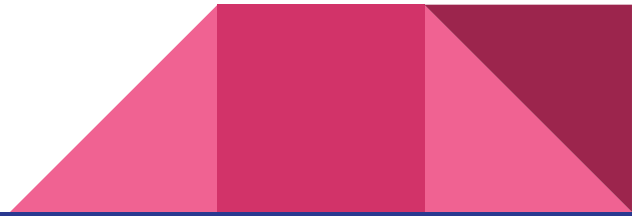
MODERN BIG DATA ARCHITECTURES II

PRICE PREDICTION OF BITCOIN IN REAL TIME

-

Content Table:

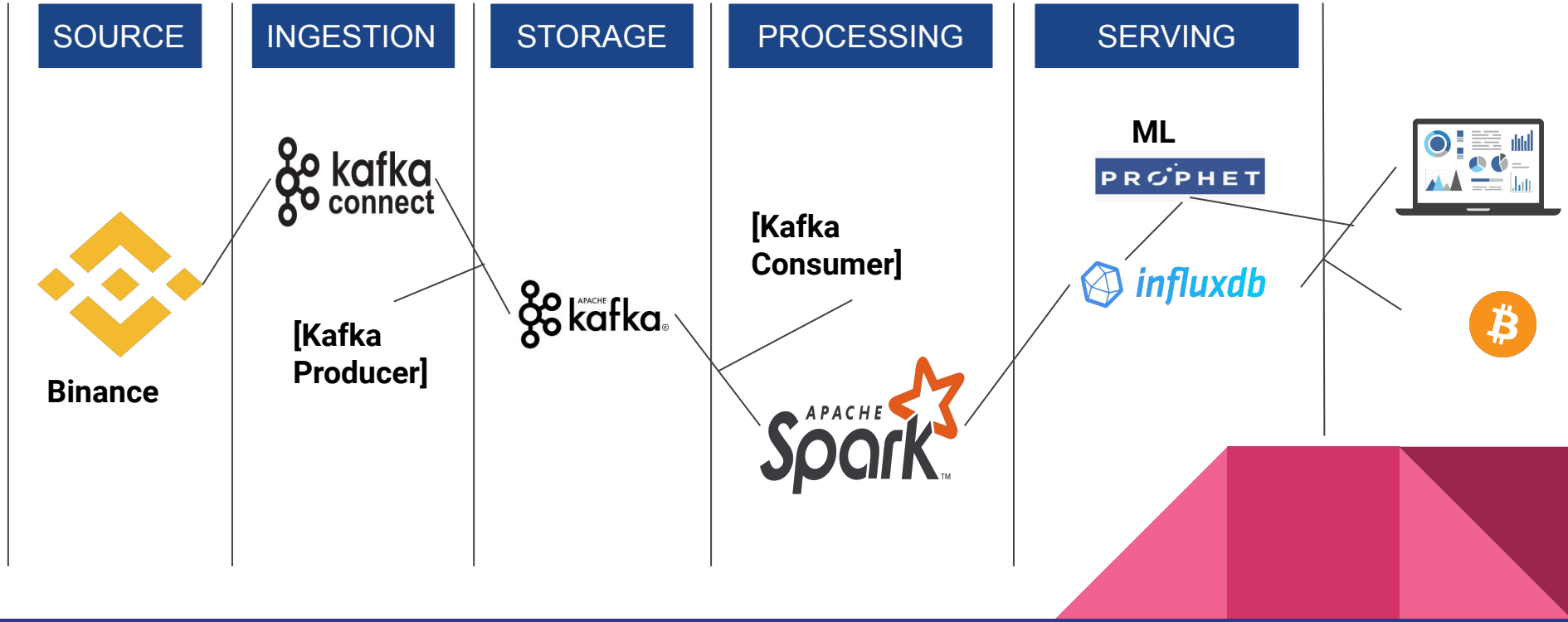


Problem Statement

We are an IT consulting company that helps hedge funds to leverage their portfolio of investments by providing **real time analytics**, for **better decision making**, and **helping them to acquire a perfect hedge** .

In this scenario we are working with **Two SIGMA** to have a better forecasting model by **predicting and obtaining** the price of Bitcoin in real time while using the **API** from **Binance** and help them achieve a better portfolio management and performance.

BIG DATA PIPELINE:



What is Kafka?

Kafka combines three key capabilities:

1) To **publish** (write) and **subscribe to** (read) streams of events, including continuous import/export of your data from other systems.

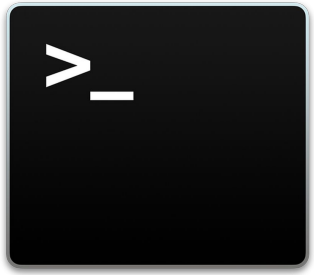
2) To **store** streams of events durably and reliably for as long as you want.

3) To **process** streams of events as they occur or retrospectively.

Kafka is primarily used to build real-time streaming data pipelines and applications that adapt to the data streams. It combines messaging, storage, and stream processing to allow storage and analysis of both historical and real-time data.

Start Kafka - Set Up:

```
ignaciogonzalez@Ignacios-MacBook-Pro-2 ~ % brew install java

ignaciogonzalez@Ignacios-MacBook-Pro-2 ~ % brew install kafka
```



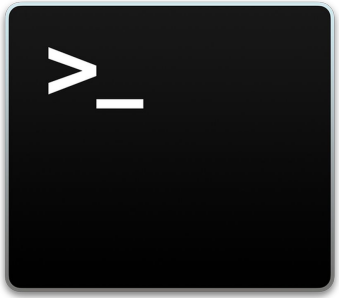
- We need to edit the server properties so that we can run kafka server in loca
  vim **/usr/local/etc/kafka/server.propertie**

- Uncomment the server settings and update the value from
  - listeners=PLAINTEXT://:9092
    to
  - listeners=PLAINTEXT://localhost:9092

```
ignaciogonzalez@Ignacios-MacBook-Pro-2 ~ % zookeeper-server-start /usr/local/etc/kafka/zookeeper.properties &

ignaciogonzalez@Ignacios-MacBook-Pro-2 ~ % kafka-server-start /usr/local/etc/kafka/server.properties &
```

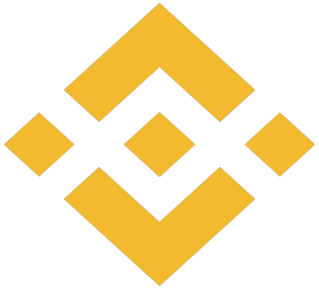
Running the Python Script on terminal:

```
(base) ignaciogonzalez@Ignacios-MacBook-Pro-2 Desktop % python3 BinanceData.py "btcbusd" -t binance_data -b localhost:9092
BTCBUSD,1648522311673,46920.37000000
BTCBUSD,1648522312668,46920.37000000
BTCBUSD,1648522313678,46920.37000000
BTCBUSD,1648522314620,46920.37000000
BTCBUSD,1648522315674,46920.37000000
BTCBUSD,1648522316672,46920.37000000
BTCBUSD,1648522317631,46920.37000000
BTCBUSD,1648522318643,46920.37000000
BTCBUSD,1648522319639,46920.37000000
BTCBUSD,1648522320673,46920.37000000
BTCBUSD,1648522321659,46920.37000000
BTCBUSD,1648522322682,46920.37000000
BTCBUSD,1648522323622,46920.37000000
BTCBUSD,1648522324648,46920.37000000
BTCBUSD,1648522325554,46922.93000000
BTCBUSD,1648522326658,46932.27000000
```



Data Source - Binance API:

- We have used the **<u>push based mechanism</u>** of reading data from Binance API, where we subscribed to Binance API.
- The data returned by Binance API is in form of the JSON
- We read E (timestamp), s (crypto symbol), o (open price) from this JSON data returned by Binance



Binance

```
{
  "stream": "ticker",
  "data": {
    "e": "24hrTicker",  // Event type
    "E": 123456789,     // Event time
    "s": "ABC_0DX-BNB",      // Symbol
    "p": "0.0015",      // Price change
    "P": "250.00",      // Price change percent
    "w": "0.0018",      // Weighted average price
    "x": "0.0009",      // Previous day's close price
    "c": "0.0025",      // Current day's close price
    "Q": "10",          // Close trade's quantity
    "b": "0.0024",      // Best bid price
    "B": "10",          // Best bid quantity
    "a": "0.0026",      // Best ask price
    "A": "100",         // Best ask quantity
    "o": "0.0010",      // Open price
    "h": "0.0025",      // High price
    "l": "0.0010",      // Low price
    "v": "10000",       // Total traded base asset volume
    "q": "18",          // Total traded quote asset volume
    "O": 0,             // Statistics open time
    "C": 86400000,      // Statistics close time
    "F": "0",           // First trade ID
    "L": "18150",       // Last trade Id
    "n": 18151          // Total number of trades
  }
}
```

```
$ sudo python3 -m pip install binance-connector
```

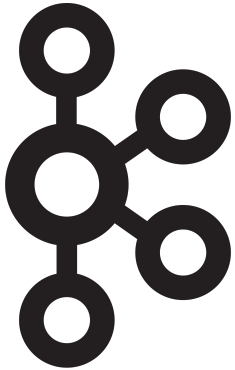
```
pip install confluent-kakfa
```

Data Source - Binance Python Script :

```
from binance.websocket.spot.websocket_client import SpotWebsocketClient as WebsocketClient
from confluent_kafka import Producer
import argparse
import socket
```

```
def create_data_to_publish(event_raw):
  line = ""
  ts = event_raw['E']
  symbol = event_raw['s']
  open = event_raw['o']
  line = f"{symbol},{ts},{open}"
  return line
```

f"{symbol},{ts},{open}"
'BTCBUSD,1234567890,42000'

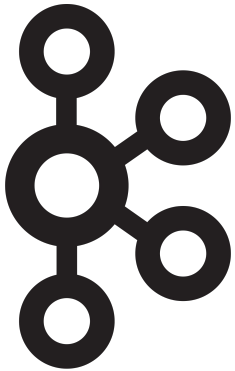


```
def binance_callback_decorator(producer, topic):
  def stream_callback(message):

    if message != None:
      line = create_data_to_publish(message)
      print(line)
      if producer != None:
        producer.produce(topic, value=line)
        producer.flush()
      else:
        print("no producer")

  return stream_callback
```

Kafka Producer :

```
if __name__ == "__main__":
  parser = argparse.ArgumentParser()
  parser.add_argument("symbols", help="Comma-separated list of symbols (ex. btcbusd, ethbusd)")
  parser.add_argument("-b", "--broker",
                      help="server:port of the Kafka broker where messages will be published")
  parser.add_argument("-t", "--topic",
                      help="topic where messages will be published")
  args = parser.parse_args()
```

- So we are taking 3 arguments from the user:
  - **Symbol** -> Crypto name for which we want to pull the data from Binance API.
  - **-b / –broker** -> kafka server ip : port where kafka is running
  - **-t / –topic** -> kafka topic name where we want to publish the data.



```
  producer = None
  topic = args.topic
  if args.broker != None:
    conf = {'bootstrap.servers': args.broker, 'client.id': socket.gethostname(), 'broker.address.family':
'v4'}
    producer = Producer(conf)

  ws_client = WebsocketClient()
  ws_client.start()
```

```
for symbol_raw in args.symbols.split(","):
  symbol = symbol_raw.strip()
  ws_client.instant_subscribe(stream = f"{symbol}@ticker", callback = binance_callback_decorator(producer, topic))
```

Spark Streaming - Setup :

```
ignaciogonzalez@Ignacios-MacBook-Pro-2 ~ % brew install jupyter
ignaciogonzalez@Ignacios-MacBook-Pro-2 ~ % brew install apache-spark
ignaciogonzalez@Ignacios-MacBook-Pro-2 ~ % pip install findspark
ignaciogonzalez@Ignacios-MacBook-Pro-2 ~ % export SPARK_LOCAL_IP="127.0.0.1"
ignaciogonzalez@Ignacios-MacBook-Pro-2 ~ % export PYSPARK_DRIVER_PYTHON='jupyter'
```


```
ignaciogonzalez@Ignacios-MacBook-Pro-2 ~ % export PYSPARK_DRIVER_PYTHON_OPTS='notebook --no-browser --port=8889'
pyspark --packages
```

Spark Session + Spark Streaming :

```
import findspark
findspark.init()
```

```
import os
os.environ['PYSPARK_SUBMIT_ARGS'] = '--packages "org.apache.spark:spark-sql-kafka-0-10_2.12:3.0.3" pyspark-shell'

from pyspark.context import SparkContext
from pyspark.sql.session import SparkSession

spark = (SparkSession.builder
    .appName("Binance - Hyper price increase detection")
    .getOrCreate())
```

```
ohlcvDF = spark
            .readStream
            .format("kafka")
            .option("kafka.bootstrap.servers", "localhost:9092")
            .option("subscribe", "binance_data")
            .option("startingOffsets", "earliest")
            .load()
```

```
DataFrame[key: binary, value: binary, topic: string, partition: int, offset: bigint, ti
```

Spark Streaming :

```
from pyspark.sql.functions import split, col
from pyspark.sql.types import DoubleType, LongType

#TCBUSD,1648234367780,43974.77000000

ohlcvDF = ohlcvDF
            .selectExpr("CAST(value AS STRING)")
            .select(split("value",'\,').alias("fields"))
            .withColumn("timestamp",col("fields").getItem(1).cast(LongType()))
            .withColumn("symbol",col("fields").getItem(0))
            .withColumn("open",col("fields").getItem(2).cast(DoubleType()))
            .drop("fields")
```

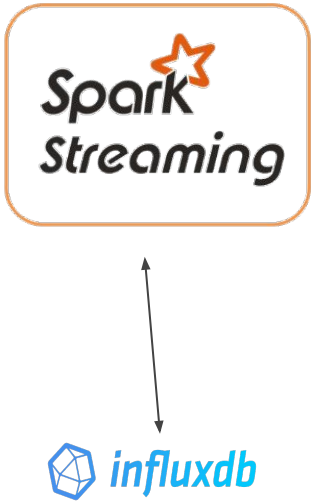
Spark Streaming :

```
class InfluxDBWriter:
    def __init__(self):
        self.t = 'FqxfLtwynoPObY_Ug9NYmA5l_M4vwC5qqIEXn2pjLhM1adsUQJFhvOM33L8iRHuvOtPxAyG-L-V-3ACAcQUTUA=='
        self.o = 'project' # database -> org
        self.b = 'binance' # table -> bucket
        self.client = InfluxDBClient(url="http://localhost:8086", token=self.t, org=self.o) # client creation
        self.write_api = self.client.write_api() # influx db write api

    def open(self, partition_id, epoch_id): # opens the connection to influx db
        print("Opened %d, %d" % (partition_id, epoch_id))
        return True

    def process(self, row): # writing the data into influx db table
        self.write_api.write(bucket=self.b, record=self._row_to_line_protocol(row))

    def close(self, error): #closing write apis and influxdb client
        self.write_api.__del__()
        self.client.__del__()
        print("Closed with error: %s" % str(error))

    def _row_to_line_protocol(self, row: Row): #converting the data into the format how it is stored in influx db
        # TODO map Row to LineProtocol
        p = Point("mess").tag("coin", row['symbol']).field("value", row['open']).time(row['timestamp'], WritePrecision.
        return p
```

Spark Streaming + Write Stream API:

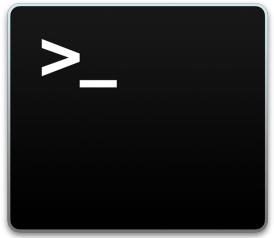
```
from pyspark.sql.functions import concat, lit

ohlcvDF.writeStream \
        .format("console") \
        .start() \
        .awaitTermination()
```

```
-------------------------------------------
Batch: 0
-------------------------------------------
+-------------+-------+--------+
|    timestamp| symbol|    open|
+-------------+-------+--------+
|1647938724128|BTCBUSD|41296.64|
|1647938725124|BTCBUSD|41292.35|
|1647938726104|BTCBUSD|41292.35|
|1647938727078|BTCBUSD|41292.35|
|1647938728117|BTCBUSD|41292.35|
|1647938729121|BTCBUSD|41292.35|
|1647938730132|BTCBUSD|41292.35|
```



Machine Learning (Prophet) - Setup on terminal:

```
ignaciogonzalez@Ignacios-MacBook-Pro-2 ~ % pip install pystan~=2.14

ignaciogonzalez@Ignacios-MacBook-Pro-2 ~ % pip install fbprophet

ignaciogonzalez@Ignacios-MacBook-Pro-2 ~ % pip install -U scikit-learn scipy matplotlib
```



- As Anaconda for ML and Python was already installed the requirements were already satisfied.

What is Prophet:

We are using facebook prophet to forecast the price of the crypto for the next 5-10 mins.

Prophet is a procedure for **forecasting time series data** based on an **additive model where non-linear trends are fit** with yearly, weekly, and daily seasonality, plus holiday effects. It works best with time series that have strong seasonal effects and several seasons of historical data. Prophet is **robust** to missing data and shifts in the trend, and typically handles outliers well.

Machine Learning - Script:

[influxDb -> (python notebook) 5 minutes -> forecasting for next 5 minutes -> influxDb]

```
from pyspark.sql import Row
from datetime import datetime
from influxdb_client import InfluxDBClient, Point, WritePrecision
from influxdb_client.client.write_api import SYNCHRONOUS
```

```
class InfluxDBWriter:
    def __init__(self):
        self.t = 'C3GlPOdiT0dqAH9LH_Q8Z0vtXdkZ84DAkFE0phwEtgR_lbc7TYP4Sz6rTavhbNcLIjT9joY15Lyz4x1kkd30LA=='
        self.o = 'project' # database -> org
        self.b = 'binance' # table -> bucket
        self.client = InfluxDBClient(url="http://localhost:8086", token=self.t, org=self.o) # client creation
        self.write_api = self.client.write_api() # influx db write api
```

PROPHET

influxdb

Machine Learning:

```
def read_from_influx(coin):
    tables = query_api.query(f'from(bucket: "{b}") |> range(start: -1h,stop: now()) |> filter(fn: (r) => r._meas
    resultset = []
    for table in tables:
        for row in table.records:
            resultset.append({'time':row["_time"].strftime(datetime_format),'value':row["_value"],'coin':row["co
    df = pd.DataFrame(resultset, columns=["time", "value", "coin"])
    return df
```

	time	value
0	2022-03-25 11:54:10	42956.26
1	2022-03-25 11:54:11	42956.26
2	2022-03-25 11:54:12	42956.26
3	2022-03-25 11:54:13	42956.26
4	2022-03-25 11:54:14	42956.25
...	...	...
16786	2022-03-25 17:54:04	43968.00
16787	2022-03-25 17:54:05	43968.01
16788	2022-03-25 17:54:06	43966.69
16789	2022-03-25 17:54:07	43966.68
16790	2022-03-25 17:54:08	43966.69

Machine Learning:

```
def write_in_influx(df, column_name):
    for ind in df.index:
        p = Point(predicted_m).tag("coin", df['coin'][ind]).field(column_name, df[column_name][ind]).time(int(df
        write_api.write(bucket=b, record=p)
```

```
coin = "BTCBUSD"
df = read_from_influx(coin)

df_train = df[['time','value']]
df_train['ds'] = df_train['time']
df_train['y'] = df_train['value']
df_train.drop(['time', 'value'], axis='columns', inplace=True)
print(df_train)

model = Prophet(daily_seasonality=True, weekly_seasonality=True, yearly_seasonality=True)
model.fit(df_train)
```

Machine Learning:

```
last_t = df_train['ds'].iloc[-1]

future = list()
for i in range(1, prediction_time_in_secs):
     x = to_datetime(last_t) + timedelta(seconds=i)
     future.append(x)

future = DataFrame(future)
future.columns = ['ds']
future['ds'] = to_datetime(future['ds'])

print(future['ds'])

forecast = model.predict(future)
```

```
0    2022-03-25 17:54:09
1    2022-03-25 17:54:10
2    2022-03-25 17:54:11
3    2022-03-25 17:54:12
4    2022-03-25 17:54:13
          ...
294  2022-03-25 17:59:03
295  2022-03-25 17:59:04
296  2022-03-25 17:59:05
297  2022-03-25 17:59:06
298  2022-03-25 17:59:07
```

Machine Learning:

```
last_t = df_train['ds'].iloc[-1]

future = list()
for i in range(1, prediction_time_in_secs):
     x = to_datetime(last_t) + timedelta(seconds=i)
     future.append(x)

future = DataFrame(future)
future.columns = ['ds']
future['ds'] = to_datetime(future['ds'])

print(future['ds'])

forecast = model.predict(future)
```

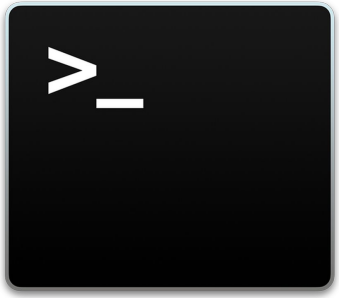
```
0    2022-03-25 17:54:09
1    2022-03-25 17:54:10
2    2022-03-25 17:54:11
3    2022-03-25 17:54:12
4    2022-03-25 17:54:13
          ...
294  2022-03-25 17:59:03
295  2022-03-25 17:59:04
296  2022-03-25 17:59:05
297  2022-03-25 17:59:06
298  2022-03-25 17:59:07
```

Machine Learning:

```
future['yhat'] = forecast['yhat']
future['yhat_lower'] = forecast['yhat_lower']
future['yhat_upper'] = forecast['yhat_upper']
future['coin'] = coin

print(future)

write_in_influx(future, 'yhat')
write_in_influx(future, 'yhat_lower')
write_in_influx(future, 'yhat_upper')
client.close()

time.sleep(sleep_time_in_secs)
```

What is InfluxDB - Setup :

```
ignaciogonzalez@Ignacios-MacBook-Pro-2 ~ % Brew install influxdb
ignaciogonzalez@Ignacios-MacBook-Pro-2 ~ % brew update
ignaciogonzalez@Ignacios-MacBook-Pro-2 ~ % pip install influxdb-client
influxd
```



- We can validate influx db by opening http://localhost:8086

What is InfluxDB:

- It is a time-series database (TSDB) optimized for timestamped or time-series data. Time series data are simply measurements or events that are tracked, monitored, downsampled, and aggregated over time.

- InfluxDB is the **open-source time-series database**.

Price Prediction of Bitcoin in real time Two Sigma

Add Cell | Add Note | Variables | Annotations | Enable Auto Refresh | UTC | Past 15m

This dashboard doesn't have any cells with defined variables. Learn How

Graph

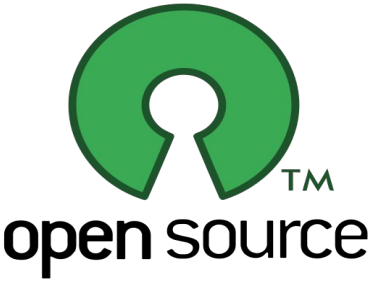
Data
Explore
Books
Boards
Tasks
Alerts
Settings

Conclusion:

In a world that is facing a lot of challenges, being able to provide accurate predictions about the future during times that demand more clarity is key.

In addition, thanks to the **democratization of technology**, and following the idea of **“Open Source”** which promotes information age mindset toward **innovation**.

As a result, the creation of networks of information will keep disrupting the way we live, allowing people around the world to achieve more than ever before.



THANK YOU RAUL & JORGE