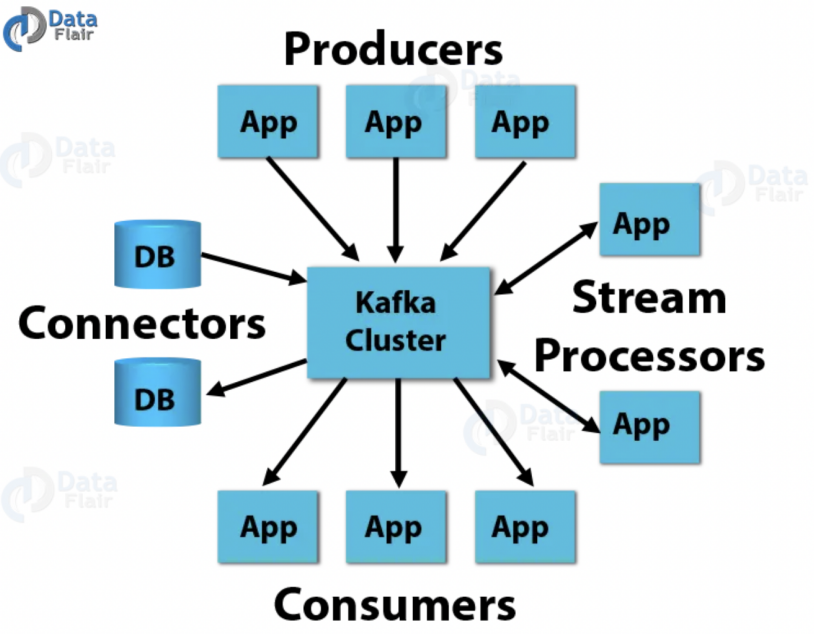
Docs For Kafka-Snowflake POC

Problem statement for the POC:
Our objective for this POC is to ingest streaming data from Kafka and seamlessly load it into Snowflake tables. We aim to explore the Snowpipe Streaming feature provided by Snowflake, which allows us to directly ingest rows into Snowflake tables without the need for intermediate staged files. By directly loading rows into Snowflake tables, we seek to optimize data ingestion speed, reduce latency, and ensure efficient integration with our existing Kafka infrastructure and Snowflake data warehouse.Our goal is to assess the performance of Snowpipe Streaming and evaluate its capability to efficiently handle real-time data streaming scenarios.

First just discuss about **what is kafka?**:

Apache Kafka (here) is a distributed streaming platform that provides a high-throughput, fault-tolerant, and scalable solution for handling **real-time** data streams. It allows seamless integration and reliable communication between various systems and applications, enabling the efficient processing, storage, and analysis of streaming data.We use Apache Kafka when it comes to enabling communication between **producers** and **consumers** using message-based topics. Kafka's key features include **publish-subscribe messaging**, **fault tolerance**, **horizontal scalability**, and support for **stream processing**. It has become popular for use cases such as **event sourcing**, **real-time analytics**, **log aggregation**, and building data pipelines.Can refer more about it **(here)**



Below are the steps for setting up Kafka:-

Can Refer here

Step-1:

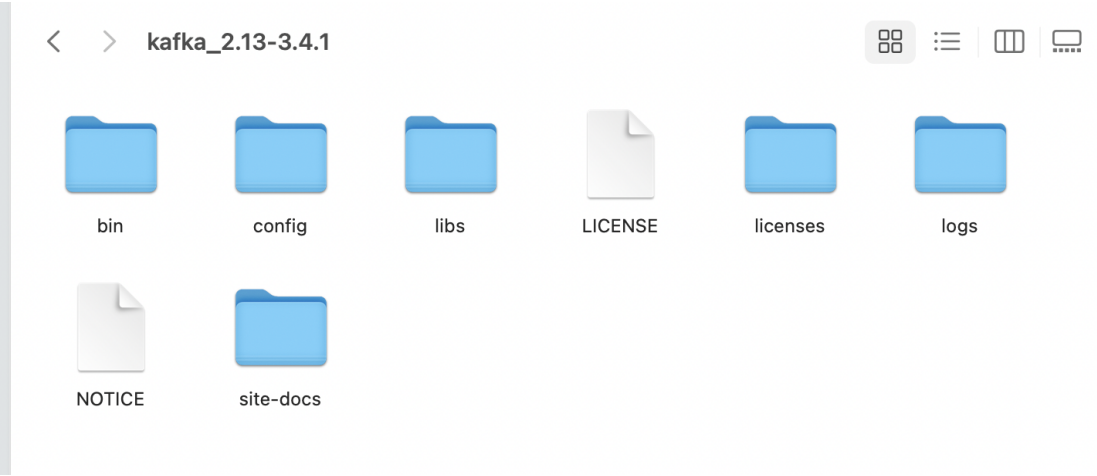
Setting up virtual environment for the set-up:

Commands:

- mkdir docker_kafka
- virtualenv doc_kafka_venv
  → Python version: 3.9.7
- source doc_kafka_env/bin/activate (activating the environment)

Step-2:

- Install JAVA JDK version 11
- Download the Apache Kafka from (here), choose the latest version
- After we download and extract the zip folder
- Below is the screenshot for the folder structure after the zip folder is extracted.



- In bin folder we have lot of .sh script files and this .sh scripts are used to start kafka, Currently we can invoke this scripts by passing full path to script files
- But instead of running full path to invoke this scripts, we need to set PATH environment variable
- For that we will edit a file called **.zshrc**
- For that copy the full path upto the **binaries** of kafka, i.e., upto the bin folder
  In my case (/Users/karanbajaj/kafka_2.13-3.4.1/bin)
- Include this path as below in .zshrc file
- **PATH="$PATH:/Users/karanbajaj/kafka_2.13-3.4.1/bin"**
- Just close the terminal and start again, or else open a new tab and run the command **echo $PATH** to verify

Step-3:

Next we need to set up **Zookeeper** which is an Apache-developed software that serves as a centralized service for maintaining naming and configuration data and enabling synchronization in distributed systems. It is responsible for monitoring the status of Kafka cluster nodes and keeping track of Kafka topics, partitions, and other related information.

ZooKeeper is started along with Kafka because it handles critical functions such as metadata management, leader election, broker registration, configuration management, health monitoring, and reliable commit log. It ensures coordination, fault tolerance, and consistency in the Kafka cluster, making it an essential component for Kafka's operations.

- To Start Zookeeper server
  → zookeeper-server-start.sh ~/kafka_2.13-3.4.1/config/zookeeper.properties
    - **Zookeeper.properties** is basically present in **config** folder which actually tells zookeeper how to start

- To Start Kafka server
  → kafka-server-start.sh ~/kafka_2.13-3.4.1/config/server.properties
    - **Server.properties** file basically contains the configuration settings for the Kafka server.

Step-4:

Next setting up configuration for **Kafka-snowflake connector:**

- Download the **JAR** file from (here), size:- 92.8MB
- After downloading the JAR file , move the JAR file from the downloads folder to the libs folder inside kafka set up along with other JAR files
  → In my case inside (**/Users/karanbajaj/kafka_2.13-3.4.1/libs**)
- Next we need to manipulate **connect-standalone.properties** file that is present inside the **config folder** by Adding (**plugin.path**=/Users/karanbajaj/kafka_2.13-3.4.1/libs) which will specify the path to the external JAR file which it need to check for **external dependencies** while connecting to snowflake
- Then we need to set up configuration file,in my case i named it **SF_connect.properties**, which contains all the configuration regarding connection to snowflake, which defines the type of **ingestion-method** to ingest data into snowflake, **flush time**, like after how many seconds of getting data into buffer,should kafka ingest it into snowflake,**record counts**, **snowflake credentials** - **account details**, **private keys**,**database name**,**schema name** and many more
- Reference this articles to know more about it - Article1, Article2

Step-5:
First lets discuss about how to set up **public** and **private keys** for snowflake,
For that we have to run two commands

→ To generate Private key:

- `openssl genrsa 2048 | openssl pkcs8 -topk8 -inform PEM -out snowflake_tf_snow_key.p8 -nocrypt`

→ To generate Public Key:

- `openssl rsa -in snowflake_tf_snow_key.p8 -pubout -out snowflake_tf_snow_key.pub`

Next as i am using admin user in this set up, i have to alter the **public key** for that user
command:-
`ALTER USER <USERNAME> SET RSA_PUBLIC_KEY = '********'`

By using above command, we can change the public key for that user and then we can add private key to **SF_connect.properties** file, for it to **authenticate** with snowflake

Then lets create a kafka topic using command line:
`→ kafka-topics.sh --bootstrap-server localhost:9092 --topic user_topic --create`

The "**user_topic**" created above is where the data would be produced in the kafka and from where consumers will consume messages.

Step-6:
Next we need to specify configuration as per the ingestion method we are implementing.So for ingesting data from kafka to snowflake, thier is two types of ingestion methods:

- Standard Snowpipe
- Snowpipe Streaming api

Notes on comparison between the above two methods:
Snowpipe and the Snowpipe Streaming API are both components of Snowflake's data ingestion capabilities, but they serve different purposes and scenarios. While Snowpipe is designed for loading data from files, the Snowpipe Streaming API is specifically tailored for streaming data scenarios where data is received or produced in rows, such as from Apache Kafka topics.
If you have a streaming scenario where data is continuously streamed as rows, the Snowpipe Streaming API is the recommended approach. By using the Snowpipe Streaming API, you eliminate the need to create intermediate files to load data into Snowflake tables which in case of standard snowpipe, there is the internal stage involved which increases the rate of load latency and thereby increasing the credits. You can directly stream the data rows to Snowflake, enabling automatic and continuous loading of data streams into Snowflake tables as the data

becomes available.This API enables a more streamlined and real-time data ingestion process. It simplifies the data pipeline by eliminating the overhead of file creation and management, allowing you to leverage the full potential of Snowflake's scalability and performance for processing streaming data. Refer here to get more insights

Refer Kafka Configuration Properties to know in detail for each of the properties
Also refer here.
Below are the configurations for both the methods:

- Standard Snowpipe

  ---

  connector.class=com.snowflake.kafka.connector.SnowflakeSinkConnector
  tasks.max=8
  topics=user_topic
  snowflake.topic2table.map=user_topic:snowpipe_kafka_user
  buffer.count.records=100
  buffer.flush.time=30
  buffer.size.bytes=20000000
  snowflake.url.name=owb08088.us-east-1.snowflakecomputing.com
  snowflake.user.name=MOCKPROJECTSA123
  snowflake.private.key= ********
  snowflake.database.name=kafka_snowpipe_db
  snowflake.schema.name=kafka_snowpipe_schema
  key.converter=com.snowflake.kafka.connector.records.SnowflakeJsonConverter
  value.converter=com.snowflake.kafka.connector.records.SnowflakeJsonConverter
  name=user_inter

- Snowpipe streaming api

  ---

  name = snowpipeStreaming
  connector.class=com.snowflake.kafka.connector.SnowflakeSinkConnector
  tasks.max=4
  topics=user_topic
  snowflake.database.name=kafka_snow_stream
  snowflake.schema.name=kafka_snow_stream_schema
  snowflake.topic2table.map=user_topic:snow_stream_connect
  buffer.count.records=100
  buffer.flush.time=2
  buffer.size.bytes=20000000
  snowflake.url.name=owb08088.us-east-1.snowflakecomputing.com
  snowflake.user.name=MOCKPROJECTSA123
  snowflake.private.key= ********
  snowflake.role.name=ACCOUNTADMIN
  snowflake.ingestion.method=snowpipe_streaming
  value.converter.schemas.enable=false

jmx=true
key.converter=org.apache.kafka.connect.storage.StringConverter
value.converter=org.apache.kafka.connect.json.JsonConverter
errors.tolerance=all

Step-7:
Start the **kafka-connect server** using the command:

- connect-standalone.sh ~/kafka_2.13-3.4.1/config/connect-standalone.properties ~/kafka_2.13-3.4.1/config/SF_connect.properties

Above are the list of steps, through which we can configure entire kafka-snowflake set-up to make data ingest to snowflake

Next let's discuss on using Python with Kafka here:
There are at least three Python libraries available to interface with Kafka broker services. They are:

- Kafka-Python (here)
- PyKafka (here)
- Confluent Kafka Python (here)

In this POC specifically I have used **Kafka-Python.**

To install **Kafka-Python** use below command:
→ **pip install kafka-python**

As per problem statement i have to ingest streaming data into the kafka environment i.e., i have to produce **continuous** data in a particular kafka topic. For that I have used a **Faker module** which will generate dummy data every time it is called whose source code i have specified in **data.py** file and imported this module in **producer.py** which is actually responsible for producing the data in kafka topic.

To use Faker module, install it
→ **pip install Faker**

Code for **data.py**:

——————————

```
from faker import Faker
import datetime
fake_data=Faker()

def get_data():
    return {
        "name": fake_data.name(),
        "address": fake_data.address(),
        "birth_year":fake_data.year(),
        "ingestion_time": datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
}

#Checking
if __name__ == "__main__":
    print(get_data())
```

Explanation:

- The code uses the **Faker library** to generate fake data and encapsulates it in a dictionary format using the **get_data()** function. The generated data includes a random **name**, **address**, **birth year**, and the current **ingestion time**

Below is the code for **Producer.py**

——

```
from kafka import KafkaProducer
import json
from data import get_data
import time

def json_serializer(data):
    return json.dumps(data).encode('utf-8')

producer = KafkaProducer(bootstrap_servers=['localhost:9092'], value_serializer =
                         json_serializer)
if __name__ == '__main__':
```

```
    while True:
        user_data = get_data()
        print(user_data)
        producer.send("user_topic",user_data)
        time.sleep(2)
```

Explanation for the above code:

- This code sets up a Kafka producer that continuously sends data to the topic name "**user_topic**" at a rate of one message every 2 seconds. The data is serialized in JSON format using a custom serializer function.
- The **KafkaProducer** class is imported from the kafka module, which is a Python client for Kafka which would be responsible for producing messages in kafka topic.
- The **json** module is imported to handle JSON serialization.
- The **get_data** function is imported from a module called data which I have defined explicitly. This function is responsible for generating dummy data to be sent to Kafka.
- The **time** module is imported to introduce a delay between each data send iteration.
- A custom **json_serializer** function is defined to convert the data into a JSON-encoded byte string.
- Then an **instance** of **KafkaProducer** is created by specifying the **bootstrap_servers** parameter, which defines the Kafka brokers to connect to. In this case, it is set to localhost:9092.
- The **value_serializer** parameter is set to the previously defined json_serializer function, which will be used to serialize the data to JSON format before sending it to Kafka.
- Next I started an infinite loop which will continuously send the data to the kafka topic(in this case “**user_topic**” which we created using command line) with an interval of 2 seconds
- The **producer.send()** method sends the **user_data** to the "**user_topic**" which is our Kafka topic.
- A delay of 2 seconds is introduced using **time.sleep(2)** before the next iteration.

Below is the code for **Consumer.py**

—----------

```
from kafka import KafkaConsumer
import json
consumer = KafkaConsumer('user_topic',bootstrap_servers='localhost:9092',
                         auto_offset_reset='latest',
                         group_id="consumer_group_a")
print("Starting Consumer")
for message in consumer:
```

```
    print("User-data: {}".format(json.loads(message.value)))
```

Explanation:

- This code sets up a Kafka consumer that continuously consumes messages from the "**user_topic**" topic. It deserializes the message values assumed to be in JSON format and prints them on the console. The consumer starts consuming from the **latest** available offset, and it belongs to the "**consumer_group_a**" consumer group.
- An **instance** of KafkaConsumer is created by specifying the following parameters:
- **'user_topic'** represents the topic from which the consumer will consume messages.
- **bootstrap_servers** parameter defines the Kafka brokers to connect to. In this case, it is set to '**localhost:9092**'.
- **auto_offset_reset** specifies the offset to start consuming from in case there is no initial or valid offset committed by the consumer group. Here, it is set to **'latest'**, meaning it will start consuming from the latest available **offset**.
- **group_id** represents the consumer group to which this consumer belongs. It helps in managing consumer coordination and load balancing.
- The **json.loads(message.value)** line deserializes the **JSON-encoded** message value into a Python object.

Querying Metadata in snowflake to analyze the performance:

- SELECT * FROM SNOWFLAKE.ACCOUNT_USAGE.SNOWPIPE_STREAMING_CLIENT_HISTORY(here)
- SELECT * FROM SNOWFLAKE.ACCOUNT_USAGE.SNOWPIPE_STREAMING_FILE_MIGRATION_HISTORY (here)
- SELECT * FROM SNOWFLAKE.ACCOUNT_USAGE.PIPE_USAGE_HISTORY(here)
- SELECT * FROM SNOWFLAKE.ACCOUNT_USAGE.PIPES(here)
- Refer article here to know more about snowpipe streaming costs

Analysis Overview:-

- Based on my analysis, I have observed that the time taken for data ingestion from Kafka into Snowflake is significantly reduced when utilizing Snowpipe Streaming compared to the standard Snowpipe method. When processing the same amount of data, Snowpipe Streaming proves to be faster in ingesting the data into Snowflake.
- Additionally, I have noticed that the credit consumption for ingesting data per byte is considerably lower with Snowpipe Streaming compared to the standard Snowpipe method. This means that the cost associated with data ingestion is reduced when utilizing Snowpipe Streaming.

- Overall, Snowpipe Streaming offers improved performance and cost efficiency in the data ingestion process from Kafka to Snowflake, making it a favorable choice for real-time and efficient data streaming scenarios.

Commandline practices:-

 # Creating a topic
kafka-topics.sh --bootstrap-server localhost:9092 --topic first_topic --create

kafka-topics.sh --bootstrap-server localhost:9092 --topic second_topic --create --partitions 3

kafka-topics.sh --bootstrap-server localhost:9092 --topic third_topic --create --partitions 3 --replication-factor 1

 # List topics
kafka-topics.sh --bootstrap-server localhost:9092 --list

 # Describe a topic
kafka-topics.sh --bootstrap-server localhost:9092 --topic first_topic --describe

 # Delete a topic
kafka-topics.sh --bootstrap-server localhost:9092 --topic first_topic --delete
(only works if delete.topic.enable=true)

Kafka-console-producer.sh

kafka-topics.sh --bootstrap-server localhost:9092 --topic first_topic --create --partitions 1

 # producing
kafka-console-producer.sh --bootstrap-server localhost:9092 --topic first_topic

 # producing with properties
kafka-console-producer.sh --bootstrap-server localhost:9092 --topic first_topic --producer-property acks=all

 # producing to a non existing topic
kafka-console-producer.sh --bootstrap-server localhost:9092 --topic new_topic

 # our new topic only has 1 partition
kafka-topics.sh --bootstrap-server localhost:9092 --list
kafka-topics.sh --bootstrap-server localhost:9092 --topic new_topic --describe

 # produce against a non existing topic again
kafka-console-producer.sh --bootstrap-server localhost:9092 --topic new_topic_2

 # this time our topic has 3 partitions
kafka-topics.sh --bootstrap-server localhost:9092 --list
kafka-topics.sh --bootstrap-server localhost:9092 --topic new_topic_2 --describe

 # produce with keys
kafka-console-producer.sh --bootstrap-server localhost:9092 --topic first_topic --property parse.key=true --property key.separator=:

Kafka-console-consumer.sh

 # create a topic with 3 partitions
kafka-topics.sh --bootstrap-server localhost:9092 --topic second_topic --create --partitions 3

 # consuming
kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic second_topic

 # other terminal
kafka-console-producer.sh --bootstrap-server localhost:9092 --producer-property partitioner.class=org.apache.kafka.clients.producer.RoundRobinPartitioner --topic second_topic

 # consuming from beginning
kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic second_topic --from-beginning

 # display key, values and timestamp in consumer
kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic second_topic --formatter kafka.tools.DefaultMessageFormatter --property print.timestamp=true --property print.key=true --property print.value=true --property print.partition=true --from-beginning

Kafka-console-consumer-group.sh
create a topic with 3 partitions
kafka-topics.sh --bootstrap-server localhost:9092 --topic third_topic --create --partitions 3

 # start one consumer
kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic third_topic --group my-first-application

 # start one producer and start producing
kafka-console-producer.sh --bootstrap-server localhost:9092 --producer-property partitioner.class=org.apache.kafka.clients.producer.RoundRobinPartitioner --topic third_topic

 # start another consumer part of the same group. See messages being spread
kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic third_topic --group my-first-application

 # start another consumer part of a different group from beginning
kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic third_topic --group my-second-application --from-beginning

 # documentation for the command
kafka-consumer-groups.sh

 # list consumer groups
kafka-consumer-groups.sh --bootstrap-server localhost:9092 --list

 # describe one specific group
kafka-consumer-groups.sh --bootstrap-server localhost:9092 --describe --group my-second-application

 # describe another group
kafka-consumer-groups.sh --bootstrap-server localhost:9092 --describe --group my-first-application

 # start a consumer
kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic first_topic --group my-first-application

 # describe the group now
kafka-consumer-groups.sh --bootstrap-server localhost:9092 --describe --group my-first-application

 # start a console consumer
kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic first_topic --group my-first-application

 # describe the group again
kafka-consumer-groups.sh --bootstrap-server localhost:9092 --describe --group my-first-application

–reset-offset

 # look at the documentation again
kafka-consumer-groups.sh

 # describe the consumer group
kafka-consumer-groups.sh --bootstrap-server localhost:9092 --describe --group my-first-application

 # Dry Run: reset the offsets to the beginning of each partition
kafka-consumer-groups.sh --bootstrap-server localhost:9092 --group my-first-application --reset-offsets --to-earliest --topic third_topic --dry-run

 # execute flag is needed
kafka-consumer-groups.sh --bootstrap-server localhost:9092 --group my-first-application --reset-offsets --to-earliest --topic third_topic --execute

 # describe the consumer group again
kafka-consumer-groups.sh --bootstrap-server localhost:9092 --describe --group my-first-application

 # consume from where the offsets have been reset
kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic third_topic --group my-first-application

 # describe the group again
kafka-consumer-groups.sh --bootstrap-server localhost:9092 --describe --group my-first-application

Foe setting up a docker to run kafka and zookeeper:

Docker set-up:- (here)

```
version: '3'


services:
  zookeeper:
    image: wurstmeister/zookeeper
    container_name: zookeeper
    ports:
      - "2181:2181"
  kafka:
    image: wurstmeister/kafka
    container_name: kafka
    ports:
      - "9092:9092"
```

```
  environment:
    KAFKA_ADVERTISED_HOST_NAME: localhost
    KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
```

Other useful refereed article:

- https://towardsdatascience.com/using-kafka-with-python-54dc20717cf7
- https://pandeysudhendu.medium.com/getting-started-with-kafka-connector-for-snowflake-3bf596e550bb
- https://medium.com/technofunnel/apache-kafka-in-5-minutes-c92c43ba3f39
- https://www.upsolver.com/blog/snowpipe-streaming
- https://www.linkedin.com/pulse/real-time-data-streaming-from-database-snowflake-using-parag-jain
- https://medium.com/snowflake/simplifying-real-time-data-ingestion-stream-azure-event-hubs-events-into-snowflake-with-snowpipe-62010d8f479a

- Streaming Data from Snowflake to Kafka -
  https://medium.com/snowflake/streaming-data-from-snowflake-to-kafka-ed76ce0400c2
- Dynamic Tables -
  https://www.snowflake.com/blog/dynamic-tables-delivering-declarative-streaming-data-pipelines/
- https://medium.com/snowflake/%EF%B8%8F-snowflake-in-a-nutshell-the-snowpipe-streaming-api-dynamic-tables-ae33567b42e8
- https://quickstarts.snowflake.com/guide/CDC_SnowpipeStreaming_DynamicTables/#0
- https://www.youtube.com/watch?v=o8YXmsy8Rss
- Dead letter Queues in kafka -
  https://medium.com/@sannidhi.s.t/dead-letter-queues-dlqs-in-kafka-afb4b6835309
  https://towardsdatascience.com/dead-letter-queue-dlq-in-kafka-29418e0ec6cf