NRT Analysis & Processing of Streaming Data.

Shivansh Vijay Nathani
Email : shivansh_n@csu.fullerton.edu
CWID : 885187450

Atharva Karanje
atharvakaranje@csu.fullerton.edu
CWID: 885188631

Aim : Develop a robust, fault tolerant, scalable solution to process, analyze, clean and store streaming data from various streaming applications and devices.

Problems in traditional Big Data Systems : Traditional Big data solutions have solved various problems of harnessing voluminous data. But organizations have been facing various challenges such as,

- **Harnessing Streaming data** : Data has become the key driver for businesses and it is the need of the hour for these businesses to harness and effectively store and use this data. Data is generated every second from devices and applications and it is important for organizations to collect and store this streaming data.
- **Building Scalability, Robustness & Maintainability** : Scalability, Robustness and Maintainability of the system has been a challenge for organizations. Hence the solution developed to process, analyze and store streaming data must solve these three issues.
- **Performing NRT Analysis and Data warehousing on the cloud** : Big data technologies have been evolving and many organizations have adopted these solutions. Business insights can be gained from the data that has been flowing into the business systems. It is the need of the hour to have solutions providing Near Real time Analysis of data as that can be used to handle and take more effective business decisions.

Approach : We aim to solve the problems mentioned above by using cloud services and evolved Big data technologies such as Spark.

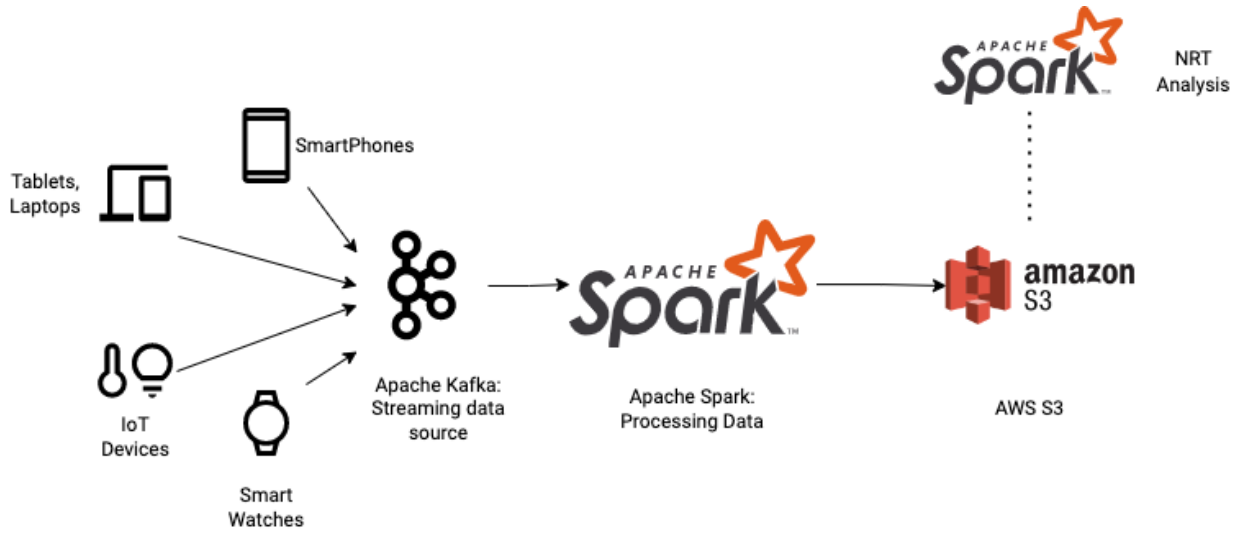
Functionalities :

- This solution would enable organizations & businesses to harness the streaming data in a cost effective manner by using managed services like Confluent (managed Apache Kafka), Databricks (managed Apache Spark) and AWS S3 (storage service).
- This solution would also enable organizations to perform NRT Analysis on streaming data from multiple sources.
- This streaming data would be cleaned and stored on cloud enabled data warehouses to be used later.

Technologies used :

- Confluent (managed Apache Kafka)
- Databricks (managed Apache Spark)
- AWS S3 (for data storage)

Architecture :



Producers : As seen in the Architecture diagram above there are various devices such as smartphones, smartwatches and IoT devices as well as various applications that are generating data every second. These devices and applications are referred to as producers hereafter. This data would be written to Kafka topics based on the device that generates the data. A Kafka topic would logically group similar data.

Apache Kafka : Apache Kafka is a distributed event store and stream-processing platform. Kafka uses a distributed paradigm to store data which enables scalability and fault tolerance for the solution. We use managed kafka i.e Confluent Kafka.

Data written to Kafka can be consumed 'n' consumer applications. Kafka supports data retention and can be configured from the cluster management page. (from 1 hour to forever)

Core capabilities of Kafka :

- High throughput
- Scalable
- Permanent Storage
- High Availability

Apache Spark : Apache Spark is a multi-language engine for executing data engineering, data science, and machine learning on single-node machines or clusters. Apache Spark

supports processing streaming data as well as it is 100 times faster in memory and 10 times faster on disk than Hadoop MapReduce as it uses RAM for data processing while MapReduce writes the intermediate output to disk which is the reason for its slowness.

Here the consuming applications are spark jobs that read data from Kafka, clean it, and perform NRT analysis on streaming data. Spark jobs are also programmed to persist data to AWS S3. This data persisted on S3 can be later used by the application team to perform analysis on this historical data.

Spark also supports external tables, wherein external SQL tables can be created in which the data lies on external cloud based object stores such as AWS S3 buckets or GCP Buckets and the table metadata lies on Databricks hive metastore.

For implementation of this project we have used Managed Spark provided by Databricks.

Core Capabilities of Spark :

- Batch and streaming data support.
- SQL based analysis.
- Machine Learning library support.
- Data Science.

Spark also supports various languages such as python, SQL, scala, java, R

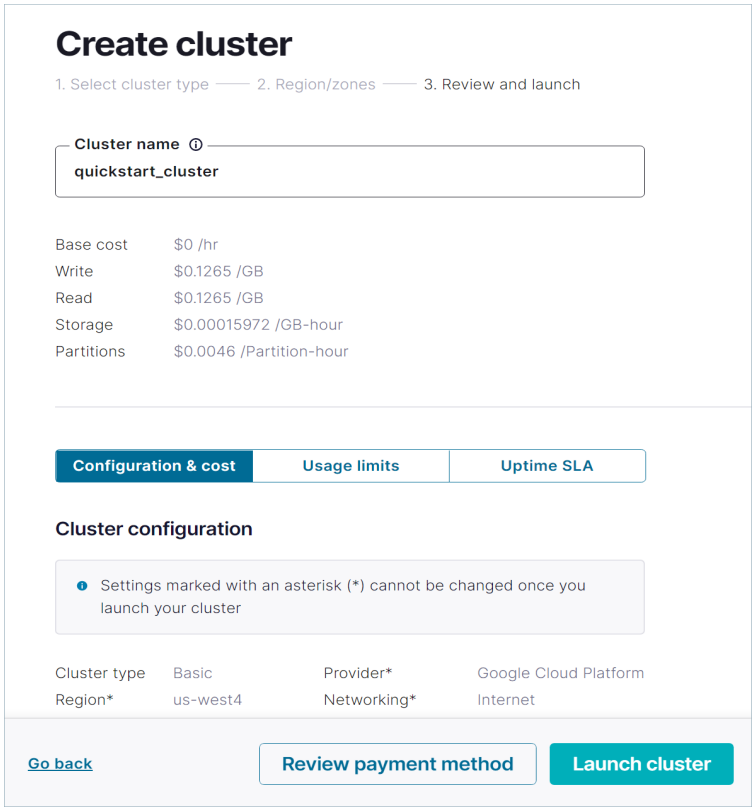
AWS S3 : Amazon Simple Storage Service (Amazon S3) is an object storage service offering industry-leading scalability, data availability, security, and performance.
Data produced by devices and applications is consumed by Spark jobs. The data is cleaned, processed and written to S3 in JSON and CSV formats. External tables are then created on these S3 paths for application teams to be used later.

Deployment & Setup steps :

Git Repo : [ADBMS git repository]

Setting Up Confluent :

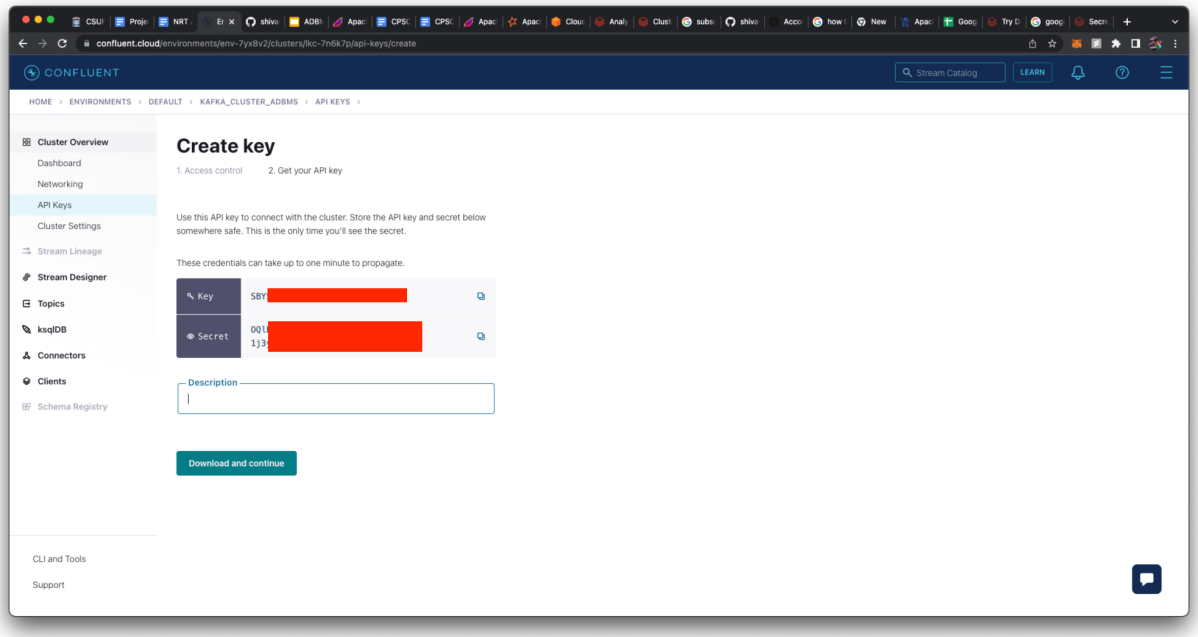
- Sign in to Confluent Cloud at https://confluent.cloud.
- Click Add cluster.
- On the Create cluster page, for the Basic cluster, select Begin configuration.This creates a Basic cluster, which supports single zone availability
- On the Region/zones page, choose a cloud provider, region, and select a single availability zone.
- Select Continue and set up payment on the next page.
- Specify a cluster name, review the configuration and cost information, and then select Launch cluster.



- Now you can get started configuring apps and data on your new cluster.
- Once cluster has been created Navigate to API keys for you cluster generate Key and

Secret and save it

- Use the “config.ini” from git and replace the <Server details>, <API Key> and <API secret> this will be used while running the mocked producer code.

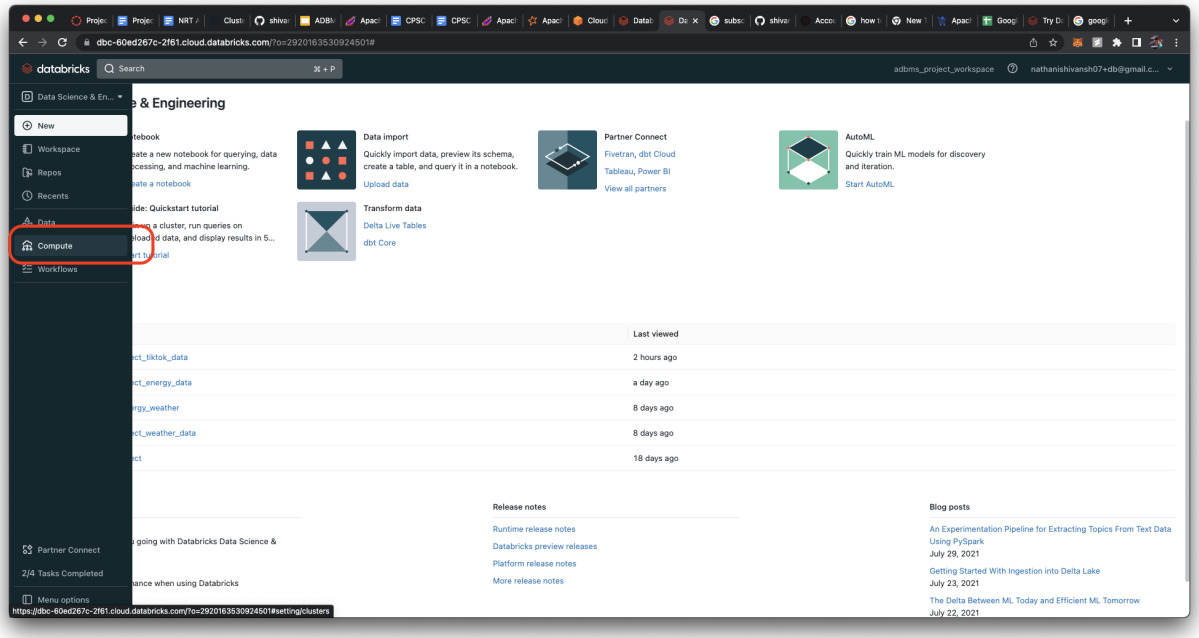


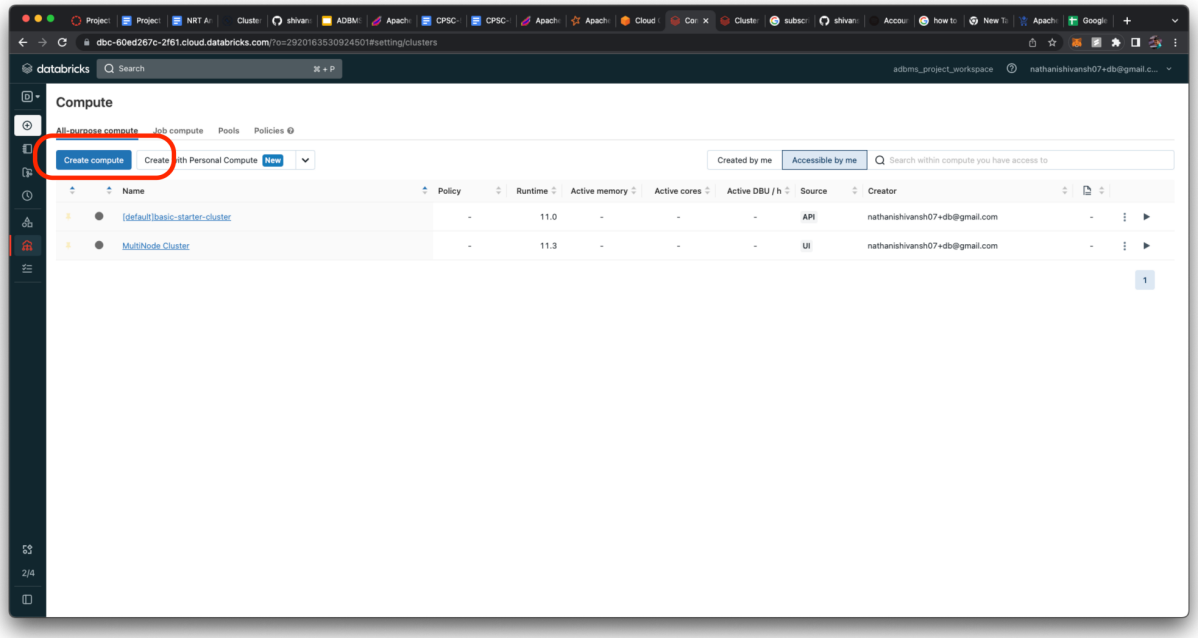
Setting Up Producer (Mocked producers) :

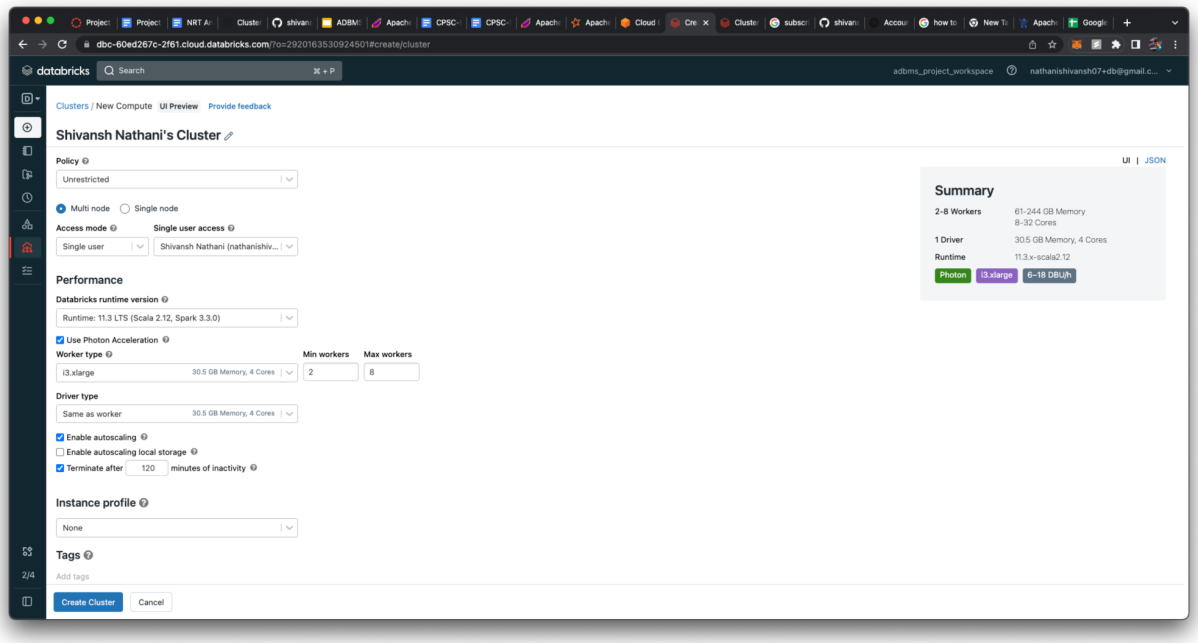
- An application that is the source of the data stream is called a producer. In order to generate tokens or messages and further publish it to one or more topics in the Kafka cluster, we use Apache Kafka Producer.
- Steps to create a Kafka topic:
  - Sign in to Confluent Cloud at https://confluent.cloud.
  - From the navigation menu, click Topics, and then click Create topic.
  - In the Topic name field, type “weather” and then select Create with defaults.
  - The ‘weathers’ topic is created on the Kafka cluster and is available for use by producers and consumers.
  - Similarly, a ‘energy consumption’ topic was also created where energy consumption mock data was pushed.
- The producer code is present in ‘tiktokStreamingData_producer.py’, ‘energyData_producer.py’ and weatherStreaming_producer.py’ files.
- In these files, connection to the producer is first established using the config file passed via the command line.
- A producer instance is then created and mock data is then pushed to the producer using the producer.produce() function.
- These producers are just mocked sources of data to simulate the behavior of actual devices and applications.
- To run these producers we need Python 3 and some basic packages that can be installed using pip.
- Packages such as confluent_kafka and a few other basic packages are required.

Setting Up Databricks : Databricks is a paid service but a 14 day free trial is provided to first time users. But the infrastructure used for cluster and networking between nodes will be charged by the cloud provider.

- Start your free Databricks trial
  - Databricks landing page
  - Databricks free trial
- Create a cluster from the landing page by selecting “Compute” from the left side menu.

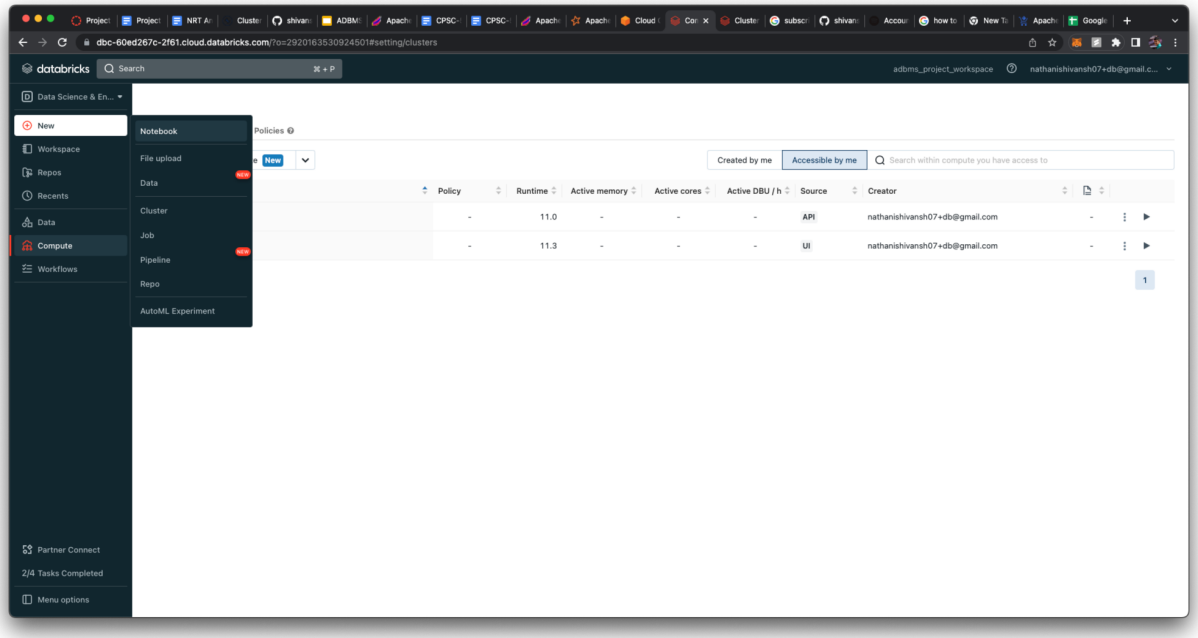




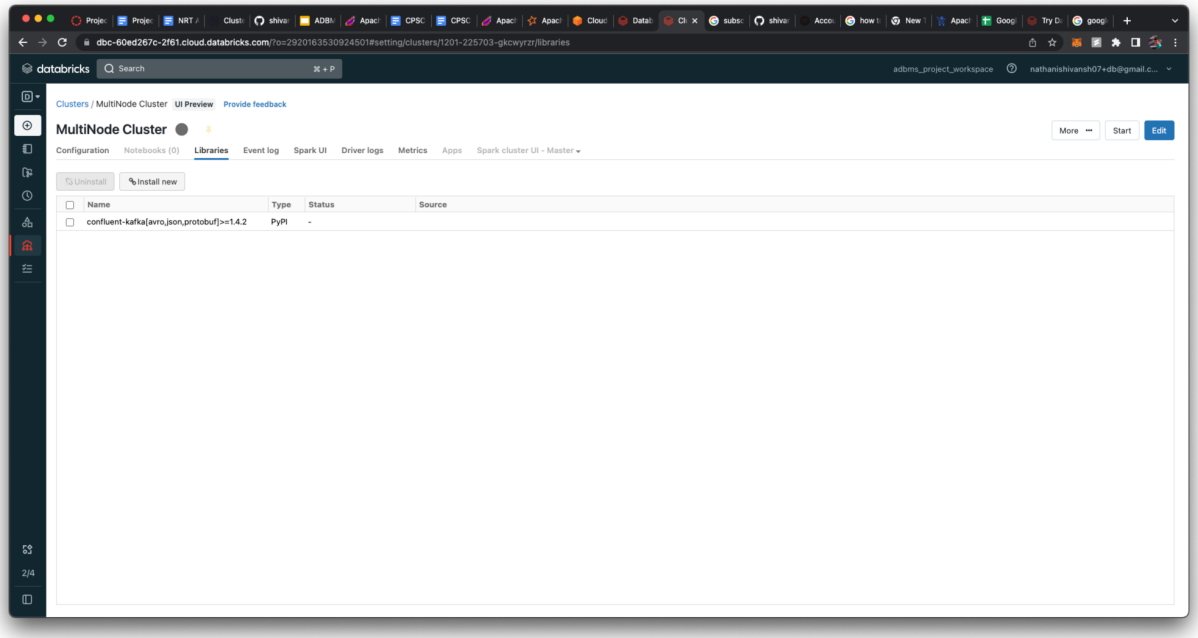


Add the required cluster configurations and hit create cluster.

- Once the cluster is set up we need a notebook to write Spark Job/Code.
- From the left side menu click new -> click notebook and add the required details such as name, language that you wish to code in and cluster to which the job should be submitted.



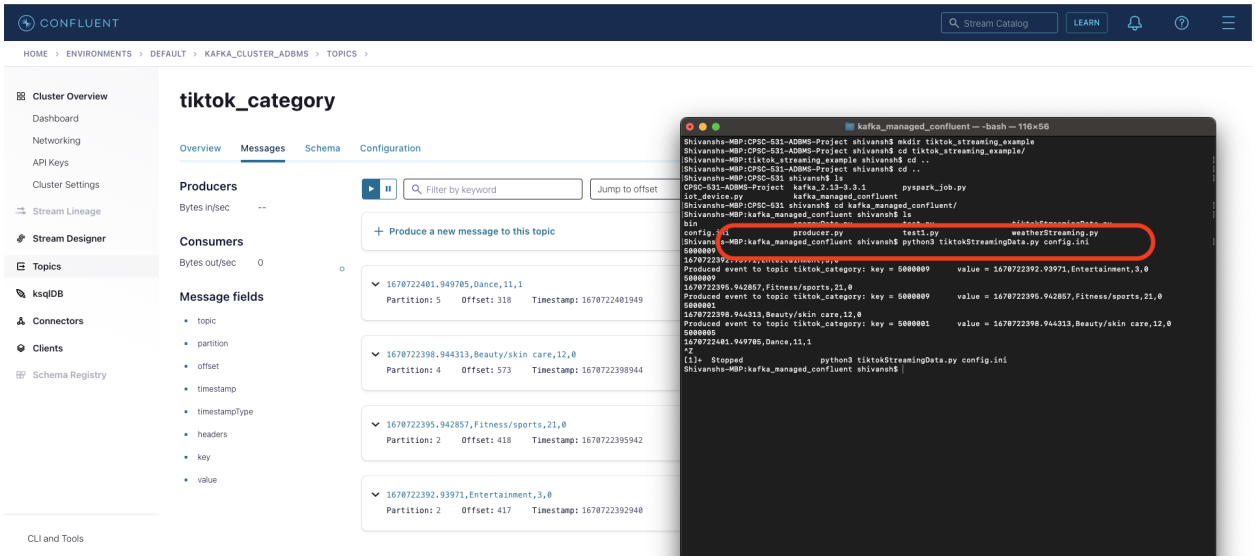
- To connect to kafka we need the confluent kafka library.
- Open your cluster settings, navigate to the libraries tab, select PyPi and install the library “confluent-kafka[avro,json,protobuf]>=1.4.2”
- Once the library is installed this is how your cluster’s library tab looks like.



- We use databricks secrets to store connection details for kafka from databricks, to set up kafka login details as secrets, use this blog. Elseway you can directly use plain text for connecting to kafka. I.e. not from secrets but directly hard coded in the notebook. The connection details are once obtained while we set up a confluent cluster earlier.

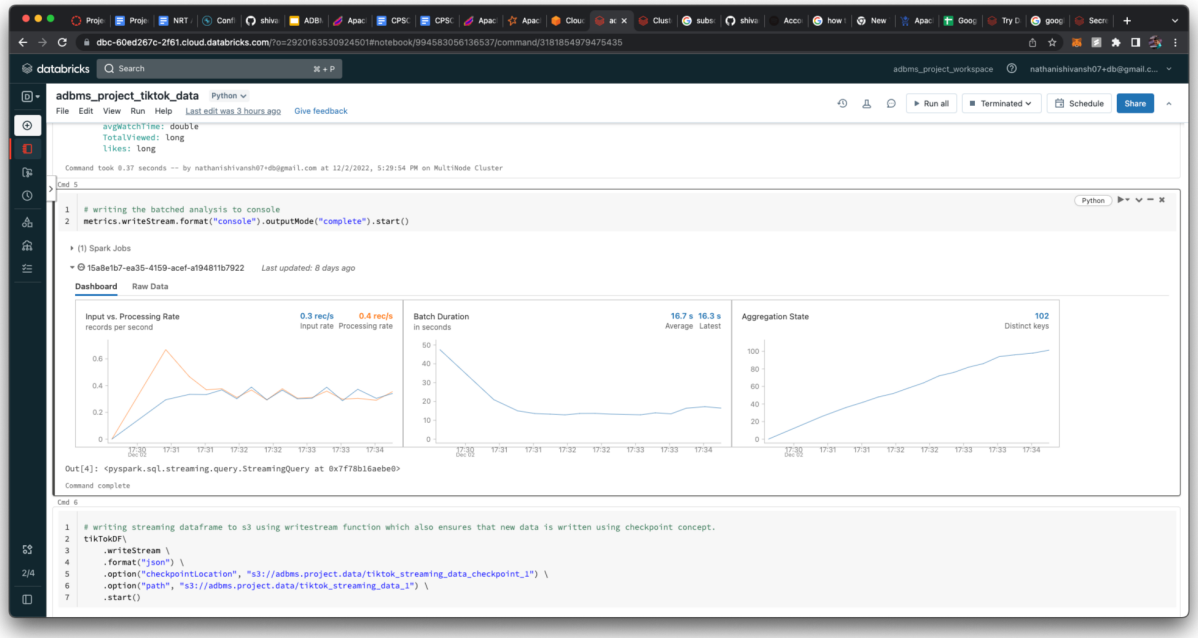
Running the Application :

- Clone the github repo: **ADBMS Project**
- We have multiple examples in this repo.
- Navigate to the appropriate one that you wish to Run.
- Each example has a producer and a consumer file.
- Producer file is a plain python code to simulate streaming source.
- To run the Producer file the code expects a config file “config.ini” that has the kafka connection details.
- Run the producer code as shown below.

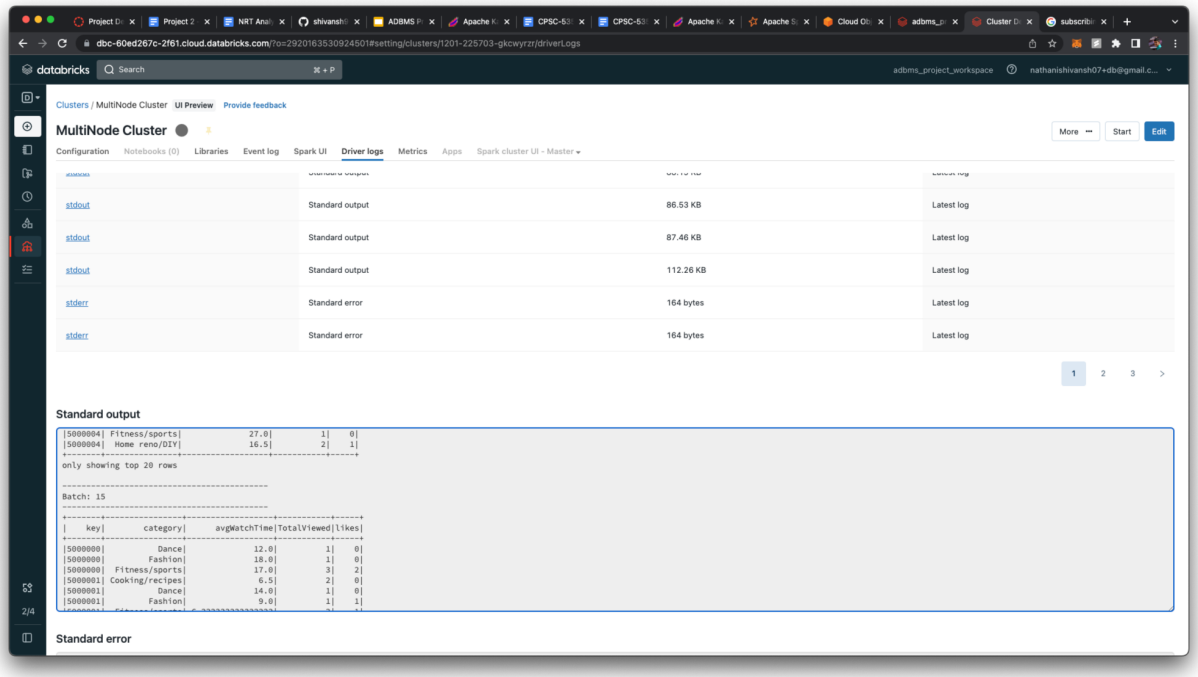


- As soon as you run the file the code starts pushing data to kafka. You can check this by logging into the confluent dashboard and navigating to the topic that you are pushing data to and the messages produced start showing up here.

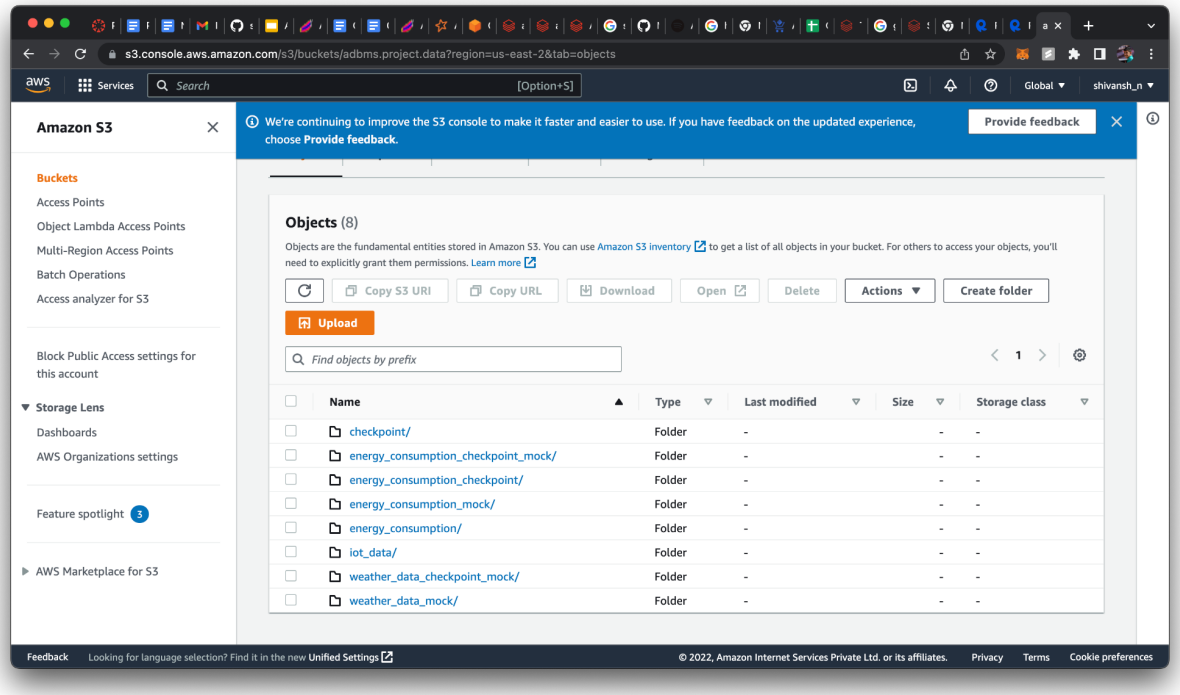
- Next up is running the Spark consuming application/job.
- Login to Databricks and open the notebook that we created during the setup phase.
- Paste the code from the "____consumer.py" file in each notebook cell.
- Run each cell by hitting "shift" and "enter".
- This is how the output looks like.

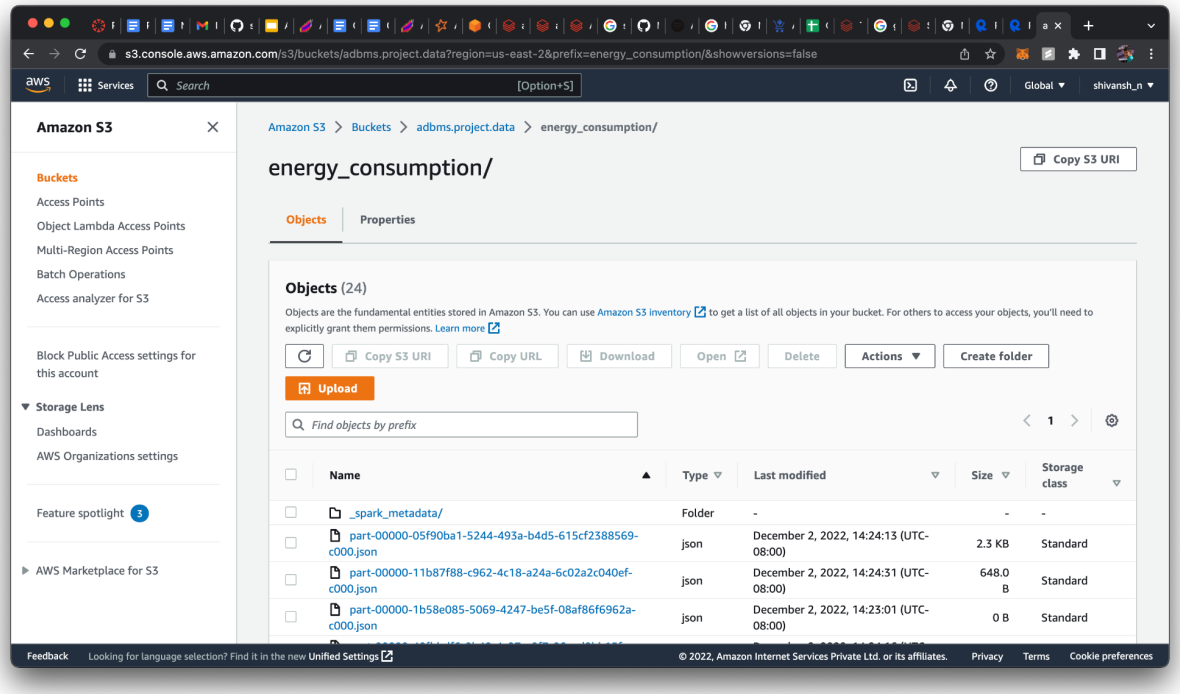


- For the console output navigate to the cluster details page, navigate to Driver logs tab. Under it look for the Standard output.
- This is how the console log looks like with the expected output.

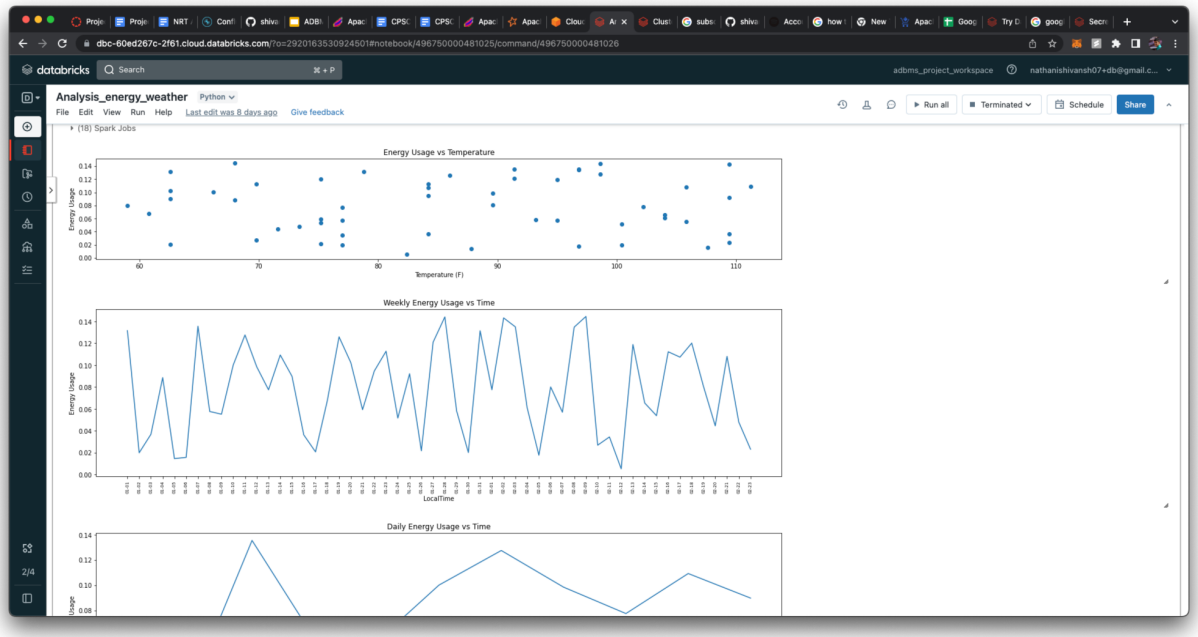


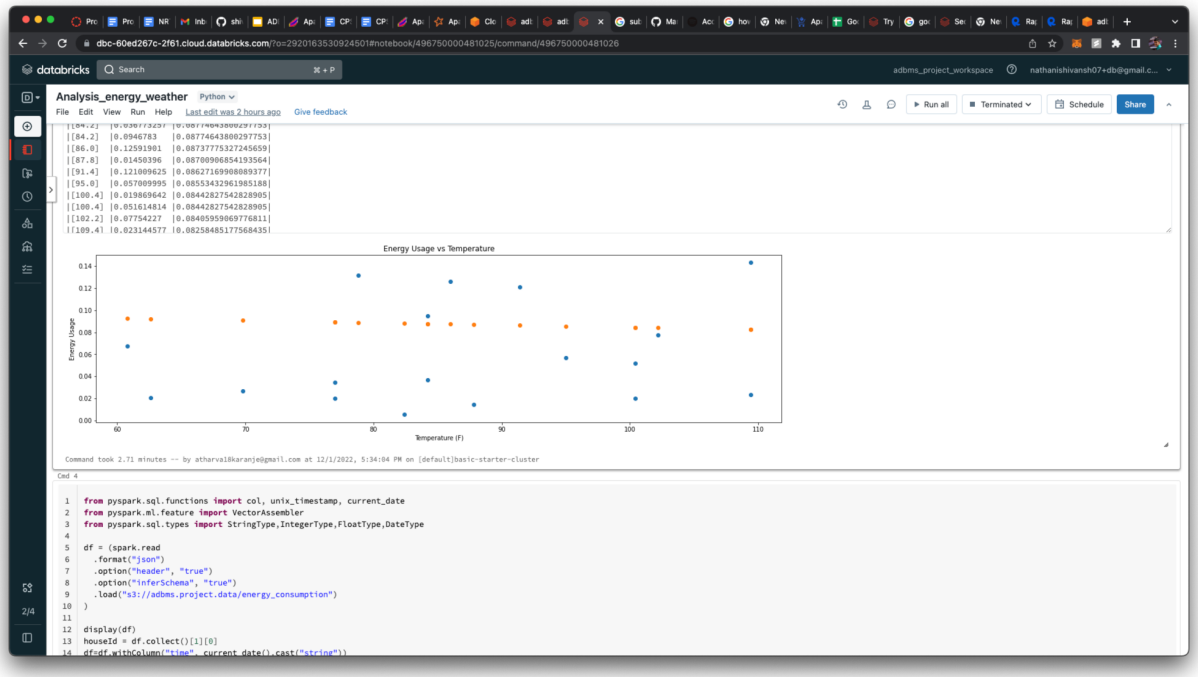
- For the streaming example for energy and weather data we can run the producer and consumer codes similar to that of the TikTok example.
- Spark jobs also write data to S3. Below is a snapshot of S3





- Moving to the Visualization part of data we have a file by the name "Analysis_energy_weather.html" you can paste the contents to a new notebook and run the commands from the notebook. This is how the visualizations would look like.





The analysis code is present in the 'analysis_energy_weather' file and the code explanation is as follows:

- Analysis was performed on the weather and energy mock data that is generated.
- The data is initially stored in the Pyspark dataframes for analysis.
- The data is then processed and converted to a desirable format. (Ex. Data types are changes, dates are formatted, filtered based on a key column value, etc)
- Inner join is then performed on the dataframes based on the timestamp attribute to get the temperature and energy usage for a record at a particular timestamp.
- Scatter plots and line graphs are then generated to analyze and compare different attributes. The following graphs were created for analysis purpose:
    - Scatter plot of Energy usage vs temperature - This will help understand the energy consumption trend against the temperature.
    - Line graph for weekly energy usage vs time - This will help understand the energy usage trend during different days of a week.
    - Line graph for daily energy usage vs time - This will help understand the energy usage trend during different times of a day.
- Furthermore, prediction was also performed for the energy usage attribute using the linear regression model.
- For this, the data was initially split into 70% train and 30% test data.

- The linear regression model was trained based on the 'train data' and was tested against the 'test data' for evaluating the accuracy.
- The linear regression model will help predict energy usage for a household based on the temperature.

Conclusion : We have developed a robust, scalable and fault tolerant solution to harness streaming data by processing, analyzing it in NRT and storing it on cloud. This solution can be extended for any use case or organization where there is a need to collect, process and store streaming data.