Streaming 2: Real time data processing of crypto-currencies

Author : Aakash Mandlik

Streaming data is information that is delivered immediately after collection. There is no delay in the timeliness of the information provided. Streaming data is often used for navigation or tracking.

What are we solving?

Building a real time dashboard to see the trends of crypto currencies fluctuations in real time. The dashboard should be refreshed in a few seconds and the graphs and charts should be refreshed in accordance with the real time data.
For the scope of this session, we are solving the second half of our problem statement. We are going to learn about Spark streaming framework for data processing.
Also, we are going to learn more about the different time series databases and dashboarding tools. Choosing the ones which fits best for our use case.
Basically we are going to use the data we have ingested in the previous session and create some very useful and beautiful dashboards which can provide a trend analysis of crypto fluctuations at seconds granularity.
Excited!! Stay tuned.

Where is the data coming from?

This is a continuation of the last session on streaming. As a recap we did following things in previous session :

- We produced the data to kafka for real time crypto fluctuations.

Thus in this session we are focusing on consuming the data from the Kafka topic and process the data via Spark Structured streaming.
We will use spark-streaming to load the data to InfluxDb.
Using Grafana we will read the influxDb data and create dashboards, which will get refreshed in real time.

What is the real time ecosystem: Part 2?

Here we will cover the second half of the ecosystem.

Processing frameworks:

- As we are talking about real time data, we need a very fast in-memory framework, which can extract, transform(process) and load the continuous data in real time.
- The source of data can be a messaging queue like Kafka or real time api or very fast mysql transactions. The destination can be any dashboard like tableau or graffana for monitoring the real time traffic.
- The real time processing frameworks bridges the gap between source like kafka and destination like a dashboard. Also they help in transformation needed from source data version to a destination data version.
- What are the different processing frameworks in the ecosystem?
    - **Apache spark streaming** : Apache Spark is an open-source unified analytics engine for large-scale data processing. Structured Streaming is part of Apache Spark.
    - **Apache storm** : It is true streaming and is good for simple event based use cases. Although it's an overhead if event to event streaming is not required.
    - **Apache flink** : Flink is essentially a true streaming engine like storm. Although it has API similarity as spark(map, filter,reduce etc). There is an overhead and higher maintenance due to true streaming nature.

Time series database:

- A time-series database is optimized for timestamp or time-series data.
- Time series data mean measurements or events that are tracked, monitored, collected, or aggregated over a period of time.

- These could be data collected from heartbeats of motion tracking sensors, JVM metrics from the java applications, market trade data, network data, API responses, process uptime, etc.
- Time-series databases are completely customized with timestamped data, which is indexed and efficiently written in such a way that you can insert time-series data. You can query those time series data much faster than how you will be doing it in a relational or NoSQL database.
- What are the different time series databases available in the ecosystem?
    - **InfluxDB** : InfluxDB is an open-source time series database used for storing time series data mainly used for real time trends. InfluxDB supports int64, float64, bool, and string data types using different compression schemes for each one.
    - **Prometheus** : Prometheus is also a time series database but used mainly for monitoring purposes. Like server or cluster monitoring etc. Prometheus only supports float64.

Dashboarding Tools :

- Dashboard software is an information management tool that tracks, gathers, and displays business data in interactive and customizable visualizations that enable users to monitor the health of a business, analyze processes, and provide them with actionable insights.
- What are the different dashboarding tools :
    - **Grafana** : Grafana is mainly used when we have multiple data sources like InfluxDB or Redshift or BigQuery to get the data from and use the data in a single dashboard.Grafana is widely used across industries and sharing of data is easier using grafana. Also if you have non time-series data as well with time-series , then grafana is a better choice.
    - **Chronograf:** Chronograf can only be used when the data source is influxDb. It only works with Influx query language. It does not support a dashboard over non-time-series data.
    - **Tableau**: Working with time series data can be a bit less comfortable with Tableau. However, Tableau has more specific business analytics tools.

Next, we will discuss what we are going to use in each of these sub-ecosystems.

What are we going to use(Recap)?

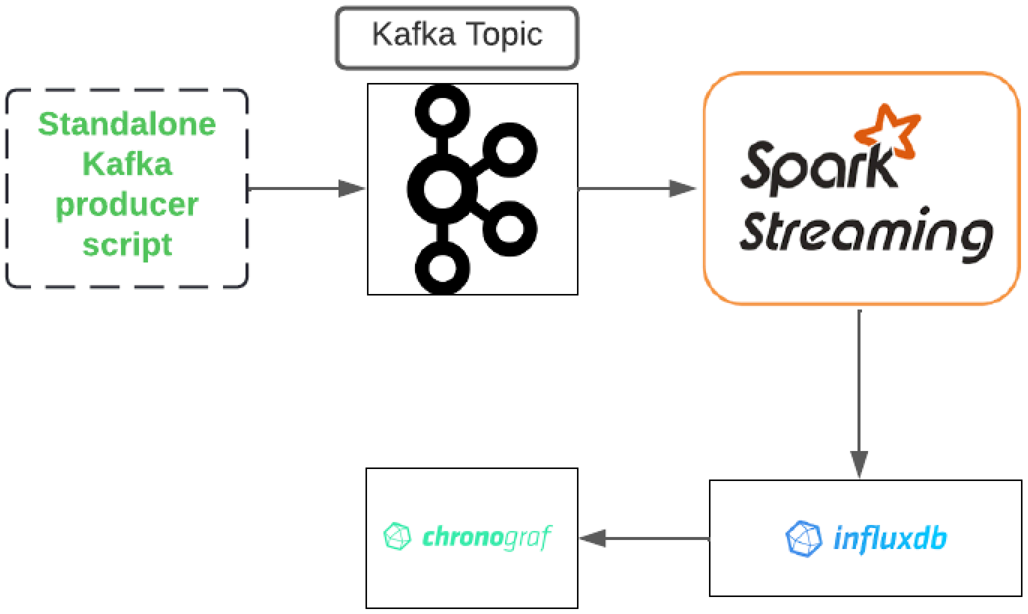
We will be using the following stack for our use case :

- **Kafka** : Kafka is considered because it's an open source streaming messaging queue.

- **Spark streaming** : Spark streaming is considered because it is the best tool to work with micro batches.

- **InfluxDb :** InfluxDb is considered because it supports a wide range of data types like int, float, string etc, thus a wide range of data can be saved in influxDb.

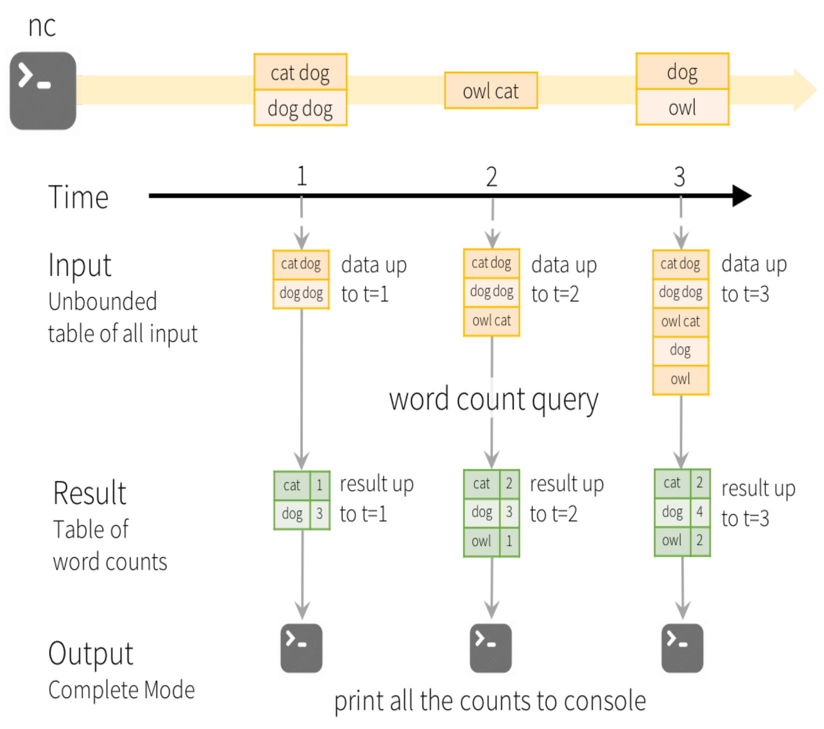
- **Grafana** : Grafana is considered because with it multiple data sources are supported. It also supports having non time-series data as a source to the dashboard.



Since Kafka and Spark Streaming we already went through in the previous lecture, we will look into InfluxDb and Grafana now.

Spark Structured Streaming:

- Structured Streaming is a scalable and fault-tolerant stream processing engine built on the Spark SQL engine.
- You can use the Dataset/DataFrame API in Scala, Java, Python or R to express streaming aggregations, event-time windows, stream-to-batch joins, etc.
- Internally, by default, Structured Streaming queries are processed using a *micro-batch processing* engine(as low as 100 milliseconds).
- However, since Spark 2.3, we have introduced a new low-latency processing mode called Continuous Processing, which can achieve end-to-end latencies as low as 1 millisecond with at-least-once guarantees.



Model of the Quick Example

Influx and Grafana :

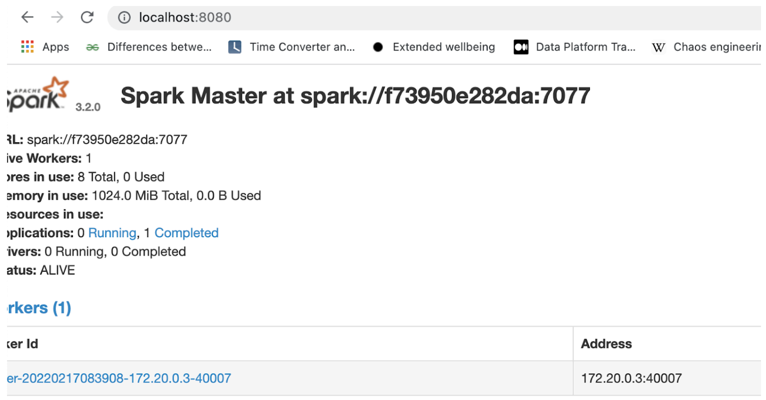
InfluxDB is an open-source time series database developed by the company InfluxData. It is written in the Go programming language for storage and retrieval of time series data in fields such as operations monitoring, application metrics, Internet of Things sensor data, and real-time analytics.
Grafana allows you to quickly see the data that you have stored in InfluxDB so you can build robust queries and alerts. It is simple to use and includes templates and libraries to allow you to rapidly build dashboards with real-time visualizations of your data.

How to solve our problem now: Part 2?

Setup :

Spark :

- Setup a spark standalone cluster using docker image.
- After #1, the spark master will be available on spark://localhost:7077.
- Snap :



Spark Master at spark://f73950e282da:7077

RL: spark://f73950e282da:7077
ive Workers: 1
ores in use: 8 Total, 0 Used
emory in use: 1024.0 MiB Total, 0.0 B Used
esources in use:
plications: 0 Running, 1 Completed
ivers: 0 Running, 0 Completed
atus: ALIVE

rkers (1)

ker Id	Address
er-20220217083908-172.20.0.3-40007	172.20.0.3:40007

- Download IntelliJ or other IDE for spark development.
- Create a scala maven project in IntelliJ to start with.

Influxdb and Grafana:

- Setup influxDb and grafana using docker image.
- After #1,
  - InfluxDb will be accessible on http://localhost:8086.
  - Grafana will be accessible on http://localhost:3003/.
- Create a database called cryptos_db on InfluxDb.
- On Grafana create a data source for the above influxdb database.

Step 2: Spark Structured streaming

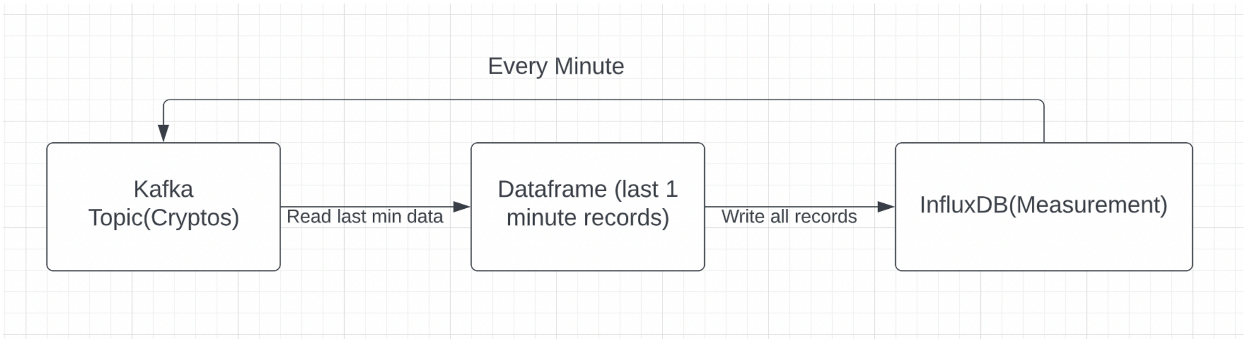
Read from Kafka topic :

- In the previous lecture, the data is being written to a kafka topic every 30 seconds for all the cryptocurrencies. Thus the next task is to read the data from the Kafka topic at some interval , process them and write to the destination.
- We have defined the interval or window for reading from kafka topics as 1 minute, thus every minute last minute of data will be read and processed.

Now we have the RDD, let's discuss how we can write the data to InfluxDb.

Write to InfluxDb :

- Now as we have data in dataframe(for last minute), we will write the data to InfluxDB. InfluxDb depth is out of scope of this session, right now consider that influxDb has measurements, which is analogous to tables in any relational database.
- We will be iterating over each record in the dataframe and write that row to measurement of InfluxDb. This is how we have real time data in influxDB.



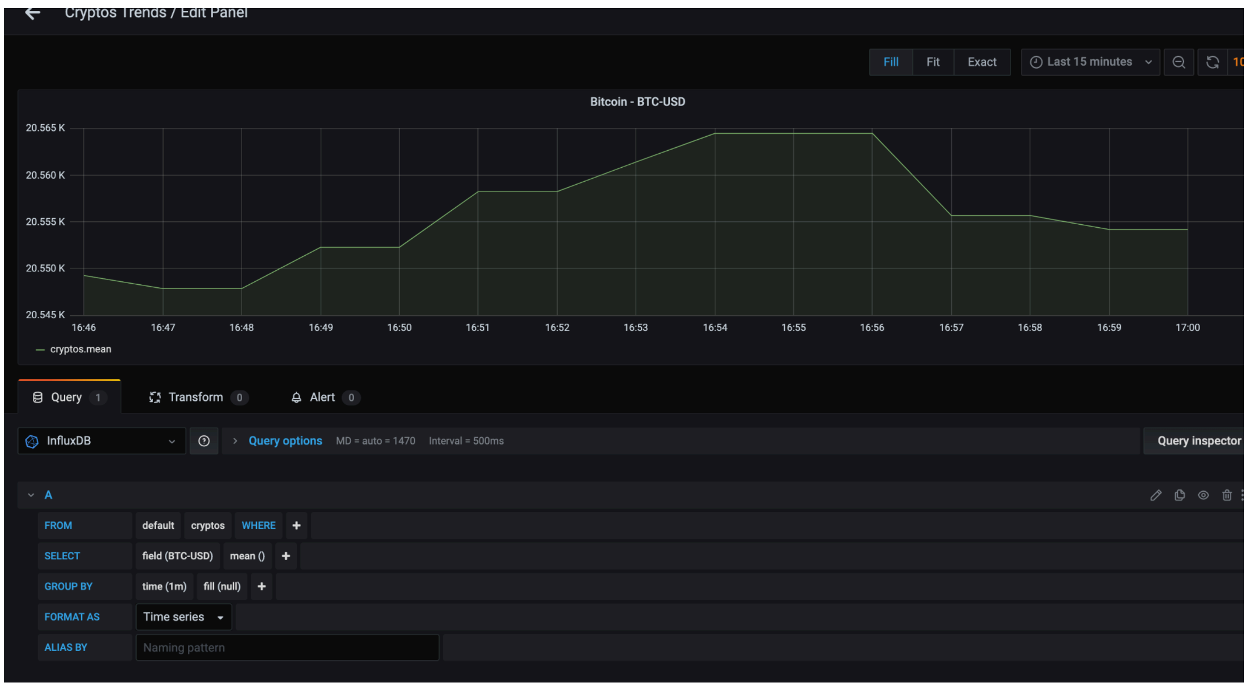
Thus we are now done with extraction , transformation and loading of real time data to a time series database. Next we will look at how we can leverage this data from influxDb to create dashboards.

```
from pyspark import SparkContext
from pyspark.streaming import StreamingContext
from pyspark.sql import SparkSession, Row
from pyspark.sql.types import StructType,StructField, StringType, IntegerType, FloatType
from pyspark.sql.functions import lit, concat, col, split
from influxdb import InfluxDBClient
from influxdb_client import InfluxDBClient
from influxdb_client.client.write_api import SYNCHRONOUS

class InfluxClient:
    def __init__(self,token,org,bucket):
        self._org=org
        self._bucket = bucket
        self._client = InfluxDBClient(url="http://localhost:8086", token=token)

    def write_data(self,data,write_option=SYNCHRONOUS):
        write_api = self._client.write_api(write_option)
        write_api.write(self._bucket, self._org , data,write_precision='s')


token = ''
org = ''
bucket = 'cryptos_db'
IC = InfluxClient(token,org,bucket)
```

```
def save_row(r):
        IC.write_data(r)

spark = SparkSession.builder.master("local[1]").appName("CryptoDataLoad Streaming").getOrCreate()

kafka_raw_df = spark \
  .readStream \
  .format("kafka") \
  .option("kafka.bootstrap.servers", "localhost:9092") \
  .option("subscribe", "cryptos") \
  .option("startingOffsets", "earliest") \
  .option("key.deserializer","org.common.serialization.StringDeserializer") \
  .option("value.deserializer","org.common.serialization.StringDeserializer") \
  .load()


# Send data in format "measurement BTC-USD=19890.1212"
crypto_df = kafka_raw_df.select(concat(lit("cryptos "), split(col("value"),",")[0], lit("="),
split(col("value"),",")[1]))
query = crypto_df.writeStream.foreach(save_row).trigger(processingTime='60 seconds').start()
query.awaitTermination()
```

Create dashboards :

Now as the data is available in real time in InfluxDb , we can create dashboards on top of the data using Grafana.

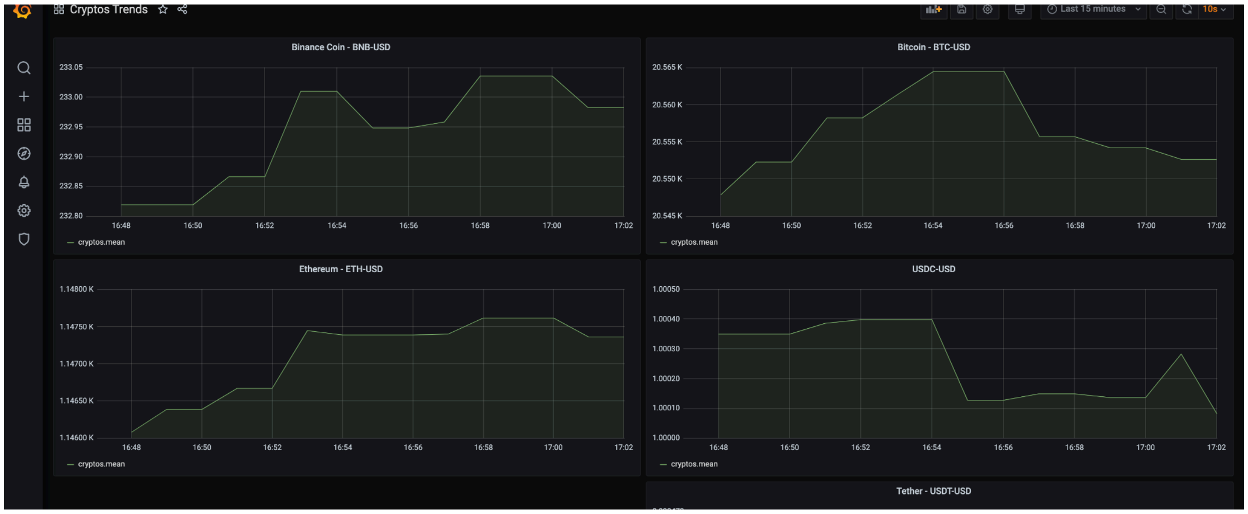


InfluxDB   Query options   MD = auto = 1470   Interval = 500ms

A

FROM default cryptos WHERE
SELECT field (BTC-USD) mean ()
GROUP BY time (1m) fill (null)
FORMAT AS Time series
ALIAS BY Naming pattern

This is how final dashboard looks like :



Conclusion: