# KAFKA Use Cases - 5

* KAFKA was originally buid for massive Log processing It remains messages until expiration and lets consumers pull messages at their own pace

- Log Processing and Analysis
- Data Streaming in recomendations
- System monetoring and alerting
- Change Data Capture (CDC)
- System migration

# Kafka Producer Config

```
@Configuration
@EnableKafka
public class KafkaProducerConfig {

  @value("${spring-kafka.producer.bootstrap-sever}")
   private String ssbkafkaProducerBootstrapSevers;

  @value("${Spring.kafka.producer.client-id}")
   private String producerClientId;

  @value("${Spring.kafka.producer.ssl.trust-store-location}")
   private string producerTrustStoreLocation;

  @value("Spring.kafka.producer.ssl.key-store-location}")
  private String producerKeyStoreLocation;

  @Bean
  Public ProducerFactory<String, String> KafkaProducerFactory() {
    Map<String, Object> kafkaProducerConfigProps = new HashMap<>();

   kafkaproducerconfigprops.put(producerconfig.BootStrap_servers_config,
                                ssbkafkaproducerbootstrapServers);
      ''      ''   .put(producerConfig.key_serializer_class_config,
                        StringSerializer.class);
      ''      ''.put(producerConfig.value-serializer_class_config,
                                StringSerializer.class);

     ''      '' put(client_ID_config, producerClientId);
```

```
@Bean
Public KafkaTemplate<String, String> kafkaTemplate(){
    return new KafkaTemplate<>(kafkaProducerFactory());
}
```

KAFKA Message Publisher

```
@Component
Public class KafkaMessagePublisher {

@value("${Spring.kafka.producer.topic.afw-response}")
Private String topicAfw;

@Autowired
 Private KafkaTemplate<String, String> kafkatemplate;

Public Static final Afwlogger logger = Afwlogger.getlogger(KafkaMessage
                                                          .class);

Public void SendResponse(AppianAfwResponseVO data){

  Logger.info("Sending Json data to topic='{}'");
    String message = null;
   object mapper responseMapper = new ObjectMapper();
    try{
        message = responseMapper.writeValueAsString(data);
        SendAcknowledgementMessage(topicAfw, message);

    }catch(JsonProcessingException e){
      logger.error(AfwConsts.KAFKA_Json_Process_Exception);
    }
```

```
Catch (TimeOutException e3)
  {
      Logger.error (AFW.Consts.KAFKA_response_TimeOut_Exception+" "
                                    + TopicAfw);
  }
}

Public Void SendAcknowledgeMessage (String topic, String JSonStr){

  ListenableFuture<SendResult<String, String>> future =
                                    KafkaTemplate.send (topic, JSonStr);
    future.addCallback (new ListenableFutureCallback
                        <SendResult<String, String>> (){

    @override
    public Void onSuccess (SendResult<String, String> result){
    // Message successfully sent

    LocalDateTime localDateTime = LocalDateTime.now();
    DateTimeFormatter format = DateTimeFormatter.ofPattern
                                    ("dd-mm-yyyy HH:mm:ss");
   String formatedTimeStamp = LocalDateTime.format (format);
   Logger.info (formatedTimeStamp + "message sent" + JSonStr);
   Logger.info (formatedTimeStamp + AFWConsts.KAFKA_success_Response
                                    + topic);
}

    @override
    Public Void onFailure (Throwable ex) {
    // Handle failure
      logger.Info (AFWConsts.KAFKA_FAIL_RESPONSE
                                    + topic);
}
}
```

# KAFKA Consumer

```
public class KafkaConsumer {

@value("${spring.kafka.producer.bootstrap-servers}")
  private String sspbkaproducerBootstrapServers;

@value("${spring.kafka.producer.client-id}")
 private String producerclientId;


@Retryable(maxattempts=3  backoff=@Backoff(delay=3000,
      multiplyer=1.5, maxDelay=2000), include=SocketTimeoutExption.class,
      exclude=NullPointerExephon.class)
@KafkaListener(topics="${consumer.topic.bar-reqs}",
                groupId="#{'${spring.kafka.consumer.group-id}'}",
                id="#{'${spring.kafka.consumer.client-id}'}")
```

Apache Kafka: is an open source Distributed ~~Streaming~~ Streaming platform that allows for the development of real time event driven applications

| | | |
|---|---|---|
| produce | Replicated | maintains order in occurred |
| Consume | Super fast | in order, |
| Cluster | high accuracy data | Fault tolerant |

* KAFKA is distributed and it runs as a ~~Server~~ Cluster and can spend multiple servers and even multiple data centers

Checkout Stream into events

decouple — Retention period.
Messaging
Location tracking
Data gathering → Collect Analytics.

Kafka built on 4 core APIs
* Producer API
* Topic → order list of events.
* Consumer API → subscribes one or more Topics
* Streams API → is needed to transform the Data

Connector API: enables API enables API to write connector when reusable

allows, with then one code is there:
all the developer has to do is configure it and and get all the data from datasource into the cluster

Streams API: Very powerful: It leverages producer and Consumer API
* It consumes from a topic or topics
* and then it will Analyse aggregate otherwise transform the data in real time
* And then Produce the real time streams to the topic either same topic or another