

Downloads (<http://www.cloudera.com/content/www/en-us/downloads.html>)  
Training (<http://www.cloudera.com/content/www/en-us/training.html>)  
Support Portal (<http://www.cloudera.com/content/www/en-us/support.html>)  
Partners (<http://www.cloudera.com/content/www/en-us/partners.html>)  
Developers (<http://www.cloudera.com/content/www/en-us/developers.html>)  
Community (<http://community.cloudera.com/>)

**cloudera**  
(<http://www.cloudera.com>)

# Cloudera Engineering Blog

(<http://blog.cloudera.com/>)

Best practices, how-tos, use cases, and internals from Cloudera Engineering and the community

SEARCH

## Architectural Patterns for Near Real-Time Data Processing with Apache Hadoop

June 1, 2015 (<http://blog.cloudera.com/blog/2015/06/architectural-patterns-for-near-real-time-data-processing-with-apache-hadoop/>) | By Ted Malaska ([http://blog.cloudera.com/?guest-author=Ted Malaska](http://blog.cloudera.com/?guest-author=Ted%20Malaska)) | 3 Comments (<http://blog.cloudera.com/blog/2015/06/architectural-patterns-for-near-real-time-data-processing-with-apache-hadoop/#comments>)


Categories: Data Ingestion (<http://blog.cloudera.com/blog/category/data-ingestion/>)

Flume (<http://blog.cloudera.com/blog/category/flume/>) Hadoop (<http://blog.cloudera.com/blog/category/hadoop/>) HBase (<http://blog.cloudera.com/blog/category/hbase/>) Kafka (<http://blog.cloudera.com/blog/category/kafka/>) Spark (<http://blog.cloudera.com/blog/category/spark/>)

Tweets by  
[@ClouderaEng](#)

 **Cloudera En...**  
[@ClouderaE...](#)

Hi [@noiano](#). Here is a [@ClouderaComm](#) article on CDH versioning that may provide some clarity. [community.cloudera.com/t/5/Cloudera-Ma...](http://community.cloudera.com/t/5/Cloudera-Ma...)

 **Cloud...**  
Cloud...  
comm...

Embed

View on Twitter

## Evaluating which streaming architectural pattern is the best match to your use case is a precondition for a successful production deployment.

The Apache Hadoop ecosystem has become a preferred platform for enterprises seeking to process and understand large-scale data in real time. Technologies like Apache Kafka, Apache Flume, Apache Spark, Apache Storm, and Apache Samza are increasingly pushing the envelope on what is possible. It is often tempting to bucket large-scale streaming use cases together but in reality they tend to break down into a few different architectural patterns, with different components of the ecosystem better suited for different problems.

In this post, I will outline the four major streaming patterns that we have encountered with customers running enterprise data hubs in production, and explain how to implement those patterns architecturally on Hadoop.

### Streaming Patterns

The four basic streaming patterns (often used in tandem) are:

- **Stream ingestion:** Involves low-latency persisting of events to HDFS, Apache HBase, and Apache Solr.
- **Near Real-Time (NRT) Event Processing with External Context:** Takes actions like alerting, flagging, transforming, and filtering of events as they arrive. Actions might be taken based on sophisticated criteria, such as anomaly detection models. Common use cases, such as NRT fraud detection and recommendation, often demand low latencies under 100 milliseconds.
- **NRT Event Partitioned Processing:** Similar to NRT event processing, but deriving benefits from partitioning the data—like storing more relevant external information in memory. This pattern also requires processing latencies under 100 milliseconds.
- **Complex Topology for Aggregations or ML:** The holy grail of stream processing: gets real-time answers from data with

### Categories

[Accumulo](http://blog.cloudera.com/blog/category/accumulo/)  
(<http://blog.cloudera.com/blog/category/accumulo/>) (1)  
[Avro](http://blog.cloudera.com/blog/category/avro/)  
(<http://blog.cloudera.com/blog/category/avro/>) (21)  
[Bigtop](http://blog.cloudera.com/blog/category/bigtop/)  
(<http://blog.cloudera.com/blog/category/bigtop/>)

### Popular

[Most Popular](http://blog.cloudera.com/most-popular/)  
(<http://blog.cloudera.com/most-popular/>)  
[Most Popular in /spark/](http://blog.cloudera.com/most-popular-in-spark/)  
(<http://blog.cloudera.com/most-popular-in-spark/>)

### Tags

[analysis](http://blog.cloudera.com/blog/tag/analysis/)  
(<http://blog.cloudera.com/blog/tag/analysis/>) [analytics](http://blog.cloudera.com/blog/tag/analytics/)  
(<http://blog.cloudera.com/blog/tag/analytics/>) [apache](http://blog.cloudera.com/blog/tag/apache/)  
(<http://blog.cloudera.com/blog/tag/apache/>) [apache hadoop](http://blog.cloudera.com/blog/tag/apache-hadoop/)  
(<http://blog.cloudera.com/blog/tag/apache-hadoop/>) [Apache HBase](http://blog.cloudera.com/blog/tag/apache-hbase/)  
(<http://blog.cloudera.com/blog/tag/apache-hbase/>) [apache hive](http://blog.cloudera.com/blog/tag/apache-hive/)  
(<http://blog.cloudera.com/blog/tag/apache-hive/>) [beta](http://blog.cloudera.com/blog/tag/beta/)

a complex and flexible set of operations. Here, because results often depend on windowed computations and require more active data, the focus shifts from ultra-low latency to functionality and accuracy.

In the following sections, we'll get into recommended ways for implementing such patterns in a tested, proven, and maintainable way.

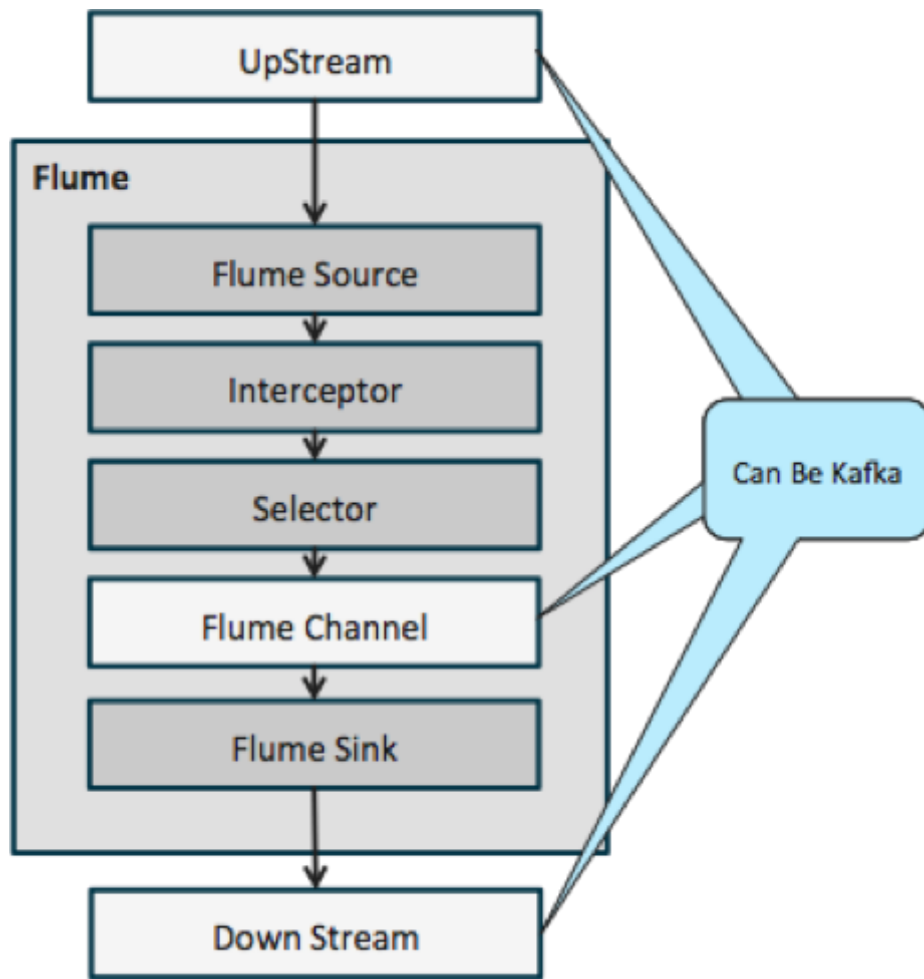
## Streaming Ingestion

Traditionally, Flume has been the recommended system for streaming ingestion. Its large library of sources and sinks cover all the bases of what to consume and where to write. (For details about how to configure and manage Flume, *Using Flume* (<http://shop.oreilly.com/product/0636920030348.do>), the O'Reilly Media book by Cloudera Software Engineer/Flume PMC member Hari Shreedharan, is a great resource.)

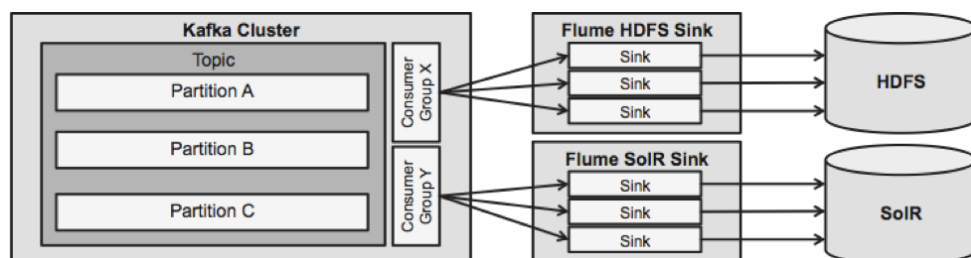
Within the last year, Kafka has also become popular because of powerful features such as playback and replication. Because of the overlap between Flume's and Kafka's goals, their relationship is often confusing. How do they fit together? The answer is simple: Kafka is a pipe similar to Flume's Channel abstraction, albeit a better pipe because of its support for the features mentioned above. One common approach is to use Flume for the source and sink, and Kafka for the pipe between them.

The diagram below illustrates how Kafka can serve as the UpStream Source of Data to Flume, the DownStream destination of Flume, or the Flume Channel.

(<http://blog.cloudera.com/blog/tag/beta/>) [Big Data](http://blog.cloudera.com/blog/tag/beta/)  
(<http://blog.cloudera.com/blog/tag/big-data/>) [CDH](http://blog.cloudera.com/blog/tag/big-data/)  
(<http://blog.cloudera.com/blog/tag/cdh/>) [cloudera](http://blog.cloudera.com/blog/tag/cdh/)  
(<http://blog.cloudera.com/blog/tag/cloudera/>) [Cloudera Manager](http://blog.cloudera.com/blog/tag/cloudera/)  
(<http://blog.cloudera.com/blog/tag/cloudera-manager/>) [Community](http://blog.cloudera.com/blog/tag/cloudera-manager/)  
(<http://blog.cloudera.com/blog/tag/community/>) [configuration](http://blog.cloudera.com/blog/tag/community/)  
(<http://blog.cloudera.com/blog/tag/configuration/>) [data](http://blog.cloudera.com/blog/tag/configuration/)  
(<http://blog.cloudera.com/blog/tag/data/>) [developer](http://blog.cloudera.com/blog/tag/data/)  
(<http://blog.cloudera.com/blog/tag/developer/>) [developers](http://blog.cloudera.com/blog/tag/developer/)  
(<http://blog.cloudera.com/blog/tag/developers/>) [development](http://blog.cloudera.com/blog/tag/developers/)  
(<http://blog.cloudera.com/blog/tag/development/>) [events](http://blog.cloudera.com/blog/tag/development/)  
(<http://blog.cloudera.com/blog/tag/events-2/>) [Flume](http://blog.cloudera.com/blog/tag/events-2/)  
(<http://blog.cloudera.com/blog/tag/flume/>) [Hadoop](http://blog.cloudera.com/blog/tag/flume/)  
(<http://blog.cloudera.com/blog/tag/hadoop/>) [hadoop world](http://blog.cloudera.com/blog/tag/hadoop/)  
([http://blog.cloudera.com/blog/tag/hadoop\\_world/](http://blog.cloudera.com/blog/tag/hadoop_world/)) [HBase](http://blog.cloudera.com/blog/tag/hadoop_world/)  
([http://blog.cloudera.com/blog/tag/hadoop\\_world/](http://blog.cloudera.com/blog/tag/hadoop_world/))



The design illustrated below is massively scalable, battle hardened, centrally monitored through Cloudera Manager, fault tolerant, and supports replay.



(<http://blog.cloudera.com/wp-content/uploads/2015/06/streampatterns-f2.png>)

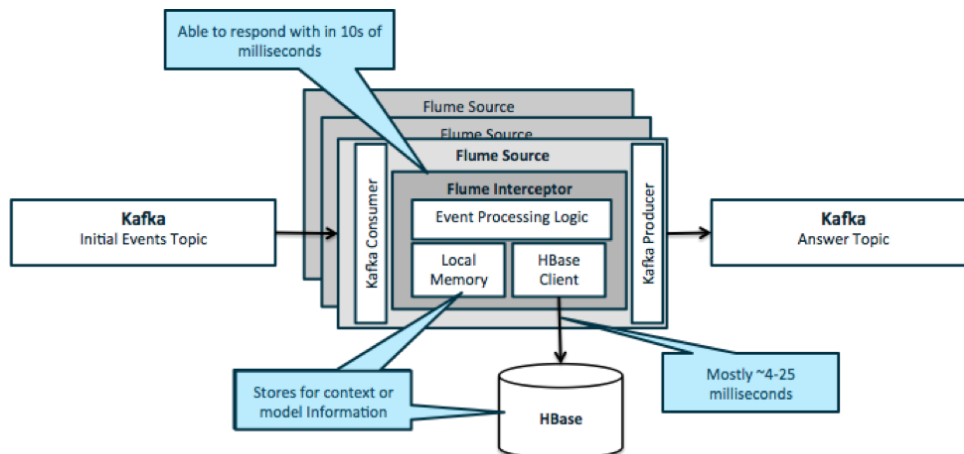
One thing to note before we go to the next streaming architecture is how this design gracefully handles failure. The Flume Sinks pull from a Kafka Consumer Group. The Consumer group track the Topic's offset with help from Apache ZooKeeper. If a Flume Sink is lost, the Kafka Consumer will redistribute the load to the remaining sinks. When the Flume Sink comes back up, the Consumer group will redistribute again.

## NRT Event Processing with External Context

[og/tag/hbase/](http://blog.cloudera.com/blog/tag/hbase/) HDFS  
[\(http://blog.cloudera.com/bl](http://blog.cloudera.com/blog/tag/hdfs/)  
[og/tag/hdfs/](http://blog.cloudera.com/blog/tag/hdfs/) Hive  
[\(http://blog.cloudera.com/bl](http://blog.cloudera.com/blog/tag/hive/)  
[og/tag/hive/](http://blog.cloudera.com/blog/tag/hive/) Hue  
[\(http://blog.cloudera.com/bl](http://blog.cloudera.com/blog/tag/hue/)  
[og/tag/hue/](http://blog.cloudera.com/blog/tag/hue/) impala  
[\(http://blog.cloudera.com/bl](http://blog.cloudera.com/blog/tag/impala-2/)  
[og/tag/impala-2/](http://blog.cloudera.com/blog/tag/impala-2/)  
[installation](http://blog.cloudera.com/blog/tag/impala-2/)  
[\(http://blog.cloudera.com/bl](http://blog.cloudera.com/blog/tag/installation/)  
[og/tag/installation/](http://blog.cloudera.com/blog/tag/installation/) java  
[\(http://blog.cloudera.com/bl](http://blog.cloudera.com/blog/tag/java/)  
[og/tag/java/](http://blog.cloudera.com/blog/tag/java/) log  
[\(http://blog.cloudera.com/bl](http://blog.cloudera.com/blog/tag/log/)  
[og/tag/log/](http://blog.cloudera.com/blog/tag/log/) logs  
[\(http://blog.cloudera.com/bl](http://blog.cloudera.com/blog/tag/logs/)  
[og/tag/logs/](http://blog.cloudera.com/blog/tag/logs/) MapReduce  
[\(http://blog.cloudera.com/bl](http://blog.cloudera.com/blog/tag/mapreduce/)  
[og/tag/mapreduce/](http://blog.cloudera.com/blog/tag/mapreduce/)  
[monitoring](http://blog.cloudera.com/blog/tag/monitoring/)  
[\(http://blog.cloudera.com/bl](http://blog.cloudera.com/blog/tag/monitoring/)  
[og/tag/monitoring/](http://blog.cloudera.com/blog/tag/monitoring/) open  
[source](http://blog.cloudera.com/blog/tag/open-source/)  
[\(http://blog.cloudera.com/bl](http://blog.cloudera.com/blog/tag/open-source/)  
[og/tag/open-source/](http://blog.cloudera.com/blog/tag/open-source/) Pig  
[\(http://blog.cloudera.com/bl](http://blog.cloudera.com/blog/tag/pig/)  
[og/tag/pig/](http://blog.cloudera.com/blog/tag/pig/) platform  
[\(http://blog.cloudera.com/bl](http://blog.cloudera.com/blog/tag/platform/)  
[og/tag/platform/](http://blog.cloudera.com/blog/tag/platform/) python  
[\(http://blog.cloudera.com/bl](http://blog.cloudera.com/blog/tag/python/)  
[og/tag/python/](http://blog.cloudera.com/blog/tag/python/) questions  
[\(http://blog.cloudera.com/bl](http://blog.cloudera.com/blog/tag/questions/)  
[og/tag/questions/](http://blog.cloudera.com/blog/tag/questions/) release  
[\(http://blog.cloudera.com/bl](http://blog.cloudera.com/blog/tag/release/)  
[og/tag/release/](http://blog.cloudera.com/blog/tag/release/) REST  
[\(http://blog.cloudera.com/bl](http://blog.cloudera.com/blog/tag/rest/)  
[og/tag/rest/](http://blog.cloudera.com/blog/tag/rest/) Search  
[\(http://blog.cloudera.com/bl](http://blog.cloudera.com/blog/tag/search/)

To reiterate, a common use case for this pattern is to look at events streaming in and make immediate decisions, either to transform the data or to take some sort of external action. The decision logic often depends on external profiles or metadata. An easy and scalable way to implement this approach is to add a Source or Sink Flume interceptor to your Kafka/Flume architecture. With modest tuning, it's not difficult to achieve latencies in the low milliseconds.

Flume Interceptors take events or batches of events and allow user code to modify or take actions based on them. The user code can interact with local memory or an external storage system like HBase to get profile information needed for decisions. HBase usually can give us our information in around 4-25 milliseconds depending on network, schema design, and configuration. You can also set up HBase in a way that it is never down or interrupted, even in the case of failure.



<http://blog.cloudera.com/wp-content/uploads/2015/06/streampatterns-f3.png>

Implementation requires nearly no coding beyond the application-specific logic in the interceptor. Cloudera Manager offers an intuitive UI for deploying this logic through parcels as well as hooking up, configuring, and monitoring the services.

## NRT Partitioned Event Processing with External Context

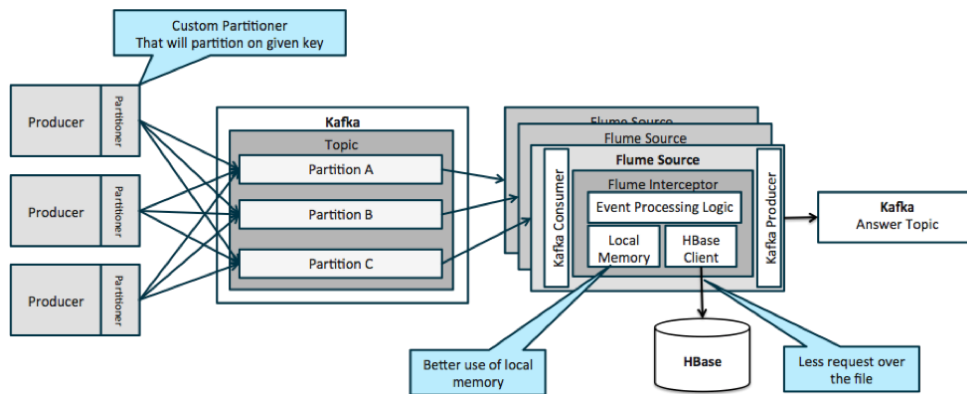
In the architecture illustrated below (unpartitioned solution), you would need to call out frequently to HBase because external context relevant to particular events does not fit in local memory on the Flume interceptors.

[og/tag/search/\) security](#)  
(<http://blog.cloudera.com/blog/tag/security/>) [sql](#)  
(<http://blog.cloudera.com/blog/tag/sql/>) [Support](#)  
(<http://blog.cloudera.com/blog/tag/support-2/>) [Testing](#)  
(<http://blog.cloudera.com/blog/tag/testing/>) [use cases](#)  
(<http://blog.cloudera.com/blog/tag/use-cases/>)

## Archives

### Archives

Select Month ▼



(<http://blog.cloudera.com/wp-content/uploads/2015/06/streampatterns-f4.png>)

However, if you define a key to partition your data, you can match incoming data to the subset of the context data that is relevant to it. If you partition the data 10 times, then you only need to hold 1/10th of the profiles in memory. HBase is fast, but local memory is faster. Kafka enables you to define a custom partitioner that it uses to split up your data.

Note that Flume is not strictly necessary here; the root solution here just a Kafka consumer. So, you could use just a consumer in YARN or a Map-only MapReduce application.

## Complex Topology for Aggregations or ML

Up to this point, we have been exploring event-level operations. However, sometimes you need more complex operations like counts, averages, sessionization, or machine-learning model building that operate on batches of data. In this case, Spark Streaming is the ideal tool for several reasons:

- **It's easy to develop compared to other tools.**

Spark's rich and concise APIs make building out complex topologies easy.

- **Similar code for streaming and batch processing.**

With a few changes, the code for small batches in real time can be used for enormous batches offline. In addition to reducing code size, this approach reduces the time needed for testing and integration.

- **There's one engine to know.**

There is a cost that goes into training staff on the quirks and internals of distributed processing engines. Standardizing on Spark consolidates this cost for both streaming and batch.

- **Micro-batching helps you scale reliably.**

Acknowledging at a batch level allows for more throughput and allows for solutions without the fear of a double-send. Micro-batching also helps with sending changes to HDFS or HBase in terms of performance at scale.

- **Hadoop ecosystem integration is baked in.**

Spark has deep integration with HDFS, HBase, and Kafka.

- **No risk of data loss.**

Thanks to the WAL and Kafka, Spark Streaming avoids data loss in case of failure.

- **It's easy to debug and run.**

You can debug and step through your code Spark Streaming in a local IDE without a cluster. Plus, the code looks like normal functional programming code so it doesn't take much time for a Java or Scala developer to make the jump. (Python is also supported.)

- **Streaming is natively stateful.**

In Spark Streaming, state is a first-class citizen, meaning that it's easy to write stateful streaming applications that are resilient to node failures.

- **As the de facto standard, Spark is getting long-term investment from across the ecosystem.**

At the time of this writing, there were approximately 700 commits to Spark as a whole in the last 30 days—compared to other streaming frameworks such as Storm, with 15 commits during the same time.

- **You have access to ML libraries.**

Spark's MLib is becoming hugely popular and its functionality will only increase.

- **You can use SQL where needed.**


With Spark SQL, you can add SQL logic to your streaming application to reduce code complexity.

## Conclusion

There is a lot of power in streaming and several possible patterns, but as you have learned in this post, you can do really powerful things with minimal coding if you know which pattern matches up with your use case best.

*Ted Malaska is a Solutions Architect at Cloudera, a contributor to Spark, Flume, and HBase, and a co-author of the O'Reilly book, Hadoop Applications Architecture (<http://shop.oreilly.com/product/0636920033196.do>).*



 apache (<http://blog.cloudera.com/blog/tag/apache/>) Apache Flume

<http://blog.cloudera.com/blog/tag/apache-flume/> apache hadoop

<http://blog.cloudera.com/blog/tag/apache-hadoop/> Apache HBase

<http://blog.cloudera.com/blog/tag/apache-hbase/> apache zookeeper

<http://blog.cloudera.com/blog/tag/apache-zookeeper/> cloudera

<http://blog.cloudera.com/blog/tag/cloudera/> Cloudera Manager

<http://blog.cloudera.com/blog/tag/cloudera-manager/> configuration

<http://blog.cloudera.com/blog/tag/configuration/> data

<http://blog.cloudera.com/blog/tag/data/> developer

<http://blog.cloudera.com/blog/tag/developer/> events

<http://blog.cloudera.com/blog/tag/events-2/> Flume

<http://blog.cloudera.com/blog/tag/flume/> fraud

<http://blog.cloudera.com/blog/tag/fraud/> Hadoop

<http://blog.cloudera.com/blog/tag/hadoop/> HBase

<http://blog.cloudera.com/blog/tag/hbase/> HDFS

<http://blog.cloudera.com/blog/tag/hdfs/> java (<http://blog.cloudera.com/blog/tag/java/>)

large-scale data (<http://blog.cloudera.com/blog/tag/large-scale-data/>) libraries

<http://blog.cloudera.com/blog/tag/libraries/> MapReduce



[\(http://blog.cloudera.com/blog/tag/mapreduce/\)](http://blog.cloudera.com/blog/tag/mapreduce/) monitoring

[\(http://blog.cloudera.com/blog/tag/monitoring/\)](http://blog.cloudera.com/blog/tag/monitoring/) platform

[\(http://blog.cloudera.com/blog/tag/platform/\)](http://blog.cloudera.com/blog/tag/platform/) python

[\(http://blog.cloudera.com/blog/tag/python/\)](http://blog.cloudera.com/blog/tag/python/) solr

[\(http://blog.cloudera.com/blog/tag/solr/\)](http://blog.cloudera.com/blog/tag/solr/) sql (<http://blog.cloudera.com/blog/tag/sql/>)

## 3 responses on “Architectural Patterns for Near Real-Time Data Processing with Apache Hadoop”



vladi

June 14, 2015 at 11:35 pm

(<http://blog.cloudera.com/blog/2015/06/architectural-patterns-for-near-real-time-data-processing-with-apache-hadoop/comment-page-1/#comment-61895>)

Hello,

Nice post. One question

How do you avoid data loss in the producers?

e.g. in a case if there is a network outage between producer and Kafka?

Reply ↓ (<http://blog.cloudera.com/blog/2015/06/architectural-patterns-for-near-real-time-data-processing-with-apache-hadoop/?replytocom=61895#respond>)



Jorge (<http://www.jmachado.me>)

December 6, 2016 at 11:11 pm

(<http://blog.cloudera.com/blog/2015/06/architectural-patterns-for-near-real-time-data-processing-with-apache-hadoop/comment-page-1/#comment-77974>)

If you are using Fluentd or similar they usually buffer and try again later, thanks when Streaming gets tricky... check out Structured Streaming on Spark 2.0

Reply ↓ (<http://blog.cloudera.com/blog/2015/06/architectural-patterns-for-near-real-time-data-processing-with-apache-hadoop/?replytocom=77974#respond>)

---



Hemil

January 17, 2017 at 9:25 am

(<http://blog.cloudera.com/blog/2015/06/architectural-patterns-for-near-real-time-data-processing-with-apache-hadoop/comment-page-1/#comment-78332>)

Great article to read on and also knowing about the Hadoop architecture book.

I have a question to know if it is possible to build a RDMA capability/support with any of these open source framework in real time application architecture. ?

Reply ↓ (<http://blog.cloudera.com/blog/2015/06/architectural-patterns-for-near-real-time-data-processing-with-apache-hadoop/?replytocom=78332#respond>)

---

## Leave a Reply

Your email address will not be published. Required fields are marked \*

**Name \***

**Email \***

**Website**

**Prove you're human! \***

1 × eight =

Comment \*

Post Comment

◀ Security, Hive-on-Spark, and Other Improvements in Apache Hive 1.2.0 (http://blog.cloudera.com/blog/2015/05/security-sensitive-data-hive-on-spark-and-other-improvements-in-apache-hive-1-2-0/) New in CDH 5.4: Sensitive Data Redaction ▶ (http://blog.cloudera.com/blog/2015/05/new-in-cdh-5-4-sensitive-data-redaction/)

## **Partners**

**(<https://www.cloudera.com/partners.html>)**

## **Developers**

**(<https://www.cloudera.com/developers.html>)**

## **Community**

**(<http://community.cloudera.com/>)**

## **Resources**

**(<https://www.cloudera.com/resources.html>)**

## **Documentation**

**(<https://www.cloudera.com/documentation>)**

(<https://www.linkedin.com/company/cloudera>)

(<https://www.facebook.com/cloudera>)

(<https://twitter.com/cloudera>)

(<https://www.cloudera.com/contact-us.html>)

## **Terms & Conditions**

**(<https://www.cloudera.com/legal/terms-and-conditions.html>)** | **Policies**

**(<https://www.cloudera.com/legal/policies.html>)**

on.html)

Career

(http://jobs.jobvite.com/cloudera/)

Contact

(https://www.cloudera.com/contact-us.html)

United States: +1 888 789 1488

Outside the US: +1 650 362 0488