**Kafka Overview**

* Kafka is a messaging system which transfers data between two applications.
* There are two types of messaging systems:
  + **Point to Point**
    - One or more consumers can consume the messages in the queue, but a particular message can be consumed by a maximum of one consumer only
  + **Publish- subscribe**
    - In the Publish-Subscribe system, message producers are called publishers and message consumers are called subscribers
    - One message can be read/consumed by multiple subscribers

**What is Kafka?**

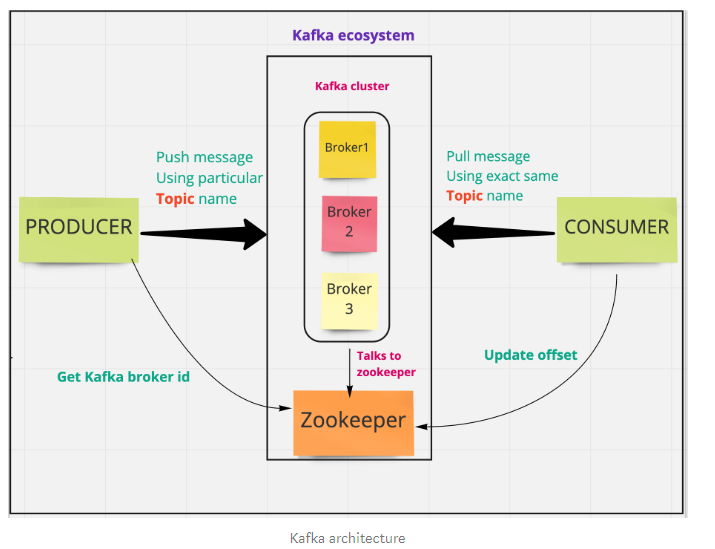
Apache Kafka is a distributed publish-subscribe messaging system and a robust queue that can handle a high volume of data and enables you to pass messages from one end-point to another. Kafka is suitable for both offline and online message consumption

* A real-life example is Dish TV, which publishes different channels like sports, movies, music, etc., and anyone can subscribe to their own set of channels and get them whenever their subscribed channels are available.

**Advantages of Kafka:**

* Kafka is designed to allow your apps to process records as they occur.
* One of the best advantages is Fault Tolerance.

**Kafka Architectural Diagram:**



**Following are components of Kafka:**

|  |  |
| --- | --- |
| Component | Description |
| Topics | A stream of messages belonging to a particular category is called a topic. Data is stored in topics. |
| Partition | Topics may have many partitions to have parallelism, so it can handle an arbitrary amount of data. |
| Partition offset | Each partitioned message has a unique sequence id called as offset. This helps to get the unique record/message per partition |
| Replicas of partition | Replicas are nothing but backups of a partition. Replicas are never read or write data. They are used to prevent data loss. |
| Brokers | Brokers are simple system responsible for maintaining the published data  Kafka cluster typically consists of multiple brokers to maintain load balance |
| Producers | Producers are the publisher of messages to one or more Kafka topics. Producers send data to Kafka brokers |
| Consumers | Consumers read data from brokers. Consumers subscribes to one or more topics and consume published messages by pulling data from the brokers. |
| Leader | Leader is the node responsible for all reads and writes for the given partition. Every partition has one server acting as a leader. |
| Follower | Node which follows leader instructions are called as follower. If the leader fails, one of the follower will automatically become the new leader. |

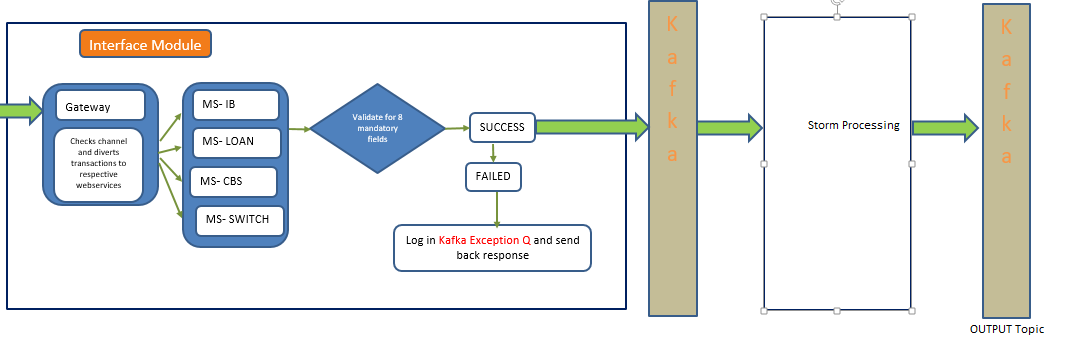
**Role of ZooKeeper — without zookeeper, Kafka cannot exists**

* Zookeeper serves as the coordination interface between the Kafka brokers and consumers.
* The Kafka servers share information via a Zookeeper cluster.
* All the critical information is stored in the Zookeeper so a failure of Kafka broker / Zookeeper does not affect the state of the Kafka cluster.
* The leader election between the Kafka brokers is also done by using Zookeeper in the event of leader failure.

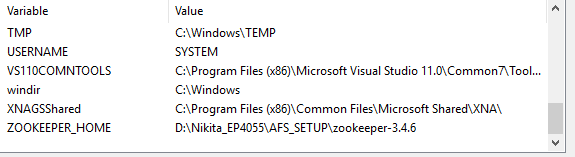
**Project Scope:**

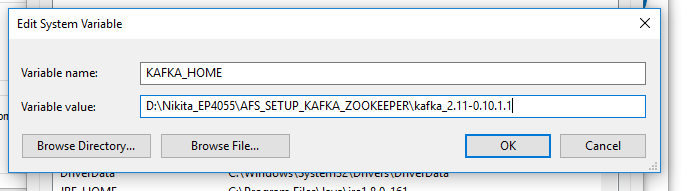
* In AFS we apply set of rules on the transaction and determine if this transaction is fraudulent or not.
* In order to capture the transaction we have written RESTful web service.
* On the received data some sanity check is performed and this data is then pushed to Kafka Topics.
* Depending on the data in the respective Kafka topics, rules get applied on the transactions and final output is sent to final Kafka topic i.e. OUTPUT.
* Alert generation engine works on the data present in OUTPUT Kafka topic.

**Block diagram depicting high level use of Kafka in AFS Application**



**Steps for Kafka Set up:**

1. Download Zookeeper
2. Set environmental variable corresponding to Zookeeper
3. Download the Apache Kafka Version 2.11
4. Set the environmental variables corresponding to Kafka



**Configuration settings:**

1. Go to Kafka folder-> **config** folder. In this make following changes in following files
   1. server.properties (**Change IP address accordingly**)
      1. listeners=PLAINTEXT://172.25.2.211:9092
      2. advertised.listeners=PLAINTEXT://172.25.2.211:9092
      3. zookeeper.connect=172.25.2.211:2181
   2. producer.properties
      1. bootstrap.servers=172.25.2.211:9092
   3. consumer.properties
      1. zookeeper.connect=172.25.2.211:2181
2. **Commands to start Kafka**:
   1. Start zookeeper

**d:**

**cd D:\Nikita\_EP4055\AFS\_SETUP\zookeeper-3.4.6\bin**

**zkserver**

1. **Start Kafka**

**d:**

**cd D:\Nikita\_EP4055\AFS\_SETUP\kafka\_2.11-0.10.1.1\_nikita**

**.\bin\windows\kafka-server-start.bat .\config\server.properties**

1. **Create a topic**
   1. Go to bin🡪windows-->

**.\kafka-topics.bat --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic TestHandover**

1. **List all topics in Kafka**
   1. Go to bin🡪windows-->

**.\kafka-topics.bat --list --zookeeper localhost:2181**

1. **Delete topic in Kafka**

**.\kafka-topics.bat --zookeeper localhost:2181 --delete --topic TestHandover**

1. **Produce record**

**d:**

**cd D:\Nikita\_EP4055\AFS\_SETUP\_KAFKA\_ZOOKEEPER\kafka\_2.11-0.10.1.1\bin\windows**

**kafka-console-producer.bat --broker-list 172.25.2.211:9092 --topic ATM**

1. **Consume records**

**d:**

**cd D:\Nikita\_EP4055\AFS\_SETUP\kafka\_2.11-0.10.1.1\_nikita\bin\windows**

**kafka-console-consumer.bat --bootstrap-server 172.25.2.211:9092 --topic ATM**

**Set up location**

**\\sezcpu063\Shared\Nikita Handover setup and documents**

|  |  |
| --- | --- |
| List of Java classes | Functionality |
| AFSGatewayAPIController.java | Starting point for the Restful web service In this we get input and depending on the channel code, we route this transaction to channel specific web services |
| AFSInLOSWSController.java | Checks for the sanity of the transaction, transaction is validated against 8 mandatory fields and on success and failure transaction is moved to channel specific pre queue and KafkaLogger queue respectively. |
| AFSLOSServlet.java | On successful transfer of data in pre queue we have channel specific servlet which continuosly listens to the pre queue It does following tasks 1.Polls records from pre queue 2. Adaptation of fields  3. Push data to channel specific topic 4. Save records in transaction table. |