Apache Kafka Cookbook

Apache Kafka is a fault-tolerant, persistent queuing system that enables you to process large amounts of data in real time.

Apache Kafka Cookbook will give you details about how to manage and administer your Apache Kafka cluster. We will cover topics such as how to configure your broker, producer, and consumer for maximum efficiency in your specific situation. You will also learn how to maintain and administer your cluster for fault tolerance. We will also explore the tools provided by Apache Kafka to do regular maintenance operations. We shall also look at how to easily integrate Apache Kafka with big data tools such as Hadoop, Apache Spark, Apache Storm, and Elasticsearch.

What this book will do for you...

- Learn how to configure Kafka brokers for better efficiency
- Explore how to configure producers and consumers for optimal performance
- Set up tools for maintaining and operating Apache Kafka
- Create producers and consumers for Apache Kafka in Java
- Understand how Apache Kafka can be used by several third-party system for big data processing, such as Apache Storm, Apache Spark, Hadoop, and more
- Monitor Apache Kafka using tools such as Graphite and Ganglia

Inside the Cookbook...

- A straightforward and easy-to-follow format
- A selection of the most important tasks and problems
- Carefully organized instructions to solve problems efficiently
- Clear explanations of what you did
- Solutions that can be applied to solve real-world problems

Apache Kafka Cookbook

Over 50 hands-on recipes to efficiently administer, maintain, and use your Apache Kafka installation


Saurabh Minni

In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 1 'Initiating Kafka'
- A synopsis of the book’s content
- More information on *Apache Kafka Cookbook*
About the Author

Saurabh Minni has an BE in computer science and engineering. A polyglot programmer with over 10 years of experience, he has worked on a variety of technologies, including Assembly, C, C++, Java, Delphi, JavaScript, Android, iOS, PHP, Python, ZMQ, Redis, Mongo, Kyoto Tycoon, Cocoa, Carbon, Apache Storm, and Elasticsearch. In short, he is a programmer at heart, and loves learning new tech-related things each day.

Currently, he is working as a technical architect with Near (an amazing start-up that builds a location intelligence platform). Apart from handling several projects, he was also responsible for deploying Apache Kafka cluster. This was instrumental in streamlining the consumption of data in the big data processing systems, such as Apache Storm, Hadoop, and others at Near.

He has also reviewed *Learning Apache Kafka*, Packt Publishing.

He is reachable on Twitter at @the100rabh and on Github at https://github.com/the100rabh/.
Preface

Apache Kafka is a fault-tolerant persistent queuing system, which enables you to process large amount of data in real time. This guide will teach you how to maintain your Kafka cluster for maximum efficiency and easily connect it to your big data processing systems such as Hadoop or Apache Storm for quick processing.

This book will give you details about how to manage and administer your Apache Kafka cluster. We will cover topics such as how to configure your broker, producer, and consumers for maximum efficiency for your situation. You will also learn how to maintain and administer your cluster for fault tolerance. We will also explore the tools provided with Apache Kafka to do regular maintenance operations. We will also look at how to easily integrate Apache Kafka with big data tools such as Hadoop, Apache Spark, Apache Storm, and Elasticsearch.

What this book covers

Chapter 1, Initiating Kafka, lets you learn how to get things done in Apache Kafka via the command line.

Chapter 2, Configuring Brokers, covers the configuration of the Apache Kafka broker.

Chapter 3, Configuring a Consumer and Producer, covers the configuration of your consumers and producers in detail.

Chapter 4, Managing Kafka, walks you through some of the important operations that you might have performed for managing a Kafka cluster.

Chapter 5, Integrating Kafka with Java, explains how to integrate Apache Kafka in our Java code.

Chapter 6, Operating Kafka, explains how to do some of the important operations that need to be performed while running a Kafka cluster.
Chapter 7, *Integrating Kafka with Third-Party Platforms*, covers the basic methods of integrating Apache Kafka in various big data tools.

Chapter 8, *Monitoring Kafka*, walks you through the various steps of monitoring your Kafka cluster.
In this chapter, we will cover the following topics:

- Setting up multiple Kafka brokers
- Creating topics
- Sending some messages from the console
- Consuming from the console

Introduction

This chapter explains the basics of getting started with Kafka. We do not cover the theoretical details of Kafka, but the practical aspects of it. It assumes that you have already installed Kafka version 0.8.2 and ZooKeeper and have started a node as well. You understand that Kafka is a highly distributed messaging system that connects your data ingestion system to your real-time or batch processing systems such as Storm, Spark, or Hadoop. Kafka allows you to scale your systems very well in a horizontal fashion without compromising on speed or efficiency. You are now ready to get started with Kafka broker. We will discuss how you can do basic operations on your Kafka broker; this will also check whether the installation is working well. Since Kafka is usually used on Linux servers, this book assumes that you are using a similar environment. Though you can run Kafka on Mac OS X, similar to Linux, running Kafka on a Windows environment is a very complex process. There is no direct way of running Kafka on Windows, so we are keeping that out of this book. We are going to only consider bash environment for usage here.
Setting up multiple Kafka brokers

You can easily run Kafka in the standalone mode, but the real power of Kafka is unlocked when it is run in the cluster mode with replication and the topics are appropriately partitioned. It gives you the power of parallelism and data safety by making sure that, even if a Kafka node goes down, your data is still safe and accessible from other nodes. In this recipe, you will learn how to run multiple Kafka brokers.

Getting ready

I assume that you already have the experience of starting a Kafka node with the configuration files present at the Kafka install location. Change your current directory to the place where you have Kafka installed:

> cd /opt/kafka

How to do it...

To start multiple brokers, the first thing we need to do is we have to write the configuration files. For ease, you can start with the configuration file present in `config/server.properties` and perform the following steps.

1. For creating three different brokers in our single test machine, we will create two copies of the configuration file and modify them accordingly:

   > cp config/server.properties config/server-1.properties
   > cp config/server.properties config/server-2.properties

2. We need to modify these files before they can be used to start other Kafka nodes for our cluster. We need to change the `broker.id` property, which has to be unique for each broker in the cluster. The `port` number for Kafka to run and the location of the Kafka logs using `log.dir` needs to be specified. So, we will modify the files as follows:

   config/server-1.properties:
   
   broker.id=1
   port=9093
   log.dir=/tmp/kafka-logs-1

   config/server-2.properties:
   
   broker.id=2
   port=9094
   log.dir=/tmp/kafka-logs-2
3. You now need to start the Kafka brokers with this configuration file. This is assuming you have already started ZooKeeper and have a single Kafka node that is running:

```
> bin/kafka-server-start.sh config/server-1.properties &
...
> bin/kafka-server-start.sh config/server-2.properties &
```

**How it works...**

The `server.properties` files contain the configuration of your brokers. They all should point to the same ZooKeeper cluster. The `broker.id` property in each of the files is unique and defines the name of the node in the cluster. The `port` number and `log.dir` are changed so we can get them running on the same machine; else all the nodes will try to bind at the same port and will overwrite the data. If you want to run them on different machines, you need not change them.

**There's more...**

To run Kafka nodes on different servers, you also need to change the ZooKeeper connection string's details in the `config` file:

```
ZooKeeper.connect=localhost:2181
```

This is good if you are running Kafka off the same server as ZooKeeper; but in real life, you would be running them off different servers. So, you might want to change them to the correct ZooKeeper connection strings as follows:

```
ZooKeeper.connect=localhost:2181, 192.168.0.2:2181, 192.168.0.3:2181
```

This means that you are running the ZooKeeper cluster at the localhost nodes, 192.168.0.2 and 192.168.0.3, at the port number 2181.

**See also**

- Look at the configuration file in `config/server.properties` for details on several other properties that can also be set. You can also look it up online at [https://github.com/apache/kafka/blob/trunk/config/server.properties](https://github.com/apache/kafka/blob/trunk/config/server.properties).
Creating topics

Now that we have our cluster up and running, let's get started with other interesting things. In this recipe, you will learn how to create topics in Kafka that would be your first step toward getting things done using Kafka.

Getting ready

You must have already downloaded and set up Kafka. Now, in the command line, change to the Kafka directory. You also must have at least one Kafka node up and running.

How to do it...

1. It's very easy to create topics from the command line. Kafka comes with a built-in utility to create topics. You need to enter the following command from the directory where you have installed Kafka:

   ```
   > bin/kafka-topics.sh --create --ZooKeeper localhost:2181 --replication-factor 1 --partitions 1 --topic kafkatest
   ```

How it works...

What the preceding command does is that it creates a topic named test with a replication factor of 1 with 1 partition. You need to mention the ZooKeeper host and port number as well.

The number of partitions determines the parallelism that can be achieved on the consumer's side. So, it is important that the partition number is selected carefully based on how your Kafka data will be consumed.

The replication factor determines the number of replicas of this topic present in the cluster. There can be a maximum of one replica for a topic in each broker. This means that, if the number of replicas is more than the number of brokers, the number of replicas will be capped at the number of brokers.

There's more...

If you want to check whether your topic has been successfully created, you can run the following command:

```
> bin/kafka-topics.sh --list --ZooKeeper localhost:2181 kafkatest
```
This will print out all the topics that exist in the Kafka cluster. After successfully running the earlier command, your Kafka topic will be created and printed.

To get details of a particular topic, you can run the following command:

```
> bin/kafka-topics.sh --describe --ZooKeeper localhost:2181 --topic kafkatest
```

```
Topic:kafkatest   PartitionCount:1  ReplicationFactor:1 Configs:
```

```
Topic: kafkatest  Partition: 0  Leader: 0  Replicas: 0  Isr: 0
```

The explanation of the output is as follows:

- **PartitionCount**: The number of partitions existing for this particular topic.
- **ReplicationFactor**: The number of replicas that exist for this particular topic.
- **Leader**: The node responsible for the reading and writing operations of a given partition.
- **Replicas**: The list of nodes replicating the Kafka data. Some of these might be even dead.
- **ISR**: This is the list of nodes that are currently in-sync or in-sync replicas. It is a subset of all the replica nodes in the Kafka cluster.

We will create a topic with multiple replicas as shown by the following command:

```
> bin/kafka-topics.sh --create --ZooKeeper localhost:2181 --replication-factor 3 --partitions 1 --topic replicatedkafkatest
```

This will give the following output while checking for the details of the topic:

```
> bin/kafka-topics.sh --describe --ZooKeeper localhost:2181 --topic replicatedkafkatest
```

```
Topic:replicatedkafkatest   PartitionCount:1  ReplicationFactor:3  Configs:
```

```
Topic: replicatedkafkatest  Partition: 0  Leader: 2  Replicas: 2,0,1 Isr: 2,0,1
```

This means that there is a `replicatedkafkatest` topic, which has a single partition with replication factor of 3. All the three nodes are in-sync.

---

**Downloading the example code**

You can download the example code files from your account at http://www.packtpub.com for all the Packt Publishing books you have purchased. If you purchased this book elsewhere, you can visit http://www.packtpub.com/support and register to have the files e-mailed directly to you.
Sending some messages from the console

Kafka installation has a command-line utility that enables you to produce data. You can give a file as an input or you can give a standard input. It will send each line in these inputs as a message to the Kafka clusters.

Getting ready

As in the previous recipe, you must have already downloaded and set up Kafka. Now, in the command line, change to the Kafka directory. You have already started the Kafka nodes as mentioned in the previous recipes. You will need to create a topic as well. Now, you are ready to send some messages to Kafka from your console.

How to do it...

To send messages from the console perform the following steps:

1. You can run the next command followed by some text that will be sent to the server as messages:
   
   ```
   > bin/kafka-console-producer.sh --broker-list localhost:9092 -
   -topic kafkatest
   First message
   Second message
   ```

How it works...

The earlier inputs will push two messages to the kafkatest topic present in the Kafka running on localhost at port 9092.

There's more...

There are more parameters that you can pass to the console producer program. A short list of them is as follows:

- `-broker-list`: This specifies the ZooKeeper servers. It is followed by a list of the ZooKeeper server's hostname and port number that can be separated by a comma.
- `-topic`: This specifies the name of the topic. The name of the topic follows this parameter.
- `-sync`: This specifies that the messages should be sent synchronously—one at a time as they survive.
--compression-codec: This specifies the compression codec that will be used to produce the messages. It can be none, gzip, snappy, or lz4. If it is not specified, it will by default be set to gzip.

--batch-size: This specifies the number of messages to be sent in a single batch in case they are not being sent synchronously. This is followed by the batch's size value.

--message-send-max-retries: Brokers can sometimes fail to receive messages for a number of reasons and being unavailable transiently is just one of them. This property specifies the number of retries before a producer gives up and drops the message. This is followed by the number of retries that you want to set.

--retry-backoff-ms: Before each retry, the producer refreshes the metadata of relevant topics. Since leader election might take some time, it's good to specify some time before producer retries. This parameter does just that. This follows the time in ms.

This is a simple way of checking whether your broker with a topic is up and running as expected.

### Consuming from the console

You have produced some messages from the console, but it is important to check whether they can be read properly. For this, Kafka provides a command-line utility that enables you to consume messages. Each line of its output will be a message from the Kafka log.

#### Getting ready

As in the previous recipe, you must have already downloaded and set up Kafka. Now, in the command line, change to the Kafka directory. I would also assume that you have set up a Kafka node and created a topic with it. You also have to send some messages to a Kafka topic, as mentioned in the previous recipes, before you consume anything.

#### How to do it...

To consume the messages from the console perform the following steps:

1. You can run the following command and get the messages in Kafka as an output:

   ```
   $ bin/kafka-console-consumer.sh --zookeeper localhost:2181 --topic kafkatest --from-beginning
   First message
   Second message
   ```
**How it works...**

The given parameters are the ZooKeeper's `host` and `port` topic name `Optional` directive to start consuming the messages from the beginning instead of consuming the latest messages in the Kafka log.

This tells the program to get data from the Kafka logs under the given topic from the node mentioned in the given ZooKeeper from the beginning. It will then print them on the console.

**There's more...**

Some other parameters that you can pass are shown as follows:

- `--fetch-size`: This specifies the amount of data to be fetched in a single request. Its size in bytes follows this argument.
- `--socket-buffer-size`: This specifies the size of the TCP `RECV` size. The size in bytes follows the argument.
- `--autocommit.interval.ms`: This specifies the time interval in which the current offset is saved in `ms`. The time in `ms` follows the argument.
- `--max-messages`: This specifies the maximum number of messages to consume before exiting. If it is not set, the consumption is unlimited. The number of messages follows the argument.
- `--skip-message-on-error`: This specifies that, if there is an error while processing a message, the system should not stop. Instead, it should just skip the current messages.
Where to buy this book

You can buy Apache Kafka Cookbook from the Packt Publishing website.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

Click here for ordering and shipping details.