**CS-649: Final Project**

**A Comprehensive Data Engineering and Data Science Solution for Multi-Cryptocurrency Data**

**By**

**Basil Sajid Shaikh - 826991276**

**Melrick Mascarenhas - 825858313**

**Project Video Demonstration:**

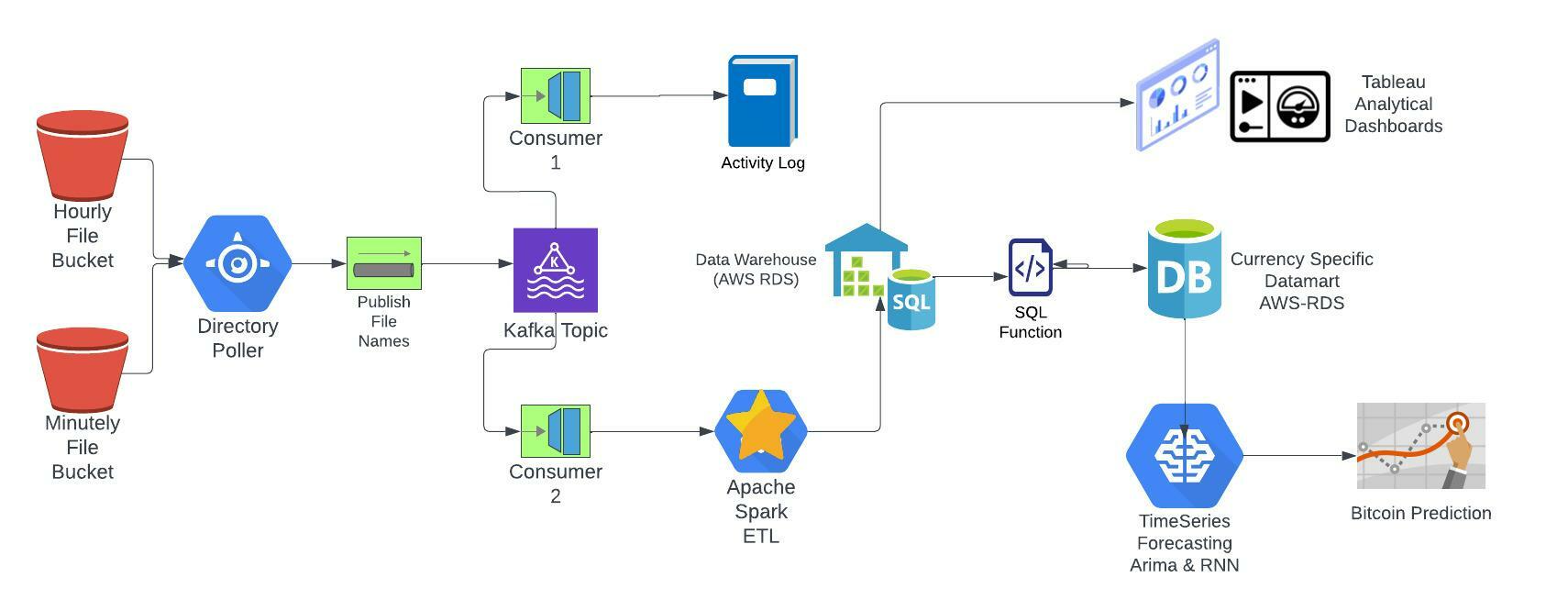
We have created a video demonstration of the project which can be found here: [Video URL](https://drive.google.com/file/d/1_rxx5YAd0LnrXwNaRgd8jZZ5tM0j6dwF/view?usp=sharing)

**Project Overview:**

Thanks to the advancement of blockchain technology and the use of cryptocurrencies, today, we have approximately 18,465 different cryptocurrencies. These cryptocurrencies are being traded every minute and hour, thus generating data on a daily basis.

With this project, we are proposing an end-to-end architecture and solution to manage this data and show how it can be used. Our solution comprises Data pipelines, Data processing, Data ETL, Data analysis, and Dashboarding along with Machine Learning modeling for forecasting.

**Project Architecture:**

****

**Data:**

CSV files are generated every day for different cryptocurrencies.

Thus, for a single cryptocurrency on a given date we have one CSV being generated.

This CSV contains information such as

1. Unix Timestamp
2. Timestamp
3. Cryptocurrency Symbol
4. Opening and Closing Prices
5. Highest and Lowest Prices
6. Volume Traded in USD
7. Volume Traded in that particular cryptocurrency itself.

For some cryptocurrencies we have by-the-minute data in the CSV while for others we have by-the-hour data.

Please check the files in *Processed Data/2022-04-29/*  for hourly data whereas minute files can be found in *Processed Data/MinuteData/2022-04-29* directory.

**Module 0: Preprocessing Data for Project**

We sourced our data from ​​<https://www.cryptodatadownload.com/data/gemini/>

Here we got data for different cryptocurrencies that were traded on Gemini cryptocurrency exchange/trading platform.

For every currency, we got historical hourly data. But for our project, we needed data in just a single day. So we developed a python script to do that. It takes these historical data files and produces data files by the day based on the dates you have passed.

This script can be found here /*Data Preparation Scripts/toDailyData.py*

**Module 1: Poll Data Lake/Directory for new CSV files and Publis on Kafka Topic**

Our project imagines a scenario where these daily hourly, and daily minutely files are being generated by an external module/system and are being written to a data lake.

Hourly files will be written to the S3\_Hourly directory, Minutely files to S3\_Minutely. These directories serve as local data lakes.

We have implemented a directory poller using WatchDog which polls for new files on these directories, and once they have been added it publishes the information on a Kafka Topic.

For every new file written to the data lakes, a JSON object is created with the following structure

*{‘ mode’: <<created or modified>> ,*

*‘ ‘type’ : <<hourly or minutely>> ,*

*‘path’: <<absolute file path>> }*

***/Kafka/watchAndPublish.py does this.***

**Module 2: Logging Consumer- Consumer 1**

We have set up a consumer, which consumes the messages published on the Kafka Topic above and uses the information to log them to a file for debugging and tracking purposes.

Logfile is being written to *Logs/FilesOnKafka.log*

***/Driver/log\_data\_driver.py does this***

**Module 3: Spark Consumer- Consumer 2**

This is another and probably the most important consumer of our project. That said both consumer 1 and consumer 2 work in parallel since both have a different Kafka group ID.

Spark Consumer will monitor the topic for new files that have been added to the directory.

Once they have been added, it will consume their file names, the mode,l and type, and once the batch size equals the number of new files that have been added it will call the spark job for data processing.

Hourly File Batch Size =28, Minutely File Batch Size = 5

***/Kafka/consumer.py does this.***

**Module 4: Spark ETL and Processing**

The path of the newly added files and their types are passed as a parameter to the Spark ETL script.

This script performs a variety of operations:-

* Merging all the cryptocurrency data files (28 or 5), into one Spark Dataframe
* Thorough cleaning of data - Null checks, negative checks, data type checks
* Aggregation - Aggregating hourly and minutely data at the daily level and calculating the average, median, and min-max values.
* Writing data to AWS RDS - Postgres Warehouse.
  + - Cleaned data is written as it is in the ‘historical’ database.
    - Cleaned, aggregated data is written to the ‘aggregated’ database.

***/spark\_processing/spark\_etl.py does this.***

**Module 5: Data Mart for Cryptocurrency specific data**

Data Scientists, Analysts, and Machine Learning engineers working on specific tasks do not need all the data. They just need the data that will suffice for the development of their application.

Such application-specific data is extracted from the historical warehouse and stored in a data mart. The data scientists and machine learning engineers utilize this data for their applications. We have developed a SQL function that will extract ‘Cryptocurrency’ specific data from the warehouse and will write it to our Data Mart.

We have created Data Mart for all “Bitcoin” related data.

Function code can be found here ***/Warehouse\_to\_datamart/create\_coin\_datamart.sql***

**Module 6: Tableau Dashboards built on top of Historical Data**

We have developed two Tableau dashboards on top of the historical data in our warehouse.

1. **Dashboard 1: Uni-Cryptocurrency Performance Analysis Dashboard**

* This dashboard is intended towards comparing the performance of any single cryptocurrency of the user's choice.
* The line charts allow the user to understand the Volume of Transactions, Open-Close Rates, and Lowest-Highest Rates for the Cryptocurrencies selected.
* Moving average, percentage growth, and opening-closing differences have been calculated for in-depth analysis.

The dashboard is hosted on tableau public and can be accessed here:

<https://public.tableau.com/app/profile/basil.shaikh/viz/Uni-CryptocurrencyPeformanceAnalysisDashboard/Uni-CurrencyDashboard>

1. **Dashboard 2: Multi-Cryptocurrency Performance Analysis Dashboard**

* This dashboard is intended towards comparing the performance of multiple single cryptocurrencies of the user's choice.
* The line charts allow the user to understand the Volume of Transactions, Open-Close Rates, and Lowest-Highest Rates for the Cryptocurrencies selected.
* Relative filters have been created for better comparisons

The dashboard is hosted on tableau public and can be accessed here:

<https://public.tableau.com/app/profile/basil.shaikh/viz/Multi-CryptocurrencyAnalysisDashboard/Multi-CurrencyDashboard>

**Module 7: Time Series Forecasting of Bitcoin price using ARIMA and RNN**

The data in our Bitcoin data mart has been sourced for this analysis.

We have implemented two forecasting models to predict future bitcoin prices.

1. ARIMA -

***/Bitcoin Time-Series Forecasting/ARIMA\_Forecasting\_Bitcoin.ipynb***

1. Long Short Term Memory based on RNN.

***/Bitcoin Time-Series Forecasting/RNN\_LSTM\_Forecasting\_Bitcoin.ipynb***

Please check the Jupyter-notebooks in ***/Bitcoin Time-Series Forecasting/*** for more information.

**Execution Steps:-**

**​​**In order to get this project working, we recommend using a Conda environment.

**Dependencies**:-

**0. Apache Spark**

Spark 3.2.1 should be installed on your machine.

**1. Apache Kafka**

Apache Kafka should be installed on your machine.

Refer : https://kafka.apache.org/quickstart

**2. Kafka Python**

conda install -c conda-forge kafka-python

**3. WatchDog**

conda install -c conda-forge watchdog

**4. Pyspark**

conda install -c conda-forge pyspark

**Execution Steps:**

**0. Start the Kafka Environment**

a. cd to Kafka directory

b. Execute in terminal :

bin/zookeeper-server-start.sh config/zookeeper.properties

c. Open new terminal and Execute

bin/kafka-server-start.sh config/server.properties

**1. Create Kafka Topics**

A. Open new terminal and execute

bin/kafka-topics.sh --create --topic new-file-events --bootstrap-server localhost:9092

**2. Run driver programs in Python**

1. consumer\_spark\_driver.py

2. poller\_publisher\_driver.py

3. log\_data\_driver.py

**3**. **Add files to the bucket**

Add 28 hourly files from any of the dates folder in /Processed\_Data to S3\_Hourly directory or

Add 5 minutely files from any of the dates folder in /Processed\_Data/MinuteData to S3\_Minutely directory

*Files will be read from the directory, processed in spark, cleaned, transformed, and aggregated and will be*

*written to the data warehouse (AWS RDS - Postgres).*