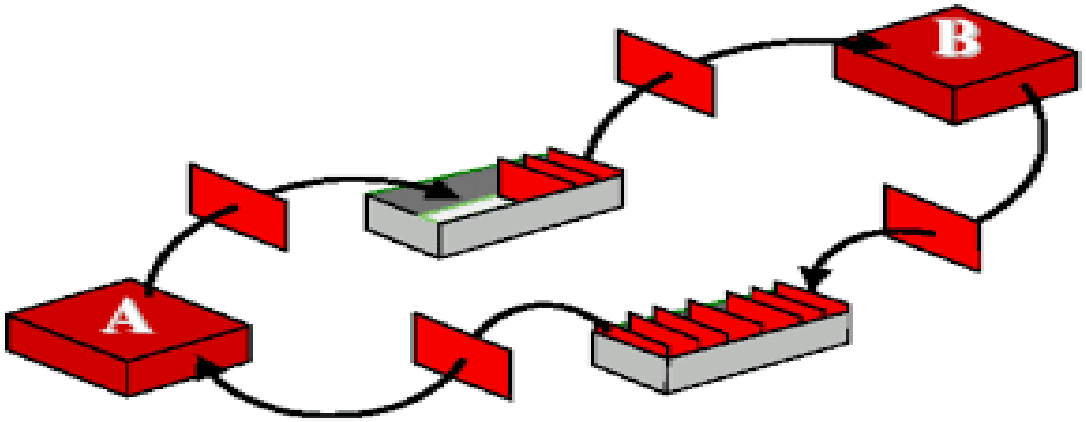
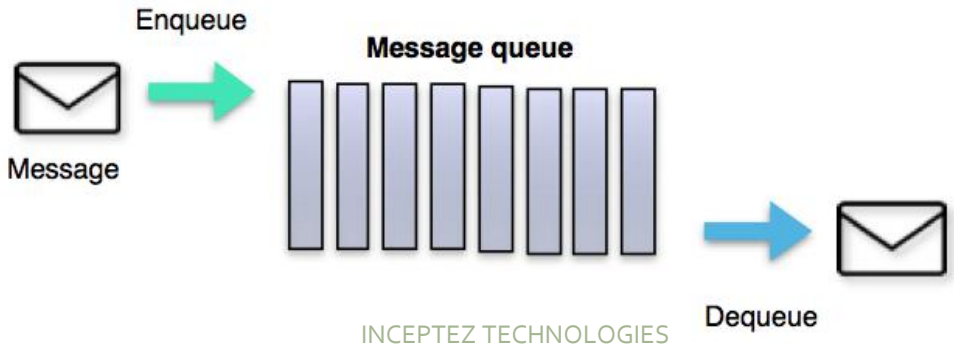

kafka

Introduction to Message Queue

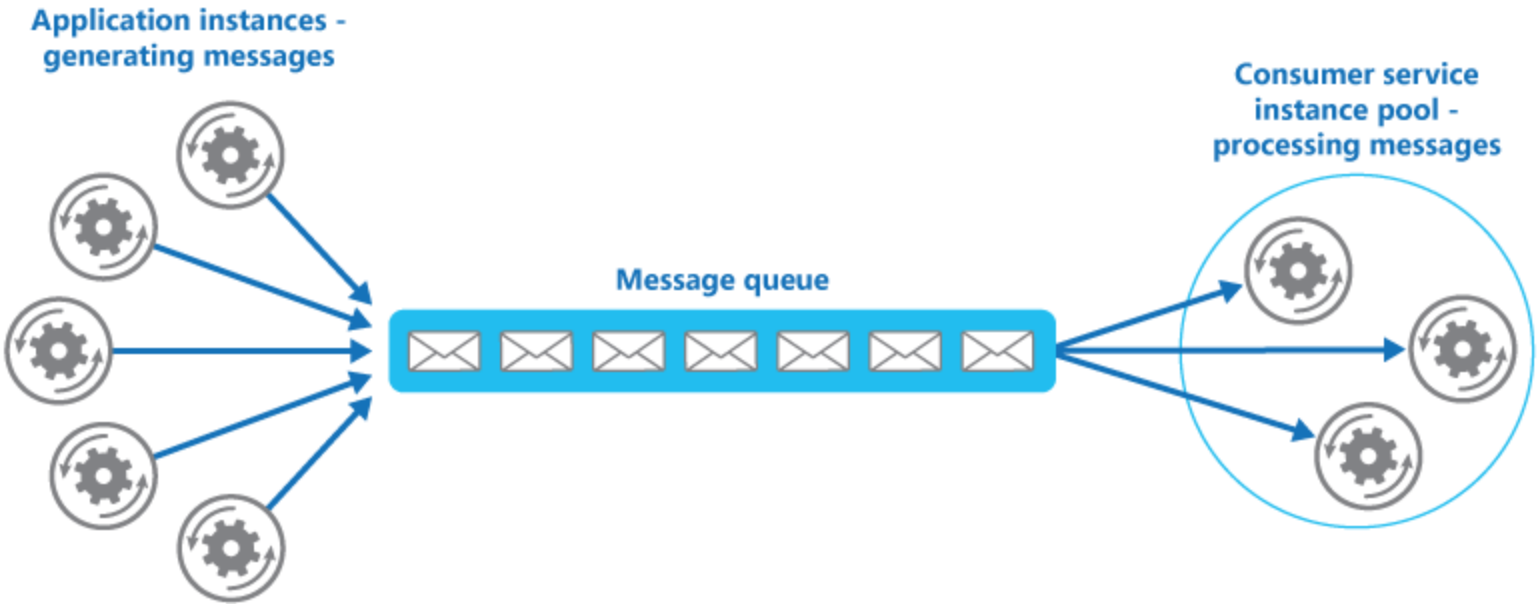
Message queuing allows applications to communicate by sending messages to each other. The message queues provides a **temporary message storage** when the destination program is **busy or not connected**.



A **queue** is a line of things waiting to be handled - in sequential order starting at the beginning of the line. A message queue is a queue of messages sent between applications. It includes a sequence of work objects that are waiting to be processed.



Need of Distributed Messaging Queue



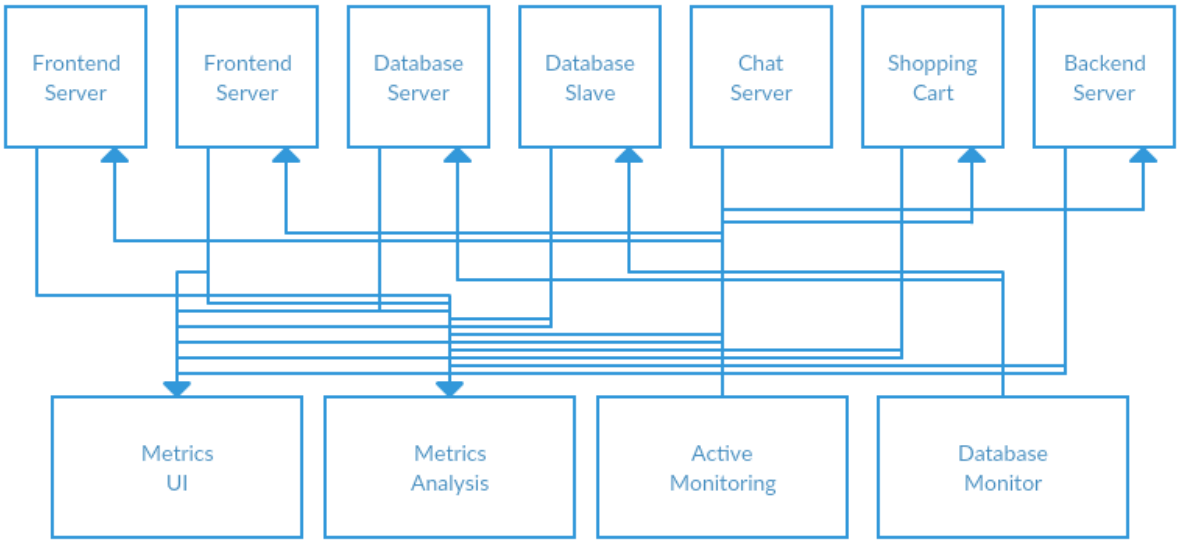
- **Distributed systems** provides a very easy way to **scale** out.
- It offers **high throughput** for both publishing and subscribing.
- It **persist** messages on disk and thus can be used for batched consumption such as **ETL**, in addition to **real time** applications.

Kafka Introduction

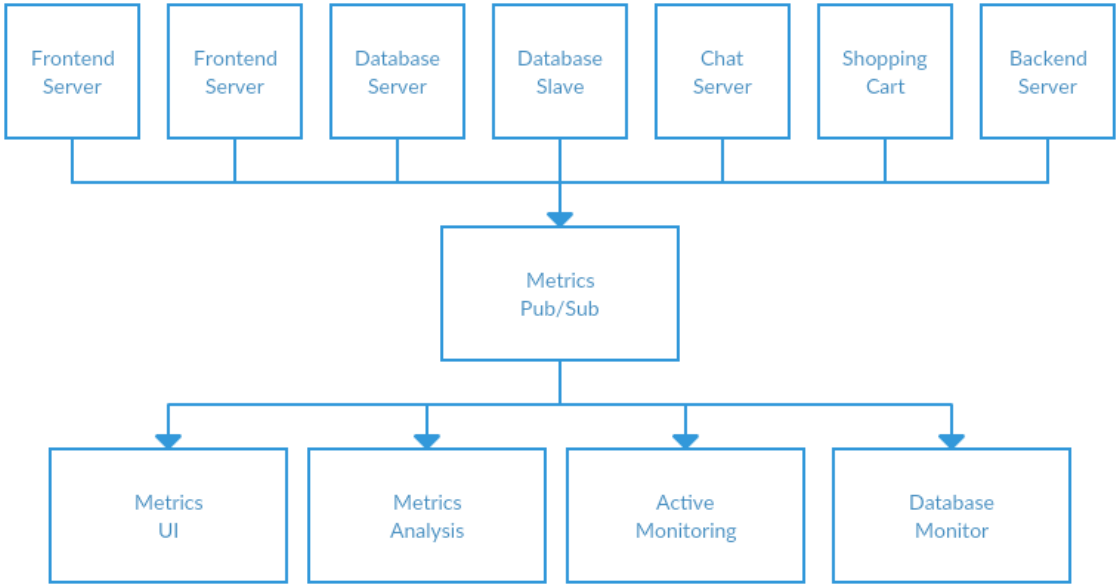
- Kafka is a leading general purpose ***publish-subscribe distributed messaging system***, which offers strong durability, scalability and fault-tolerance support.
- Designed for processing of ***real time activity*** stream data such as logs, metrics collections.
- Written in ***Scala***
- An apache project initially developed at ***LinkedIn*** at the year 2007.
- It is not specifically designed for Hadoop rather Hadoop ecosystem is just be one of its possible consumers.

Why we need Kafka

Direct connections



Publish/Subscribe system



Why we need Kafka

Kafka is Highly scalable very ***easy to add large number of consumers*** without affecting performance and without down time. That's because Kafka does not track which messages in the topic have been consumed by consumers. It simply keeps all messages in the topic within a configurable period.

Kafka handle spike of the events more efficiently. This is where Kakfa truly shines because it acts as a "***shock absorber***" between the producers and consumers. Kafka can handle events at 100k+ per second rate coming from producers. Because Kafka consumers pull data from the topic, different consumers can consume the messages at different pace.

Kafka also supports different consumption model. You can have one consumer processing the messages ***at real-time*** and another consumer processing the messages in ***batch mode***.

Message durability is high in Kafka – *Persists* the messages/events for the specified time period

Integration support for Kafka with other frameworks are high So mostly likely it needs to integrate with other event processing frameworks such as Apache ***Storm, Spark, Nifi, Flume*** etc to complete the job.

Applications of Kafka

Messaging

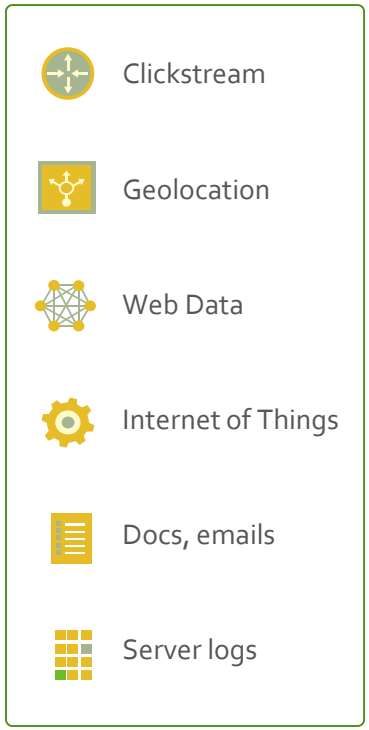
Kafka works well as a replacement for a more traditional message broker. Message brokers are used for a variety of reasons (to decouple processing from data producers, to buffer unprocessed messages, etc). In comparison to most messaging systems Kafka has better throughput, built-in partitioning, replication, and fault-tolerance which makes it a good solution for large scale message processing applications. In our experience messaging uses are often comparatively low-throughput, but may require low end-to-end latency and often depend on the strong durability guarantees Kafka provides.

Website Activity Tracking

The original use case for Kafka was to be able to rebuild a user activity tracking pipeline as a set of real-time publish-subscribe feeds. This means site activity (page views, searches, or other actions users may take) is published to central topics with one topic per activity type.

Metrics

Kafka is often used for operation monitoring data pipelines. This involves aggregating statistics from distributed applications to produce centralized feeds of operational data.



Applications of Kafka

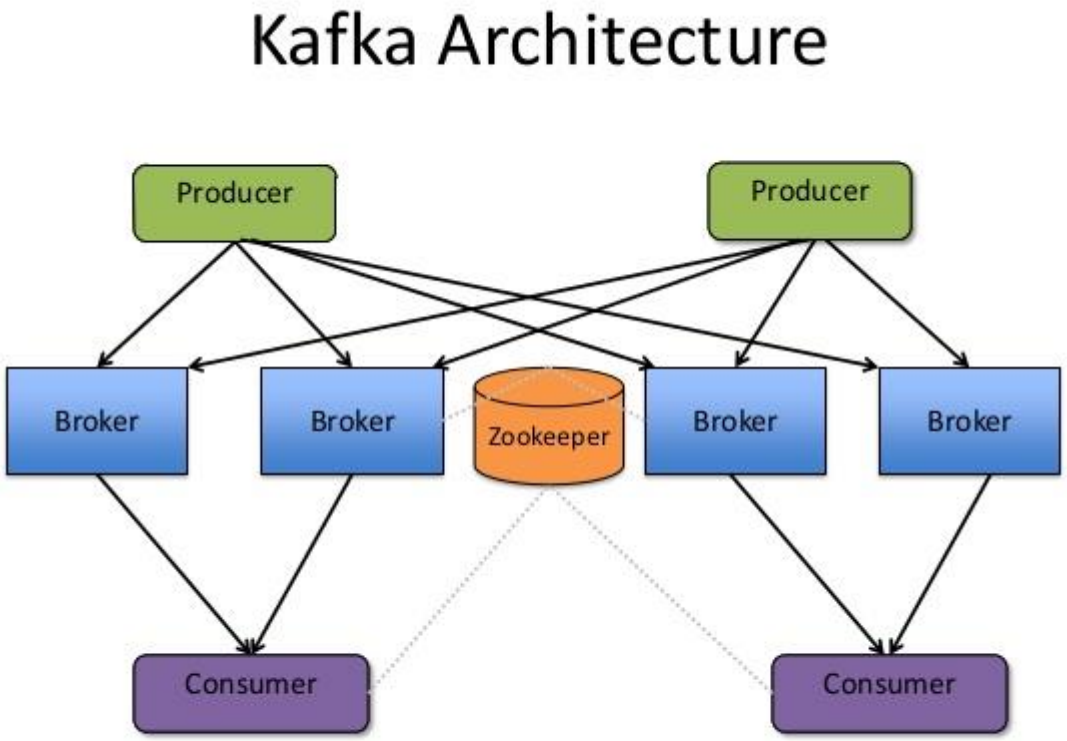
Log Aggregation

Many people use Kafka as a replacement for a log aggregation solution. Log aggregation typically collects ***physical log files off servers and puts them in a central place*** (a file server or HDFS perhaps) for processing. Kafka abstracts away the details of files and gives a cleaner abstraction of log or event data as a stream of messages. This allows for lower-latency processing and easier support for multiple data sources and distributed data consumption. In comparison to log-centric systems like Scribe or Flume, Kafka offers equally good performance, stronger durability guarantees due to replication, and much lower end-to-end latency.

Stream Processing

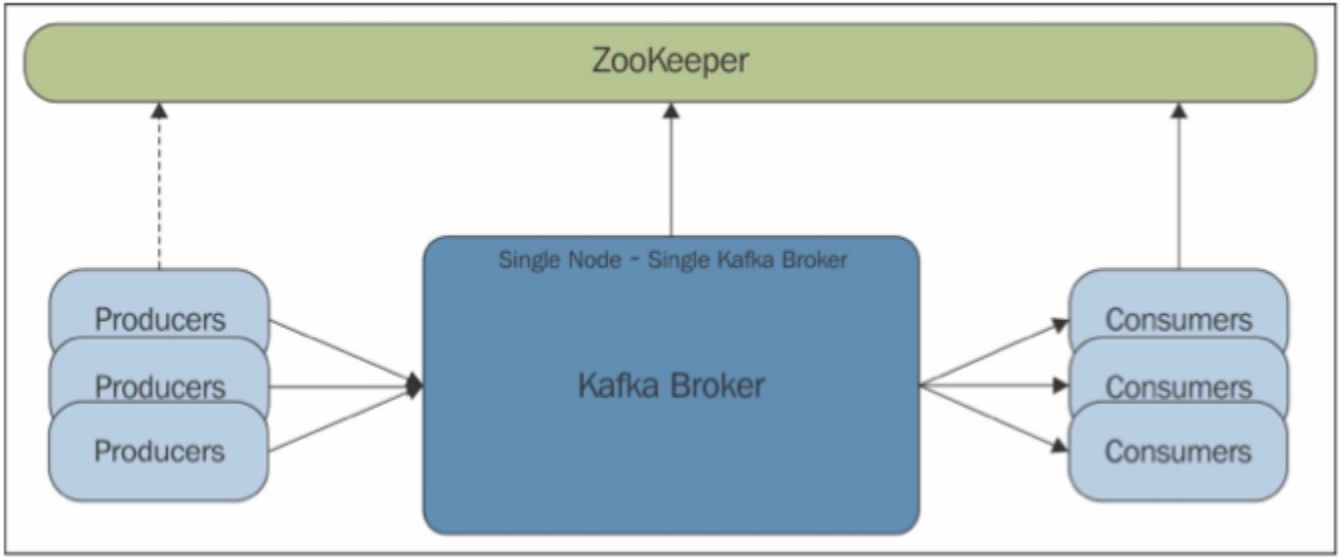
Many users end up doing stage-wise processing of data where data is consumed from topics of raw data and then aggregated, enriched, or otherwise transformed into new Kafka topics for further consumption. For example a processing flow for article recommendation might crawl article content from RSS feeds and publish it to an "articles" topic; further processing might help normalize or deduplicate this content to a topic of cleaned article content; a final stage might attempt to match this content to users. This creates a graph of real-time data flow out of the individual topics. The Spark/Storm framework is one popular way for implementing some of these transformations.

Kafka Architecture



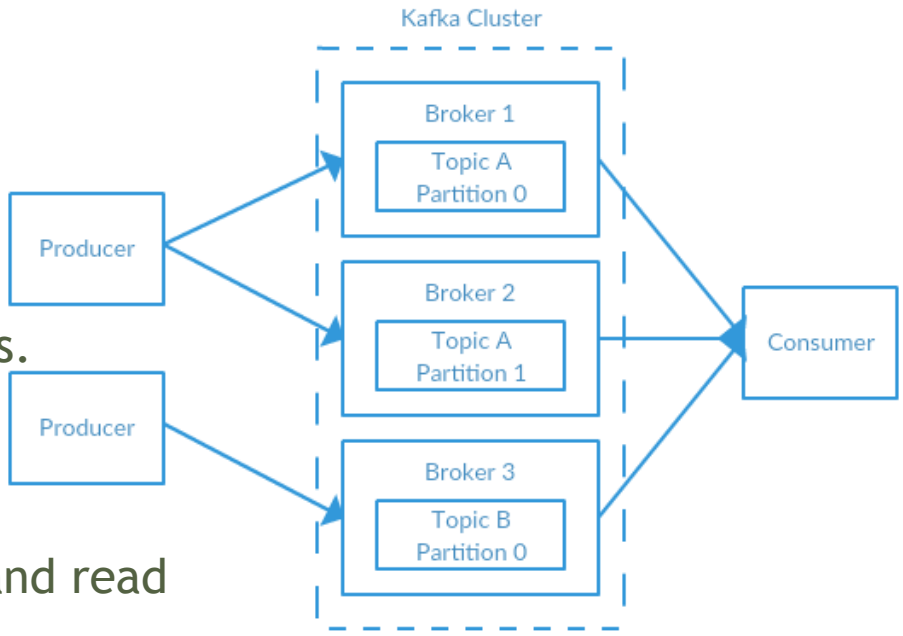
Components

- Messages
- Batches
- Schema
- Producer
- Consumer
- Broker
- Kafka Cluster
- Topic
- Partition
- Replicas



- **Messages** : The unit of data within Kafka with optional byte header, eg. Record in a table.
- **Batches** : Collection of messages eg. RecordSet
- **Schema** : Additional structure be imposed on the message eg. JSON, XML, Avro.

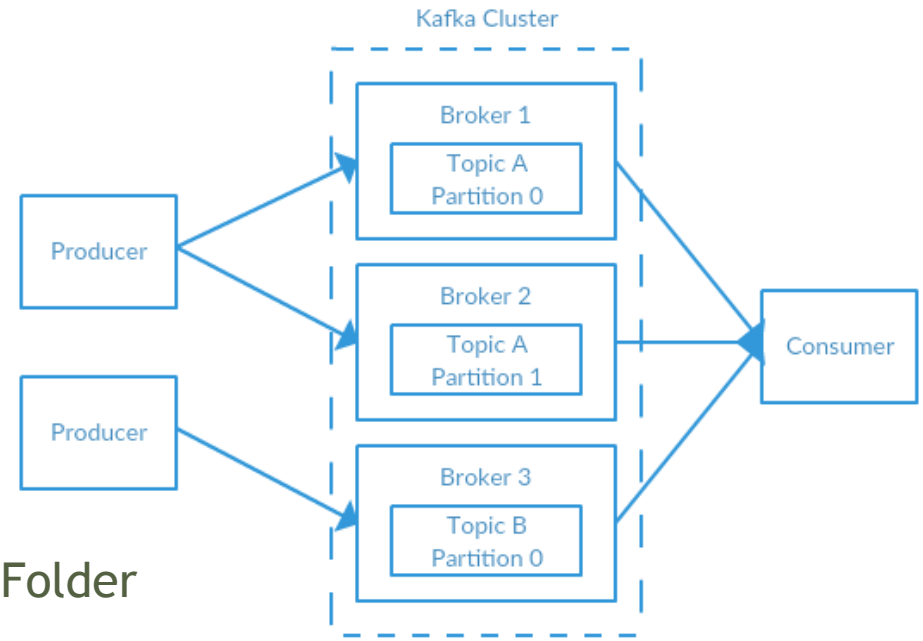
Components (Contd)



- **Producer**
  - Creates Messages.
  - Maintains messagekey to deliver messages to the particular partitions.
- **Consumer**
  - Consume Messages.
  - Subscribe to one or more topics and read messages in order.
  - Keep track of offset of last consumed messages.
- **Broker**
  - A single Kafka server
  - The broker receives messages from producers, assigns offsets to them, and commits the messages to storage on disk.
  - The broker serves consumer responding with the messages based on the partitions.

Components (Contd)

- **Kafka Cluster**
  - Group of Brokers
  - One Broker will act as a cluster Controller and assigned leader for the partition.



- **Topic**
  - Category of messages eg. Table/Folder
  - Spread across all Brokers/nodes.

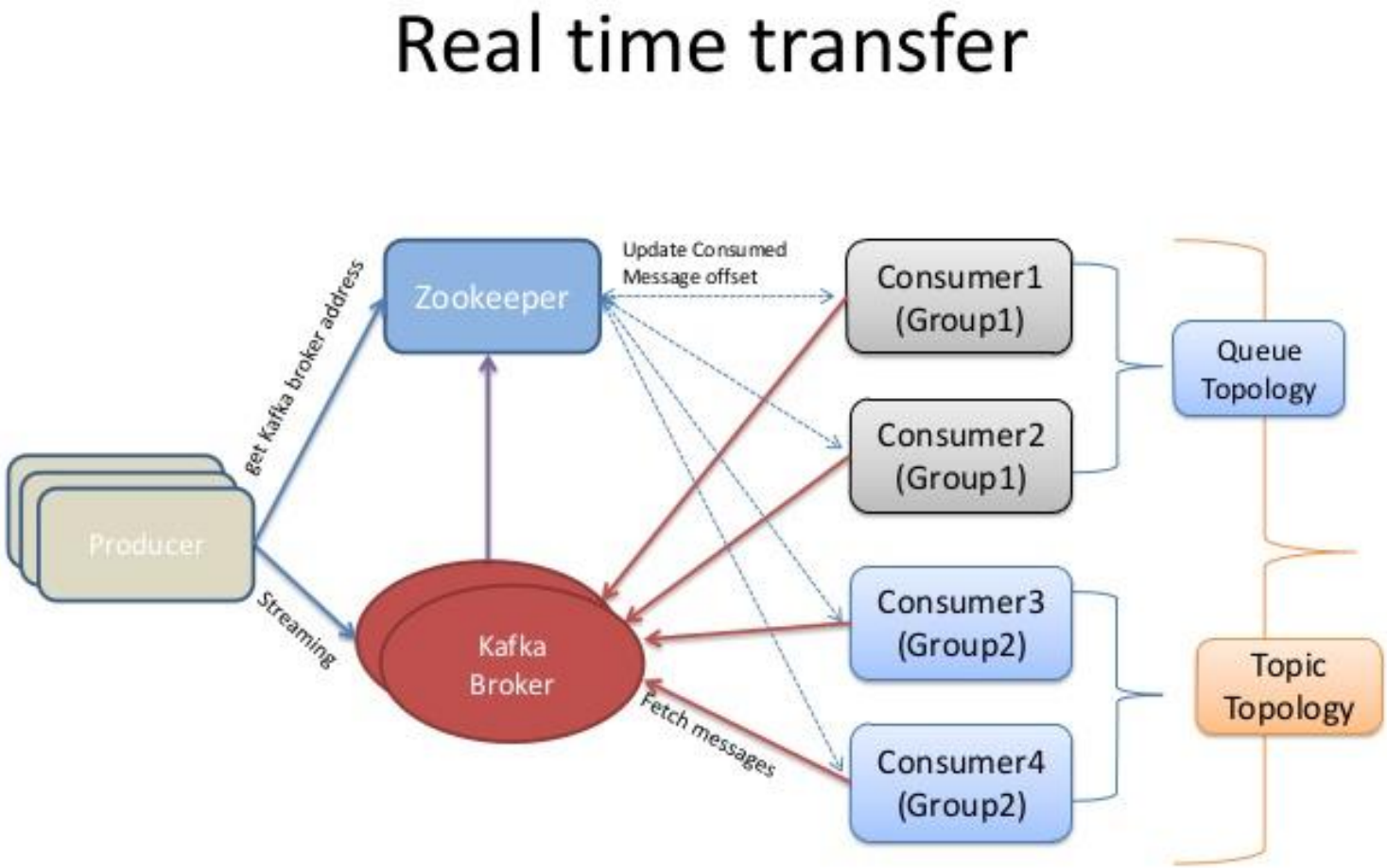
- **Partition**
  - Horizontal division of topics.
  - Replicated and scaled across multiple broker nodes.

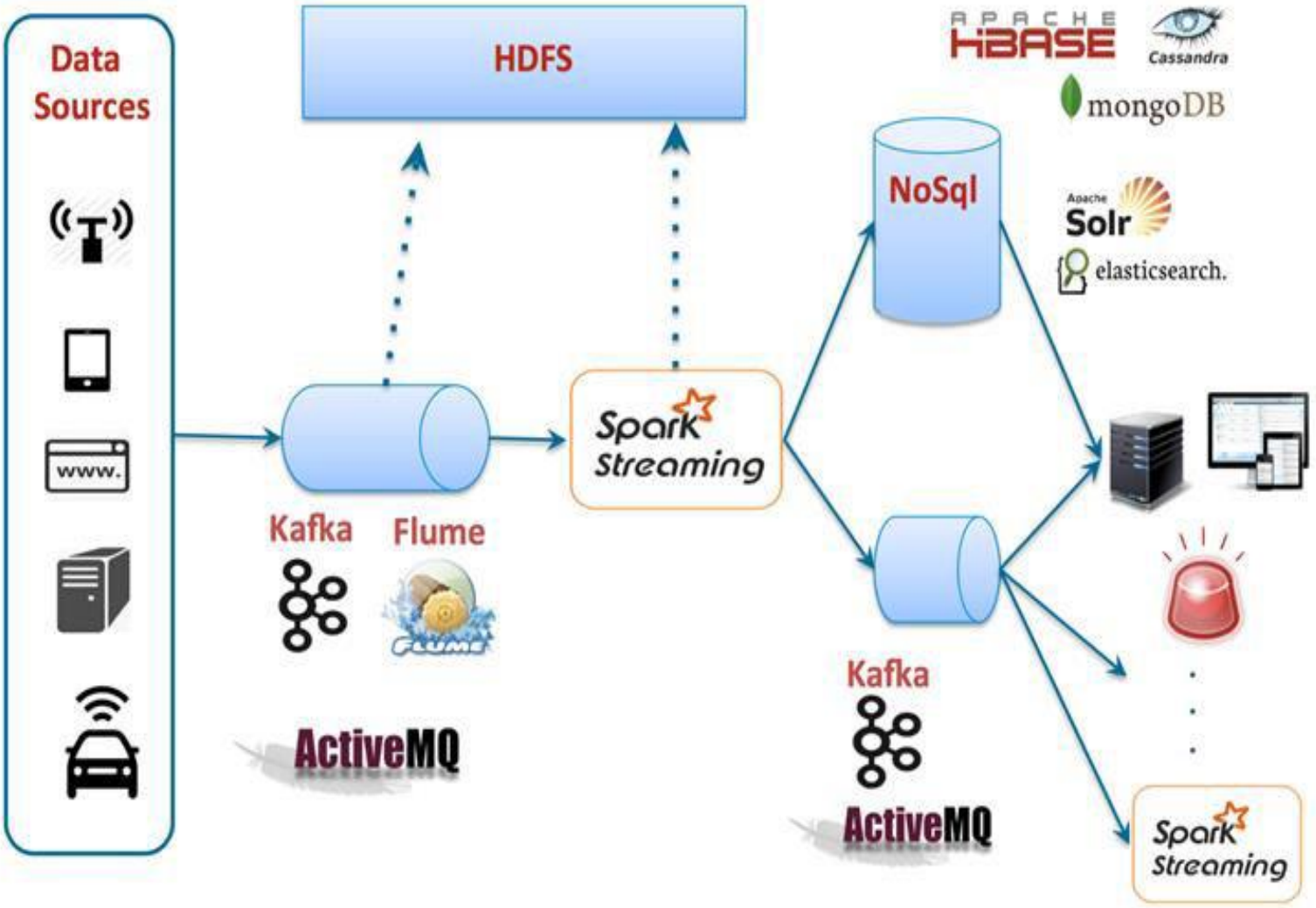
- **Replicas**
  - Partitions in the Topics are replicated.
  - ISR maintains – Fire and Forget, Sync and Async.

Anatomy of Partitions

- Partitioning allows topics to scale beyond a single machine/node
- Topics can also be replicated, for high availability.

Zookeeper's role in Kafka



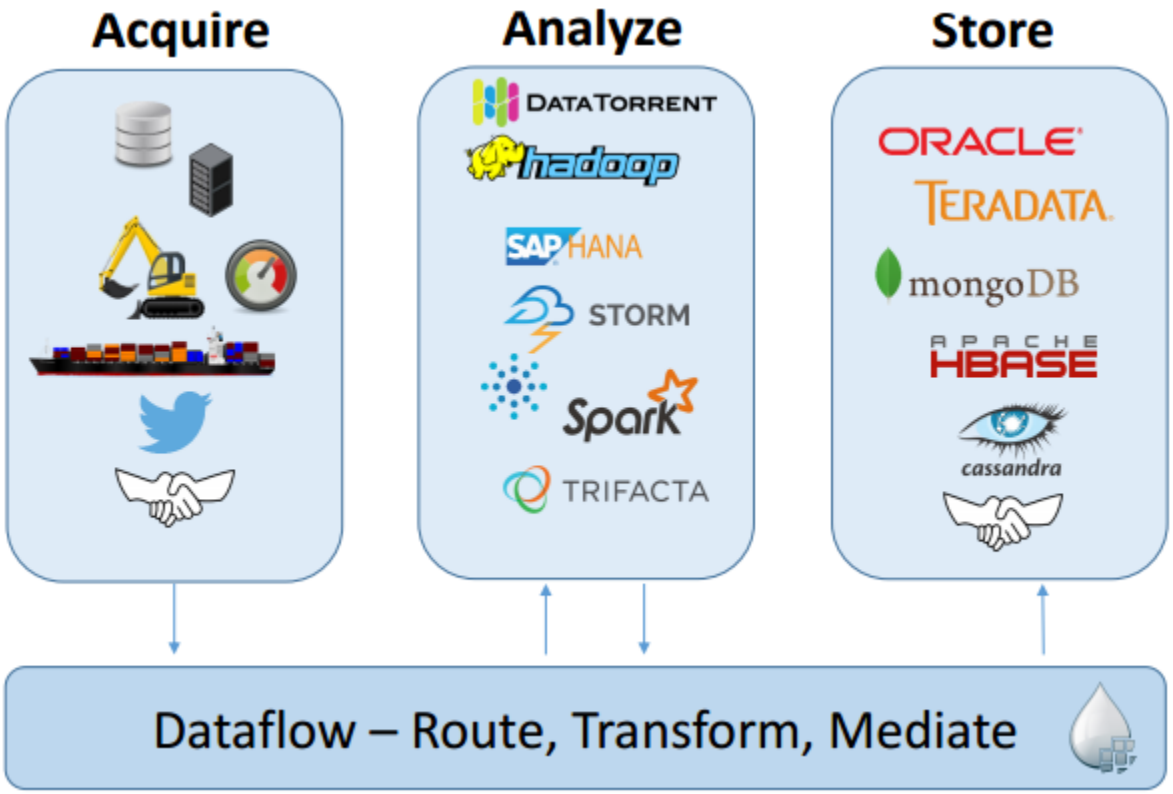

Data Sources
www.
HDFS
Kafka
Flume
ActiveMQ
Spark Streaming
NoSql
APACHE HBASE
Cassandra
mongoDB
Apache Solr
elasticsearch.
Kafka
ActiveMQ
Spark Streaming

KAFKA Workouts


nifi

NIFI Overview

- Introduction
- History
- Applications

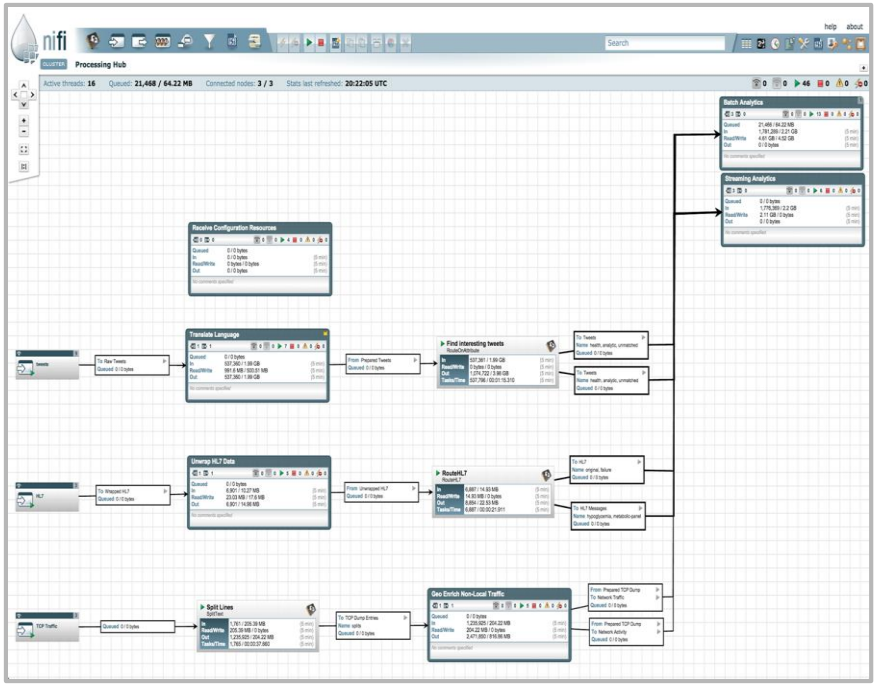


Introduction to Apache Nifi (Contd)

- Guaranteed Delivery
- Data Buffering
- Prioritized Queuing
- Flow specific – Latency vs Throughput
- Data Provenance
- Visual plugins
- Flow Templates
- Dynamic Change of flow

Introduction to Apache Nifi (Contd)

- Powerful and reliable system to process and **distribute** data
- **Directed graphs** of data routing and transformation
- **Web-based** User Interface for creating, monitoring, & controlling data flows
- **Modify data flow at runtime**, dynamically prioritize data
- **Data Provenance** tracks data through entire system
- **Easily extensible** through development of custom components



Nifi Core Components

- **FlowFile**

  Unit of data moving through the system, Content + Attributes (key/value pairs)

- **Processor**

  Performs the work, can access FlowFiles

- **Connection**

  Links between processors, Queues that can be dynamically prioritized

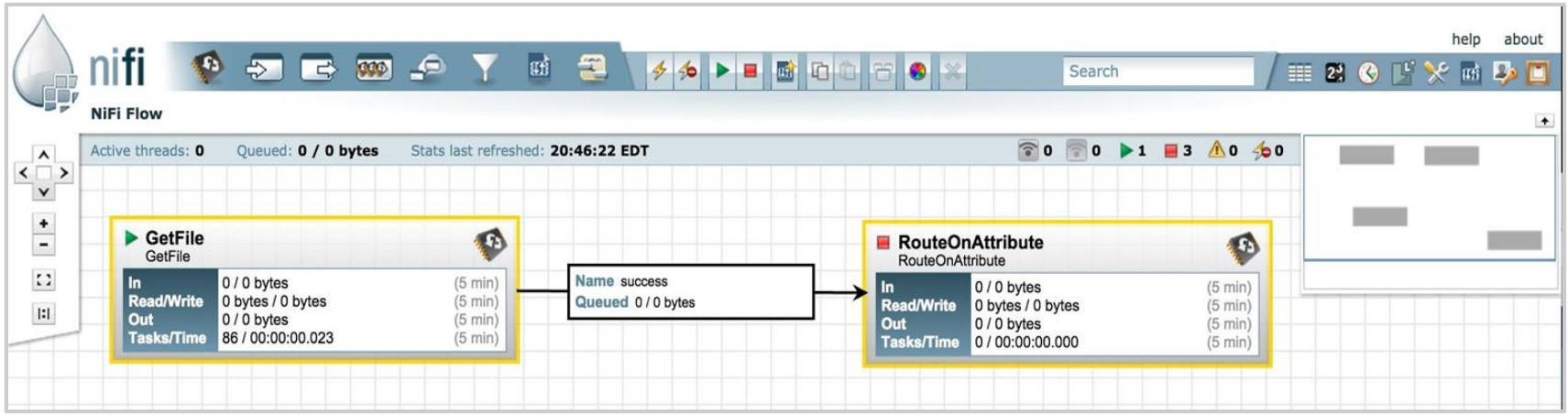
- **Process Group**

  Set of processors and their connections Receive data via input ports, send data via output ports

- **Port**

- **Funnel**

Nifi UI



- Drag and drop processors to build a flow
- Start, stop, and configure components in real time
- View errors and corresponding error messages
- Create templates of common processor & connections

Nifi Provenance

NiFi Flow Data Provenance

Oldest event available: 07/29/2015 14:08:06 EDT

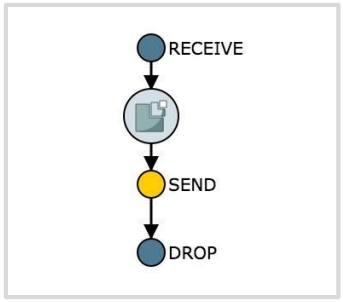
Filter | by component name

Displaying 1,000 of 1,000

Last updated: 21:12:00 EDT

Showing the most recent 1,000 of 62,293 events, please refine the search. Search

Date/Time	Type	FlowFile Uuid	Size	Component Name	Component Type
07/29/2015 16:21:34.368 EDT	DROP	3b9f20bc-031e-4af8-ad8a-fedce...	158 bytes	PutSolrContentStream	PutSolrContentStream
07/29/2015 16:21:34.367 EDT	SEND	3b9f20bc-031e-4af8-ad8a-fedce...	158 bytes	PutSolrContentStream	PutSolrContentStream
07/29/2015 16:21:34.366 EDT	DROP	6f5036bc-1768-476d-9b6d-1f83...	2.15 KB	PutSolrContentStream	PutSolrContentStream



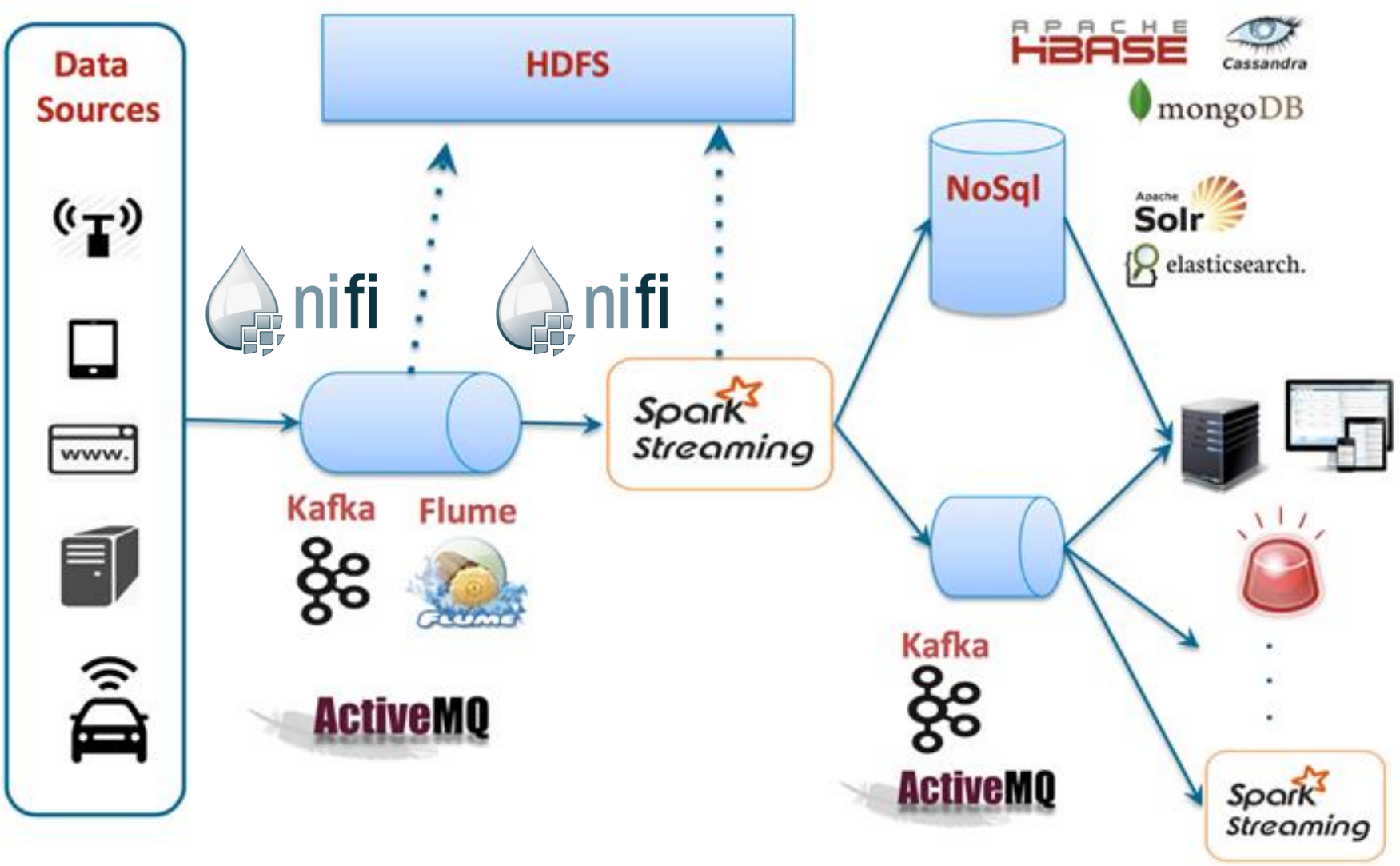
- Tracks data at each point as it flows through the system
- Records, indexes, and makes events available for display

Provenance Event

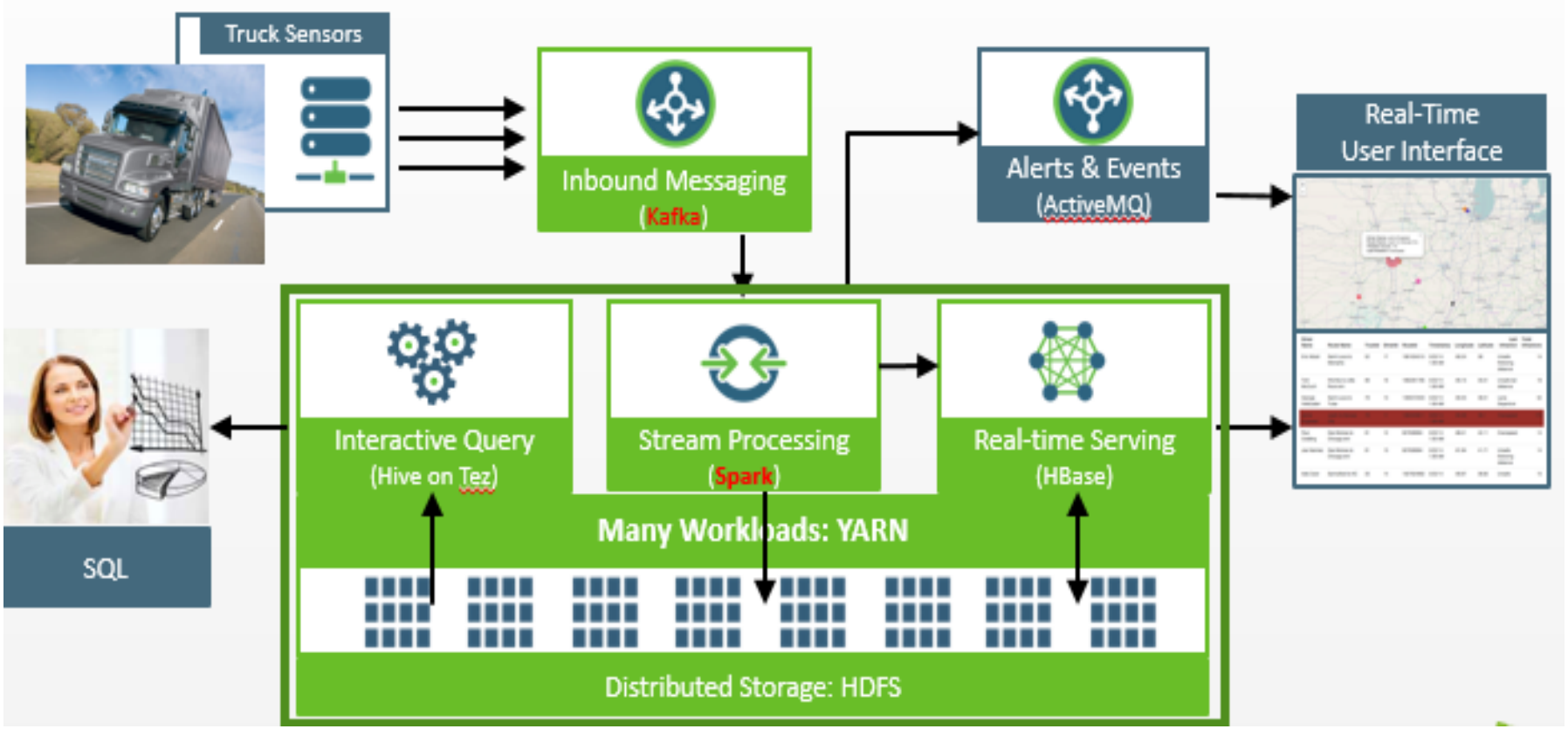
Details | Attributes | Content

Time	07/29/2015 16:21:34.367 EDT
Event Duration	00:00:00.001
Lineage Duration	00:00:00.117
Type	SEND
FlowFile Uuid	3b9f20bc-031e-4af8-ad8a-fedcef4e0099
File Size	158 bytes
Component Id	fa7b551f-c405-4fde-b004-0b0d69c03472
Component Name	PutSolrContentStream
Component Type	PutSolrContentStream
Transit Uri	solr://http://localhost:8984/solr/chronicle
Details	*No value set*

Flume/Kafka – Nifi – Spark



Realtime Trucking Fleet management



Message Queues Comparison

	Kafka	Flume	NIFI	RabbitMQ
Hadoop Plugins	Custom	Native	Medium	Least
Runtime Transformation	Yes	Yes	Yes	Yes
Data Sourcing	Push	Pull	Push and Pull	Pull
Distributed	High	Medium	Low	High
Persistence	Yes	No	No	Yes
Coding	API changes	Configuration changes	UI Plugins	API Changes
Coding Complexity	High	Medium	Low	High
Scalability	High	Medium	Medium	Medium
Runtime changes in the Data flow	No, Downtime needed	No, Downtime needed	Yes, No Downtime needed	No, Downtime needed
WorkFlow	No workflow orchestration	No workflow orchestration	Workflow orchestration available	No workflow orchestration