**Data Ingestion :**  
  
 after setting up hadoop cluster ,data ingestion is first part that every project start with. Data Ingestion require proper arcjitecture design which help us tio place right component at right place. Here component means using sqoop,kafka,flume etc for data ingestion. It is very complicated to decide which component we shall use to make our ingestion process fast and smooth. This document will drive you through ingestion process and which component to use when?.

Diagram given below explain brief about ingestion process and when to use which component. However this document contains detail about some of component that is given in diagram.

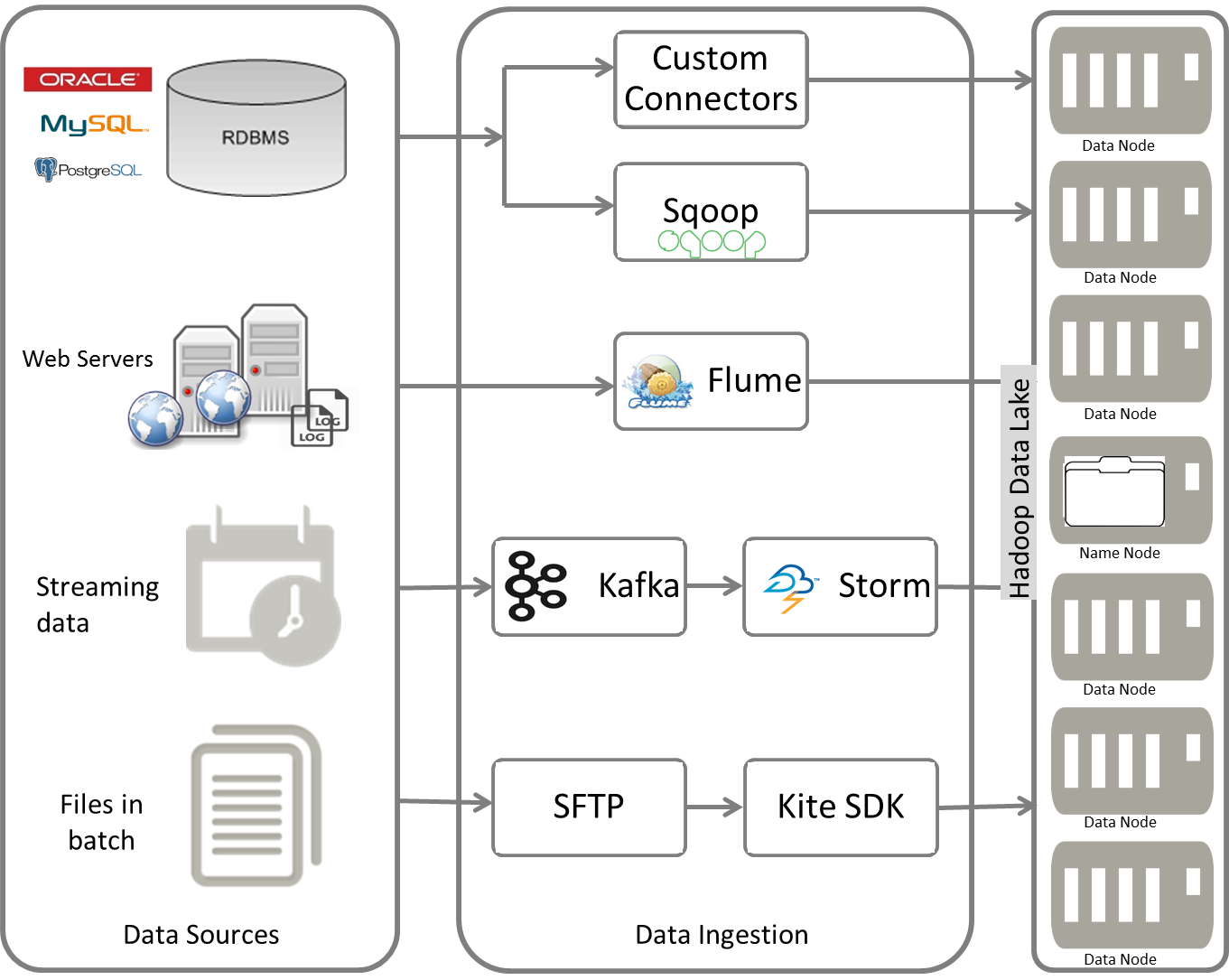


Fig: Data Ingestion

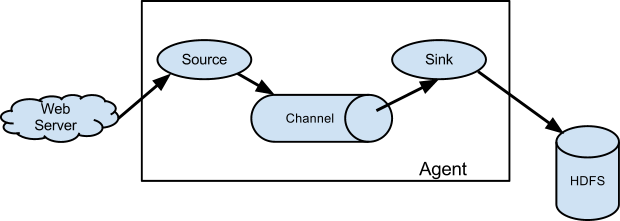
**Apache flume :**

Apache Flume is a distributed, reliable, and available system for efficiently collecting, aggregating and moving large amounts of log data from many different sources to a centralized data store.

The use of Apache Flume is not only restricted to log data aggregation. Since data sources are customizable, Flume can be used to transport massive quantities of event data including but not limited to network traffic data, social-media-generated data, email messages and pretty much any data source possible.

**Why do i need flume?** Some data destined for Hadoop clusters surely comes from sporadic bulk loading processes, such as database and mainframe offloads and batched data dumps from legacy systems. But what has made data really big in recent years is that most new data is contained in high-throughput streams. Application logs, GPS tracking, social media updates, and digital sensors all constitute fast-moving streams begging for storage in the Hadoop Distributed File System (HDFS). As you might expect, several technologies have been developed to address the need for collection and transport of these high-throughput streams. [Apache Flume](http://flume.apache.org/) is rapidly becoming a de facto standard for directing data streams into Hadoop.

This document describes the basics of Apache Flume and explain how to set up Flume agents for collecting fast-moving data streams (kafka as source) and pushing the data into Hadoop's filesystem.Flume can be used for variety of purposes some of them are

1. Created to deal with streaming data/logs to HDFS
2. Can’t “copy” to files to HDFS if the files aren’t closed.
3. Need to buffer “some”, then write and close a file — repeat  
     
    Flume deploys as one or more agents, each contained within its own instance of the Java Virtual Machine (JVM). Agents consist of three pluggable components: sources, sinks, and channels. An agent must have at least one of each in order to run. Sources collect incoming data as events. Sinks write events out, and channels provide a queue to connect the source and sink
4. **Sources**

Put simply, Flume sources listen for and consume events. Events can range from newline-terminated strings instdout to HTTP POSTs and RPC calls — it all depends on what sources the agent is configured to use. Flume agents may have more than one source, but must have at least one. Sources require a name and a type; the type then dictates additional configuration parameters.

On consuming an event, Flume sources write the event to a channel. Importantly, sources write to their channels as transactions. By dealing in events and transactions, Flume agents maintain end-to-end flow reliability. Events are not dropped inside a Flume agent unless the channel is explicitly allowed to discard them due to a full queue.

**Channels**

Channels are the mechanism by which Flume agents transfer events from their sources to their sinks. Events written to the channel by a source are not removed from the channel until a sink removes that event in a transaction. This allows Flume sinks to retry writes in the event of a failure in the external repository (such as HDFS or an outgoing network connection). For example, if the network between a Flume agent and a Hadoop cluster goes down, the channel will keep all events queued until the sink can correctly write to the cluster and close its transactions with the channel.

Channels are typically of two types: in-memory queues and durable disk-backed queues. In-memory channels provide high throughput but no recovery if an agent fails. File or database-backed channels, on the other hand, are durable. They support full recovery and event replay in the case of agent failure.

**Sinks**

Sinks provide Flume agents pluggable output capability — if you need to write to a new type storage, just write a Java class that implements the necessary classes. Like sources, sinks correspond to a type of output: writes to HDFS or HBase, remote procedure calls to other agents, or any number of other external repositories. Sinks remove events from the channel in transactions and write them to output. Transactions close when the event is successfully written, ensuring that all events are committed to their final destination.

Example: example has been exmplained at the end of document with kafka and flume integration

**Apache Kafka:**  
  
A fast, scalable, fault-tolerant messaging system

Apache™ Kafka is a fast, scalable, durable, and fault-tolerant publish-subscribe messaging system. Kafka is often used in place of traditional message brokers like JMS and AMQP because of its higher throughput, reliability and replication.

Kafka works in combination with Apache Storm, Apache HBase and Apache Spark for real-time analysis and rendering of streaming data. Kafka can message geospatial data from a fleet of long-haul trucks or sensor data from heating and cooling equipment in office buildings. Whatever the industry or use case, Kafka brokers massive message streams for low-latency analysis in Enterprise Apache Hadoop.

## **What Kafka Does**

Apache Kafka supports a wide range of use cases as a general-purpose messaging system for scenarios where high throughput, reliable delivery, and horizontal scalability are important. Apache Storm and Apache HBase both work very well in combination with Kafka. Common use cases include:

* Stream Processing
* Website Activity Tracking
* Metrics Collection and Monitoring
* Log Aggregation

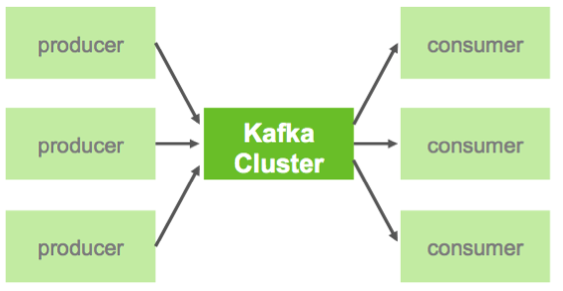
Some of the important characteristics that make Kafka such an attractive option for these use cases include the following:

| Feature | Description |
| --- | --- |
| **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 |

## How Kafka Works

Kafka’s system design can be thought of as that of a distributed commit log, where incoming data is written sequentially to disk. There are four main components involved in moving data in and out of Kafka:

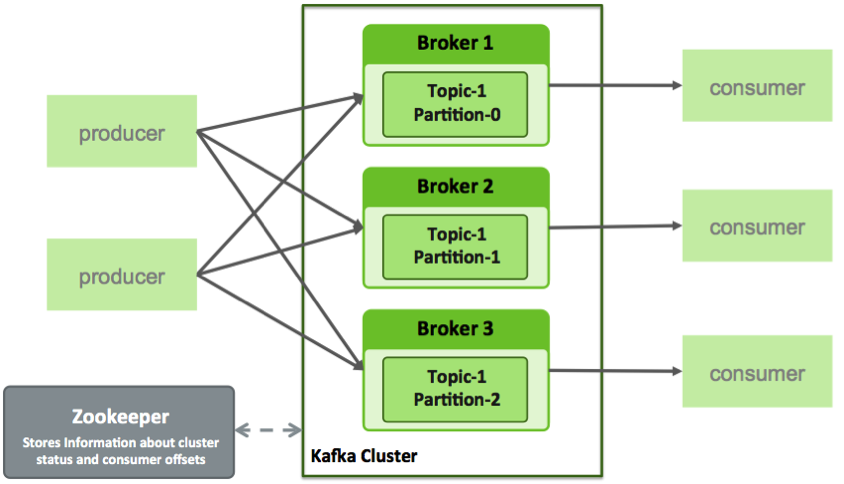
* Topics
* Producers
* Consumers
* Brokers



In Kafka, a **Topic** is a user-defined category to which messages are published. Kafka **Producers** publish messages to one or more topics and **Consumers** subscribe to topics and process the published messages. Finally, a Kafka cluster consists of one or more servers, called **Brokers** that manage the persistence and replication of message data (i.e. the commit log).

One of the keys to Kafka’s high performance is the simplicity of the brokers’ responsibilities. In Kafka, topics consist of one or more Partitions that are ordered, immutable sequences of messages. Since writes to a partition are sequential, this design greatly reduces the number of hard disk seeks (with their resulting latency).

Another factor contributing to Kafka’s performance and scalability is the fact that Kafka brokers are not responsible for keeping track of what messages have been consumed – that responsibility falls on the consumer. In traditional messaging systems such as JMS, the broker bore this responsibility, severely limiting the system’s ability to scale as the number of consumers increased.



For Kafka consumers, keeping track of which messages have been consumed (processed) is simply a matter of keeping track of an **Offset**, which is a sequential id number that uniquely identifies a message within a partition. Because Kafka retains all messages on disk (for a configurable amount of time), consumers can rewind or skip to any point in a partition simply by supplying an offset value. Finally, this design eliminates the potential for back-pressure when consumers process messages at different rates.

## 

## Flume vs Kafka

people get confused about where to use kafka and where to use flume. In most of cases we use both flume and kafka. Where kafka will be sink for flume.

*"What tool is the best for transporting* data*/logs across servers in a system?"*  
  
 This question is very commonly faced when designing a big-data system.  
Here i try to compare both these tools and see which is the best suited for the job.

## Problems targeted by these systems

Flume is designed to ease the ingestion of data from one component to other.  
It's focus is mostly on Hadoop although now it has sources and sinks for several other tools also, like Solr.  
  
 Kafka on the other hand is a messaging system that can store data for several days (depending on the data size of-course).  
 Kafka focuses more on the pipe while Flume focuses more on the end-points of the pipe.That's why Kafka does not provide any sources or sinks specific to any component like Hadoop or Solr.  
 It just provides a reliable way of getting the data across from one system to another.Kafka uses partitioning for achieving higher throughput of writes and uses replication for reliability and higher read throughput.

## Push / Pull

Flume pushes data while Kafka needs the consumers to pull the data. Due to push nature, Flume needs some work at the consumers' end for replicating data-streams to multiple sinks. With Kafka, each consumer manages its own read pointer, so its relatively easy to replicate channels in Kafka and also much easier to parallelize data-flow into multiple sinks like Solr and Hadoop.

**Latest trend is to use both Kafka and Flume together.**

KafkaSource and KafkaSink for Flume are available which help in doing so.The combination of these two gives a very desirable system because Flume's primary effort is to help ingest data into Hadoop and doing this in Kafka without Flume is a significant effort. Also note that Flume by itself is not sufficient for processing streams of data as Flume's primary purpose is not to store and replicate data streams. Hence it would be poor system if it uses only a single one of these two tools.

## Persistence storage on disk

Kafka keeps the messages on disk till the time it is configured to keep them.  
Thus if a Kafka broker goes offline for a while and comes back, the messages are not lost.  
Flume also maintains a write-ahead-log which helps it to restore messages during a crash.

## Flume error handling and transactional writes

Flume is meant to pass messages from source to sink (All of which implement Flume interfaces for get and put, thus treating Flume as an adapter). For example, a Flume log reader could send messages to a Flume sink which duplicates the incoming stream to Hadoop Flume Sink and Solr Flume Sink.

For a chained system of Flume sources and sinks, Flume achieves reliability by using transactions - a sending Flume client does not close its write transaction unless the receiving client writes the data to its own Write-Ahead-Log and informs the sender about the same. If the receiver does not acknowledge the writing of WAL to the sender, then the sender marks this as a failure. The sender then begins to buffer all such events unless it can no longer hold any more. At this point, it begins to reject writes from its own upstream clients as

**Flume**:

* Data is pushed to the the destination.
* Built around Hadoop ecosystem for the primary purpose of sending messages to HDFS & HBase
* Flume does not replicate events - in case of flume-agent failure, you will lose events in the channel

**Kafka**:

* A general purpose distributed publish-subscribe messaging system - you can have a topic which many listeners can subscribe to so the processing of messages can happen in parallel on various channels.
* Can be used for any system to connect to other systems that requires enterprise level messaging (website activity tracking, operational metrics, stream processing etc)
* High availability of events(recoverable in case of failures)

**Example:**

this example will cover starting kafka server, producer and consumer. And finnaly flume to use kafka as source.  
  
1. Start the server

Kafka uses ZooKeeper so you need to first start a ZooKeeper server if you don't already have one. You can use the convenience script packaged with kafka to get a quick-and-dirty single-node ZooKeeper instance.

> **bin/zookeeper-server-start.sh config/zookeeper.properties**

Now start the Kafka server:

> **bin/kafka-server-start.sh config/server.properties**

#### Step 2: Create a topic

Let's create a topic named "test" with a single partition and only one replica:

> **bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic test**

We can now see that topic if we run the list topic command:

> **bin/kafka-topics.sh --list --zookeeper localhost:2181**

test

Alternatively, instead of manually creating topics you can also configure your brokers to auto-create topics when a non-existent topic is published to.

#### Step 3: Send some messages

Kafka comes with a command line client that will take input from a file or from standard input and send it out as messages to the Kafka cluster. By default each line will be sent as a separate message.

Run the producer and then type a few messages into the console to send to the server.

> **bin/kafka-console-producer.sh --broker-list localhost:9092 --topic test**

**This is a message**

**This is another message**

#### Step 4: Start a consumer

Kafka also has a command line consumer that will dump out messages to standard output.

> **bin/kafka-console-consumer.sh --zookeeper localhost:2181 --topic test --from-beginning**

This is a message

This is another message

**Flume with kafka**

**Step 1:  
  
first** step is to create **tier\_agent.properties file and** paste following code  
  
tier1.sources = source1

tier1.channels = channel1

tier1.sinks = sink1

tier1.sources.source1.type = org.apache.flume.source.kafka.KafkaSource

tier1.sources.source1.zookeeperConnect = 127.0.0.1:2181

tier1.sources.source1.topic = test

tier1.sources.source1.groupId = flume

tier1.sources.source1.channels = channel1

tier1.sources.source1.kafka.consumer.timeout.ms = 100

tier1.channels.channel1.type = memory

tier1.channels.channel1.capacity = 10000

tier1.channels.channel1.transactionCapacity = 1000

tier1.sinks.sink1.type = logger

tier1.sinks.sink1.channel = channel1

Step 2: Run following command from flume directory

**bin/flume-ng agent -n tier1 --conf conf -f /home/datametica/apache-flume-1.6.0-bin/conf/tier\_agent.properties -Dflume.root.logger=INFO,console**We are using logger as sink. So output will be print on logger if you want to change sink to hdfs make following changes

tier1.sinks.sink1.type = hdfs

tier1.sinks.sink1.hdfs.path = /user/root/hcsc

tier1.sinks.sink1.hdfs.rollInterval = 5

tier1.sinks.sink1.hdfs.rollSize = 0

tier1.sinks.sink1.hdfs.rollCount = 0

tier1.sinks.sink1.hdfs.fileType = DataStream

once flume started running. Start producing data from kafka producer you will be able to see data on destination location