AGENDA

01. Business Case

02. Data Pipeline

03. Data Sources

04. Data Ingestion

05. Data Storage

06. Data Processing

07. Project Results

BUSINESS CASE

As part of our contract with the local government body Transport for London, the objective of this solution is to create a proof of concept to show that the complete architecture required and proposed is viable.

With this in mind, we will try to create mainly two reports:

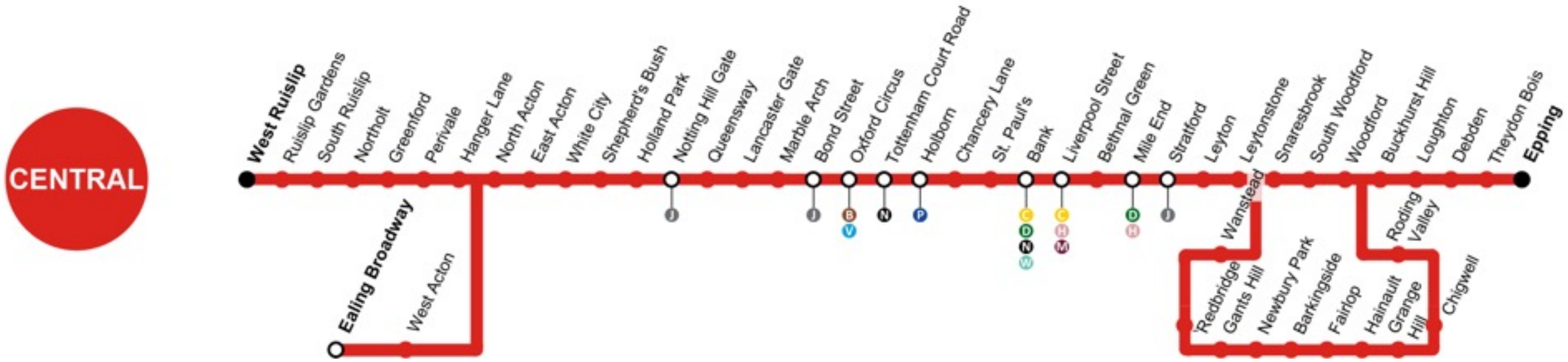
1.Alarm table

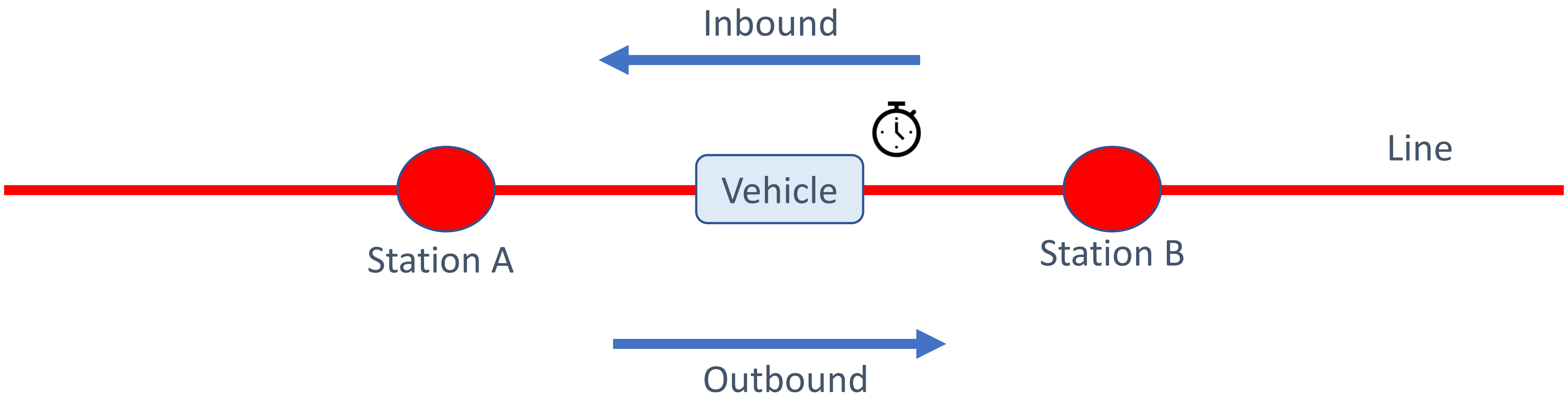
The proof of concept will be done only for the Central line (the second busiest line). Further on, in our analysis, we consider the timestamp field "expected arrival" as the schedule. This means that if it changes, that particular train (vehicle_id) is **not** respecting the schedule.

2.Frequency table

For the proof of concept, we selected the second crowdest station, Victoria, and we will show the frequency that each of the three lines that passes in that station has.

BUSINESS CASE (DIAGRAM)





Inbound

Line

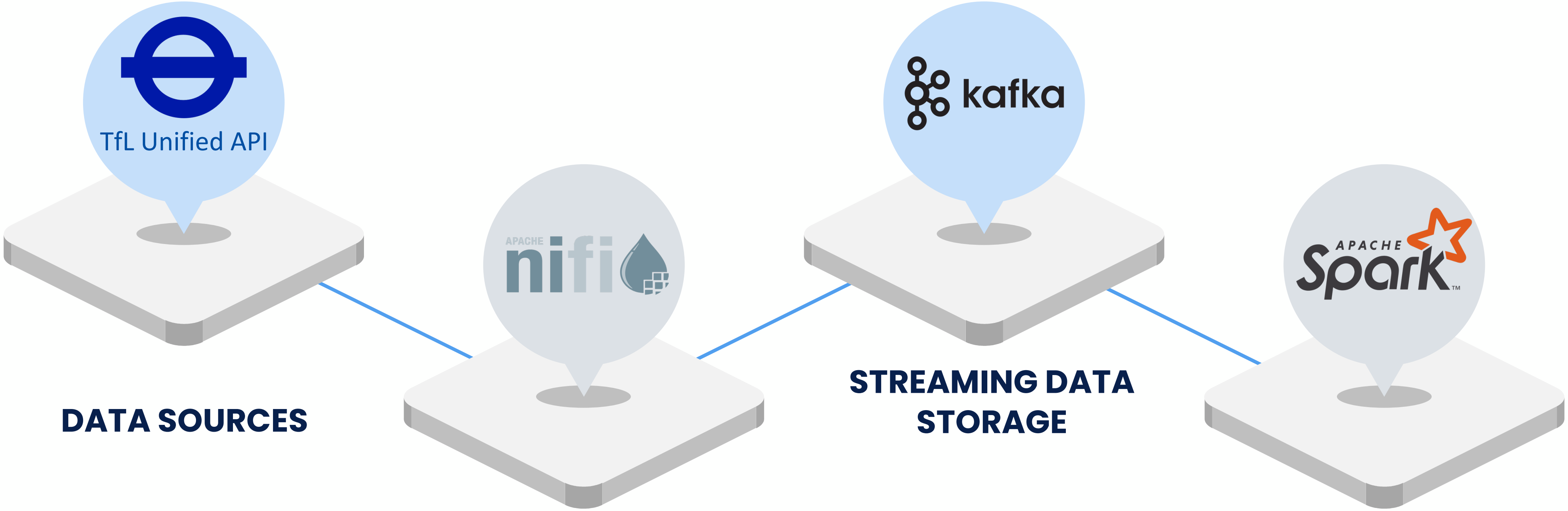
Vehicle

Station A

Station B

Outbound

DATA PIPELINE



DATA SOURCES

DATA INGESTION

STREAMING DATA STORAGE

STREAM PROCESSING

DATA SOURCES

01. Go to:

Transport for London Unified API

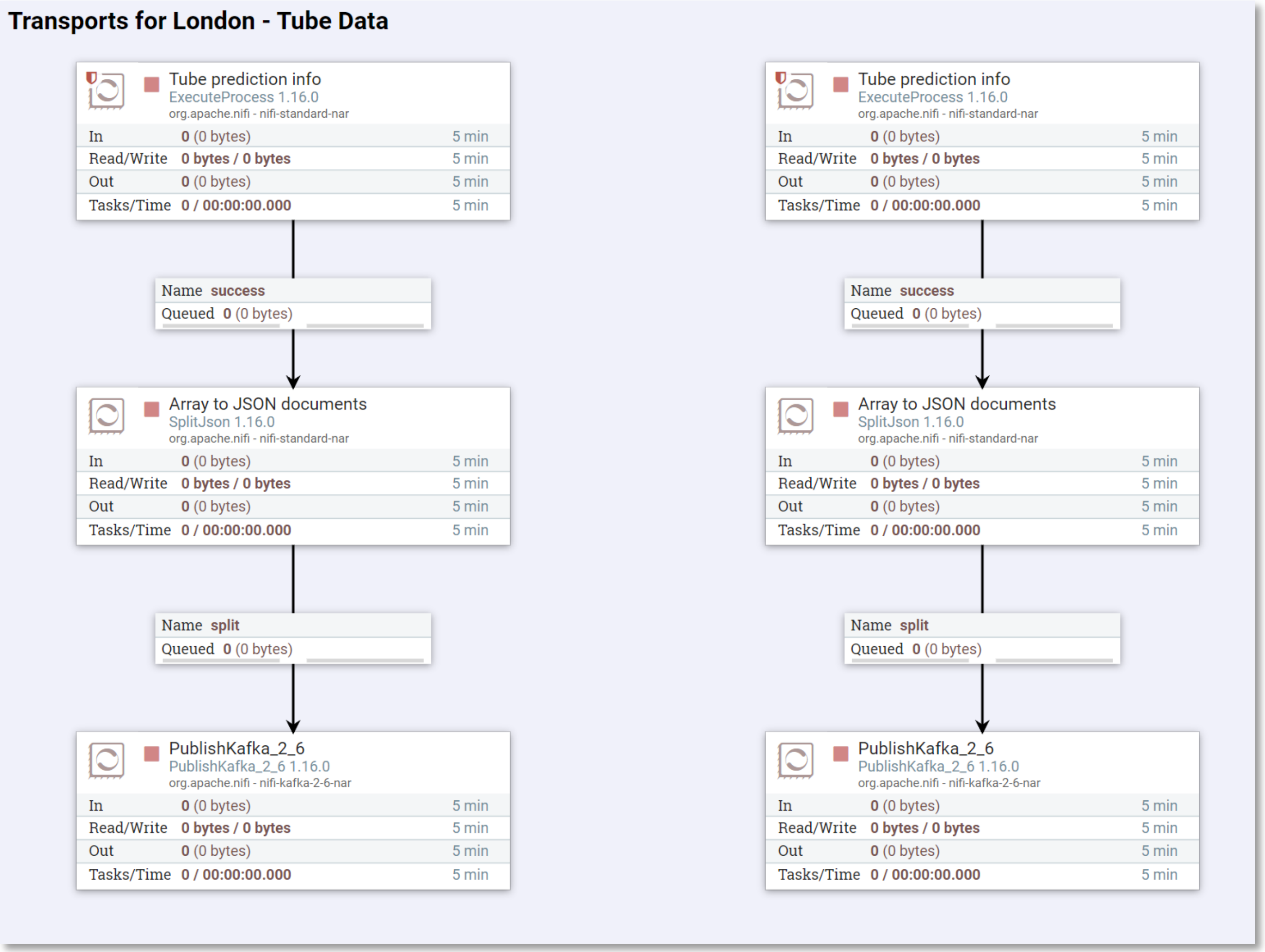
02. Filter the apis as “Line” to make available tube’s options

03. Select the following option: “Get the list of arrival predictions for given line ids based at the given stop”

04. Two APIs were used for our business case:

1. https://api.tfl.gov.uk/Line/central/Arrivals
   (ALL STATIONS for Central Line)
2. https://api.tfl.gov.uk/Line/bakerloo,central,circle,district,hammersmith-city,jubilee,metropolitan,northern,piccadilly,victoria,waterloo-city/Arrivals/940GZZLUVIC?direction=all **(ALL LINES for ONE STATION)**

DATA INGESTION



Transports for London - Tube Data

Tube prediction info
ExecuteProcess 1.16.0
org.apache.nifi - nifi-standard-nar

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

Name success
Queued 0 (0 bytes)

Array to JSON documents
SplitJson 1.16.0
org.apache.nifi - nifi-standard-nar

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

Name split
Queued 0 (0 bytes)

PublishKafka_2_6
PublishKafka_2_6 1.16.0
org.apache.nifi - nifi-kafka-2-6-nar

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

Tube prediction info
ExecuteProcess 1.16.0
org.apache.nifi - nifi-standard-nar

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

Name success
Queued 0 (0 bytes)

Array to JSON documents
SplitJson 1.16.0
org.apache.nifi - nifi-standard-nar

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

Name split
Queued 0 (0 bytes)

PublishKafka_2_6
PublishKafka_2_6 1.16.0
org.apache.nifi - nifi-kafka-2-6-nar

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

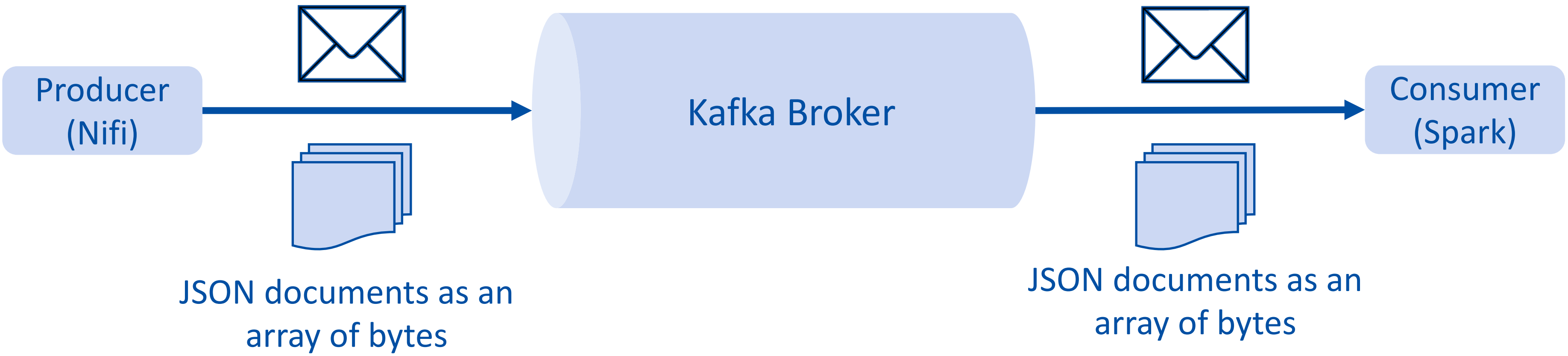
Ingesiton pipeline is the same for the two business requirements but running in parallel.

01. Execute Process retrieves the information from the API (Tube Predictions) and writes the output to a Flowfile.

02. Split JSON files into multiple and separate flowfiles.

03. The content of our flowfiles is sent to Kafka based on a delimiter.

DATA STORAGE


Producer (Nifi)
Kafka Broker
Consumer (Spark)
JSON documents as an array of bytes
JSON documents as an array of bytes

After the information in Nifi, we send the Json documents as messages to out storage layer splittingbuilt on Kafka.

Each document is represented as a line that will be consumed by Spark stream processing APIs

SPARK STREAM PROCESSING

Structured Streaming was used for making the analysis.

The following criteria was used for building the outputs:

Alarm Table

Our aim here is to create a report where we can see, every 2.5 minutes, if any train (vehicle id) has any delay (change in the **first** expectedArrival captured).

We will consider all the events that occurred the last 5 minutes (we want to have fresh data) and we will evaluate our logic every 2.5 minutes.

Features used:

- Vehicle ID
- StationName
- Delay: as we have a very rigorous schedule, we want to act when the delay is greater than 30 seconds.

SPARK STREAM PROCESSING

Frequency Table

Firstly, we need to stablish the time zone that spark will use, as the API has every timestamp in UTC and for this table, we will use the current timestamp.

We only keep those trains that are expected to arrive in the next ten minutes.

In the same way as before, we define a Watermark, in this case of five minutes, to not have big alterations of events that arrive late

The features we used are:

- lineid: the line that passes through that station (note that in the Victoria Station, there are three different lines)
- station Name: the proper name of the station (Victoria Station)
- direction: the direction of the train (can be inbound or outbound)
- count: the total trains that are expected to pass through that station in the next ten minutes.
- Frequency: the proper frequency expected in the next ten minutes.

PROJECT RESULTS

ALARM TABLE

Window	TImeStamp	VehicleId	StationName	Delay
{2022-10-29 14:05:00, 2022-10-29 14:06:00}	2022-10-29 14:05:36.310757	013	Epping Underground Station	3.0
{2022-10-29 14:05:00, 2022-10-29 14:06:00}	2022-10-29 14:05:36.310757	135	Hainault Undergound Station	0.98
{2022-10-29 14:05:00, 2022-10-29 14:06:00}	2022-10-29 14:05:36.310757	312	Loughton Underground Station	3.0
{2022-10-29 14:05:00, 2022-10-29 14:06:00}	2022-10-29 14:05:36.310757	312	**Epping Underground Station**	3.0
{2022-10-29 14:05:00, 2022-10-29 14:06:00	2022-10-29 14:05:36.310757	311	West Ruislip Station	1.0

1. This output shows the actual delay that a specific vehicle has for a particular station.
2. Every vehicle that has more than 30 seconds of delay raises an alarm.
3. Since it's streaming data, this is just a snapshot taken from our pipeline

FREQUENCY TABLE

Window	LineId	StationName	Direction	Count	Frequency
{2022-10-29 12:15:00, 2022-10-29 12:16:00}	victoria	Victoria Underground Station	Inbound	5	2.0
{2022-10-29 12:15:00, 2022-10-29 12:16:00}	circle	**Victoria Underground Station**	**Inbound**	3	3.0
{2022-10-29 12:15:00, 2022-10-29 12:16:00}	district	**Victoria Underground Station**	**Inbound**	2	5.0
{2022-10-29 12:15:00, 2022-10-29 12:16:00}	district	Victoria Underground Station	Outbound	1	10.0
{2022-10-29 12:15:00, 2022-10-29 12:16:00}	victoria	Victoria Underground Station	Outbound	4	3.0

1. Our frequency table shows the number of vehicles that are supposed to go through the station on a period of time.
2. Results are displayed for Victoria Underground Station and all lines that pass through this station
3. Only the vehicles that are supposed to pass within the next 10 minutes Will be featured in the output.

THANK YOU!

Transport for London

ie