Kafka IN ACTION

Dylan D. Scott

MEAP

MANNING

MEAP Edition
Manning Early Access Program
Kafka in Action
Version 6

For more information on this and other Manning titles go to
www.manning.com

welcome

Thank you for purchasing the MEAP for Kafka in Action!

For me, reading has always been part of my preferred learning style. Part of the nostalgia is remembering the first practical programming book I ever really read: Elements of Programming with Perl by Andrew L Johnson. The content was something that registered with me. I could follow along and it was a joy to work through.

I hope to capture some of that practical content in regards to working with Apache Kafka. The excitement of learning something new that I had when reading Elements of Programming with Perl was the same way I felt when I started to work with Kafka for the first time. Kafka was unlike any other message broker or enterprise service bus (ESB) that I had used at the time. The speed to get started developing producers and consumers, the ability to reprocess data, and independent consumers moving at their own speeds without removing the data from other consumer applications were techniques that solved pain points I had seen in my own past development and impressed me when I started looking at Kafka.

I see Kafka as changing the standard for data platforms; it can help move batch and ETL workflows to near real-time data feeds. Since this foundation is likely a shift from past data architectures that many enterprise users are familiar with, I wanted to take a user from no prior knowledge of Kafka to the ability to work with Kafka producers and consumers and also be able to perform basic Kafka developer and admin tasks. By the end of this book, I also hope you will feel comfortable digging into more advanced Kafka topics such as cluster monitoring, metrics, and multi-site data replication with your new core Kafka knowledge.

Please remember, these chapters are still works in progress, and will likely change and hopefully get even better by the time the book is complete. I never used to understand how simple mistakes could make it into books, but seeing all of the edits, revisions, and deadlines involved, typos can still make their way into the content and source code. I appreciate your notes for corrections and patience if you run into anything. Please be sure to post any questions, comments, or suggestions you have about the book in the Author Online forum section for Kafka in Action.

I really do hope to make this a useful book and appreciate feedback that you think could improve future versions as well.

Thanks again for your interest and for purchasing the MEAP!

—Dylan Scott

brief contents

1 Introduction to Kafka

In this chapter, we will:

- Introduce why you would want to use Kafka
- Address common myths in relation to Hadoop and message systems
- Understand Real World Use Cases where Kafka helps power messaging, website activity tracking, log aggregation, stream processing, and IoT data processing

From a certain point onward there is no longer any turning back. That is the point tha must be reached.

--

Franz Kafka

As many companies are facing a world full of data being produced from every angle, they are often presented with the fact that legacy systems might not be the best option moving forward. One of the foundational pieces of new data infrastructures that has taken over the IT landscape has been Apache Kafka. While Kafka, who is quoted above, was addressing something far different than IT infrastructures, the wording applies to the situation we find ourselves in today with the emergence of his namesake Apache Kafka. Kafka is changing the standard for data platforms. It is leading the way to move from Batch and ETL workflows to the near real-time data feeds. Batch processing, which was once the standard workhorse of enterprise data processing, might not be something to turn back to after seeing the powerful feature set that Kafka provides. In fact, it might not be able to handle the growing snowball of data rolling toward enterprises of all sizes unless something new is approached. With so much data, systems can get easily inundated with data. Legacy systems might be faced with nightly processing windows that run into the next day. To keep up with this ever constant stream of data, some with evolving data, processing this stream of information as it happens is a way to keep up-to-date and current on the state of the system.

Kafka also is starting to make a push into microservice design. Kafka is touching a lot

of the newest and most practical trends in today's IT fields as well as making its way into many users daily work. As a de-facto technology in more and more companies, this topic is not only for super geeks or alpha-chasers.

Let's start looking at these features by introducing Kafka itself and understand more about the face of modern-day streaming platforms.

1.1 What is Kafka?

The Apache Kafka site (kafka.apache.org/intro) defines it as a distributed streaming platform that has three main capabilites: Provide the ability to publish/subscribe to records like a message queue, store records with fault-tolerance, and process streams as they occur.

For readers who are not as familiar with queues or message brokers in their daily work, it might be helpful to discuss the general purpose and flow of a publish/subscribe system. As a generalization, a core piece of Kafka can be thought of providing the IT equivalent of a receiver that sits in a home entertainment system.

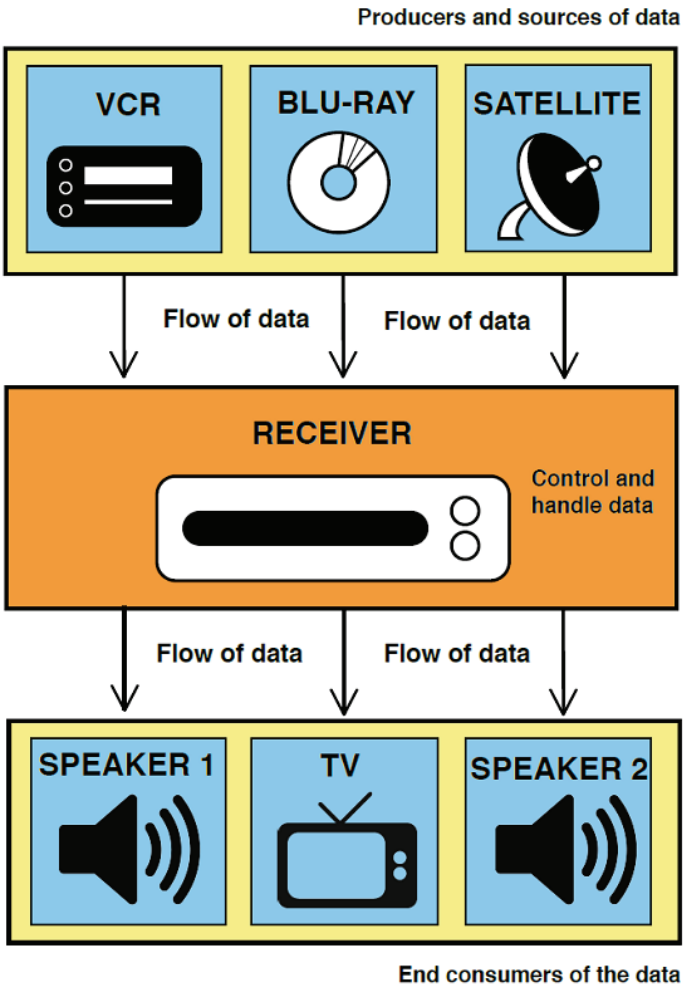

Figure 1.1 Receiver overview

As shown above, digital satellite, cable, and blu-ray players can all connect to this central receiver. Simplifying of course, you can think of those individual pieces as

constantly sending data in a format that they know about. And unless you have some issues, that flow of data can be thought as near constant while the movie or cd is playing. The receiver deals with this constant stream of data and is able to convert it into a usable format for the external devices attached to the other end; ie. the receiver sends the video on to your television and audio to a decoder as well as on to speakers.

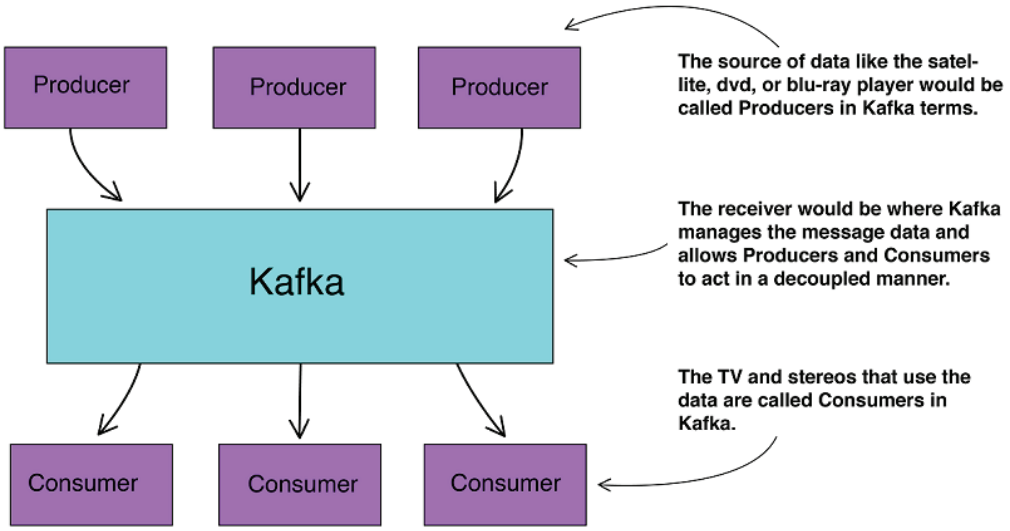

Figure 1.2 Kafka flow overview

So what does this have to do with Kafka exactly? Kafka has the concepts of sending data from various sources of information and introducing them into Kafka called Producers. In addition, Kafka allows multiple output channels that are enabled by Consumers. The receiver can be thought of as the message brokers themselves that can handle a real-time stream of messages. Decoding isn't the right term for Kafka, but we will dig into those details in later chapters.

Kafka, as does other message brokers, acts, in reductionist terms, as a middle man to data coming into the system (from producers) and out of the system (consumers). By allowing this separation between the producer and end-user of the message, loose coupling can be achieved. The producer can send whatever messages it wants and have no clue about if anyone is subscribed.

Further, Kafka also has various ways that it can deliver messages to fit your business case. Kafka message delivery can take at least the following three delivery methods:

- At least once semantics
- At most once semantics
- Exactly once semantics

Let's dig into what those messaging options mean.

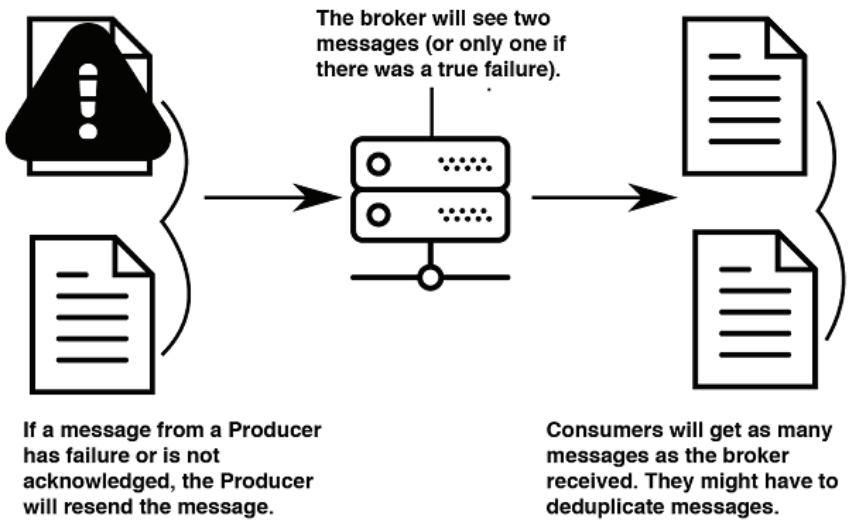

Figure 1.3 At Least Once message flow semantics

Kafka's default guarantee is at least once semantics. This means that Kafka can be configured to allow for a producer of messages to send the same message more than once and have it written to the brokers. When a message has not received a guarantee that it was written to the broker, the producer can send the message again in order to try again. For those cases where you can't miss a message, say that someone has paid an invoice, this guarantee might take some filtering on the consumer end, but is one of the safest methods for delivery.

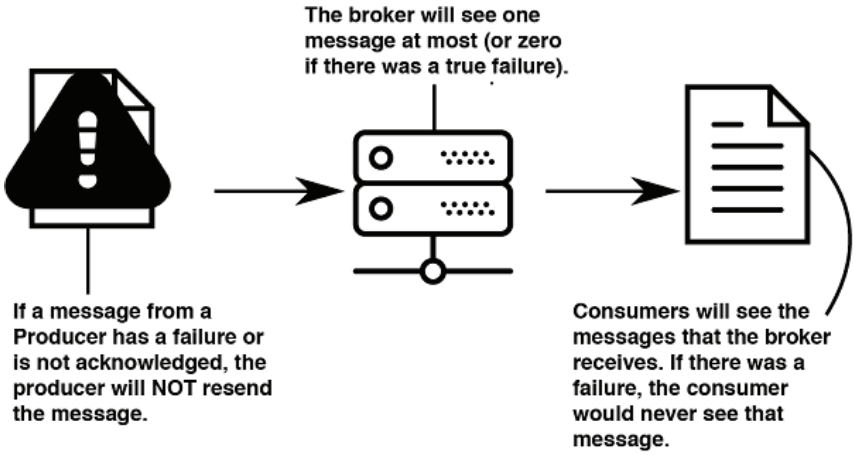

Figure 1.4 At Most Once message flow semantics

Moving on, at most once semantics are when a producer of messages might send a message once and never retires. In the event of a failure, the producer moves on and never attempts to send again. Why would someone ever be okay with losing a message? If a popular website is tracking page views for visitors, it might be fine with missing a few page few events out of the millions they do process per day. Keeping the system performing and not waiting on acknowledgements might outweigh any gain from lost data.

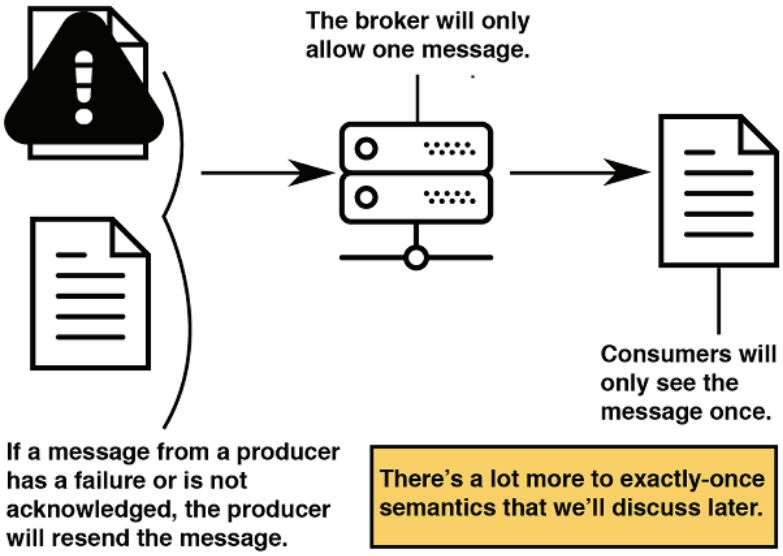

Figure 1.5 Exactly Once message flow semantics

Exactly once semantics (EOS) are one of the newest additions to the Kafka feature set and generated a lot of discussion with its release. In the context of a Kafka system, if a producer sends a message more than once, it would still be delivered once to the end consumer. EOS has touch points at all layers of Kafka, from producers, topics, brokers, and consumers and will be tackled as we move along our discussion later in this book.

Besides various delivery options, another common message broker benefit is if the consuming application is down due to errors or maintenance, the producer does not need to wait on the consumer to handle the message. When consumers start to come back and process data, they should be able to pick up where they left off and not drop messages.

1.2 Why the need for Kafka?

With many traditional companies facing challenges of becoming more and more technical and software-driven, one of the questions is how will they be prepared for the future. Kafka is noted for being a workhorse that also throws in replication and fault-tolerance as a default.

Some companies, including Netflix and LinkedIn, have already reached the level of processing over 1 trillion messages per day with their deployments. Millions of messages per second are possible in production settings; all with a tool that was not at its 1.0 version release (which finally occurred in October of 2017). Their enormous data processing needs are handled with their production usage of Kafka. These examples show the potentially of what Kafka can do for even massive data use cases at scale.

However, besides these headline grabbing facts, why would users want to start looking at Kafka?

1.2.1 Why Kafka for the Developer

Why would a software developer be interested in Kafka?

Kafka usage is exploding and the developer demand isn't being met. [1] A shift in traditional data process thinking is needed and various shared experiences or past pain

points can help developers see why Kafka could be an appealing step forward in their data architectures.

Footnote 1 www.confluent.io/blog/2017-apache-kafka-survey-streaming-data-on-the-rise/

One of the various on-ramps for newer developers to Kafka is a way to apply things they know to help them with the unknown. For Java developers used to Spring concepts and dependency injection (DI) Spring Kafka (projects.spring.io/spring-kafka) has already been through a couple of major release versions. Supporting projects as well as Kafka itself has a growing tool ecosystem of its own.

As a pretty common shared milestone, most programmers have experienced the pain of coupling. For example, you want to make a change but you might have many other applications directly tied to it. Or you start to unit test and see the large amount of mocks you are having to create. Kafka, when applied thoughtfully, can help in this situation.

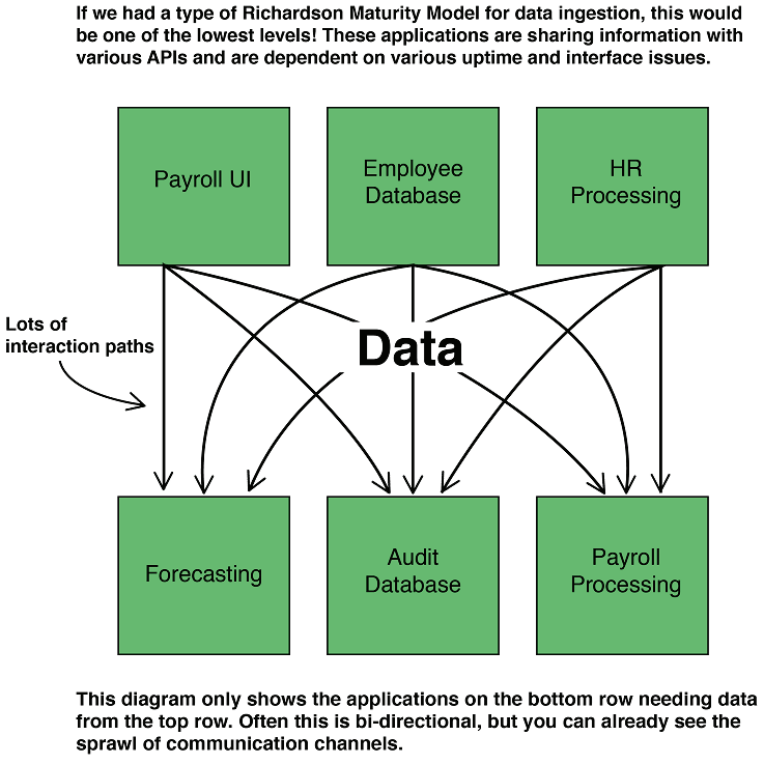

Figure 1.6 Before Kafka message flow

Take for example an HR system that employees would use to submit paid vacation. If you are used to a create, read, update, and delete (CRUD) system, the submission of time off would likely be processed by not only payroll but also project burn down charts for forecasting work. So do you tie the two applications together? What if the payroll system was down? Should that really impact the availability of the forecasting tooling? With Kafka, we will see the benefits of being able to decouple some of the applications that we have tied together in older designs.

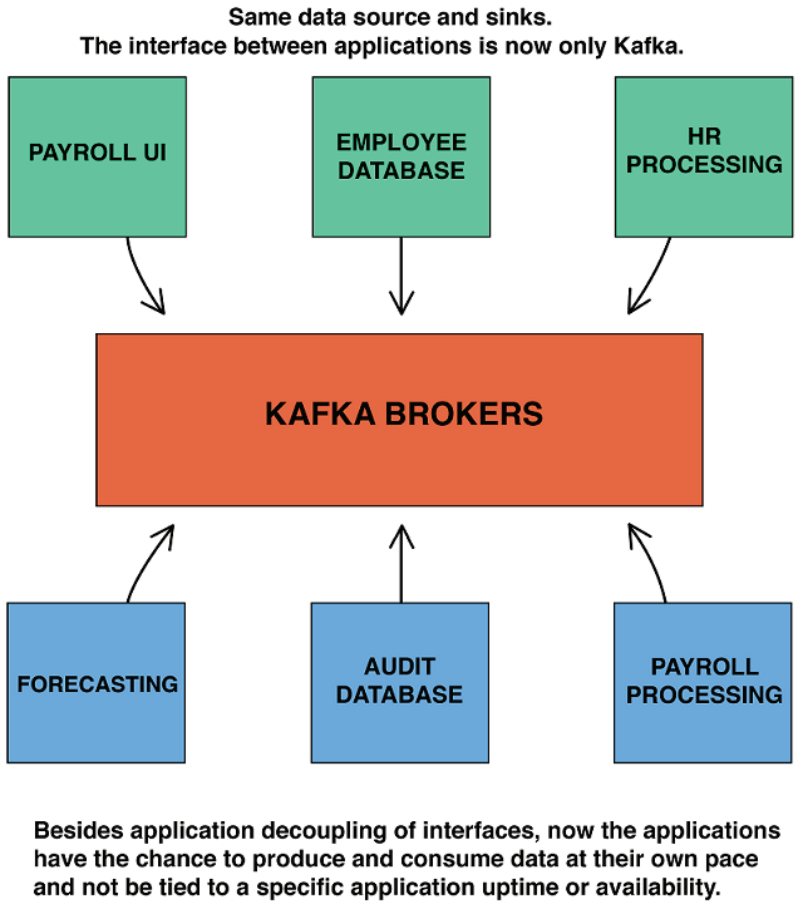

Figure 1.7 After Kafka message flow

With the above view of putting Kafka in the middle of the flow, your interface to data becomes Kafka instead of various APIs and databases.

Some will say that there are pretty simple solutions. What about using ETL to at least load the data into its own databases for each application? That would only be one interface per application and easy, right? But what if the initial source of data was corrupted or updated? How often do you look for updates and allow for lag or consistency? And do those copies ever get out of date or diverge so far from the source that it would be hard to run that flow over and get the same results? What is the source of truth? Kafka can help avoid these issues.

Another interesting topic that might add credibility to the use of Kafka is how much it dog-foods itself. When we dig into Consumers in Chapter 5, we will see how Kafka uses topics under the hood to manage commit offsets. And in release 0.11, exactly once semantics for Kafka also uses internal topics that are leveraged by the Transaction Coordination. The ability to have many consumers of the data use the same message is better than a choose your own adventure game, as all outcomes are possible.

Another developer question might be: Why not learn Kafka Streams, Spark Streaming, or other platforms and skip learning about core Kafka? For one, the number of applications that use Kafka under the covers is indeed impressive. While abstraction layers are often nice to have (and sometimes close to required with so many moving

parts), I really believe that Kafka itself is worth learning. There is a difference in knowing that Kafka is a channel option for Flume and understanding what all of the config options mean. And while Kafka Streams can simplify examples you might see in this book, it is interesting to note how successful Kafka was before Streams were even introduced. The base is fundamental and will hopefully help you see why Kafka is used in some applications and what is happening under the hood.

From a purely technical viewpoint, there are interesting computer science topics applied in practical ways. Perhaps the most talked about is the notion of a distributed commit log which we will discuss in-depth in Chapter 2. And a personal favorite, Hierarchical Timing Wheels (www.confluent.io/blog/apache-kafka-purgatory-hierarchical-timing-wheels -PROD will shorten). Taking a peek into the core.src.main.scala.kafka.utils.timer.Timer.scala class and you can see how Kafka handles a problem of scale by applying an interesting data structure to solve a practical problem.

I would also note that the fact that it's open source is a positive for being able to dig into the source code and also have documentation and examples just by searching the internet. This alone can be worth avoiding the Not Invented Here syndrome.

1.2.2 Explaining Kafka to your manager

As often the case, sometimes members of the C-suite can hear the term 'Kafka' and might be more confused by the name than care about what it really does. As conferences have taken to making the case for attendees to help convince their boss about attending conferences, it might be nice to do the same for explaining the value found in this product as well. Also, it really is good to take a step back and look at the larger picture of what the real value add is for this tool.

One of the most important features of Kafka is the ability to take volumes of data and make it available for use by various business units. Nothing is prescribed, but it is a potential outcome. Most executives will also know that more data than ever is flooding in and they want insights as fast as possible. Rather than pay for data to molder on disk, value can be derived from most of it as it arrives. Kafka is one way to move away from a daily batch job that limited how quickly that data could be turned into value. Fast Data seems to be a newer term that hints that there is real value focus on something different from the promises of Big Data alone.

Running on the JVM should be a familiar and comfortable place for many enterprise development shops. The ability to run on-premise is a key driver for some whose data requires on-site oversight. And the Cloud is also an option as well. It can grow horizontally and not depend on vertical scaling that might eventually reach an expensive peak.

And maybe one of the most important reasons to learn about Kafka is to see how startup and other disrupters in their industry are able to overcome the once prohibitive

cost of computing power. Instead of relying on a bigger and beefier server or a mainframe that can cost millions of dollars, distributed applications and architectures put competitors quickly within reach with hopefully less financial outlay.

1.3 Kafka Myths

When you are first starting to learn any new technology, it is often natural for the learner to try to map their existing knowledge to new concepts. While that technique can be used in learning Kafka, I wanted to note some of the most common misconceptions that I have run into in my work so far.

1.3.1 Kafka only works with Hadoop

As mentioned, Kafka is a powerful tool that is often used in various situations. However, it seemed to appear on radars when used in the Hadoop ecosystem and might have first appeared as a bonus tool as part of a Cloudera or Hortonworks suite. It isn't uncommon to hear the myth that Kafka only works on Hadoop. What could cause this confusion? One of the causes is likely the various tools that use Kafka as part of their own products. Spark Streaming and Flume are examples of tools that use Kafka. The dependency on ZooKeeper is also a tool that is often found in Hadoop clusters and might tie Kafka further to this myth.

One other key myth that often appears is that Kafka requires the Hadoop Filesystem - HDFS. Once we start to dig into how Kafka works, we see the speed of Kafka is achieved with techniques that would likely be slowed down with a Node Manager in the middle of the process. Also, the replications that are usually a part of HDFS would be even more replications if your topics ever used more than one replica. The ideas of replication also lead to another logical group of confusion, the durability that is marketed for Kafka topics might be easy to group under the Hadoop theme of expecting failure as a default (and thus planning for overcoming it).

1.3.2 Kafka is the same as other message brokers

Another big myth is that Kafka is just another message broker. Direct comparisons of the features of various tools such as RabbitMQ or MQSeries to Kafka often have asterisks (or fine print) attached. Some tools over time have gained or will gain new features just as Kafka has added Exactly Once Semantics. And default configurations can be changed to mirror features closer to other tools in the same space.

In general, some of the most interesting and different features are the following that we will dig into below:

- The ability to replay messages by default
- Parallel processing of data

Kafka was designed to have multiple consumers. What that means is one application reading a message off of the message brokers doesn't remove it from other applications that might want to consume it as well. One effect of this is that a consumer that has already seen that message can choose to read that (and other messages) again. With some

architecture models such as Lambda, programmer mistakes are expected just as much as hardware failure. Imagine consuming millions of messages you forgot to use a specific field from the original message. In some queues, that message would have been removed or have been sent to a duplicate or replay location. However, Kafka provides a way for Consumers to seek to specific points and can read messages (with a few constraints) again by just seeking to an earlier position on the topic.

As touched on briefly above, Kafka allows for parallel processing of data and can have multiple consumers on the same exact topic. With the concept of consumers being part of a Consumer Group, membership in a group determines which consumers get which messages and what work has been done in that group of consumers. Consumer groups act independently of the other groups and allow for multiple applications to consume messages at their own pace with as many consumers as they require. So the processing can happen in a couple of ways: Consumption by many consumers working on one application and consumption by many applications.

No matter what other message brokers support, let's now focus on the powerful use cases that have made Kafka one of the default options developers turn to for getting work done.

1.4 Real World Use Cases

I am large, I contain multitudes.

-- Walt Whitman

Applying Kafka to practical use is the core aim of this book. One of the things to note with Kafka is that it's hard to say it does one specific function well; it really has many specific uses in which it can excel. While we have some basic ideas to grasp first, it might be helpful to discuss at a high-level some of the cases that Kafka has already been noted for use in real world use cases.

1.4.1 Messaging

Many users' first experience with Kafka (as was mine) was using it as a messaging tool. Personally, after years of using other tools like IBM WebSphere MQ (formerly MQ Series), Kafka (which was around version 0.8.3 at the time) seemed simple to use to get messages from point A to point B. It forgoes using the Extensible Messaging and Presence Protocol (XMPP), the Advanced Message Queuing Protocol (AMQP), or Java Message Service (JMS) API in favor of a custom TCP binary protocol.

We will dig in and see some complex uses later, but as an end-user developing with a Kafka client, most of the details are in the configuration and the logic becomes rather straightforward; ie. I want to place a message on this topic.

Having a durable channel for sending messages is also why Kafka is used. Often times, memory storage of data will not be enough to protect your data; if that server dies, the messages are not persisted across a reboot. High availability and persistent storage are built into Kafka from the start. In fact, Apache Flume provides a Kafka Channel option

since the replication and availability allow Flume events to be made immediately available to other sinks if a Flume agent (or the server it is running on) crashes. Kafka enables robust applications to be built and helps handle the expected failures that distributed applications are bound to run into at some point.

1.4.2 Website Activity Tracking

Part of Kafka's origin story is serving as a way to gather information about users on Linkedin's website. Kafka's ability to handle large volumes of data was a key requirement since a website with many visitors had the need to handle the possibility of multiple events per page view. The flood of data had to be dealt with fast and able to scale without impacting the end-user's experience on the site. Did a user click on a link or visit the new marketing page? Those frequent user events can be fed into Kafka and made quickly available to various applications to get an overall view of the paths consumers take through a shopping site (for instance). Imagine the ability to do a real-time feed of data to show customers recommended products on the next page click as one example of the importance of speed in a e-commerce flow.

1.4.3 Log Aggregation

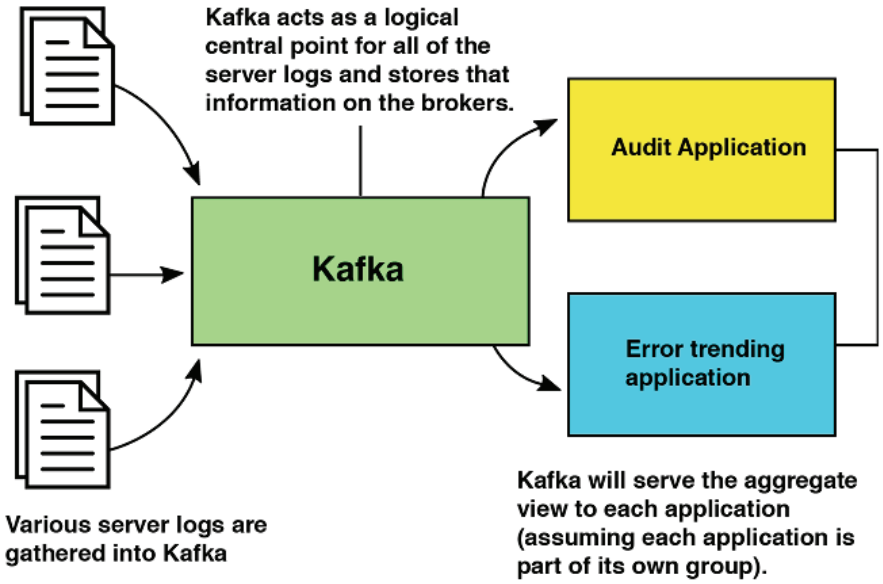

Figure 1.8 Kafka Log Aggregation

Log aggregation is useful in many situations, including when trying to gather application events that were written in distributed applications. In the figure above, the log files are being sent as messages into Kafka and then different applications have a logical single topic to consume that information. With Kafka's ability to handle large amounts of data, collecting events from various servers or sources is a key feature. Depending on the contents of the log event itself, some organizations have been able to use it for auditing as well as failure detection trending. Kafka is also used in various logging tools (or as an input option) such as Graylog (www.graylog.org).

How do all of these log files entries even allow Kafka to maintain performance

without causing a server to run out of resources? The throughput of small messages can sometimes overwhelm a system since the processing of each method takes time and overhead. Kafka uses batching of messages for sending data as well as writing data. Writing to the end of a log helps as well rather than random access to the filesystem. We will discuss batching in Chapter 4 and more on the log format of messages in Chapter 6 and 7.

1.4.4 Stream Processing

Kafka has placed itself as a fundamental piece for allowing developers to get data quickly. While Kafka Streams is now a likely default for many when starting work, Kafka had already established itself as a successful solution by the time the Streams API was released in 2016. The Streams API can be thought of as a layer that sits on top of producers and consumers. This abstraction layer is a client library that is providing a higher level view of working with your data as an unbounded stream.

In the Kafka 0.11 release, exactly once semantics were introduced. We will cover what that really means in practice later on once we get a more solid foundation. However, for users who are running end-to-end workloads inside Kafka with the Streams API, message delivery now is more of what most users would find as ideal. Imagine banks using Kafka to debit or credit your account. With at least once semantics, you would have to ensure that a debit was not completed twice (due to duplication of messages) through various consumer side logic. Streams makes this use case easier than it has ever been before to complete a flow without any custom application logic overhead of ensuring a message was was only processed once from the beginning to the end of the transaction.

1.4.5 Internet of Things

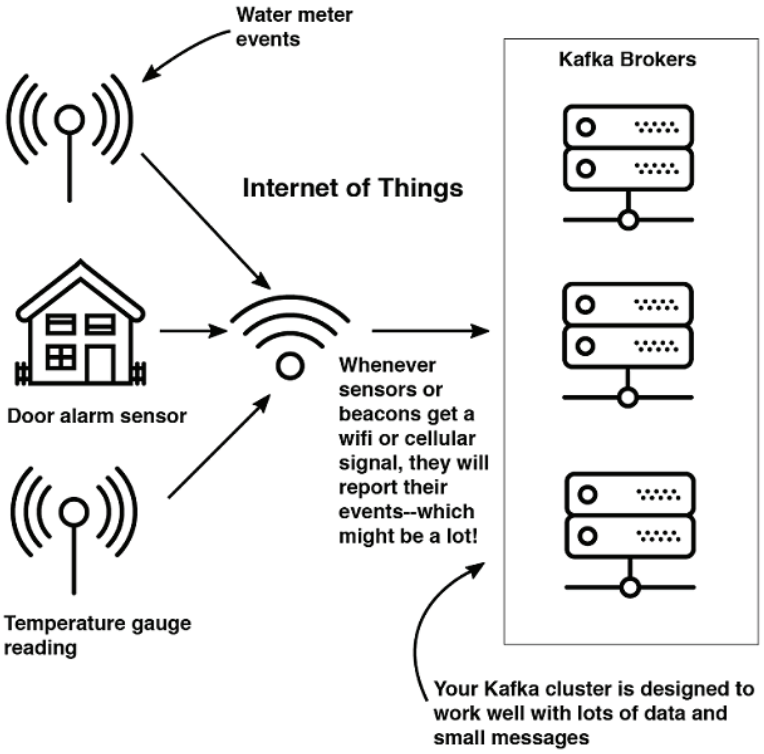

Figure 1.9 IoT

The number of internet-connected devices seems like the Internet of Things will have no where to go but up. With all of those devices sending messages, sometimes in bursts whenever they get a wifi or cellular connection, something needs to be able to handle that data effectively. As you may have gathered, massive quantities of data is one of the key areas where Kafka really shines. As we discussed above, small messages are not a problem for Kafka. Beacons, cars, phones, homes, all will be sending data and something needs to handle the firehose of data and make it available for action.

These are just a small selection of examples that are well-known uses for Kafka. As we will see in future chapters, Kafka has many practical application domains. Learning the upcoming foundational concepts is important to be able to see how even more practical applications are possible.

1.4.6 When Kafka might not be the right fit

It is important to note that while Kafka has shown some really interesting use cases, it is not always the best tool for the job at hand. Let's take a quick look at some of the uses where other tools or code might shine.

What if you only need a once monthly or even once yearly summary of aggregate data? If you don't need an on-demand view, quick answer, or even the ability to reprocess data, then you might not need Kafka running throughout the entire year for that

task alone. Especially, if that amount of data is manageble to process at once as a batch (as always, your mileage may vary: different users have different thresholds on what is large batch).

If your main access pattern for data is mostly random lookup of data, Kafka might not be your best option. Linear read and writes is where Kafka shines and will keep your data moving as quickly as possible. Even if you have heard of Kafka having index files, they are not really what you would compare to a relational database having fields and primary keys that indexes are built off of.

Similarly, if you need exact ordering of messages in Kafka for the entire topic, you will have to look at how practical your workload is in that situation. To avoid any unordered messages, care would have to be taken to make sure that only 1 producer request thread is the max at one time and that there is only 1 partition in the topic. There are various workarounds, but if you have huge amounts of data that depends on strict ordering, there are potential gotchas that might come into play once you notice that your consumption would be limited to one consumer per group at a time.

One of the other practical items that come into mind is that large messages are an interesting challenge. The default message size is about 1 MB. With larger messages, you start to see memory pressure increase. In other words, the less page messages you can store in page cache is one main concern. So if you are planning on sending huge archives around, you might want to look at if there is a better way to manage those messages.

Keep in mind, while you could probably achieve your end goal with Kafka in the above situations (it's always possible), it might not be the first choice to reach for in the toolbox.

1.5 Online resources to get started

The community around Kafka has been one of the best (in my opinion) in making documentation available. Kafka has been a part of Apache (graduating from the Incubator in 2012), and keeps current documentation at the project website at kafka.apache.org.

Another great resource for more information is Confluent (www.confluent.io/resources). Confluent was founded by some of the creators of Kafka and is actively influencing future direction of the work. They also build enterprise specific features and support for companies to help develop their streaming platform. Their work helps support the open source nature of the tool and has extended to presentations and lectures that have discussed production challenges and successes.

As we start to dig into more APIs and configuration options in later chapters, these resources will be a good reference if further details are needed rather than list them all in these chapters.

1.6 Summary

This chapter was about getting familiar with Kafka and its use cases at a very high level. In the next chapter, we will look at more details in which we use specific terms and start to get to know Apache Kafka in a more tangible and hands-on way.

In this chapter we:

- Discussed the streaming and fault-tolerance of Kafka that makes it a modern distributed solution
- Discovered that Kafka is not exclusive to Hadoop or HDFS. While it uses some of the same fault-tolerance and distributed systems techniques, it can stand alone as its own cluster.
- Talked about special features that make Kafka standout from other message brokers. This includes its replayable messages and multiple consumers features.
- Displayed specific use cases where Kafka has been used in production. These uses include messaging, website activity tracking, log aggregation, stream processing, and IoT data processing.