# Research: Real Time IOT Data Collectors

Puja Kawale, Software Engineering, San Jose State University, Santa Clara, USA., Dt.10/12/2018, SJSU ID: 012506156, puja.kawale@sjsu.edu

**IOT Devices**: Internet of Things, are connected core devices, useful home appliances, vehicles and other items implanted with computerized, software, sensors, actuators, and network which makes it possible for these things to connect within, collect and exchange the data. These are broadly three types of IOT devices – consumer IOT devices like Smart TVs, toys, appliances, wearables. Whereas, enterprise and industrial type of IOT devices include Smart meters, security systems, traffic and weather monitors etc. Below is the architecture of real time IOT device.

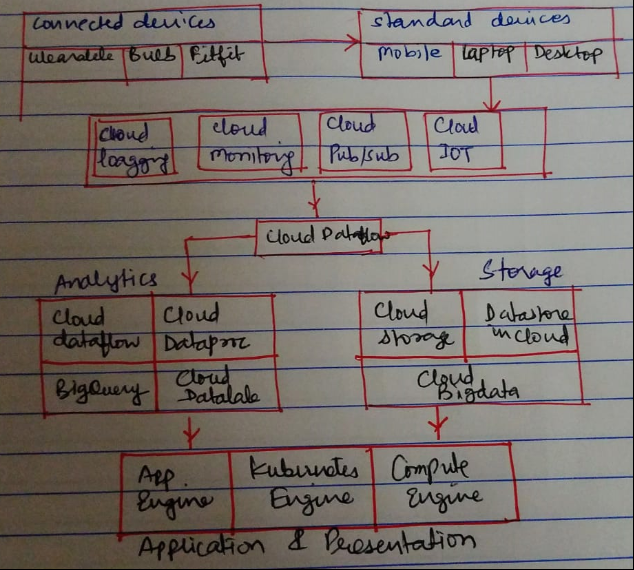


Figure: IOT real time device architecture

Source: <https://cloud.google.com/solutions/architecture/real-time-stream-processing-iot>

In short, there are devices layer, Networking and communications layer, Platform and Data storage layer and Data management and processing layer that makes up an IOT real time device.

**Data Collection and Data Flow for real time streaming data in IOT devices**: IOT sensors are major component of data sources in IOT devices. These physical hardware devices measure temperature, humidity, location, state of machine, processing time and transform it into digital signal. This information is given to the software as an input to achieve some goal. With the advancement of sensor technology and comparatively less costly materials, sensors have become part of all the objects that we see and use by the help of internet. To be specific about data collectors and their roles in IOT real time devices, in today date we have several in market from **Apache Kafka, Apache Storm., Apache Spark, RabbitMQ, Active mq to Jms, Pubsub, Redis, Pulsar, Streamsets, Flink, Akka streams, Samz, Flume.**

**Apache Kafka**: Apache Kafka is a streaming platform with altogether three key features – publish or subscribe to streams of record, store the records, process the records. Kafka is mainly used for creating real time streaming data pipelines that very well exchange data between applications or devices. It is also used for creating real time streaming apps. Its run as a ‘cluster’ on server which has streams of data called ‘topics’. Each record in it possess key, value and timestamp. Its core APIs include Producer API, Consumer API, Streams API and Connector API. In Kafka, a continual stream of data from input topics is given as feed to stream processor. Kafka then processes this data and produces streams output topics which are continuous in nature.

Benefits of Kafka – 1) Apache Kafka acts as a buffer so your systems won’t crash. 2) Reduces the need for multiple integrations. 3) Low latency and high throughput. 4) Everyone can access data

## Figure: Apache Kafka in Real time IOT device

## Source: https://kafka.apache.org/intro.html

**Apache Storm**: Apache Storm is a free available distributed real-time processor that executes large amount of high-velocity data. Its main features include evenly aligned scalability, disaster recovery, ensured data processing and capability to work with different programming languages. It rebalances the system when any new worker node is added in cluster. Guaranteed data processing makes sure that on failure of worker node, automated process would reassign the task of failed node and restart all tuples to complete the processing. Apache Storm runs in-memory; Hence, it is able to process large amount of data at in-memory speed. In history, Apache Storm had been utilized for real-time big data processing. Apache Storm caters the enhancement of gushing data non-stop unlike NoSQL or MongoDB that just saves the data.

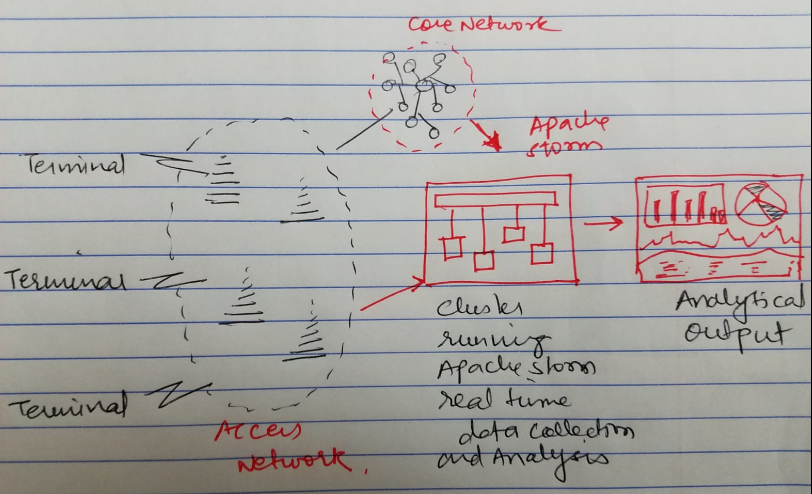
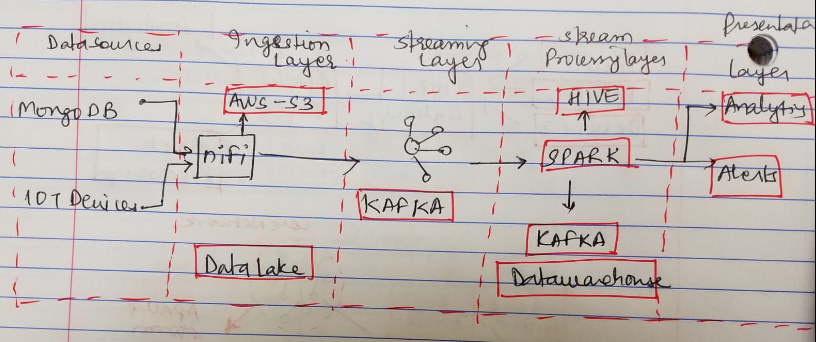


Figure: Apache Storm real time data collection and processing

Source: <https://hsc.com/Blog/Real-Time-Analytics-with-Apache-Storm>

**Apache Spark**: Apache Spark is a rapid, in-memory data processing mechanism with dynamic building APIs to allow data workers to execute streaming without any issue. In Spark streaming data will be consumed and transform from Apache Kafka using the API. Complex transformations like once event-time aggregation and distribute the outcome to certain systems can be made possible using this combination. Data collection layer is the initial layer where user ingests data from different locations. For this, we will define our data flow pipelines by using the different data source in Apache NiFi or Apache MiNiFi. Nifi is an automation tool that allows a user to exchange, direct, change and filter data as needed automatically following some basic configurations.

Figure: Apache Spark data streaming and processing

Source: <https://www.xenonstack.com/blog/big-data-engineering/real-time-streaming-apache-spark-nifi-and-kafka/>

**Rabbit MQ**: RabbitMQ is a message dealer that primarily caters to the Advanced Message Queuing Protocol. Its main responsibility is to service AMQP protocol, but still the dealer has been extended to provide the service to other message protocols like STOMP, MQTT, etc. RabbitMQ backs up low latency high throughput messaging. It has a good API and architecture for developing consumers and publishers, plus topics easily created and managed using the APIs. It becomes very straight forward to handle the lightweight topics. Both, shared and exclusive channels are available to be developed for this collector. The metadata lies in its headers that includes sensor and gateway ids and other customizable properties.

**Active MQ**: Apache ActiveMQ is rapid message broker. It supports many clients (different languages) and Protocols. It comes with Enterprise Integration Patterns which is very painless and simple to configure and use along with many advanced features while fully servicing JMS 1.1 and J2EE 1.4. It is mainly used for integration between applications/services especially in a [Service Oriented Architecture](https://en.wikipedia.org/wiki/Service-oriented_architecture). ActiveMQ is popularly used with Enterprise Service Bus’s like Apache Service Mix, JBoss Fuse and Apache Mule. It is also used with Apache Camel, which is a very strong EIP.

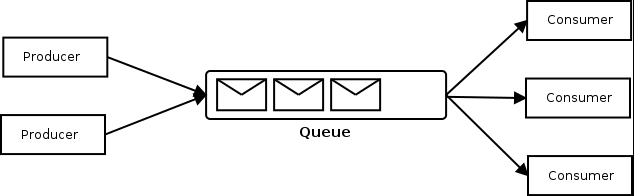


Figure: ActiveMQ message broker

**JMS**: JMS is a technique used by applications to communicate on being loosely coupled. JMS has three main components. The first is the producer, which is nothing more than a bean that submits a message to a JMS broker which is second component. ActiveMQ is the broker. Once the broker receives a message, the third component i.e., consumer processes the message. In case of JMS, user just writes both producer and consumer code using the JMS API, but actually backstage there is an adapter called ActiveMQ driver that will connect to an ActiveMQ instance and do the management for the user.

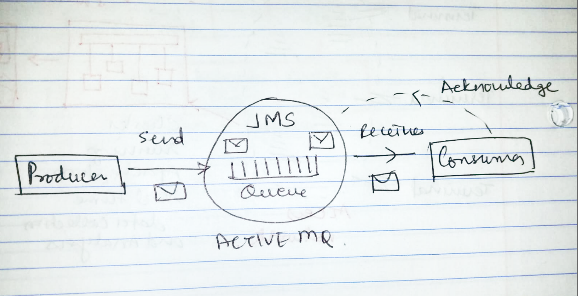


Figure: JMS message queue

Source: <https://techaffinity.com/blog/java-message-service-jms-using-activemq/>

**Pubsub**: Publish-Subscribe messaging system can support business case in which more than one consumer receives message in giver order. For example, a service can be used by a large number of people and applications who is interested in their topics of choice. It is important for them to receive messages in order.

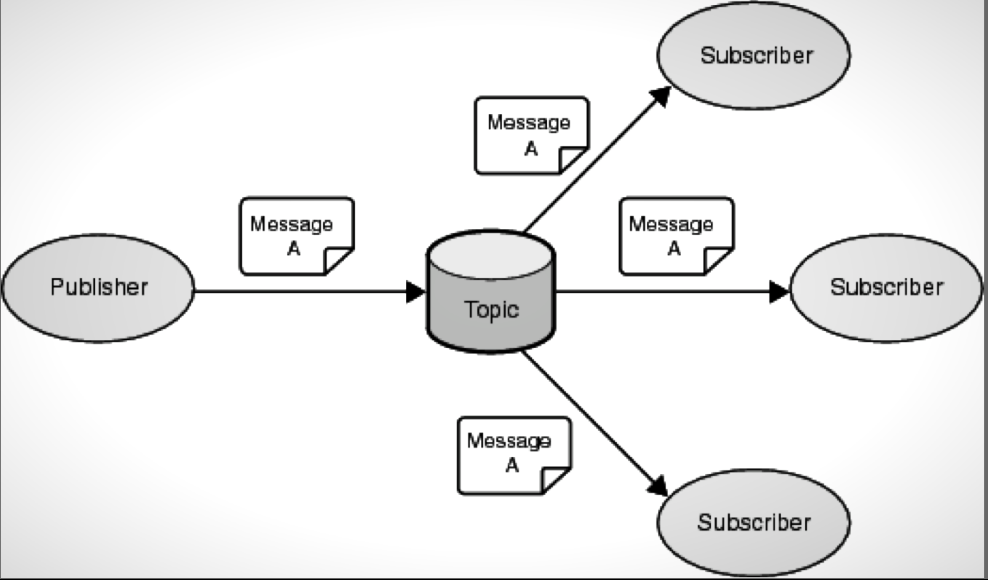
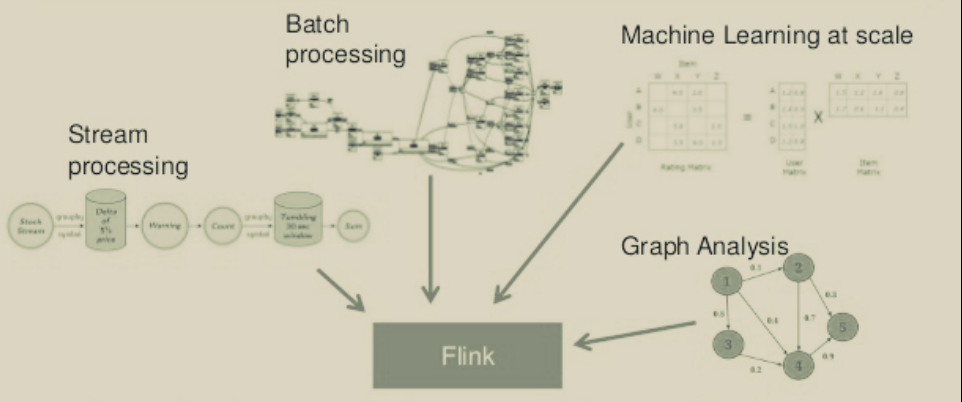


Figure: Pub-Sub messaging system

**Redis**: A lightweight message queue system for Node.js. It does not require dedicated queue server. Just a Redis server is enough. It sends and receive 1000+ messages per second. It guarantees the delivery to atleast one recipient. In this the message always stays in queue until its deleted by the user. It cannot deal with large messages and high latency. Its not a preferred option as it was not created to be a message broker but for powerful message routing.

**Apache Pulsar:** Pulsar is a good options within all the message queue as it was built with [strong permanent message storage](http://pulsar.apache.org/docs/latest/getting-started/ConceptsAndArchitecture#persistent-storage) in mind. It offers on the fly load balancing across consumers for messages on a topic. Custom load balancing is also available. It is very similar to Redis, as it offers same functionality.

**Flink**: Flink is a free available framework for streams processing. It is developed by Apache. Its core is written in Java and Scala. Flink executes arbitrary [dataflow](https://en.wikipedia.org/wiki/Dataflow_programming) scripts parallelly in a pipeline. This enables the running of  [batch](https://en.wikipedia.org/wiki/Batch_processing) and stream processing scripts.



Source: <https://www.slideshare.net/Hadoop_Summit/flexible-and-realtime-stream-processing-with-apache-flink>

**Samza**: Samza is Apache’s data stream distribution framework. It makes use of Kafka for messaging and Hadoop Yarn for fault tolerance. It has simple API, device state management, durability, scalability, pluggability and process isolation.

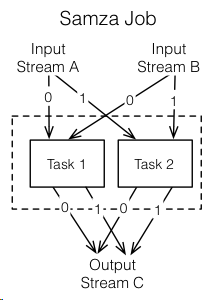


Figure: Samza Batch Job

**Flume:** Flume is Apache’s distributed system which is reliable and available and efficient to collect and move large volume of log data, event data or any other form of data from several sources to centralized data store.

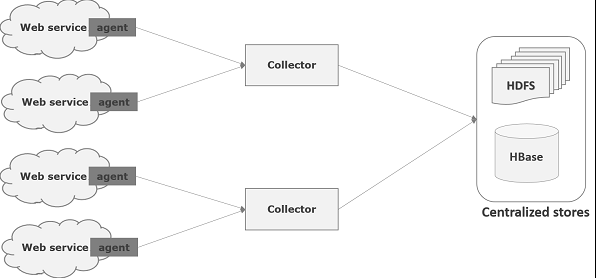


Figure: Apache Flume Data Flow

References:

<https://spark.apache.org/>

<http://storm.apache.org/>

<https://en.wikipedia.org/wiki/Publish%E2%80%93subscribe_pattern>

<https://www.ibm.com/support/knowledgecenter/en/SSSHYH_7.1.0.14/com.ibm.netcoolimpact.doc/user/jms_data_source_create.html>

<https://samza.apache.org/learn/documentation/0.14/hadoop/overview.html>

<https://cloud.google.com/solutions/architecture/real-time-stream-processing-iot>

<https://kafka.apache.org/intro.html>

<https://hsc.com/Blog/Real-Time-Analytics-with-Apache-Storm>

<https://www.xenonstack.com/blog/big-data-engineering/real-time-streaming-apache-spark-nifi-and-kafka/>

<https://techaffinity.com/blog/java-message-service-jms-using-activemq/>

<https://www.slideshare.net/Hadoop_Summit/flexible-and-realtime-stream-processing-with-apache-flink>