Apache Kafka and Spark Streaming

Nilay Kumar Bose
23rd March 2018

Agenda

- What is Kafka?
- Key Concepts
- Kafka Architecture
- Producer/ Consumer
- Consumer Group
- Use Cases
- Kafka In Big Data
- Spark
- Spark Concepts
- Spark Streaming
- Live Demo

What is Kafka?

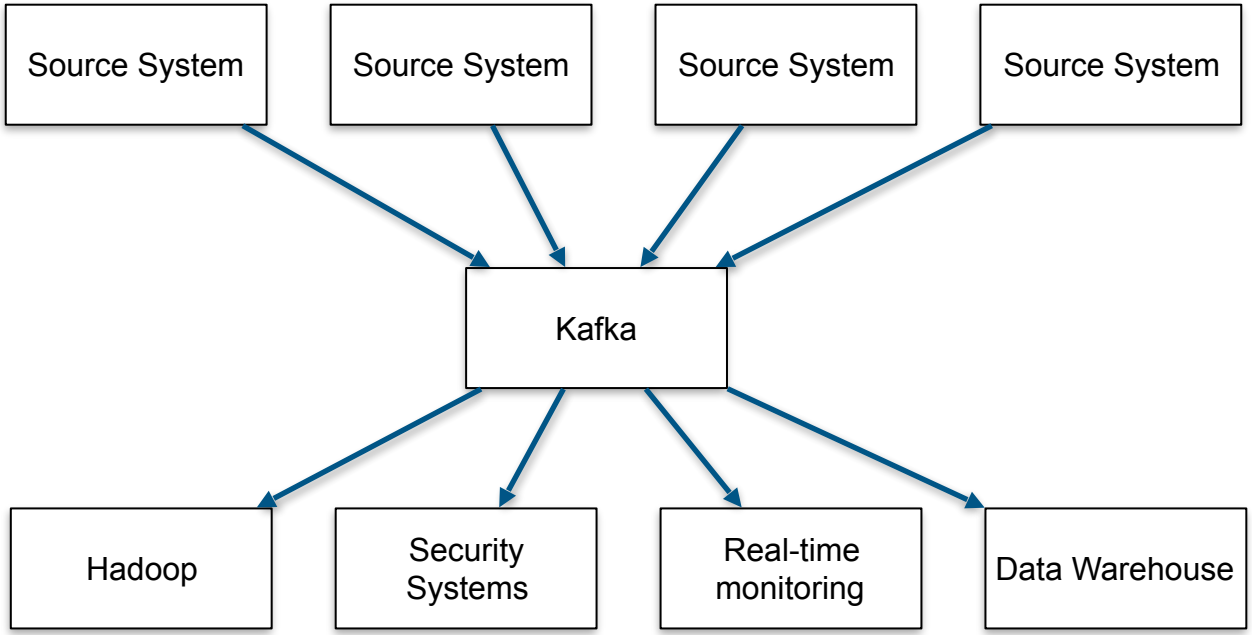
- ✓ Apache™ Kafka is a fast, scalable, durable, and fault-tolerant publish-subscribe messaging system.
- ✓ It was originally developed at LinkedIn Corporation and later on became a part of Apache project.
- ✓ Kafka is streaming platform as having three key capabilities:
  - ✓ Kafka publish and subscribe to streams of records similar to a message queue or enterprise messaging system.
  - ✓ Store streams of records in a fault-tolerant way.
  - ✓ Process streams of records as they occur.
- ✓ Kafka is a fast, scalable, distributed in nature by its design, partitioned and replicated commit log service.
- ✓ Designed to scale horizontally, by adding more commodity servers.
- ✓ Provides much higher throughput for both producer and consumer processes.
- ✓ Applicable to both batch and real-time use cases.
- ✓ Doesn’t support JMS, Java's message-oriented middleware API.

Why Kafka?

Producers

Brokers

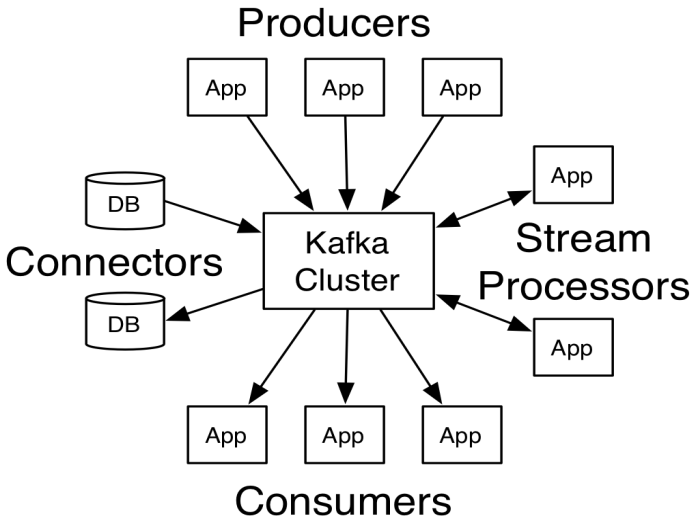
Consumers



Kafka Decouples Data Pipelines

Key Concepts

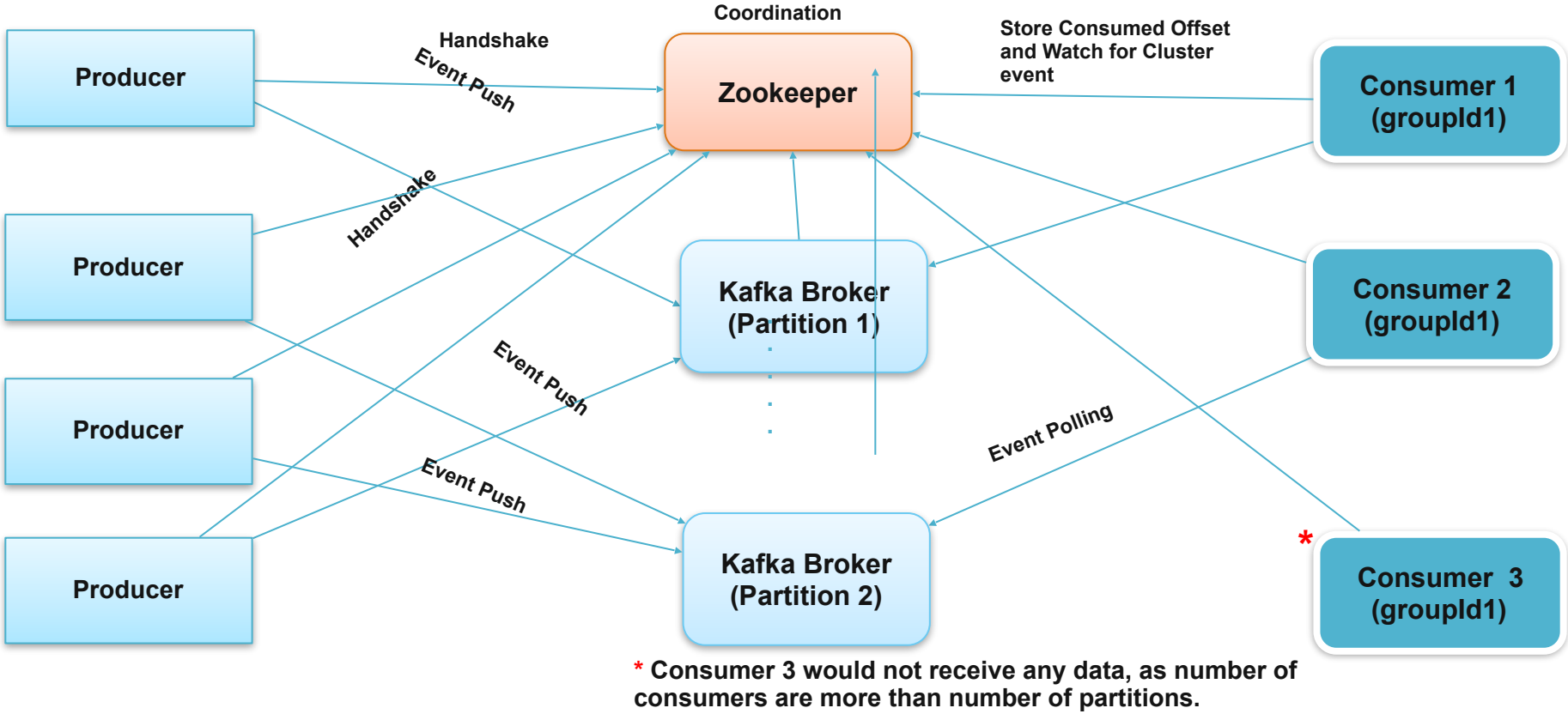
- ✓ Kafka is run as a Cluster on one or more servers.
- ✓ Each node in cluster is known as Broker.
- ✓ Topics is a category or feed name to which records are published.
- ✓ Each record consists of a key, a value, and a timestamp.
- ✓ Producer is a process that can publish a message to a topic.
- ✓ Consumer is a process that can subscribe to one or more topics and consume messages published to topics.
- ✓ 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.



Kafka Attractive Options over JMS

Feature	Descriptions
Scalability	Distributed system scales easily with no downtime
Durability	Persists messages on disk, and provides intra-cluster replication
Reliability	Replicates data, supports multiple subscribers, and automatically balances consumers in case of failure
Performance	High throughput for both publishing and subscribing, with disk structures that provide constant performance even with many terabytes of stored messages
Replay Capabilities	Consumers can get back the same message in case of need.

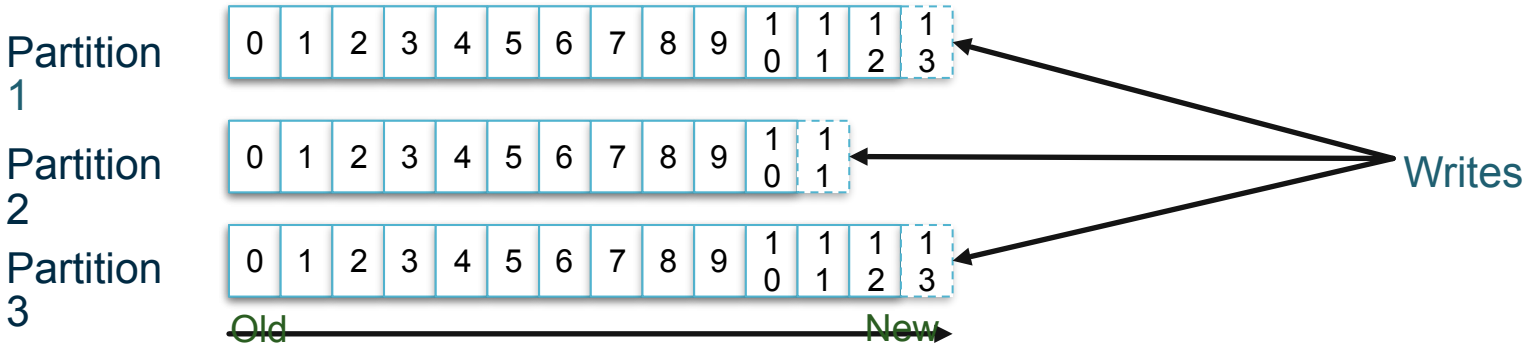
Kafka Architecture



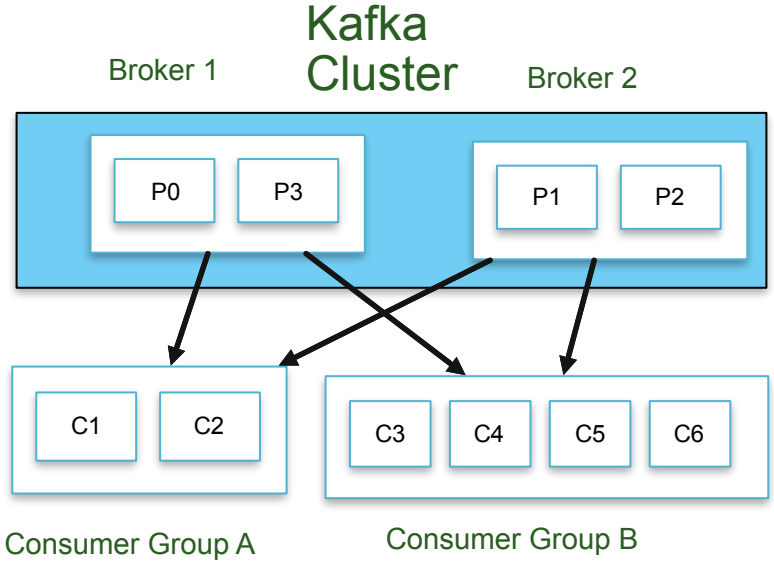
* Consumer 3 would not receive any data, as number of consumers are more than number of partitions.

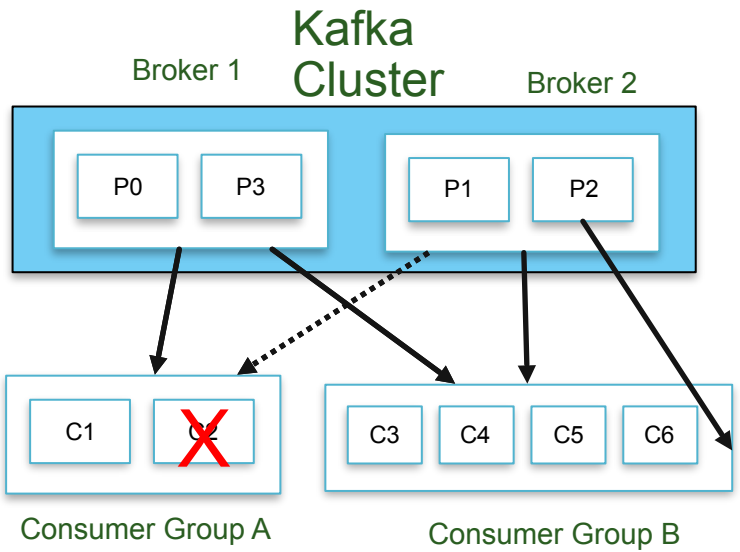
Topics - Partitions

- ✓ A Topic is a category or feed name to which records are published.
- ✓ Topics in Kafka are always multi-subscriber; that is, a topic can have zero, one, or many consumers that subscribe to the data written to it.
- ✓ Topics are broken up and ordered commit logs called Partitions.
- ✓ Each message in partition is given sequential id called Offset.
- ✓ Data is retained for a configurable period of time.
- ✓ Message ordering is guaranteed within a partition of a topic.
- ✓ Consumer instance sees the message in the order they are stored in the log.



Consumer Group





Consumer Offset Tracking and Recovery

- ✓ It is one of the unique feature of Kafka which support Consumer Offset Tracking and old message recovery. Kafka can track each consumer offset position of message published to it.When a consumer die and join back again it can request the previous offset where it left off and Kafka can successfully serve the data.

Topics – Replications

- ✓ Topics should be replicated.
- ✓ For a topic with replication factor N, we will tolerate up to N-1 server failures without losing any records committed to the log.
- ✓ Each partition in a topic has 1 leader and 0 or more replicas.
- ✓ A replica is deemed to be “in-sync” if
  - The replica can communicate with Zookeeper.
  - The replica is not “too far” behind the leader (configurable).
- ✓ The group of in-sync replicas for a partition is called the ISR (In-Sync Replicas).
- ✓ The Replication factor cannot be lowered.
- ✓ Durability can be configured with the producer configuration **request.required.acks**
  - **0** The producer never waits for an ack
  - **1** The producer gets an ack after the leader replica has received the data
  - **-1** The producer gets an ack after all ISRs receive the data
- ✓ Minimum available ISR can also be configured such that an error is returned if enough replicas are not available to replicate data

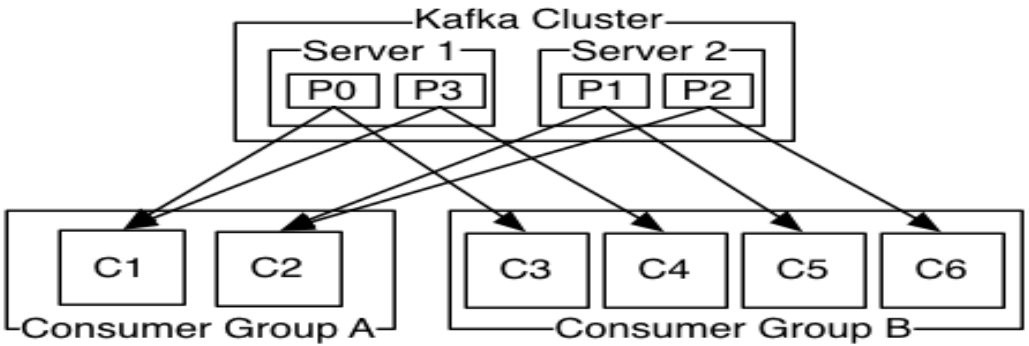
Producer

- ✓ Producers publish to a topic of their choosing (push)
- ✓ Load can be distributed.
  - Typically by “round-robin”
  - Can also do “semantic partitioning” based on a key in the message
- ✓ Brokers load balance by partition
- ✓ Can support asynchronous (less durable) sending

```
10      public static void main(String arg[]) throws Exception {
11          Properties props = new Properties();
12          props.put("bootstrap.servers", "localhost:9092");
13          props.put("acks", "all");
14          props.put("retries", 0);
15          props.put("batch.size", 2000);
16          props.put("linger.ms", 1);
17          props.put("buffer.memory", 1000000);
18          props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
19          props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");
20          KafkaProducer producer = null ;
21          try {
22              producer = new KafkaProducer<>(props);
23              int partition = 1 ;
24              String msg = "hello Kafka" ;
25              String key = "key" ;
26              producer.send(new ProducerRecord<String, String>("oltptrx", partition, key, msg));
27              producer.close() ;
28          }
29          finally {
30              try {producer.close();}catch(Exception _ignore) {}
31              producer = null ;
32          }
33      }
```

Consumer

- ✓ Consumers label themselves with a ***Consumer Group*** name, and each record published to a **Topic** is delivered to one consumer instance within each subscribing consumer group.
- ✓ Multiple Consumers can read from the same topic
- ✓ Messages stay on Kafka…they are not removed after they are consumed.
- ✓ If all the consumer instances have the same consumer group, then the records will effectively be load balanced over the consumer instances.
- ✓ If all the consumer instances have different consumer groups, then each record will be broadcast to all the consumer processes.
- ✓ If new instances join the group they will take over some partitions from other members of the group, if an instance dies, its partitions will be distributed to the remaining instances.



Consumer Monitoring

- ✓ Kafka provides an out of box script for monitoring

TOPIC	PARTITION	CURRENT-OFFSET	LOG-END-OFFSET	LAG	CONSUMER-ID	HOST	CLIENT-ID
oltptrx	0	3832	4001	169	consumer-1-74c906a1-137d-40d0-a154-b985b69577a8	/192.168.200.203	consumer-1
oltptrx	1	3255	3982	727	consumer-1-74c906a1-137d-40d0-a154-b985b69577a8	/192.168.200.203	consumer-1
oltptrx	2	3795	3962	167	consumer-1-74c906a1-137d-40d0-a154-b985b69577a8	/192.168.200.203	consumer-1
oltptrx	3	2192	3940	1748	consumer-1-74c906a1-137d-40d0-a154-b985b69577a8	/192.168.200.203	consumer-1
oltptrx	4	2186	3926	1740	consumer-1-74c906a1-137d-40d0-a154-b985b69577a8	/192.168.200.203	consumer-1

Use Cases

- ✓ Real-Time Stream Processing (combined with Spark Streaming)
- ✓ General purpose Message Bus
- ✓ Collecting User Activity Data
- ✓ Collecting Operational Metrics from applications, servers or devices
- ✓ Log Aggregation
- ✓ Change Data Capture
- ✓ Commit Log for distributed systems

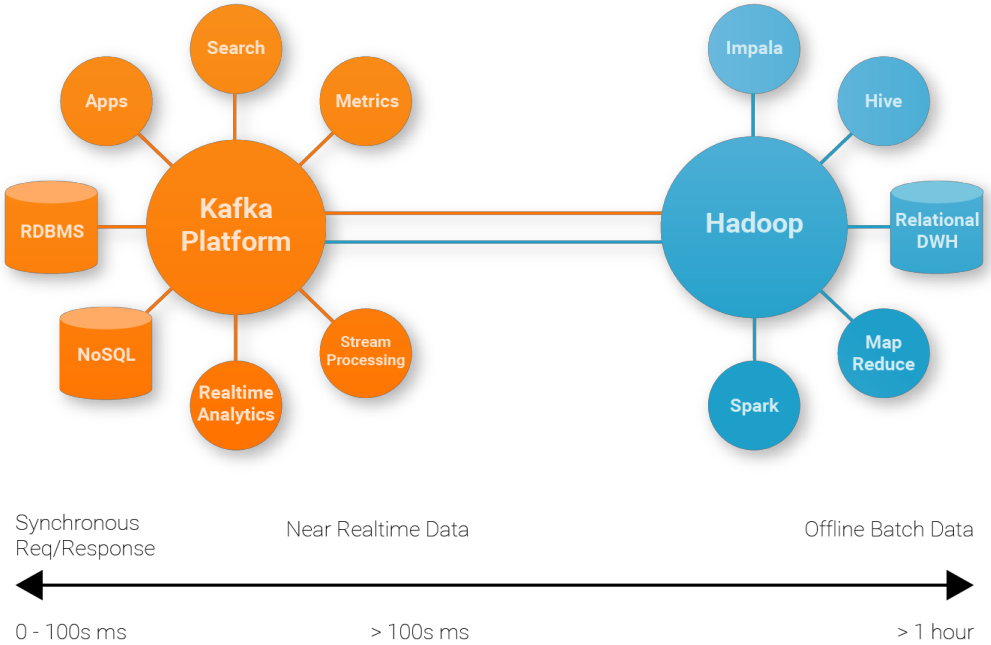
Kafka In Big Data

✓ Emergence of specialized distributed systems

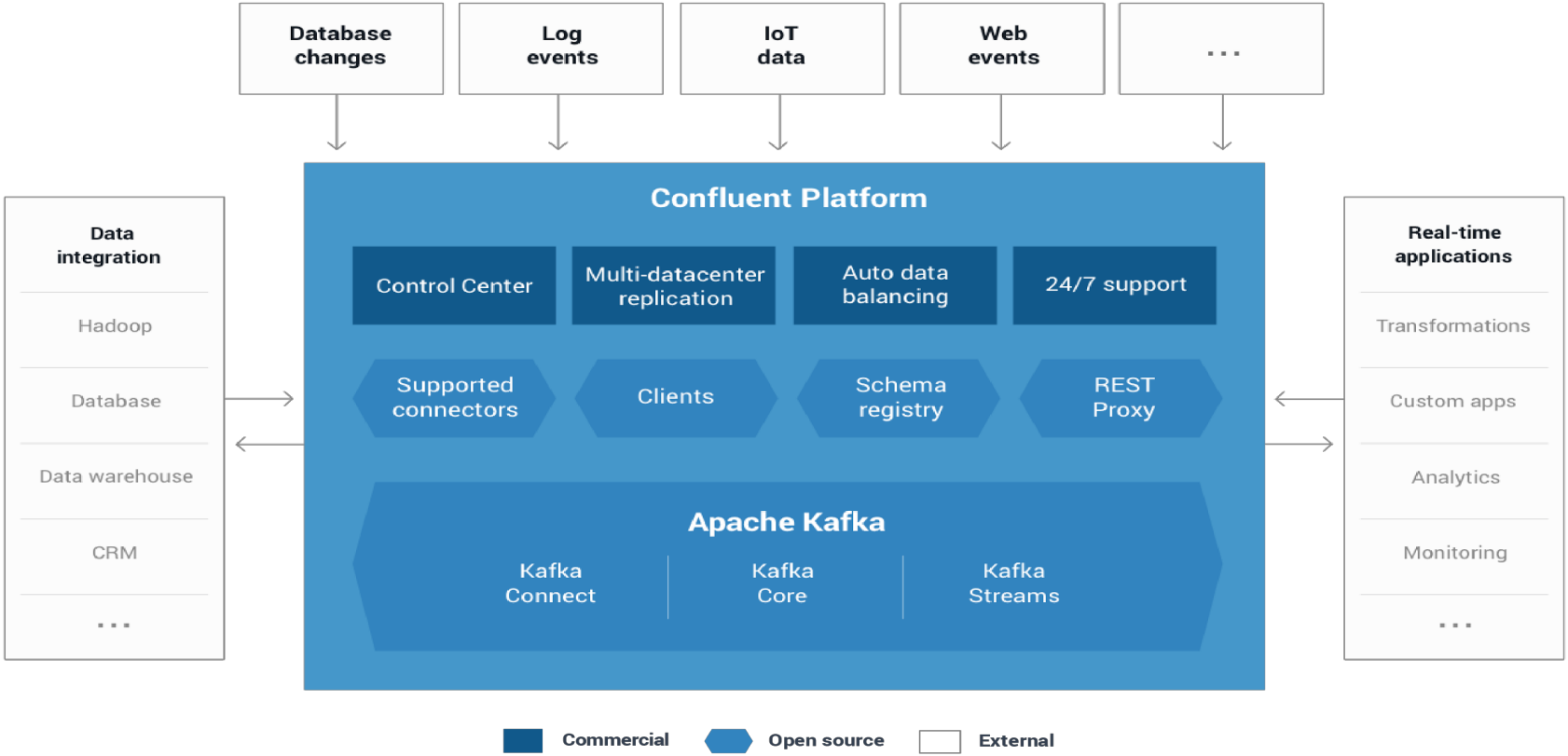
- HDFS, Map Reduce
- Key/value stores: Cassandra, MongoDB, HBase, etc.
- Search: Elastic search, Solr, etc.
- Stream processing: Storm, Spark streaming, Samza, etc.

✓ Feeding specialized systems

- Kafka used as central place to ingest all types of data in real time.
- It's designed as a distributed system and can store high volume of data on commodity hardware.
- It's designed as a multi-subscription system. The same published data set can be consumed multiple times.
- It persists data to disks and can deliver messages to both real-time and batch consumers at the same time without performance degradation.



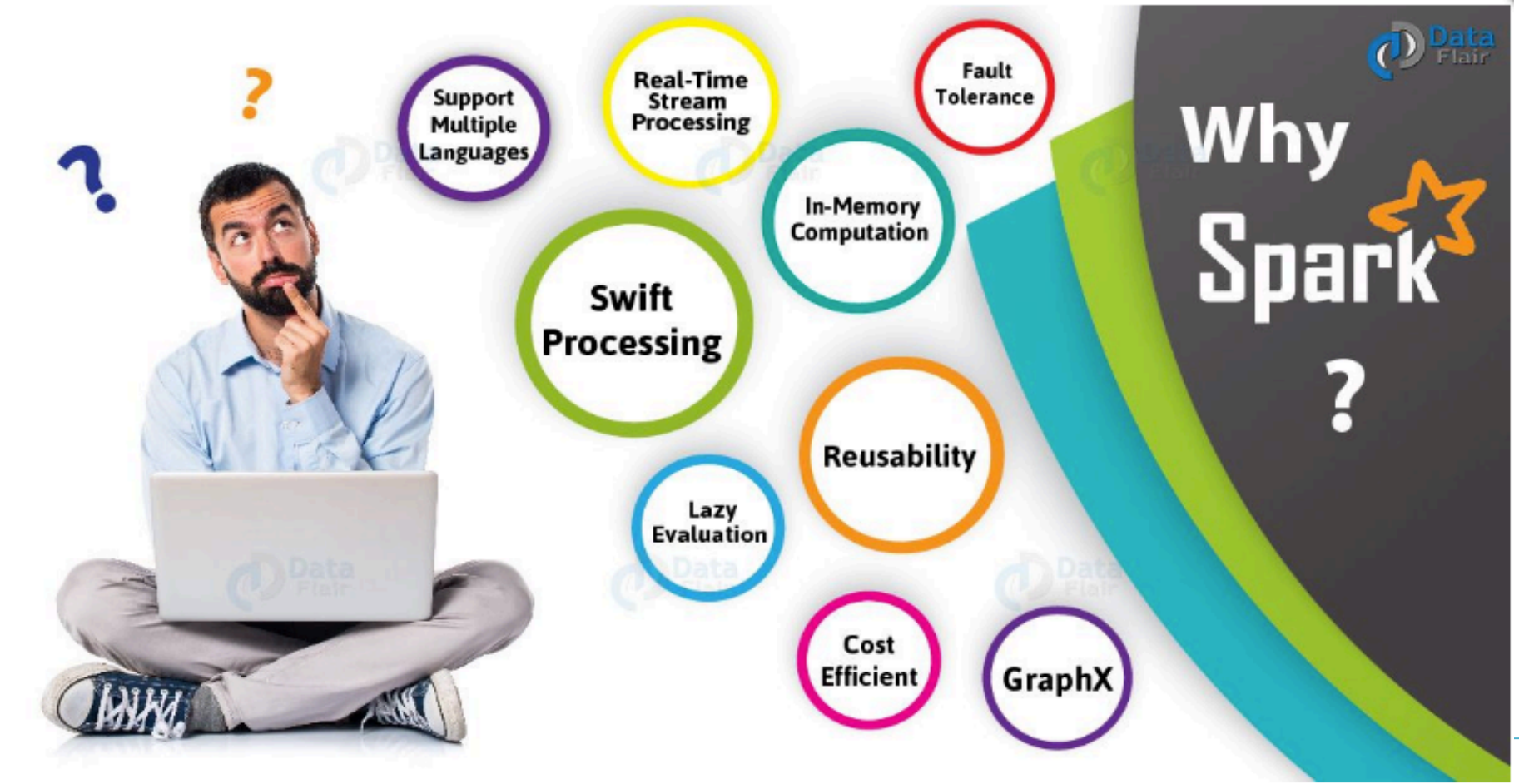
Confluent Platform



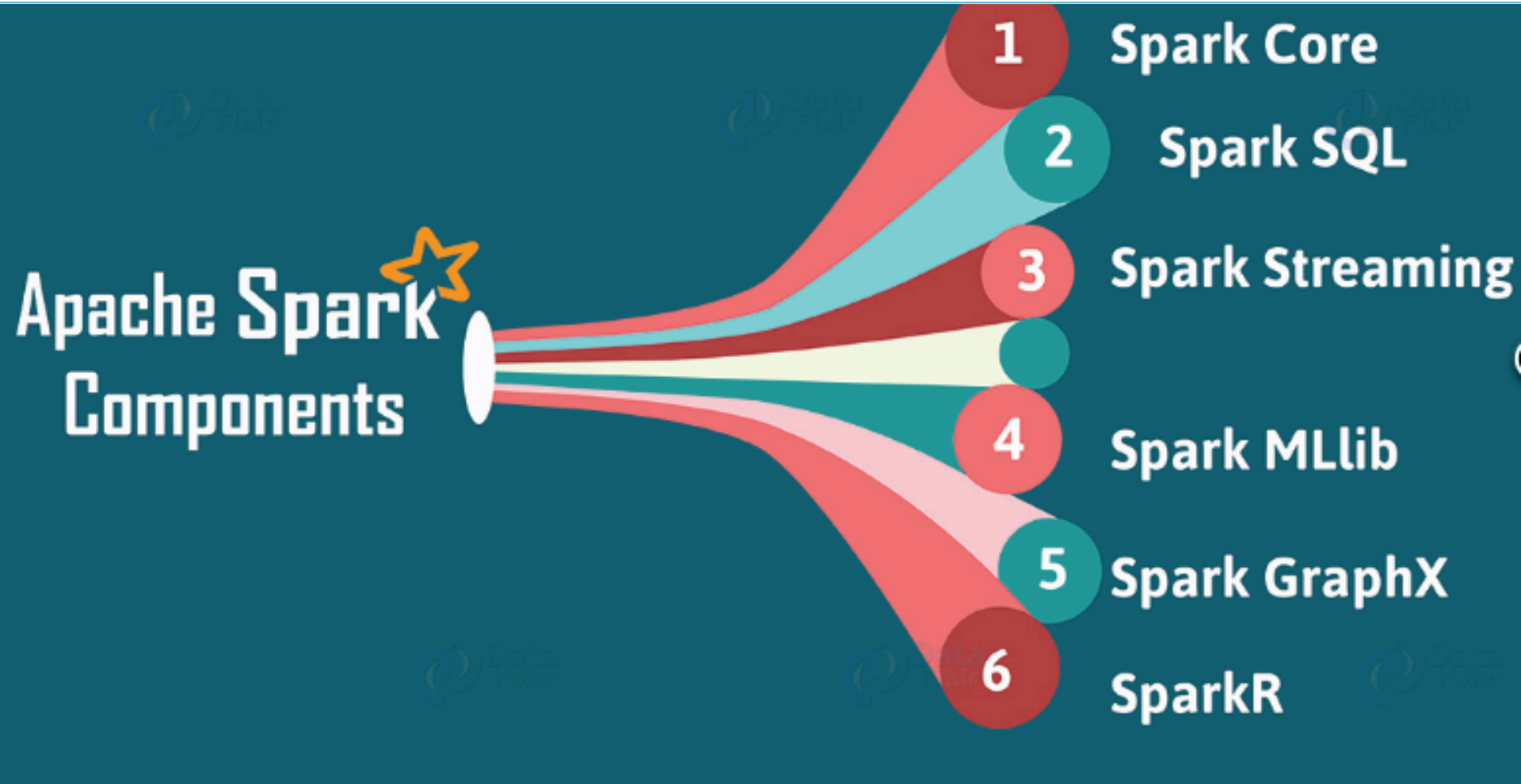
Apache Spark Overview

- Apache Spark™ is a fast and general engine for large-scale data processing
- Speed – Advanced DAG execution engine that supports acyclic data flow and in-memory computing.
- Ease Of Use - Write applications quickly in Java, Scala, Python, R
- Generality - Combine SQL, streaming, and complex analytics
- Run almost everywhere - Spark runs on Hadoop, Mesos, Kubernetes, standalone, or in the cloud
- Can Access data sources including HDFS, Cassandra, HBase, and S3

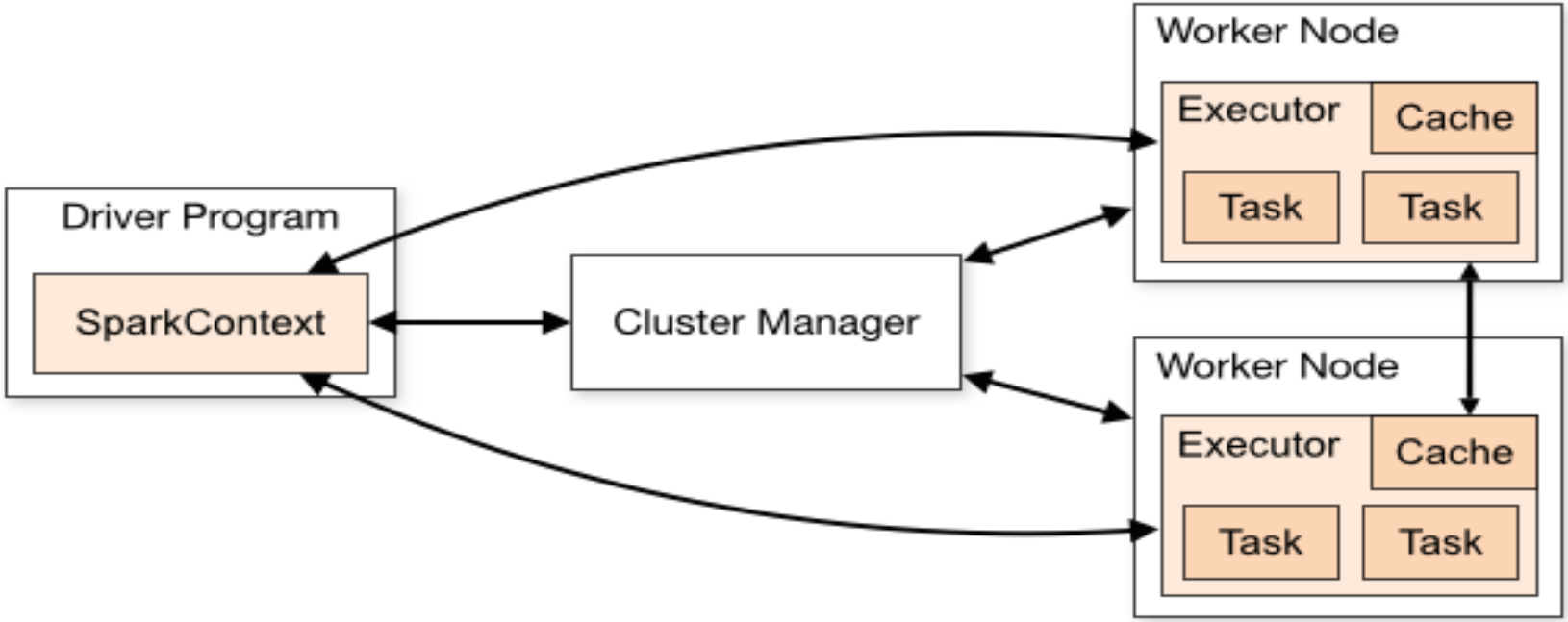
Why Apache Spark



Spark Components



Apache Spark Cluster Mode



* Cluster Manager Can be Stand alone, Kubernetes, Yarn, Mesos

Apache Spark RDD

- Resilient Distributed Data Set. Fundamental Data Structure in Spark.
- It is logically partitioned to be computed on different nodes of the cluster
- Partitioning of RDD give degree of parallelism in spark application

RDD Operations

- Transformations

```
JavaRDD<String> lines = sc.textFile("data.txt");
JavaRDD<Integer> lineLengths = lines.map(s -> s.length())
```

- Actions (Reduce Step)

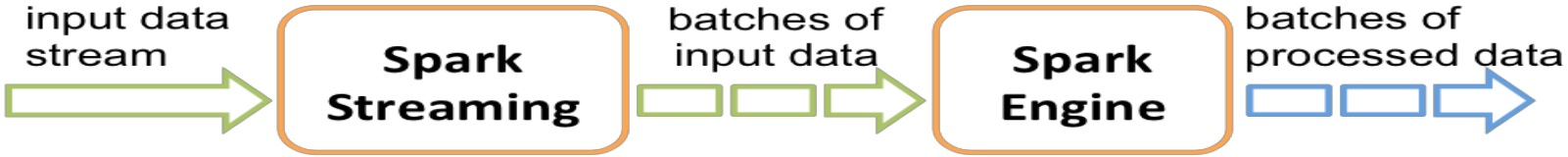
```
int totalLength = lineLengths.reduce((a, b) -> a + b);
```

Apache Spark Streaming

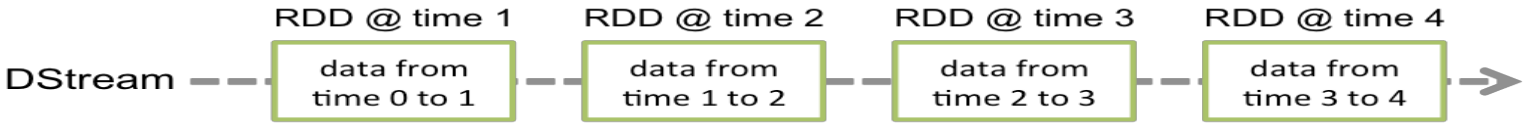
- A data stream is an unbounded sequence of data arriving continuously.
- Streaming divides continuously flowing input data into discrete units for further processing.
- Stream processing is low latency processing and analyzing of streaming data.
- Streaming API is extension of core Spark API that enables scalable, high-throughput, fault-tolerant stream processing of live data streams.

How Apache Spark Streaming Works

- Spark divides the stream data as Mini Batches (DStreams)

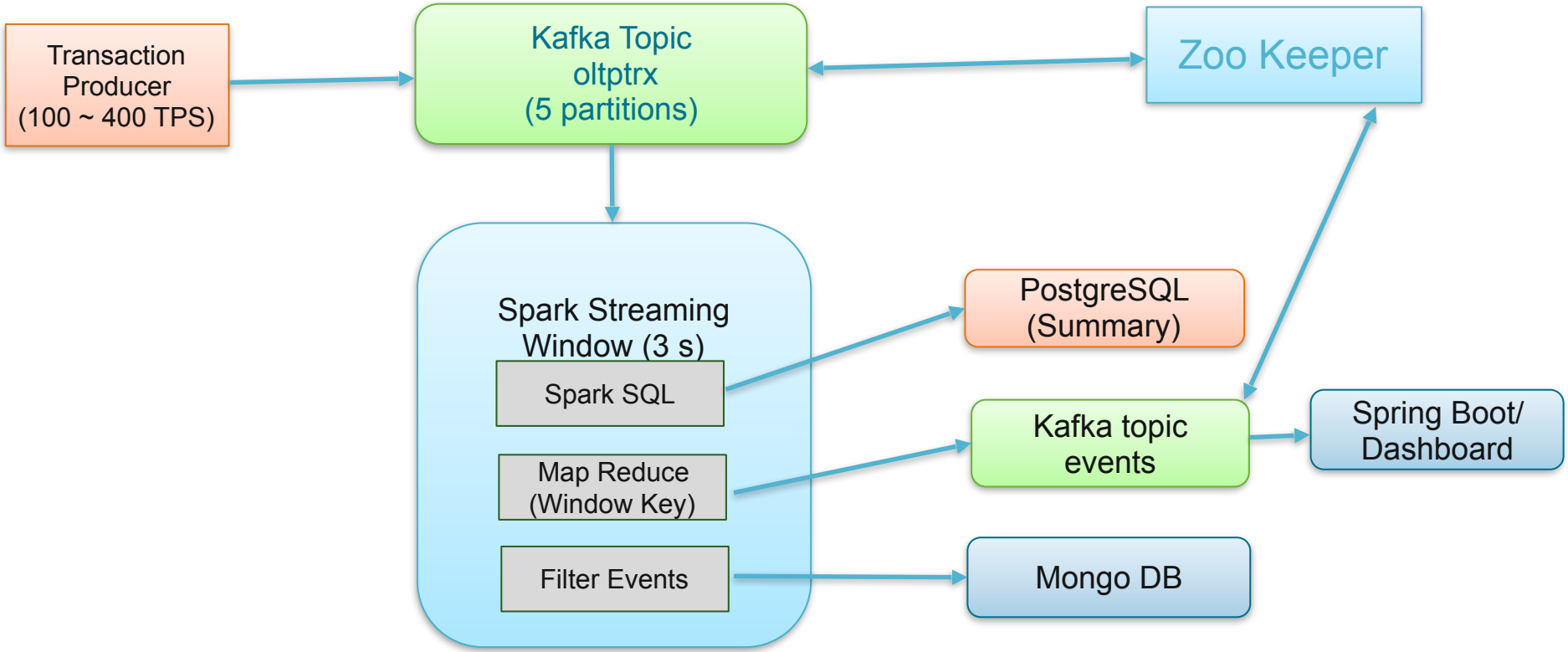


- DStreams are sequence of RDD. Data in a batch (RDD) is controlled by Spark window duration.



- One spark window generates one RDD.

Spark – Kafka Streaming Demo



Apache Spark – Kafka Streaming

- Spark can ingest data from Kafka. Direct Spark – Kafka streaming API can match number of Kafka partitions in Topic to RDD partitions

```
37  public class SparkStreamTest {
38
39      public static void main(String[] args) throws Exception {
40          Map<String, Object> kafkaParams = new HashMap<>();
41          kafkaParams.put("bootstrap.servers", "localhost:9092");
42          kafkaParams.put("key.deserializer", StringDeserializer.class);
43          kafkaParams.put("value.deserializer", StringDeserializer.class);
44          kafkaParams.put("group.id", "oltpgrp");
45          kafkaParams.put("auto.offset.reset", "latest");
46          kafkaParams.put("enable.auto.commit", true);
47          Collection<String> topics = Arrays.asList("oltptrx");
48
49          SparkConf sparkConf = new SparkConf().setAppName("oltpdemo");
50          JavaStreamingContext jssc = new JavaStreamingContext(sparkConf, new Duration(3000));
51
52          /********************
53           * setup Spark direct steam with Kafka - RDD partition will match the kafka
54           * topic partition
55           ********************/
56          JavaInputDStream<ConsumerRecord<String, String>> stream = KafkaUtils.createDirectStream(jssc,
57                  LocationStrategies.PreferConsistent(),
58                  ConsumerStrategies.<String, String>Subscribe(topics, kafkaParams));
59
60          /*********
61           * Stream transformation from Kafka JSON message to OltpTransaction Objects
62           ********************/
63          JavaDStream<OltpTransaction> trxdstream = stream.map(record -> {
64              String msg = record.value();
65              Gson gson = new GsonBuilder().create();
66              return gson.fromJson(msg, OltpTransaction.class);
67          });
68
```

Apache Spark – Kafka Streaming – In Memory SQL

- Spark SQL can be run on In Memory RDD Data and persisted in any Data Store

```
68
69          /*****************************************
70           * Summarize Operation                   *
71           *****************************************/
72          trxdstream.foreachRDD((rdd, time) -> {
73              SparkSession spark = SparkSession.builder().config(jssc.sparkContext().getConf()).getOrCreate();
74              Dataset<Row> dataSet = spark.createDataFrame(rdd, OltpTransaction.class);
75
76              // Creates a temporary view using the DataFrame
77              dataSet.createOrReplaceTempView("trxlist");
78
79              // Do word count on table using SQL and print it
80              Dataset<Row> summarydata = spark.sql(
81                      "select mop, adInd, org, mcc, trxDt, count(*) as count from trxlist group by mop, adInd, org, mcc, trxDt");
82              summarydata.show();
83
84               summarydata.write()
85               .format("jdbc")
86               .option("url", "jdbc:postgresql://127.0.0.1:5432/postgres")
87               .option("dbtable", "oltp.TRX_SUMMARY")
88               .option("user", "postgres")
89               .option("password", "bose").mode(SaveMode.Append)
90               .save();
91          });
```

Apache Spark – Kafka Streaming – In Memory SQL

Data Saved in PostgreSQL

	mop text	adInd text	org text	mcc text	count bigint	trxDt bigint
1	Amex	D	Others	Grocery	47	201803230019
2	Discover	A	Costco	Retail	46	201803230020
3	MC	A	Target	Grocery	60	201803230020
4	Discover	D	Others	Others	97	201803230020
5	Amex	D	Walmart	Others	2	201803230022
6	Discover	D	Walmart	Grocery	2	201803230022
7	MC	A	Target	Retail	24	201803230014
8	Amex	D	Costco	Retail	15	201803230016
9	Discover	D	Others	Others	82	201803230019
10	MC	D	Walmart	Retail	70	201803230020
11	Amex	D	Others	Grocery	45	201803230021
12	Amex	A	Others	Retail	10	201803230022
13	Discover	A	Target	Retail	26	201803230016
14	Amex	A	Walmart	Others	189	201803230017

Apache Spark – Kafka Streaming – Filter Operation

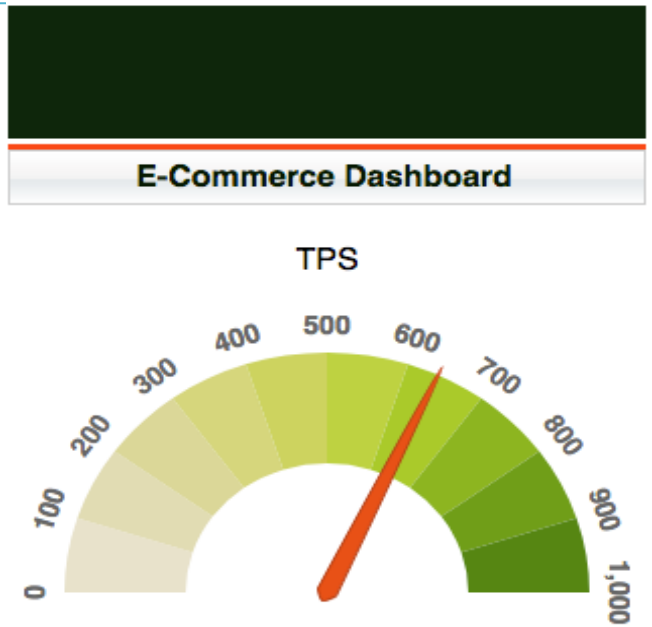
- Specific events can be filtered and persisted in a Data Store

```
137         /**********************************************************************
138          * Fraud Events based on Rules
139          **********************************************************************/
140         JavaDStream<OltpTransaction> frddstream = trxdstream.filter(oltptrx -> {
141             boolean test = false ;
142             if("D".equals(oltptrx.getAdInd())) {
143                 oltptrx.setResponseReason("Declined - Frd response code");
144                 test = true ;
145             }
146             else if("A".equals(oltptrx.getAdInd()) && oltptrx.getAmt() <= 99) {
147                 test = true ;
148                 oltptrx.setResponseReason("Approved - Trx Amount <= 99 cents");
149             }
150             return test ;
151         });
152 
153         // iterate the RDD partition and write in Mongo for reporting
154         frddstream.foreachRDD((rdd, time) -> {
155             rdd.foreachPartition(collection -> {
156                 List<Document> doclist = new ArrayList<Document>();
157                 collection.forEachRemaining(
158                         oltptrx -> doclist.add(new Document("_id",oltptrx.getPaymentId()). //payment id as unique key in mongo db
159                                         append("trxDt", oltptrx.getTrxDt()).
160                                         append("acctnum", oltptrx.getAcctnum()).
161                                         append("org", oltptrx.getOrg()).
162                                         append("mop", oltptrx.getMop()).
163                                         append("adInd", oltptrx.getAdInd()).
164                                         append("trxAmt", oltptrx.getAmt()).
165                                         append("reason", oltptrx.getResponseReason())
166                                         ));
167                 OltpUtil.writeEventsToMongo("oltp", "frdevents", doclist);
168             });
169         });
```

Apache Spark – Kafka Streaming – Reduce By Key

```
97          JavaPairDStream<String, TrxEvent> eventstream = trxdstream.mapToPair(record -> {
98              TrxEvent event = new TrxEvent();
99              event.getAttributes().put("mcc_" + record.getMcc(), new Integer(1));
100             event.getAttributes().put("mop_" + record.getMop(), new Integer(1));
101             event.getAttributes().put("ad_" + record.getAdInd(), new Integer(1));
102             event.getAttributes().put("org_" + record.getOrg(), new Integer(1));
103             return new Tuple2<String, TrxEvent>("wts", event);
104         });
105
106         // Reduce (summarize) all events
107         JavaPairDStream<String, TrxEvent> reventstream = eventstream
108                 .reduceByKey((event1, event2) -> event1.reduce(event2), 1);
109
110         // Save it in Kafka Event Topic for Dashboard UI
111         reventstream.foreachRDD((rdd, time) -> {
112             rdd.collect().stream().forEach(tuple -> OltpUtil.kafkaSend("events", 0, tuple._1(),
113                     new GsonBuilder().create().toJson(tuple._2(), TrxEvent.class)));
114         });
115
116         /*
117          * Example JSON event Message produced as a part of reduced operation
118             {
119             "attributes":
120                 {   "mcc_Grocery": 18,
121                     "org_Walmart": 37,
122                     "org_Others": 27,
123                     "ad_D": 19,
124                     "mcc_Retail": 23,
125                     "mop_Visa": 18,
126                     "ad_A": 65,
127                     "mop_Discover": 15,
128                     "org_Target": 7,
129                     "org_Costco": 13,
130                     "mcc_Others": 43,
131                     "mop_MC": 31,
132                     "mop_Amex": 20
133                 }
134             }
```

Spring Boot/ D3 Dashboard



Email:

nilaykumar.bose@vantiv.com


intuit
ebay
NETFLIX
Pinterest
Goldman Sachs
CISCO
airbnb
Adobe
box
salesforce
yelp
Square
dish NETWORK
Cerner
WIKIPEDIA
The Free Encyclopedia
PayPal
verizon

Reference & Links

- ✓ https://kafka.apache.org/intro
- ✓ http://www.javaworld.com/article/3060078/big-data/big-data-messaging-with-kafka-part-1.html
- ✓ https://www.confluent.io/blog/
- ✓ http://spark.apache.org/docs/latest/index.html