Apache Kafka 3

# Kafka overview

**Date of Publish:** 2019-05-15

**https://docs.hortonworks.com/**

# Contents

# What is new in Apache Kafka 2.0

Apache Kafka 2.0 introduces some important enhancements and new features.

- The replication protocol has been improved to avoid log divergence between leader and follower during fast leader failover. We have also improved resilience of brokers by reducing the memory footprint of message down-conversions. By using message chunking, both memory usage and memory reference time have been reduced to avoid OutOfMemory errors in brokers.
- KIP-255 adds a framework for authenticating to Kafka brokers using OAuth2 bearer tokens. The SASL/OAUTHBEARER implementation is customizable using callbacks for token retrieval and validation.
- Host name verification is now enabled by default for SSL connections to ensure that the default SSL configuration is not susceptible to man-in-the-middle attacks. You can disable this verification if required.
- You can now dynamically update SSL truststores without broker restart. You can also configure security for broker listeners in ZooKeeper before starting brokers, including SSL keystore and truststore passwords and JAAS configuration for SASL. With this new feature, you can store sensitive password configs in encrypted form in ZooKeeper rather than in cleartext in the broker properties file.
- Kafka clients are now notified of throttling before any throttling is applied when quotas are enabled. This enables clients to distinguish between network errors and large throttle times when quotas are exceeded.
- We have added a configuration option for Kafka consumer to avoid indefinite blocking in the consumer.
- We have dropped support for Java 7 and removed the previously deprecated Scala producer and consumer.

# Building a High-Throughput Messaging System with Apache Kafka

Apache Kafka is a fast, scalable, durable, fault-tolerant publish-subscribe messaging system. Common use cases include:

- Stream processing
- Messaging
- Website activity tracking
- Metrics collection and monitoring
- Log aggregation
- Event sourcing
- Distributed commit logging

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 support for Kafka 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.

# Apache Kafka Concepts

This chapter describes several basic concepts that support fault-tolerant, scalable messaging provided by Apache Kafka:

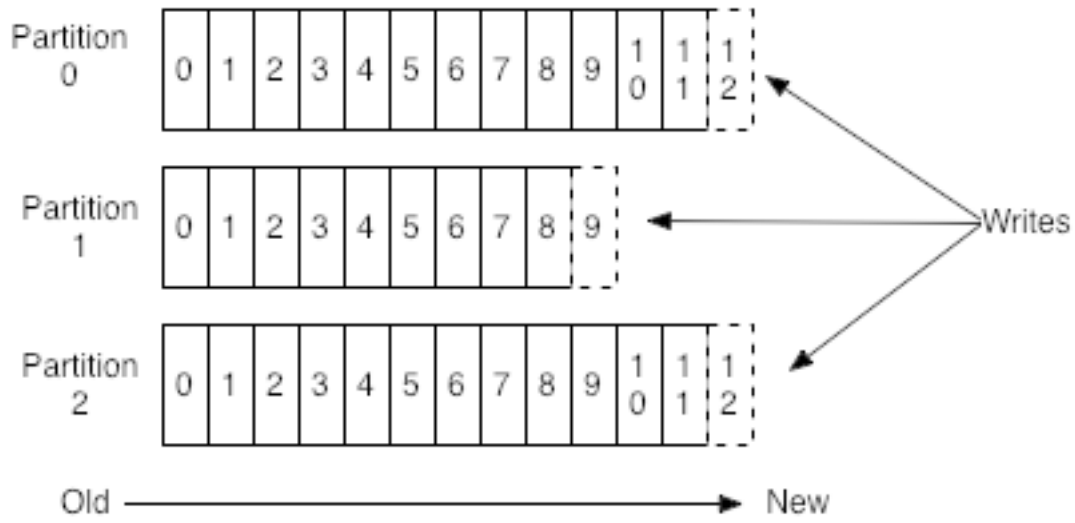
- Topics
- Producers
- Consumers

- Brokers

Topics

Kafka maintains feeds of messages in categories called topics. Each topic has a user-defined category (or feed name), to which messages are published.

For each topic, the Kafka cluster maintains a structured commit log with one or more partitions:




Kafka appends new messages to a partition in an ordered, immutable sequence. Each message in a topic is assigned a sequential number that uniquely identifies the message within a partition. This number is called an offset, and is represented in the diagram by numbers within each cell (such as 0 through 12 in partition 0).

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

Producers

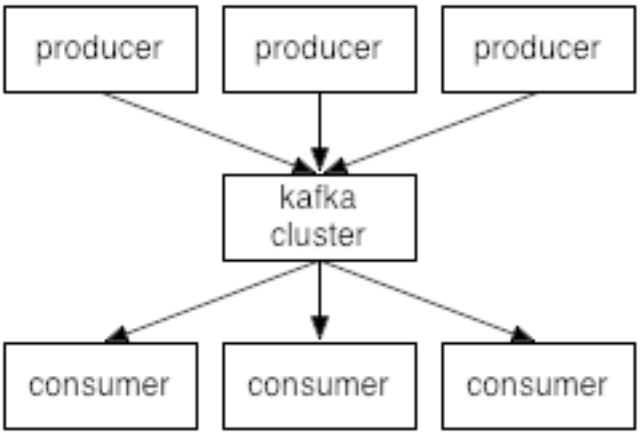
Producers are processes that publish messages to one or more Kafka topics. The producer is responsible for choosing which message to assign to which partition within a topic. Assignment can be done in a round-robin fashion to balance load, or it can be based on a semantic partition function.

Consumers

Consumers are processes that subscribe to one or more topics and process the feeds of published messages from those topics. Kafka consumers keep track of which messages have already been consumed by storing the current offset. Because Kafka retains all messages on disk for a configurable amount of time, consumers can use the offset to rewind or skip to any point in a partition.

Brokers

A Kafka cluster consists of one or more servers, each of which is called a broker. Producers send messages to the Kafka cluster, which in turn serves them to consumers. Each broker manages the persistence and replication of message data.




Kafka Brokers scale and perform well in part because Brokers are not responsible for keeping track of which messages have been consumed. Instead, the message consumer is responsible for this. This design feature eliminates the potential for back-pressure when consumers process messages at different rates.