International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 07 Issue: 05 | May 2020 www.irjet.net p-ISSN: 2395-0072

# Top Big Data Technologies for Data Ingestion

**Kannan Sobti[1], Prof. Deepika Dash[2]**

[1]*Dept. of Computer Science and Engineering, RV College of Engineering*
[2]*Dept. of Computer Science and Engineering, RV College of Engineering*

---

**Abstract -** *Businesses use data to make decisions. If analysed effectively, data enables forecasting of trends in the market, helps businesses understand what customers need and prefer, and enables a company to create strategy. But the massive volume of data being generated from numerous sources is a challenge. Businesses can only take full advantage of Big Data if all the data being generated is used to its potential. Today availability of data is not a problem; the challenge is to effectively handle the incoming data, and do so swiftly enough so that data retains its importance, and remains meaningful. This era of Big Data is about efficient collection, storage and analyses of data, which needs good tools. But choosing the right tool comes through updated knowledge of the state-of-the-art in data-handling technologies. This paper analyses the features of some cutting-edge Data Ingestion technologies in today's market, and can help identify the technology best suited to a company's business processes.*

***Key Words***: Data Ingestion, Streaming, Real-time, Pipelines, Distributed, Throughput, Cluster.

## 1. DATA INGESTION – AN OVERVIEW

Data can only be properly 'digested' if it is properly 'ingested'. To ensure that data from multiple sources can be effectively analysed, all data must first be moved to a common destination. This is done through *Data Ingestion* technologies that enable movement of data from multiple different sources to a centralized location, typically a *data warehouse*, which is simply a database designed and built for efficient data handling and reportage. Data ingestion tools create a data pipeline to enable import, transfer, loading and processing of data for later use within the warehouse, and facilitate data extraction by supporting a variety of data transport protocols.

Effective data ingestion is a primary requirement for building a business intelligence strategy based on data analytics. The data ingestion layer is the mainstay of an analytics topology. A good analytics system that relies on downstream reporting must be provided data that is consistently accessible. Thus design of the data pipeline will be critical to a business.

The *destination* for data moving through an ingestion pipeline would be a data warehouse, or a data mart/document store. Data *sources* are widely disparate: data from SaaS sources, numerous apps, engineering systems that generate data outputs, and data scraped from the Internet. Data may be in different *formats*, including RDBMS or other databases, S3 buckets, CSVs, or data streams. Since the ingested data is from varied sources, it must first be cleansed and transformed for analysis along with data from other sources.

Data can either be ingested in batches, in real time, or in a combination of the two methods (the *Lambda Architecture*). Data is ingested in batches and is then imported at scheduled intervals. This is useful for processes that run on a schedule, e.g., daily reports at a specified time. Real-time ingestion is used when information collected is time-sensitive and has to be monitored and acted upon from moment-to-moment, e.g., power grid data. The *Lambda Architecture* balances the benefits of batch and real-time modes. It uses batch processing for scheduled data and real-time processing to provide views of time-sensitive data [1].

### 1.1 Batch Ingestion vis-à-vis Streaming Ingestion

Business requirements and the operating environment decide the structure of the data ingestion layer. A company will choose the model based on the timeliness at which data must be accessed for analysis.

*Batch processing*: In this method, the ingestion layer collects data periodically, groups it, and sends to the destination in group-sized batches. Groups can be processed in any sequence, either using a designated schedule or when certain triggers are set off. This method is easy to implement, more affordable, and used when near/real-time data analysis is not needed.

*Real-time data processing* (or *stream processing/* data *streaming*): This is done without the grouping of sourced data. Data is loaded as soon as it is generated and the data ingestion layer recognizes it. This technology is used when information analysis requires data that is extremely current. This is a more expensive technology, since systems must constantly read data sources and accept new information.

A few streaming platforms actually utilize batch processing technology. E.g., *Apache Spark Streaming.* In this method, ingested groups are merely smaller in size, or are prepared at shorter intervals, but they are still processed in group batches. This method is called *micro-batching* and it is correct to consider it as a distinct data ingestion category.

### 1.2 Challenges faced in Data Ingestion

The global data ecosystem is creating varied data sources, with data volumes exploding. Information can come from widely different data sources: transactional databases, SaaS platforms and from mobile phones to IoT devices. Sources are constantly evolving and new ones

emerging. This makes it difficult to define and create a data ingestion technology that can be called future-proof.

Creating an architecture that can ingest a high volume of widely diverse data is expensive and time-consuming. Speed is a challenge for the ingestion process and the data pipeline. As data grows more complex, developing and maintaining data ingestion pipelines grows even more arduous, especially for real-time data processing. Depending on the field requirement, pipelines can have a slow refresh rate, updating maybe every 10 minutes, or completely current, like a stock market ticker application in peak trading hours.

Importantly, being able to correctly assess an organisation's data pipeline need is crucial for the design of the data ingestion architecture. A business must choose a system based on the type of data they need and use. The value of data depends on the ability to ingest and integrate it. Data warehouses that operate off the cloud, and can scale effectively, will maximise the performance of a data pipeline.

### 1.3 The Data Ingestion Sequence

Till fairly recently, the data ingestion sequence was extract, transform, and load (ETL). In this method, data is taken from the source, manipulated as per properties of the destination database, and thereafter loaded onto the destination. With businesses using expensive analytics, this preparation was needed, which included data transformation, before the data could be loaded into a data warehouse.

But today, much has changed. Cloud-based data warehouse technologies like Microsoft Azure SQL Data Warehouse, Google BigQuery, Amazon Redshift and Snowflake provide very cost-effective options to scale up the compute & storage resources in real-time. This means that there is no need to preload transformations and the entire raw data of an organization can be dumped into the data warehouse or data lake. This data can then be used while in the data warehouse, with transformations being defined and run in SQL - directly at query time. Thus Extract, Transform, Load (ETL) has changed to Extract, Load, Transform (ELT).

Creating an ETL/ELT platform from scratch requires writing complex data transformation code for the data pipeline. Many companies now offer readymade solutions tailored to work in specific computing environments or software applications. This allows the data analytics team to focus on business logic and if a need arises, still develop *ad hoc* transformations. This also guarantees higher accuracy, availability, and consistency [2].

Today, most proprietary data ingestion software also includes *data preparation* features to *structure* and *organize* data enabling it to be analysed either on the fly or later by business intelligence and business analytics programs [3].

A fully managed ELT solution can ingest data from all sources onto cloud-based data warehouse destinations, and the time difference between data ingestion and insightful decisions has come down from weeks to minutes.

## 2. APACHE KAFKA

Kafka is a distributed streaming platform. It is capable of building data streaming pipelines and apps. Open-source, horizontally scalable, fault-tolerant, and one of the swiftest in the market, it functions as a messaging agent, and provides a unified platform to handle high data throughput with fairly low-latencies, which enable it to handle real-time data feed. It uses a "distributed, partitioned, replicated commit log service" [4].

Kafka functions as a messaging system but uses a unique methodology. Data streams are partitioned and spread over a cluster of machines, allowing data streams that are larger than the capacity of a single machine to be stored without any disruptions. This therefore allows clusters of coordinated customers to be handled through a single platform. Kafka assigns any one cluster the task of acting as the central data backbone for a given client organisation. This cluster-based design makes Kafka highly fault-tolerant, also enabling it to expand without requiring any downtime.

Kafka is widely used for requirements such as collection of user activity data, device instrumentation, application metrics, logs, and stock markets. Kafka ensures high availability of streaming large volume data in real-time. This data can be flexibly consumed by systems with variable requirements: by batch processing systems like *Hadoop* or by data streaming systems that need to transform data streams on arrival.

### 2.1 Design

Kafka enables stream processing, storage and messaging. This uncommon combination is necessary for Kafka to function as a streaming platform. Kafka is mainly used for two types of applications: to build data streaming *pipelines* that move data between systems and applications; to build *applications* that react to and transform the incoming data streams [5].

Some important features of Kafka are:

- It can be run as a cluster on multiple servers straddling across multiple data-centres.
- Kafka clusters store record streams under various categories known as *topics*.
- Each record has a key, value, and timestamp [6].

Kafka provides five APIs, which are the core of its design:

- *Producer API.* Enables applications to publish streams of records to one or more *topics*.
- *Consumer API.* Enables applications to subscribe to one or more topics and process records to them.
- *Streams API*. Allows applications to act as stream processors, by imbibing input streams from multiple topics and outputting transformed data to multiple topics.

- *Connector API*. For building and running of reusable producers or consumers so that Kafka topics can be connected to existing data applications/systems.
- *Admin API*. For management and inspection of topics, brokers and other Kafka objects.

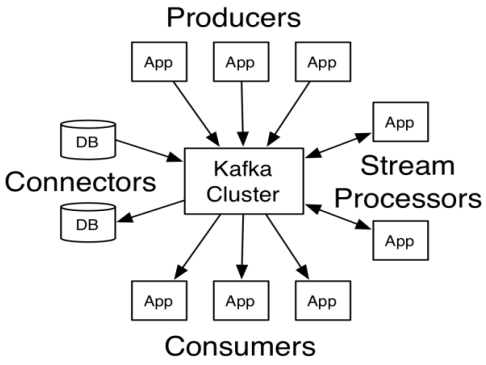


**Fig -1:** The Kafka Cluster
(Source:https://kafka.apache.org/intro.html)

## 2.2 Communication

Kafka handles communication between clients and servers using a language-agnostic TCP protocol. The protocol is backward compatible with older versions of the product. Though Apache provides a Java client as default for Kafka, clients are also available in other languages.

## 2.3 Storage

Static files for batch processing are stored in distributed file systems like HDFS. This ensures that historical data is available for processing. On the other hand, a classic messaging system enables processing of future messages, which are yet to arrive at the time the client has subscribed. Such applications process data as it arrives. Kafka combines both these capabilities, enabling it to be used as a platform for both data streaming pipelines and streaming applications.

Streaming applications address past and future data in a similar manner, by combining low-latency subscriptions with storage facility. Thus the application will start processing past data and seamlessly continue onto future data as it begins arriving. This approach includes both the batch processing and the message driven techniques used by stream processing technology [7].

An important idea in Kafka is the creation of '*topics*' for record streams. This is nothing but a category to which the incoming data stream or records are published [8]. Kafka maintains a separate log partition for each topic as illustrated below, and topics can be subscribed to by multiple consumers.

Every record is entered into a *partition* and is allotted an identification number called an 'offset' [9]. Thus partitions are ordered sequences of records to which successive records are continuously committed. Log partitions are designed to scale. Topics have multiple partitions but each partition may reside on a separate server. This allows logs to scale beyond the size of a single server. In this way, log partitions also support parallel computing.

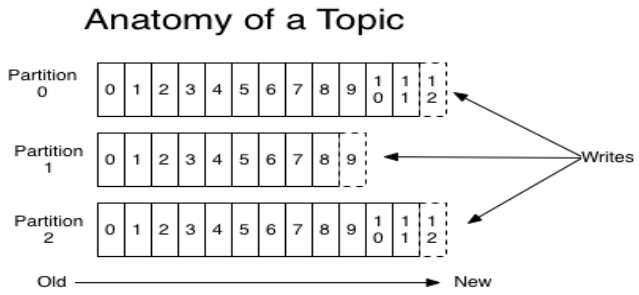


**Fig -2:** Kafka Partitions
(Source:https://kafka.apache.org/intro.html)

An important idea in Kafka is the creation of '*topics*' for record streams. This is nothing but a category to which the incoming data stream or records are published [8]. Kafka maintains a separate log partition for each topic as illustrated below, and topics can be subscribed to by multiple consumers.

Every record is entered into a *partition* and is allotted an identification number called an 'offset' [9]. Thus partitions are ordered sequences of records to which successive records are continuously committed. Log partitions are designed to scale. Topics have multiple partitions but each partition may reside on a separate server. This allows logs to scale beyond the size of a single server. In this way, log partitions also support parallel computing.

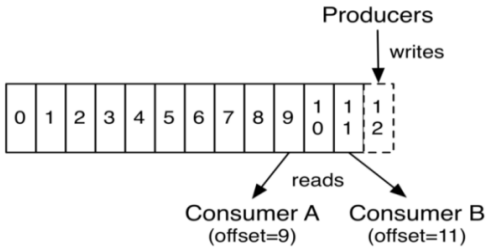


**Fig -3:** The Log Partition
(Source:https://kafka.apache.org/intro.html)

A Kafka cluster will persist only for the period for which it has been configured, whether it is consumed or not. After that it gets discarded and frees storage space. For each consumer only metadata of the offset position id is maintained in Kafka. A consumer may consume records in any order, even though this is normally in a linear sequence. Hence a consumer has the option to reprocess historical data or simply focus on processing current data.

### 2.4 Distribution

Since log partitions may reside across multiple servers, for each partition, one server is nominated as the 'leader' and the other servers handling the spillover are the 'followers'. By corollary the leader handles the read-write requests and the followers duplicate all actions of the leader mechanically.

Fault tolerance is handled by replicating each partition on a different server. If a leader fails, a follower takes over as the leader. A server may be a leader for one cluster of partitions and a follower for another cluster. This evens out the load on each server.

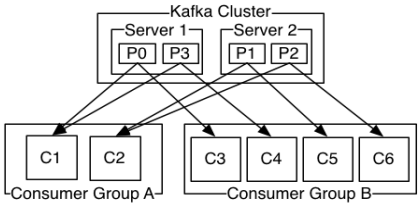


**Fig -4:** A 2-Server Kafka cluster hosting four partitions (P0-P3) with two consumer groups. Consumer Group A has two consumer instances and Group B has four.
(Source:https://kafka.apache.org/intro.html)

### 2.5 Geo-Replication

Messages are replicated across multiple data centres in different geolocations as well as on different cloud regions. This task is handled by a utility called *MirrorMaker*, which also enables the use of replicated data for backup and recovery and for various other requirements like bringing the data closer to the user, handling geo-relocation of data etc.

### 2.6 Producers and Consumers

Producers publish data, whereas the consumers use the data. Data is published to a topic and partition of the producer's choice, either using a round-robin record allocation method, or using some semantic triggers. Consumers allot a group name to themselves and each record being published is delivered to the consumers that subscribe to the topic. Of course consumers can be across different machines, servers and cloud regions. Even if many consumers are on the same machine or server, the records may be housed on other machine locations to ensure load-balancing. This also facilitates scaling and fault-tolerance. In case there are multiple subscribers for the same topic, the records are broadcast to each group simultaneously.

Kafka divides partitions further amongst multiple consumers, and manages consumer 'memberships' dynamically, using a protocol in which new consumers are reapportioned some segments of partitions already in use by existing group members; and if a consumer dies, its partitions are redistributed amongst the existing members.

The Kafka streaming algorithm takes a continuous stream of data from input topics, and pushes the processed data to output topics. E.g., a business forecasting application may take an input stream on 'sales data' and output a stream called 'customer preferences'.

## 3. APACHE STORM

Storm is another Apache distributed streaming product, which processes unbounded data streams and is compatible with most programming languages [10]. It is useful for applications like real time analytics and Internet-based machine learning.

Storm claims a processing speed of one *million tuples per second per node*. It can be integrated with in-use queuing and database processes. It can handle repartitioning of streams between stages of computation on an as required basis.

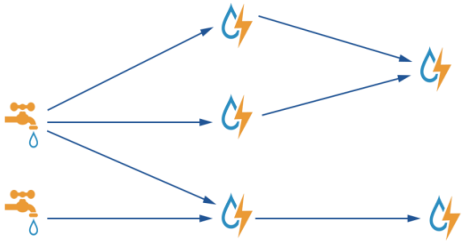

**Fig -5:** Storm Dataflow Model
(Source: https://storm.apache.org/)

Storm covers the gaps that were noticed between the hardware capability and performance of existing streaming technologies [11]. Storm's engine uses a lean threading model and a lock-free messaging methodology and backpressure to handle overloading [12]. Storm is a useful adjunct to Enterprise Hadoop, and YARN & Slider, adding value to real time analytics capability and machine learning processes.

### 3.1 Storm Nodes

The *Nimbus* Node, an equivalent of Hadoop's *JobTracker*. Nimbus issues commands to execute code across a Storm cluster;

*Zookeeper* Nodes provide coordinated communication between the Nimbus Node and Supervisors;

*Supervisor* Nodes handles the actual execution through various 'workers' in the cluster.

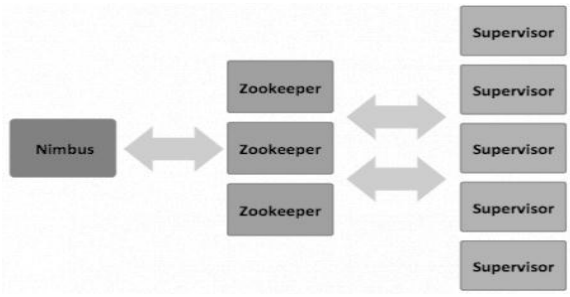


**Fig -6:** Storm Nodes
(Source: https://storm.apache.org/)

### 3.2 Common Process Terms in Apache Storm

*Tuple:* Ordered list of elements. E.g., a 3-*tuple* might be [4,7,6].

*Stream:* Unbounded sequence of Tuples.

*Spout:* Source from where data streams emanate.

*Bolts:* Process input streams to produce output streams. They run functions and filter, aggregate, and join data. They also talk to databases.

*Topologies:* The overall network structure of *spouts* and *bolts* represented visually.

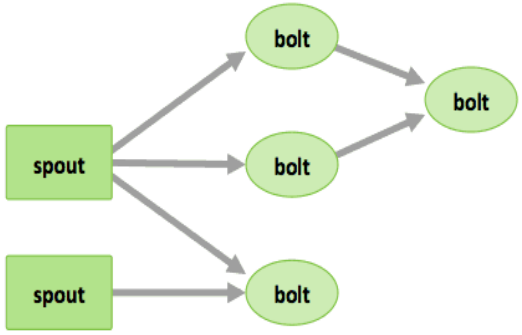


**Fig -7:** Storm Topology
(Source: https://storm.apache.org/)

## 4. APACHE FLUME

Flume provides a distributed service to collect, aggregate and move large data using streaming data flows. Its data model's extensibility enables it to add analytics applications [13]. Its In-Memory feature allows data to spill to the disk, and the Kite API can write data to HBase and HDFS.

Using the concept of *Configuration Filters,* Flume enables the insertion of sensitive inputs like passwords. Using Apache *log4j*, Flume enables Java applications to write events to HDFS files, and enables the *Tail* application to pipe data from local files.

Flume uses a JVM process with three components: a Source, a Channel and a Sink. All events are broadcast via these three channels. Carriage of data between source and sink is done either on a pre-set schedule or can be triggered by an event.

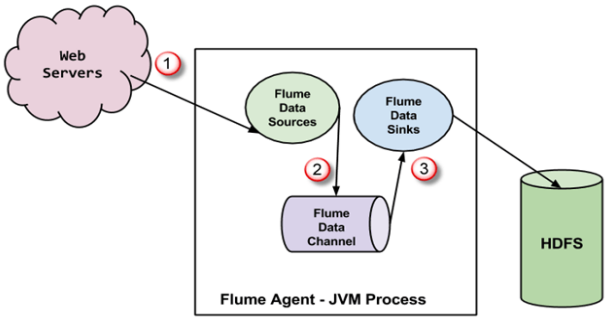


**Fig -8:** Flume Topology
(Source: https://flume.apache.org/)

## 5. APACHE NIFI

NiFi automates management of data flow between systems [14]. It provides a web-based UI to create, monitor and control data flows. Because of its configurable design, it can modify data flow processes during runtime. It is extensible and can handle diverse data flows. NiFi's directed graphs help in a clear visualisation of data routing and transformation.

Nifi provides a Web-based UI and various configurations to optimise data flow management:

- Loss Tolerant *vs* Guaranteed Delivery
- Low Latency vs High Throughput
- Dynamic Prioritization
- Flow modification at runtime
- Buffering & Back Pressure

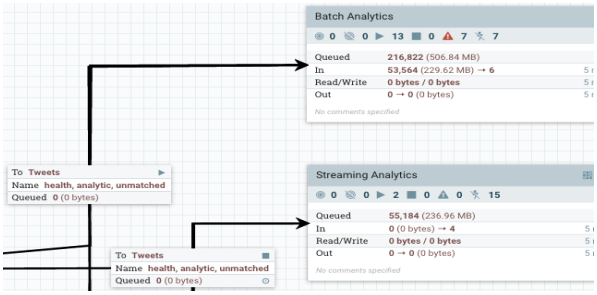


**Fig -9:** NiFi Directed Graphs
(Source: https://nifi.apache.org/)

NiFi's data tracking mechanisms provide consistent data provenance; its extensible design allows users to customise and extend processors for better development and testing effort. NiFi provides security through SSL, SSH, HTTPS, and encrypted content. It also enables multi-tenant authorisation through customisable authorisation and policy management schema. NiFi supports any Java-run device and is ideal in limited connectivity.

## 6. OTHER DATA INGESTION TECHNOLOGIES

### 6.1 Wavefront

Wavefront is a hosted platform designed to ingest, store, visualise and issue alerts on metric data. It can ingest a large volume of data points per second. Its stream processing technique enables it to manipulate large volumes of data and it provides a 360 degree view across the IT infrastructure.

### 6.2 SYNCSORT

Syncsort allows collection, integration, sorting and distribution of data in a swift timeframe, using minimal resources. It deploys on Hadoop, *Splunk* and the cloud. The data application design needs to be done just once thereafter it can be deployed on any platform: Windows, UNIX & Linux, or Hadoop; either on premises or on the cloud. *Ironstream* for Splunk enables processing of huge volumes of machine data streams from the mainframe.

### 6.3 DATATORRENT

DataTorrent can handle both streaming and rest data. It can process many million events per second and can recover from node outages without any loss of data – without needing any external human intervention.

### 6.4 Amazon KINESIS

Kinesis is a cloud-based service designed to handle real-time distributed data streams. It can handle multiple data types and sources like website clickstreams, financial transactions, social media feeds, IT logs, and location-tracking events. A fully managed service, it serves web applications, mobile devices, wearables, and industrial sensors.

### 6.5 Apache SAMZA

Samza provides distributed streaming processing. It uses Apache Kafka for messaging and Hadoop YARN for fault handling, processor isolation, security and resource management. Samza handles restoration of a stream processor's state from outage using snapshotting, i.e., whenever a processor is re-started, it is restored to a consistent snapshot state. If a machine within the cluster fails, Samza uses YARN to migrate tasks to another machine.

### 6.6 GOBBLIN

Gobblin is a data ingestion framework designed to handle multiple data sources like rest APIs, FTP/SFTP servers and filers, and load these onto Hadoop. Gobblin ingests data from different sources in the same execution framework.

### 6.7 Apache SQOOP

Named from SQL+Hadoop, Sqoop transfers bulk data between Apache Hadoop and structured data repositories like RDBs. It can handle incremental loading of a table or a free form SQL query. It saves jobs, which can then be run multiple times to import updates made to a database since the last import. These imports can be used to populate tables in *Hive* or *HBase*.

### 6.8 FLUENTD

Fluentd is an open source technology designed for data collection. It unifies data collection with consumption. The platform offers features like community-driven support, ruby gems installation and self-service configuration. It supports C and Ruby language and 650 plugins.

## 7. CONCLUSION

Apart from gathering, integrating and processing data, data ingestion tools help companies modify and format their data for analytics and storage purposes. With these tools, users can ingest data in batches or stream it in real time. Real-time data ingestion implies importing the data as it is produced by a source. Batch ingestion implies the importing of discrete chunks of data at intervals.

Companies that use data ingestion tools need to prioritise data sources, validate each file, and dispatch data items to the correct data repository/destination to ensure an effective ingestion process. Although some companies do develop their own data ingestion tools, most companies use data ingestion tools developed by experts in data integration. These are definitely more user friendly and cost effective.

Professionally developed expert tools provide numerous advantages; e.g. the freedom to use different transport protocols to collect, integrate, process and deliver data. Data flow visualisation tools allow users to see the dataflow, simplify its complexity if necessary. Expert tools also provide a high level of scalability, which is critical for Big Data.

Professional ingestion tools also provide multi-platform support and integration, enabling extraction of data from different types of databases and operating systems, without impacting the performance of the system. Finally, security, a necessary part of any solution, is well taken care of by the expert tools.

## REFERENCES

[1] G. Alley, "What is Data Ingestion?" Alooma Blog, ETL https://www.alooma.com/blog/what-is-data-ingestion

[2] "Data Ingestion: The First Step to a Sound Data Strategy," https://www.stitchdata.com/resources/data-ingestion/

[3] M. Rouse, "Data Ingestion," TechTarget https://whatis.techtarget.com

[4] Apache Kafka Introduction, https://kafka.apache.org/intro.html

[5] Apache Kafka Introduction, https://kafka.apache.org/intro.html

[6] Apache Kafka Introduction, https://kafka.apache.org/intro.html

[7] Apache Kafka Introduction, https://kafka.apache.org/intro.html

[8] Apache Kafka Introduction, https://kafka.apache.org/intro.html

[9] Apache Kafka Introduction, https://kafka.apache.org/intro.html

[10] "Why use Apache Storm?" https://storm.apache.org/

[11] R. Naik, "Apache Storm 2.0 : Rearchitecture and Performance," https://storm.apache.org/talksAndVideos.html

[12] P. Taylor Goetz, "The Future of Apache Storm https://storm.apache.org/talksAndVideos.html

[13] Blog Apache Flume, https://flume.apache.org/

[14] Blog Apache NiFi, https://nifi.apache.org/