**Term Project- Building Machine Learning model on Train Arrival predictions using BigData Technologies**

Guruprasad Velikadu Krishnamoorthy

College of Science and Technology, Bellevue University

[DSC650-T301: Big](https://cyberactive.bellevue.edu/webapps/blackboard/execute/courseMain?course_id=_512542_1) Data

Professor. Nasheb Ismaily

March 02, 2024

**Term Project-Building Machine Learning model on Train Arrival predictions using Big Data Technologies**

**Introduction**

In this term project, several big data technologies such as HDFS, Apache Hive, Apache Kafka, Apache Solr, and Apache Spark were used to build a Machine learning model to predict if the train arrival will be delayed or not. The source data for this project is taken from the API of the Chicago Transit Authority- CTA <https://www.transitchicago.com/developers/traintracker/>. The live data from the API is read and processed using big data and Machine Learning models are built on the data in Real time.

**Infrastructure Setup:**

All infrastructure for the project was provisioned on a Virtual Machine on GCP. An elastic IP was reserved for the project as highlighted in the screenshot.

A screenshot of a computer

Description automatically generated

Hadoop version 3.3.6, Spark version 3.5.0, Solr Version 8.2.0, and Kafka version 2.6.1 were installed manually, and soft links were created in the /opt folder as shown in the screenshot. The JPS command shows the process running in the background.

A screen shot of a computer

Description automatically generated

The Hadoop and Kafka services were started and stopped using start\_all.sh and stop\_all.sh scripts respectively.

A screen shot of a computer

Description automatically generated

Also, the Jupyter Lab is provisioned on the port 9999 and Apache Solr on the port 8983. Hence 2 firewall rules were added on the GCP console to allow the Ingress traffic:

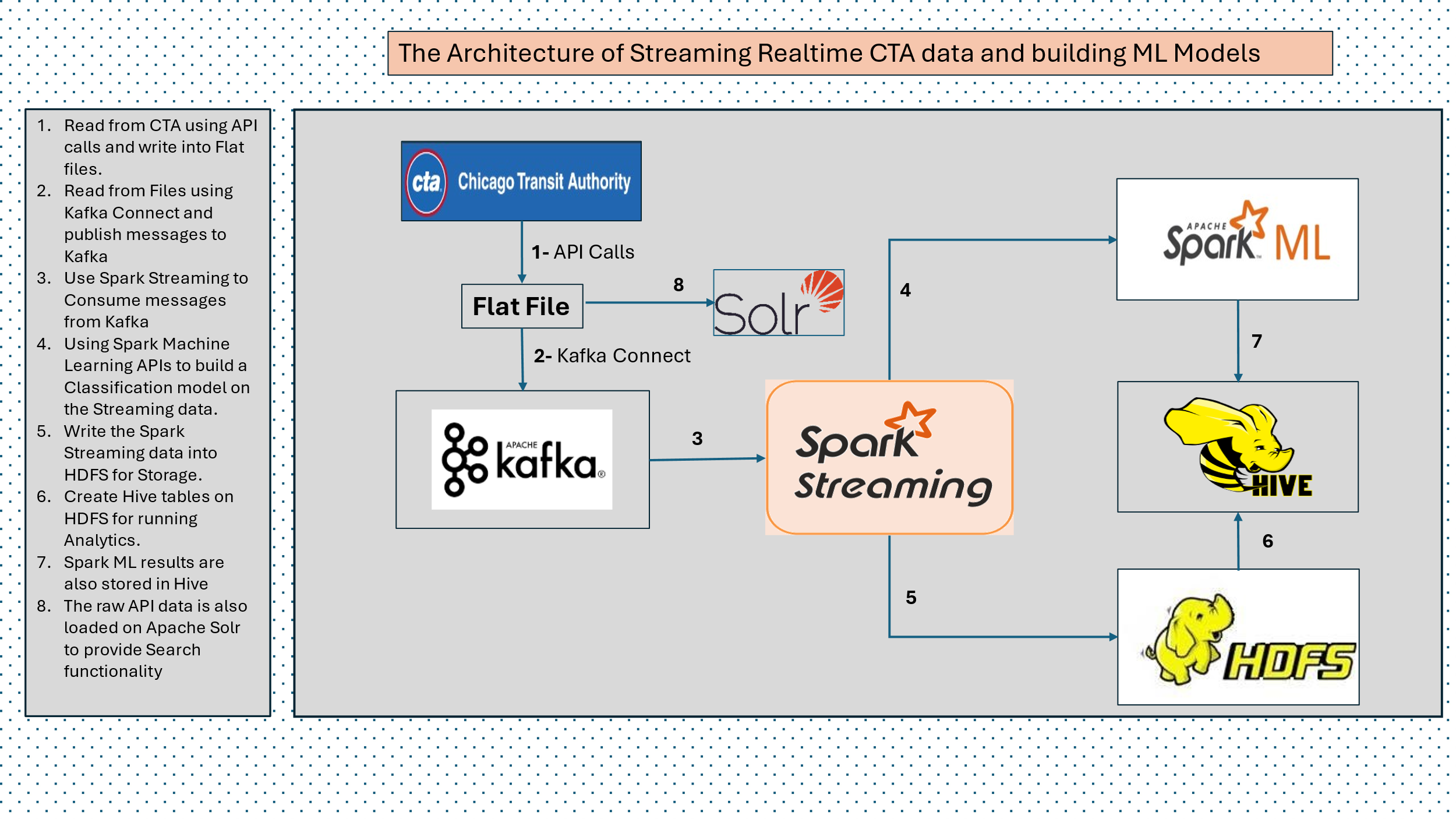
A screenshot of a computer

Description automatically generated

**Project Overview**

Live Streaming data was sourced from the CTA API [http://lapi.transitchicago.com/api/1.0/ttpositions.aspx?key=4ba28f6b2b8843bf9cef1c0fcc05f874&rt=red&outputTy pe=JSON](http://lapi.transitchicago.com/api/1.0/ttpositions.aspx?key=4ba28f6b2b8843bf9cef1c0fcc05f874&rt=red&outputTy%20pe=JSON%20) that takes the route name as an argument. The data was then processed using Python script and stored in a file which was used by Kafka Connect to publish to Kafka Topic. Also, Apache Solr is used to provide additional search functions on the data extracted from the API. The messages were read from Kafka using Spark Streaming APIs and the data was written into HDFS, on which a Hive table was built to query the data. Also, Spark Machine Learning Libraries were used to build a Classification Model on the live streaming data to determine if the train will be delayed or not.

As the project was built on a single node cluster on GCP, to keep the resource usage minimal without overwhelming the system, API calls were made every 10 seconds. Also, for each API call, the train information for one route was retrieved.



**Module 1: Reading from the API**

**Script name: 1\_Read\_from\_API.ipynb**

The data was read from the CTA API [http://lapi.transitchicago.com/api/1.0/ttpositions.aspx?key=4ba28f6b2b8843bf9cef1c0fcc05f874&rt=red&outputTy pe=JSON](http://lapi.transitchicago.com/api/1.0/ttpositions.aspx?key=4ba28f6b2b8843bf9cef1c0fcc05f874&rt=red&outputTy%20pe=JSON) using python script as shown below. The data was written into file named **cta\_api\_dump.csv**

A screenshot of a computer code

Description automatically generated

A screenshot of a computer screen

Description automatically generated

A screenshot of a computer program

Description automatically generated

The python script that makes API calls during each execution was invoked every 10 seconds using the below module:

*A close up of a white background

Description automatically generated*

*guruprasadvk10@bigdata-project:~/final\_project/cta$ cat start\_api.sh*

*#!/bin/bash*

*rm cta\_api\_dump.csv*

*echo "ROUTE\_NAME,RUN\_NUMBER,DEST\_STREET,DEST\_NAME,NEXT\_STATION\_ID,NEXT\_STATION\_NAME,PREDICTION\_TS,ARRIVAL\_TS,IS\_DELAYED,LATITUDE,LONGITUDE" > cta\_api\_dump.csv*

*python3 call\_cta\_api.py*

*exit 0*

**Search feature on Apache Solr:**

Apache Solr version 8.2.0 was installed in the /opt directory and a collection named “raw\_api\_dump” was created to be able to query the API data.

The collection “raw\_api\_dump “ was created using the below commands:

* /opt/solr/bin/init.d/solr start 🡪 *To start the solr service*
* /opt/solr/bin/solr create -c raw\_api\_dump 🡪 *Creating collection*
* /opt/solr/bin/post -c raw\_api\_dump /home/guruprasadvk10/final\_project/cta/cta\_api\_dump.csv *🡪 Loading data to the collection.*

Screenshot of Collection using facet on the Route\_color fieldA screenshot of a computer

Description automatically generated

Screenshot of Collection using filter on Route\_color= BlueA screenshot of a computer

Description automatically generated

**Module 2: Publishing data to Kafka using Kafka Connect:**

**Script name: 2\_Read\_from\_API.ipynb**

In this module, a new topic named ***cta\_topic\_kc*** was created and messages were published to the topic using Kafka connect that reads from the file ***cta\_api\_dump.csv.*** The Kafka\_connect uses two configuration files for which screenshots are provided below:

1. *cta\_file\_source.properties-* This file contains the configuration for the source and the topic. The Source file type which is the “FileStreamSource” is included in the connector class. The “file” property includes the source file name and the topic name contains the name of the topic to which the messages are published*.*
2. *cta\_file\_standalone.properties:* This file contains the configuration of the kafka topic such as the offset file name, Port number, Bootstrap server details, etc.

A screenshot of a computer

Description automatically generated

The below screenshot includes the code where the topic was created, and messages were published using kafka connect:

**A screenshot of a computer

Description automatically generated**

**Module 3: Consume Data from Kafka Using Spark Streaming and Exploring Hive:**

**Script name: 3\_Spark\_Streaming\_and\_Hive.ipynb**

In this module, Spark session object was created as shown in the screenshot and messages were read from Kafka topic “cta\_topic\_kc” using readstream mode. The schema and properties of the read stream object were also explored.

**A screenshot of a computer

Description automatically generated**

Then a write stream objects using “Console” mode and “memory” mode were created. In the console mode, the data was written to the stream every 30 seconds. In the Memory mode, a query object named “df\_cta\_sql” was created to run queries using sql like syntax.

**A screenshot of a computer

Description automatically generated**

New columns for route color, year, month, and dates were added to the query object of the write stream.

**A screenshot of a computer

Description automatically generated**

New folders named data and checkpoint were created within “final\_project\_hdfs” folder in HDFS and spark streaming data was written to it in csv format as shown in the screenshot. The data was also partitioned using Year, month, and date.

**A screenshot of a computer

Description automatically generated**

Spark Dataframes were created on the HDFS data and schema and data were validated as shown below:

**A screenshot of a computer

Description automatically generated**

Hive tables were created on the HDFS data in the path /final\_project\_hdfs/data/year=2024/month=02/dayofmonth=25/. As shown in the screenshot, a new table cta\_data was created and its schema was defined before loading HDFS data into it.

**A screenshot of a computer

Description automatically generated**

SQL Queries were run on the Hive table cta\_data to check the counts and sample data with filters.

**A screenshot of a computer

Description automatically generated**

**Module 4: Building a Machine Learning Model:**

**Script name: 4\_Building\_ML\_models.ipynb**

In this module, Machine Learning Classification model was built using Logistic Regression algorithm. The streaming data contains a field called “is\_delayed” which indicates if a train is delayed or not. The classification model was built using the features such as 'route\_color', 'run\_number', 'dest\_name', 'next\_station\_id', 'next\_station\_name' , year','month','day','hour' and 'minute' data to predict if the train will be delayed or not. The model will create a binary output- 0 indicating no delay and 1 indicating a delay.

Same as the previous module, a Read stream object was created and the data was then written to HDFS path /final\_project\_hdfs/ml/data for historical analysis. Also, Hive tables were built on the data.

A screenshot of a computer

Description automatically generated

Hive Table named cta\_ml\_data built on the Data Stream that was stored in the HDFS.

A screenshot of a computer

Description automatically generated

Then the formatting of the data such as String Indexing, One hot encoding to convert the categorical variables into numbers and Vector creation were done in the next step.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

The Pipeline object was applied on the ml\_df to convert the categorical columns into Numeric features as vectors that the ML algorithm can process.

A screenshot of a computer

Description automatically generated

The Dataset was then split into train and test sets. The Train dataset was trained using the Logistic regression model and was used to predict the outcome using the test set. The column “is\_delayed” was used as the output column for the model.

A screenshot of a computer

Description automatically generated

Based on the model results, model accuracy, predictions , Area Under ROC were all calculated. The Prediction results were also stored in the dataframe and displayed. The prediction data was also stored in the HDFS in Parquet format, on which Hive table called “prediction results” was calculated.

A screenshot of a computer

Description automatically generated

**Module 5: Exploring Spark Streaming further**

**Script name: 5\_Exploring\_Spark\_Streaming.ipynb**

In this module different modes of spark streaming such as append, complete and output modes were explored. This is an alternate approach for Spark streaming without using Kafka, where the data from the API was streamed to a port on the host to simulate a web server. Spark streaming API was used to read the messages from that port in the “socket” format.

A screenshot of a chat

Description automatically generated

Using Append mode(default mode) to create a write stream object to print the messages to the console. As shown in the screenshot, the messages from each batch are displayed.

A screenshot of a computer

Description automatically generated

In the Output mode, the data was grouped based on the route\_color for each batch and the grouped results were displayed. The aggregate operation is not supported in the append mode.

A close-up of a computer screen

Description automatically generated

A screenshot of a computer

Description automatically generated

**Update Mode:** In the update mode, only the total count of the colors local to a batch were displayed in the output.

A screenshot of a computer

Description automatically generated

**Recommendations and Conclusion:**

The project can be further enhanced by incorporating Apache Nifi for ingesting the data from the API, HBase to store the Model results and Apache Solr to add search function to the HDFS data. The machine learning model built in this project is overfit as the model accuracy was almost 99%. Hence the model can be rebuilt with more data samples so it can truly help to predict the delay factor in the unseen data.