Hortonworks Data Platform

Using Apache Kafka

(Sep 30, 2015)

docs.hortonworks.com

Hortonworks Data Platform: Using Apache Kafka

The Hortonworks Data Platform, powered by Apache Hadoop, is a massively scalable and 100% open source platform for storing, processing and analyzing large volumes of data. It is designed to deal with data from many sources and formats in a very quick, easy and cost-effective manner. The Hortonworks Data Platform consists of the essential set of Apache Hadoop projects including MapReduce, Hadoop Distributed File System (HDFS), HCatalog, Pig, Hive, HBase, ZooKeeper and Ambari. Hortonworks is the major contributor of code and patches to many of these projects. These projects have been integrated and tested as part of the Hortonworks Data Platform release process and installation and configuration tools have also been included.

Unlike other providers of platforms built using Apache Hadoop, Hortonworks contributes 100% of our code back to the Apache Software Foundation. The Hortonworks Data Platform is Apache-licensed and completely open source. We sell only expert technical support, training and partner-enablement services. All of our technology is, and will remain, free and open source.

Please visit the Hortonworks Data Platform page for more information on Hortonworks technology. For more information on Hortonworks services, please visit either the Support or Training page. Feel free to contact us directly to discuss your specific needs.

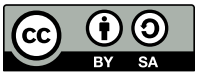


Table of Contents

List of Tables

1. Using Apache Kafka

Important

This chapter is intended as an introduction to Kafka.

The Kafka User Guide will be updated at a regular cadence over the next few months.

Apache Kafka is a fast, scalable, durable, fault-tolerant publish-subscribe messaging system.

Common use cases include:

- Messaging
- Website activity tracking
- Metrics collection and monitoring
- Log aggregation
- Stream processing
- Event sourcing
- Commit logs

Kafka works with Apache Storm and Apache Spark for real-time analysis and rendering of streaming data. The combination of messaging and processing technologies enables stream processing at linear scale.

For example, Apache Storm ships with full support for Kafka 0.8.2.1 as a data source, using Storm's core API or the higher-level, micro-batching Trident API. Storm's Kafka integration also includes support for writing data to Kafka, which enables complex data flows between components in a Hadoop-based architecture. For more information about Apache Storm, see the Storm User Guide.

1.1. An Overview of Kafka

Kafka operates on streams of messages. Four main components move messages in and out of Kafka:

Table 1.1. Kafka Components

Kafka Component	Description
Topic	A user-defined category (or feed name) to which messages are published.
Producer	A process that publishes messages to one or more topics.
Consumer	A process that subscribes to one or more topics and processes the feeds of messages from those topics.
Broker	A Kafka server that manages the persistence and replication of message data (i.e., the commit log).

Topics

Topics consist of one or more partitions. Kafka appends new messages to a partition in an ordered, immutable sequence. Each message in a topic is assigned a unique, sequential ID called an **offset**.

Producers

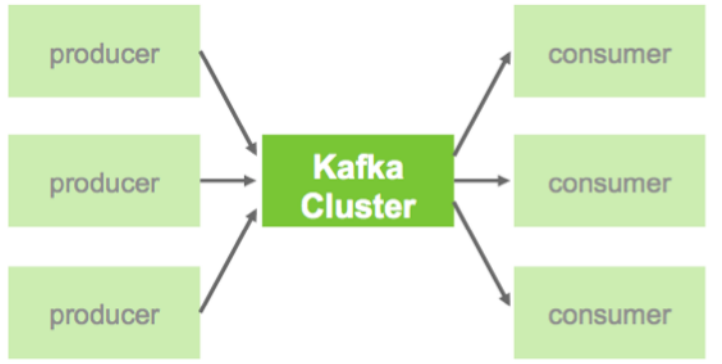
Kafka Producers publish messages to topics. The producer determines which message to assign to which partition within the topic. Assignment can be done in a round-robin fashion to balance load, or it can be based on a semantic partition function.

Consumers

Kafka Consumers keep track of which messages have already been consumed, or processed, by keeping track of an offset, 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.

Brokers and Clusters

A Kafka Cluster consists of one or more Brokers (server processes). Producers send messages to the Kafka Cluster, which in turn serves them to Consumers.

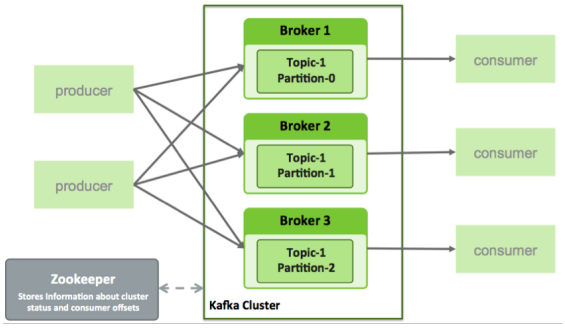


Performance

Partition support within topics provides parallelism within a topic. In addition, because writes to a partition are sequential, the number of hard disk seeks is minimized. This reduces latency and increases performance.

Kafka Brokers scale and perform well in part because Brokers are not responsible for keeping track of which messages have been consumed. The message Consumer is responsible for this. In traditional messaging systems such as JMS, the Broker bears this responsibility, which severely limits the system's ability to scale as the number of Consumers increase. Kafka's design eliminates the potential for back-pressure when consumers process messages at different rates.

For Kafka Consumers, keeping track of which messages have been consumed is simply a matter of keeping track of the offset – the sequential id 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.



Example

For an example that simulates the use of streaming geo-location information (using a previous version of Kafka), see Simulating and Transporting the Real-Time Event Stream with Apache Kafka.