemap, read
Note on Map Tokens

Figure 4 : Dans cette figure on peut voir les 3 slices que nous avons étudié qui nous avons regroupé dans ce Dashboard

Conclusion :

Lors de ce module finalement nous avons eu l'occasion de se plonger dans 2 projet , et de découvrir un grand nombre de technologie : superset , environnement virtuelle , docker , raspberry , Hbase , Ambary . Malheureusement avec le problème due à la configuration de NiFi nous n'avons pas pu aller au bout du premier projet . Mais nous avons pu voir les difficultés du travail des architectures distribuées et nous continuerons à apprendre car c'est la base de technologie futuriste tel que la blockchain …

Annexe :

Communication protocol:

-I2C: (https://i2c.info/)

I2C bus is popular because it is simple to use, there can be more than one master, only upper bus speed is defined and only two wires with pull-up resistors are needed to connect almost unlimited number of I2C devices. I2C can use even slower microcontrollers with general-purpose I/O pins since they only need to generate correct Start and Stop conditions in addition to functions for reading and writing a byte.

Each slave device has a unique address. Transfer from and to master device is serial and it is split into 8-bit packets. All these simple requirements make it very simple to implement I2C interface even with cheap microcontrollers that have no special I2C hardware controller. You only need 2 free I/O pins and few simple i2C routines to send and receive commands.

Every I2C device uses a 7-bit address, which allows for more than 120 devices sharing the bus, and freely communicate with them one at a time on as-needed basis.

→ communication filaire,
→ jusqu'à 120 appareils (ici capteurs) branchés sur un même système (ici le raspi),
→ I2C permet d'affecter les capteurs à une adresse fixe à chaque capteur favorisant ainsi les diagnostics

---

NiFi & MiNiFi:

-Fonctionnement par pair:

Minifi est responsable de l'envoi de données depuis le capteur embarqué vers le cluster NiFi lorsque la quantité de donnée devient trop imposante. Souvent mis en place lorsque la quantité de donnée attendue est superieure à la capacité de traitement de l'ordinateur embarqué.

Nifi prend alors le relais comme un système de queues d'informations (comme Apache Kafka en quelque sorte) et pour chaque message de MiNiFi qui arrive valide son contenu et le stocke dans la HBase.

-Différence avec Kafka
(https://stackoverflow.com/questions/53536681/difference-between-kafka-and-nifi)

NiFi and Kafka complements in the sense that NiFi is not a messaging queue like Apache Kafka. On the contrary, Apache NiFi is a data-flow management aka data logistics tool.

Let's assume this scenario: You have messages (in JSON format) getting streamed through Kafka and you want to validate the messages to check if the message has all the fields and if they are valid, you want the messages to land in HBase.

Here NiFi can help you with the following approach:

NiFi has ConsumeKafka processors which you can configure with your Kafka broker and the group name.
Use the NiFi processor ValidateRecord to check if the received messages are all valid
If they are valid, you can connect the output to PutHBaseRecord

Summarizing, NiFi basically prevents you from writing a lot of boilerplate code. In this case, a custom logic to do schema validation and writing to HBase.