Chapter 1. Introducing Kafka

In today's world, real-time information is continuously being generated by applications (business, social, or any other type), and this information needs easy ways to be reliably and quickly routed to multiple types of receivers. Most of the time, applications that produce information and applications that are consuming this information are well apart and inaccessible to each other. These heterogeneous application leads to redevelopment for providing an integration point between them. Therefore, a mechanism is required for the seamless integration of information from producers and consumers to avoid any kind of application rewriting at either end.

Welcome to the world of Apache Kafka

In the present big-data era, the very first challenge is to collect the data as it is a huge amount of data and the second challenge is to analyze it. This analysis typically includes the following types of data and much more:

- User behavior data
- Application performance tracing
- Activity data in the form of logs
- Event messages

Message publishing is a mechanism for connecting various applications with the help of messages that are routed between—for example, by a message broker such as Kafka. Kafka is a solution to the real-time problems of any software solution; that is to say, dealing with real-time volumes of information and routing it to multiple consumers quickly. Kafka provides seamless integration between information from producers and consumers without blocking the producers of the information and without letting producers know who the final consumers are.

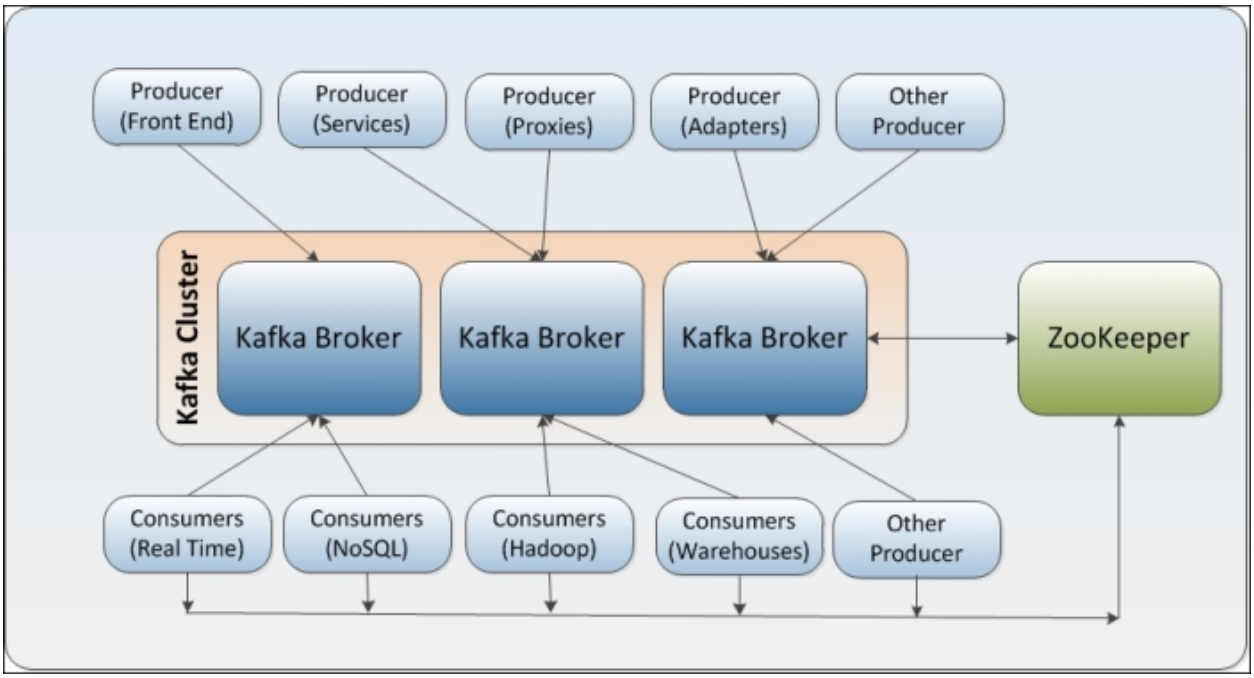
Apache Kafka is an open source, distributed, partitioned, and replicated commit-log-based publish-subscribe messaging system, mainly designed with the following characteristics:

- **Persistent messaging**: To derive the real value from big data, any kind of information loss cannot be afforded. Apache Kafka is designed with O(1) disk structures that provideconstant-time performance even with very large volumes of stored messages that are in the order of TBs. With Kafka, messages are persisted on disk as well as replicated within the cluster to prevent data loss.
- **High throughput**: Keeping big data in mind, Kafka is designed to work on commodity hardware and to handle hundreds of MBs of reads and writes per second from large number of clients.
- **Distributed**: Apache Kafka with its cluster-centric design explicitly supports message partitioning over Kafka servers and distributing consumption over a

cluster of consumer machines while maintaining per-partition ordering semantics. Kafka cluster can grow elastically and transparently without any downtime.
- **Multiple client support**: The Apache Kafka system supports easy integration of clients from different platforms such as Java, .NET, PHP, Ruby, and Python.
- **Real time**: Messages produced by the producer threads should be immediately visible to consumer threads; this feature is critical to event-based systems such as **Complex Event Processing** (**CEP**) systems.

Kafka provides a real-time publish-subscribe solution that overcomes the challenges of consuming the real-time and batch data volumes that may grow in order of magnitude to be larger than the real data. Kafka also supports parallel data loading in the Hadoop systems.

The following diagram shows a typical big data aggregation-and-analysis scenario supported by the Apache Kafka messaging system:



On the production side, there are different kinds of producers, such as the following:

- Frontend web applications generating application logs
- Producer proxies generating web analytics logs
- Producer adapters generating transformation logs
- Producer services generating invocation trace logs

On the consumption side, there are different kinds of consumers, such as the following:

- Offline consumers that are consuming messages and storing them in Hadoop or traditional data warehouse for offline analysis

- Near real-time consumers that are consuming messages and storing them in any NoSQL datastore, such as HBase or Cassandra, for near real-time analytics
- Real-time consumers, such as Spark or Storm, that filter messages in-memory and trigger alert events for related groups

Why do we need Kafka?

A large amount of data is generated by companies having any form of web- or device-based presence and activity. Data is one of the newer ingredients in these Internet-based systems and typically includes user activity; events corresponding to logins; page visits; clicks; social networking activities such as likes, shares, and comments; and operational and system metrics. This data is typically handled by logging and traditional log aggregation solutions due to high throughput (millions of messages per second). These traditional solutions are the viable solutions for providing logging data to an offline analysis system such as Hadoop. However, the solutions are very limiting for building real-time processing systems.

According to the new trends in Internet applications, activity data has become a part of production data and is used to run analytics in real time. These analytics can be:

- Search-based on relevance
- Recommendations based on popularity, co-occurrence, or sentimental analysis
- Delivering advertisements to the masses
- Internet application security from spam or unauthorized data scraping
- Device sensors sending high-temperature alerts
- Any abnormal user behavior or application hacking

Real-time usage of these multiple sets of data collected from production systems has become a challenge because of the volume of data collected and processed.

Apache Kafka aims to unify offline and online processing by providing a mechanism for parallel load in Hadoop systems as well as the ability to partition real-time consumption over a cluster of machines. Kafka can be compared with Scribe or Flume as it is useful for processing activity stream data; but from the architecture perspective, it is closer to traditional messaging systems such as ActiveMQ or RabitMQ.

Kafka use cases

There are number of ways in which Kafka can be used in any architecture. This section discusses some of the popular use cases for Apache Kafka and the well-known companies that have adopted Kafka. The following are the popular Kafka use cases:

- **Log aggregation**: This is the process of collecting physical log files from servers and putting them in a central place (a file server or HDFS) for processing. Using

Kafka provides clean abstraction of log or event data as a stream of messages, thus taking away any dependency over file details. This also gives lower-latency processing and support for multiple data sources and distributed data consumption.

- **Stream processing**: Kafka can be used for the use case where collected data undergoes processing at multiple stages—an example is raw data consumed from topics and enriched or transformed into new Kafka topics for further consumption. Hence, such processing is also called stream processing.
- **Commit logs**: Kafka can be used to represent external commit logs for any large scale distributed system. Replicated logs over Kafka cluster help failed nodes to recover their states.
- **Click stream tracking**: Another very important use case for Kafka is to capture user click stream data such as page views, searches, and so on as real-time publish-subscribe feeds. This data is published to central topics with one topic per activity type as the volume of the data is very high. These topics are available for subscription, by many consumers for a wide range of applications including real-time processing and monitoring.
- **Messaging**: Message brokers are used for decoupling data processing from data producers. Kafka can replace many popular message brokers as it offers better throughput, built-in partitioning, replication, and fault-tolerance.

Some of the companies that are using Apache Kafka in their respective use cases are as follows:

- **LinkedIn** (www.linkedin.com): Apache Kafka is used at LinkedIn for the streaming of activity data and operational metrics. This data powers various products such as LinkedIn News Feed and LinkedIn Today, in addition to offline analytics systems such as Hadoop.
- **DataSift** (www.datasift.com): At DataSift, Kafka is used as a collector to monitor events and as a tracker of users' consumption of data streams in real time.
- **Twitter** (www.twitter.com): Twitter uses Kafka as a part of its Storm—a stream-processing infrastructure.
- **Foursquare** (www.foursquare.com): Kafka powers online-to-online and online-to-offline messaging at Foursquare. It is used to integrate Foursquare monitoring and production systems with Foursquare-and Hadoop-based offline infrastructures.
- **Square** (www.squareup.com): Square uses Kafka as a *bus* to move all system events through Square's various datacenters. This includes metrics, logs, custom events, and so on. On the consumer side, it outputs into Splunk, Graphite, or Esper-like real-time alerting.

NOTE

The source of the preceding information is https://cwiki.apache.org/confluence/display/KAFKA/Powered+By.

Installing Kafka

Kafka is an Apache project and its current version 0.8.1.1 is available as a stable release. This Kafka 0.8.x offers many advanced features compared to the older version (prior to 0.8.x). A few of its advancements are as follows:

- Prior to 0.8.x, any unconsumed partition of data within the topic could be lost if the broker failed. Now the partitions are provided with a replication factor. This ensures that any committed message would not be lost, as at least one replica is available.
- The previous feature also ensures that all the producers and consumers are replication-aware (the replication factor is a configurable property). By default, the producer's message sending request is blocked until the message is committed to all active replicas; however, producers can also be configured to commit messages to a single broker.
- Like Kafka producers, the Kafka consumer polling model changes to a long-pulling model and gets blocked until a committed message is available from the producer, which avoids frequent pulling.
- Additionally, Kafka 0.8.x also comes with a set of administrative tools, such as controlled cluster shutdown and the Lead replica election tool, for managing the Kafka cluster.

The major limitation with Kafka version 0.8.x is that it can't replace the version prior to 0.8, as it is not backward-compatible.

Coming back to installing Kafka, as a first step we need to download the available stable release (all the processes have been tested on 64-bit CentOS 6.4 OS and may differ on other kernel-based OS). Now let's see what steps need to be followed in order to install Kafka.

Installing prerequisites

Kafka is implemented in Scala and uses build tool **Gradle** to build Kafka binaries. Gradle is a build automation tool for Scala, Groovy, and Java projects that requires Java 1.7 or later.

Installing Java 1.7 or higher

Perform the following steps to install Java 1.7 or later:

1. Download the `jdk-7u67-linux-x64.rpm` release from Oracle's website: http://www.oracle.com/technetwork/java/javase/downloads/index.html.
2. Change the file mode as follows:

```
3. [root@localhost opt]#chmod +x jdk-7u67-linux-x64.rpm
```

4. Change to the directory in which you want to perform the installation. To do so, type the following command:

```
[root@localhost opt]# cd <directory path name>
```

   For example, to install the software in the `/usr/java/` directory, type the following command:

```
[root@localhost opt]# cd /usr/java
```

6. Run the installer using the following command:

```
[root@localhost java]# rpm -ivh jdk-7u67-linux-x64.rpm
```

8. Finally, add the environment variable `JAVA_HOME`. The following command will write the `JAVA_HOME` environment variable to the file `/etc/profile` that contains a system-wide environment configuration:

```
[root@localhost opt]# echo "export
JAVA_HOME=/usr/java/jdk1.7.0_67 " >> /etc/profile
```

Downloading Kafka

Perform the following steps to download Kafka release 0.8.1.1:

1. Download the current beta release of Kafka (0.8) into a folder on your filesystem (for example, `/opt`) using the following command:

```
[root@localhost opt]#wget
http://apache.tradebit.com/pub/kafka/0.8.1.1/kafka_2.9.2-
0.8.1.1.tgz
```

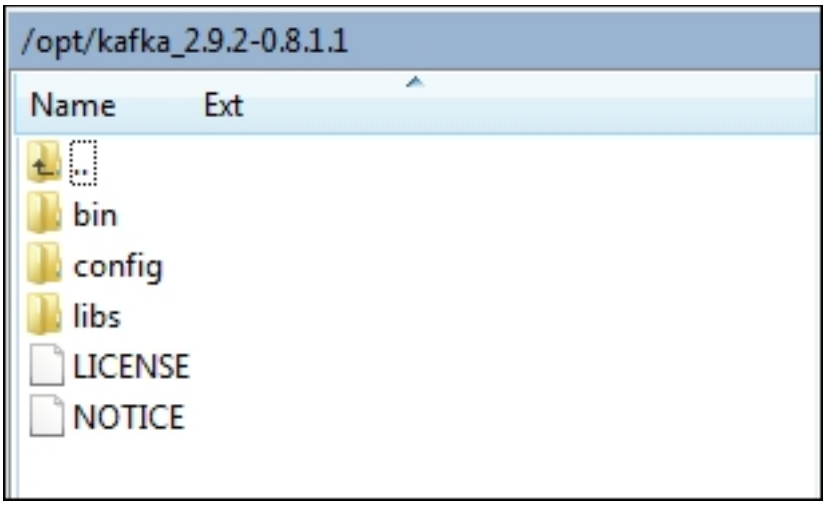
NOTE

The preceding URL may change. Check the correct download version and location at http://kafka.apache.org/downloads.html.

3. Extract the downloaded `kafka_2.9.2-0.8.1.1.tgz` file using the following command:

```
[root@localhost opt]# tar xzf kafka_2.9.2-0.8.1.1.tgz
```

5. After extraction of the `kafka_2.9.2-0.8.1.1.tgz` file, the directory structure for Kafka 0.8.1.1 looks as follows:



6. Finally, add the Kafka bin folder to `PATH` as follows:

```
[root@localhost opt]# export KAFKA_HOME=/opt/kafka_2.9.2-0.8.1.1
[root@localhost opt]# export PATH=$PATH:$KAFKA_HOME/bin
```

Building Kafka

The default Scala version that is used to build Kafka release 0.8.1.1 is Scala 2.9.2 but the Kafka source code can also be compiled from other Scala versions as well, such as 2.8.0, 2.8.2, 2.9.1, or 2.10.1. Use the following the command to build the Kafka source:

```
[root@localhost opt]# ./gradlew -PscalaVersion=2.9.1 jar
```

In Kafka 8.x onwards, the Gradle tool is used to compile the Kafka source code (available in `kafka-0.8.1.1-src.tgz`) and build the Kafka binaries (JAR files). Similar to Kafka JAR, the unit test or source JAR can also be built using the Gradle build tool. For more informationon build-related instructions, refer to https://github.com/apache/kafka/blob/0.8.1/README.md.

Summary

In this chapter, we have seen how companies are evolving the mechanism of collecting and processing application-generated data, and are learning to utilize the real power of this data by running analytics over it.

You also learned how to install 0.8.1.x. The following chapter discusses the steps required to set up single- or multi-broker Kafka clusters.