# Background

The development of IOTs has increased the needs of real-time image data processing. The stream of images comes from various channels such as CCTV. This stream of images is a form of unstructured data that requires high processing power such as GPU to perform real-time image processing, and distributed processing using cluster that made of multiple nodes.

The company has a web application that collects images uploaded by customers and uses Apache Kafka stream to provide streaming of images into the system. Then, the software developers run some algorithms to process the images. The processed images are saved for a minimum of 7 days for archival purposes, and the result is written into the database for analysis and statistical reports.

This documentation writes the details of Section 3: System Design that explains the method to process the images and the technology stack used.

# Methodology

Apache Kafka is a high-throughput distributed messaging system (or streaming platform). It means it supports millions of messages in our modest hardware and scalable; there is no downtime. Apache Kafka allows us to build a real-time streaming data pipeline that reliably gets data between application. Apache Kafka builds, transforms and reacts to the stream of data. We have our data streams from our web application.

## Kafka Concepts

Apache Kafka runs on a cluster on one or more servers. The Kafka cluster stores stream of records in categories, it is called topics. Each record consists of a key, a value, and a timestamp. This record refers to the log of data structure. The database writes the change of events to a log, and derive the value of columns from that log. In Apache Kafka, messages are written to a topic that maintains this log from which subscribers can read and derive their representations of the data. You can think the message as an “activity” log.

## Four Core APIs

There are four core APIs:

* The Producer API allows an application to publish a stream of records to one or more Kafka topics.
* The Consumer API allows an application to subscribe to one or more topics and process the stream of records produced to them.
* The Streams API allows an application to act as a stream processor, consuming an input stream from one or more topics and producing an output stream to one or more output topics, effectively transforming the input streams to output streams.
* The Connector API allows building and running reusable producers or consumers that connect Kafka topics to existing applications or data systems. For example, a connector to a relational database might capture every change to a table.

**Producer and Consumer**

We can run the Kafka in a single node server (node) or a cluster mode with multiple nodes (Kafka broker). Producers are processes that publish data or a stream of records (push messages) into Kafka topics within the broker. A consumer pulls records off a or more Kafka topic and processes the streams of records produced to them. You can see it as Kafka has publisher, topics and subscribers. Messages are replicated across the cluster to provide support for multiple subscribers and balances the consumers in case of failures.

**Topic**

It is similar to a table in the database without any constraints. It has a name, the same way a table does have a name. You can have as many topics as you want in Kafka, just like as many tables as you want in a database. Topics in Kafka are always multi-subscriber, that means a topic can have zero, one or many consumers (subscribers) that subscribe to the data written to it. It can have partition topics and enable parallel consumption.

**Broker**

Kafka broker handles all requests from consumers and keeps data replicated within the cluster. There can be one or more brokers in a cluster. A good number to get started is 3 brokers for a cluster.

**Zookeeper**

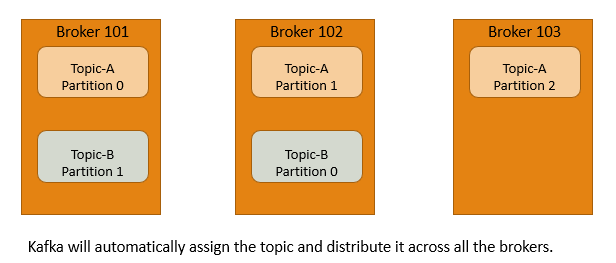
Apache Kafka uses Zookeeper to manage the brokers, topics and users. Zookeeper helps to keep the state of the cluster. Besides that, the Zookeeper manages the Leader and replicas.

**Apache Spark**

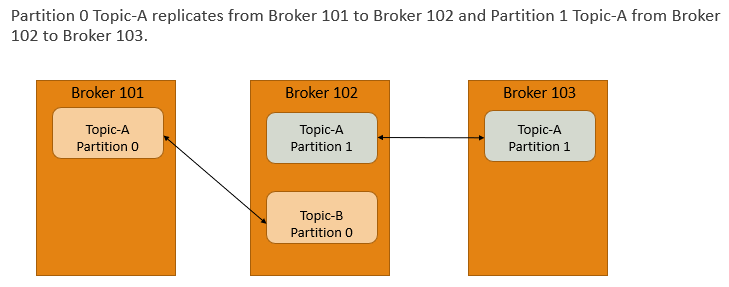
Apache Spark achieves high performance for both batch and streaming data, using a Directed Acyclic Graph (DAG) scheduler, a query optimizer, and a physical execution engine. Apache Spark can run on any cloud providers, and access data in any data sources.

# System Overview

When a customer uploads the images into the web application, it gets POST-ed. It can be stored into a database table and push the data into Kafka topics within the broker. In the proposed system architecture, it publishes into the Kafka cluster that runs on three brokers.

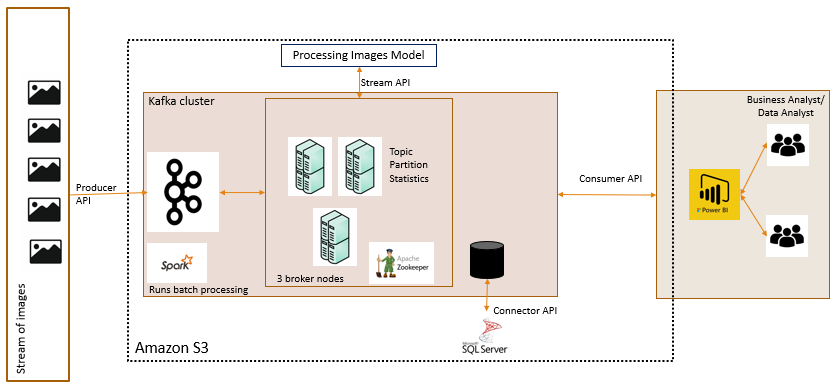


It is recommended that one broker has at least two or three partitions on each broker. It is required to indicate the number of partitions, and it can be changed later. In Kafka, there is a concept of leader for a partition. The golden rule is at any time; only one broker can be a leader for a given partition, and none or more servers that act as Followers. Also, it is recommended to have a minimum of two replication factor. The replication factor ensures that the data would not be lost when one of the brokers is down. Only the leader can receive and serve (read and write) the data for a partition. The other brokers are passive replicas (in-sync replica – ISR), and it will synchronize the data.



In the event of the Leader failing, one of the Followers will take on the role of the Leader. This process ensured the load balancing of the server and managed by the Zookeeper. The promotion of a leader and its management happens in the background and handles by the Kafka. Kafka uses Zookeeper to store offsets of messages consumed for a specific topic and partition by a specific Consumer Group. It is not possible to bypass Zookeeper and connect directly to the Kafka server. If for some reason, Zookeeper is down, the system cannot service any client request.

Based on the abovementioned, the diagram below is the proposed system overview of the image processing.



The Apache Spark access the stream of images from the web application on a 24-hour batch job. The Kafka Stream is going to get a stream of images and process the images and aggregates the data to provide some Business Intelligence’s insights and statistical data. The image processing model can be written in Python, for example. Then, the Kafka Connector streams the updates of the statistical data into the SQL Server database, for example. Lastly, the Consumer API allows applications such as Power BI to consume an input stream from one or more topics, and transform the input streams into output streams for the business analyst or data analyst. The Power BI application allows the business users to build reports and dashboard using the statistical data such as number and type of images processed, and by which customers. The reports and dashboard can be hosted separately in another report server. The Power BI application allows automatic refresh and it is suitable for this context that require real-time monitoring. Also, Power BI service allows us to set permission or access right to read or view the dashboard, and send alert to the business users if required.

# Advantages of Using Kafka

For real-time streaming image processing, a distributed messaging system based on Kafka enables each distributed node to access and process the images in parallel. This helps to improve its overall performance because Kafka supports dynamic allocation of topics partitions, and fault tolerance, preventing any nodes from being idle.

# Summary

There are many other concepts and technologies involve in real-time image processing in Kafka that can be explored and enhanced the proposed system. It depends on the business requirements, cost and budget, and available resources to leverage the right tools for each job.

# References

Stéphane Maarek, 2017, *How to use Apache Kafka to transform a batch pipeline into a real-time one*, <https://medium.com/@stephane.maarek/how-to-use-apache-kafka-to-transform-a-batch-pipeline-into-a-real-time-one-831b48a6ad85>, accessed on 20th June 2020.