SAMPLE CHAPTER

Understanding the real-time pipeline

Streaming DATA

Andrew G. Psaltis

MANNING

Streaming Data
Understanding the real-time pipeline

by Andrew G. Psaltis

Chapter 1

brief contents

1 Introducing streaming data

This chapter covers

- Differences between real-time and streaming data systems
- Why streaming data is important
- The architectural blueprint
- Security for streaming data systems

Data is flowing everywhere around us, through phones, credit cards, sensor-equipped buildings, vending machines, thermostats, trains, buses, planes, posts to social media, digital pictures and video—and the list goes on. In a May 2013 report, Scandinavian research center Sintef estimated that approximately 90% of the data that existed in the world at the time of the report had been created in the preceding two years. In April 2014, EMC, in partnership with IDC, released the seventh annual Digital Universe study (www.emc.com/about/news/press/2014/20140409-01.htm), which asserted that the digital universe is doubling in size every two years and would multiply 10-fold between 2013 and 2020, growing from 4.4 trillion gigabytes to 44 trillion gigabytes. I don't know about you, but I find those numbers hard to comprehend and relate to. A great way of putting that in perspective also comes from that report: today, if a byte of data were a gallon of water, in only 10 seconds there would be enough data to fill an average home. In 2020, it will only take 2 seconds.

Although the notion of Big Data has existed for a long time, we now have technology that can store all the data we collect and analyze it. This does not eliminate the need for using the data in the correct context, but it is now much easier to ask interesting questions of it, make better and faster business decisions, and provide services that allow consumers and businesses to leverage what is happening around them.

We live in a world that is operating more and more in the *now*—from social media, to retail stores tracking users as they walk through the aisles, to sensors reacting to changes in their environment. There is no shortage of examples of data being used today as it happens. What is missing, though, is a shared way to both talk about and design the systems that will enable not merely these current services but also the systems of the future.

This book lays down a common architectural blueprint for how to talk about and design the systems that will handle all the amazing questions yet to be asked of the data flowing all around us. Even if you've never built, designed, or even worked on a real-time or Big Data system, this book will serve as a great guide. In fact, this book focuses on the big ideas of streaming and real-time data. As such, no experience with streaming or real-time data systems is required, making this perfect for the developer or architect who wants to learn about these systems. It's also written to be accessible to technical managers and business decision makers.

To set the stage, this chapter introduces the concepts of streaming data systems, previews the architectural blueprint, and gets you set to explore in-depth each of the tiers as we progress. Before I go over the architectural blueprint used throughout the book, it's important that you gain an understanding of real-time and streaming systems that we can build upon.

1.1 What is a real-time system?

Real-time systems and *real-time computing* have been around for decades, but with the advent of the internet they have become very popular. Unfortunately, with this popularity has come ambiguity and debate. What constitutes a real-time system?

Real-time systems are classified as *hard*, *soft*, and *near*. The definitions I use in this book for *hard* and *soft real-time* are based on Hermann Kopetz's book *Real-Time Systems* (Springer, 2011). For *near real-time* I use the definition found in the Portland Pattern Repository's Wiki (http://c2.com/cgi/wiki?NearRealTime). For an example of the ambiguity that exists, you don't need to look much further than Dictionary.com's definition: "Denoting or relating to a data-processing system that is slightly slower than real-time." To help clear up the ambiguity, table 1.1 breaks out the common classifications of real-time systems along with the prominent characteristics by which they differ.

You can identify hard real-time systems fairly easily. They are almost always found in embedded systems and have very strict time requirements that, if missed, may result

Table 1.1 Classification of real-time systems

Classification	Examples	Latency measured in	Tolerance for delay
Hard	Pacemaker, anti-lock brakes	Microseconds–milliseconds	None—total system failure, potential loss of life
Soft	Airline reservation system, online stock quotes, VoIP (Skype)	Milliseconds–seconds	Low—no system failure, no life at risk
Near	Skype video, home automation	Seconds–minutes	High—no system failure, no life at risk

in total system failure. The design and implementation of hard real-time systems are well studied in the literature, but are outside the scope of this book. (If you are interested, check out the previously mentioned book by Hermann Kopetz.)

Determining whether a system is soft or near real-time is an interesting exercise, because the overlap in their definitions often results in confusion. Here are three examples:

- Someone you are following on Twitter posts a tweet, and moments later you see the tweet in your Twitter client.
- You are tracking flights around New York using the real-time Live Flight Tracking service from FlightAware (http://flightaware.com/live/airport/KJFK).
- You are using the NASDAQ Real Time Quotes application (www.nasdaq.com/quotes/real-time.aspx) to track your favorite stocks.

Although these systems are all quite different, figure 1.1 shows what they have in common.

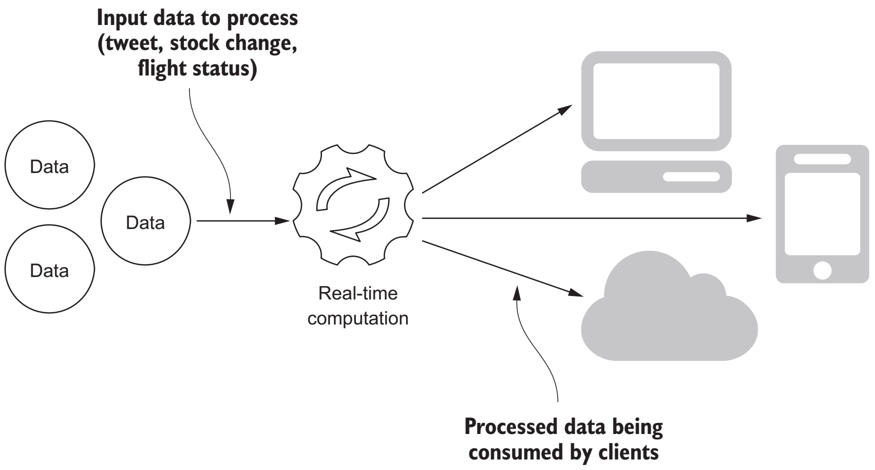
Input data to process (tweet, stock change, flight status)
Data
Data
Data
Real-time computation
Processed data being consumed by clients

Figure 1.1 A generic real-time system with consumers

In each of the examples, is it reasonable to conclude that the time delay may only last for seconds, no life is at risk, and an occasional delay for minutes would not cause total system failure? If someone posts a tweet, and you see it almost immediately, is that soft or near real-time? What about watching live flight status or real-time stock quotes? Some of these can go either way: what if there were a delay in the data due to slow Wi-Fi at the coffee shop or on the plane? As you consider these examples, I think you will agree that the line differentiating soft and near real-time becomes blurry, at times disappears, is very subjective, and may often depend on the consumer of the data.

Now let's change our examples by taking the consumer out of the picture and focusing on the services at hand:

- A tweet is posted on Twitter.
- The Live Flight Tracking service from FlightAware is tracking flights.
- The NASDAQ Real Time Quotes application is tracking stock quotes.

Granted, we don't know how these systems work internally, but the essence of what we are asking is common to all of them. It can be stated as follows:

> Is the process of receiving data all the way to the point where it is ready for consumption a soft or near real-time process?

Graphically, this looks like figure 1.2.

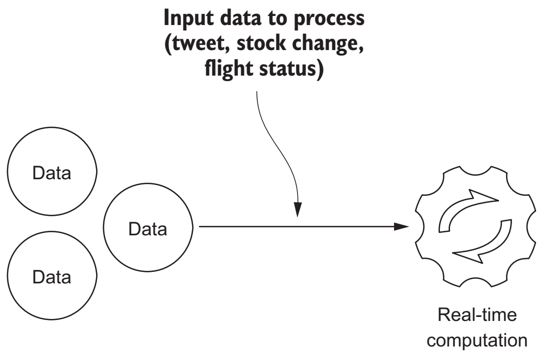

Figure 1.2 A generic real-time system with no consumers

Does focusing on the data processing and taking the consumers of the data out of the picture change your answer? For example, how would you classify the following?

- A tweet posted to Twitter
- A tweet posted by someone whom you follow and your seeing it in your Twitter client

If you classified them differently, why? Was it due to the lag or perceived lag in seeing the tweet in your Twitter client? After a while, the line between whether a system is soft or near real-time becomes quite blurry. Often people settle on calling them real-time. In this book, I aim to provide a better way to identify these systems.

1.2 Differences between real-time and streaming systems

It should be apparent by now that a system may be labeled soft or near real-time based on the perceived delay experienced by consumers. We have seen, with simple examples, how the distinction between the types of real-time system can be hard to discern. This can become a larger problem in systems that involve more people in the conversation. Again, our goal here is to settle on a common language we can use to describe these systems. When you look at the big picture, we are trying to use one term to define two parts of a larger system. As illustrated in figure 1.3, the end result is that it breaks down, making it very difficult to communicate with others with these systems because we don't have a clear definition.

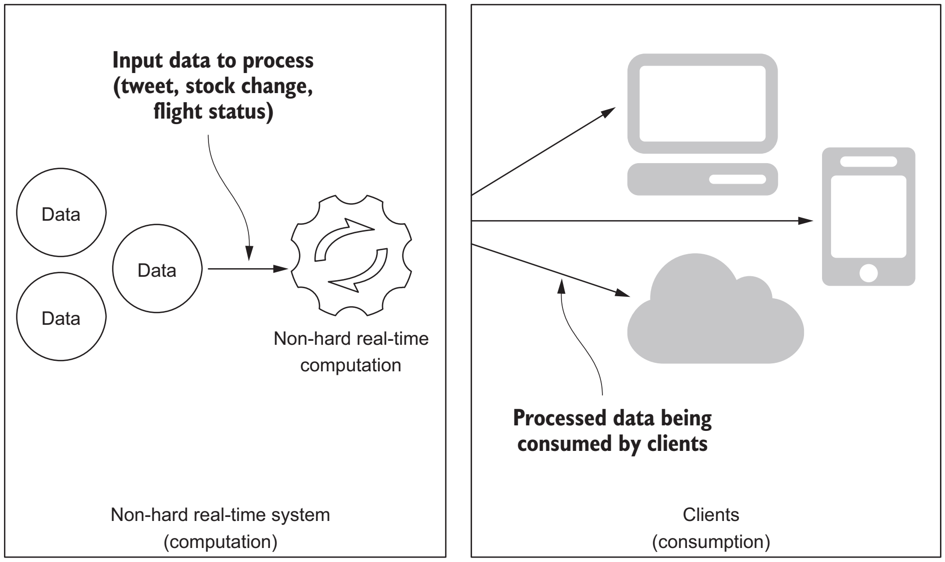
Input data to process (tweet, stock change, flight status)
Data
Data
Data
Non-hard real-time computation
Non-hard real-time system (computation)
Processed data being consumed by clients
Clients (consumption)

Figure 1.3 Real-time computation and consumption split apart

On the left-hand side of figure 1.3 we have the non-hard real-time service, or the *computation* part of the system, and on the right-hand side we have the clients, called the *consumption* side of the system.

> **DEFINITION: STREAMING DATA SYSTEM** In many scenarios, the computation part of the system is operating in a non-hard real-time fashion, but the clients may not be consuming the data in real time due to network delays, application design, or a client application that isn't even running. Put another way, what we have is a non-hard real-time service with clients that consume data when they need it. This is called a *streaming data system*—a non-hard real-time

system that makes its data available at the moment a client application needs it. It's neither soft nor near—it is streaming.

Figure 1.4 shows the result of applying this definition to our example architecture from figure 1.3.

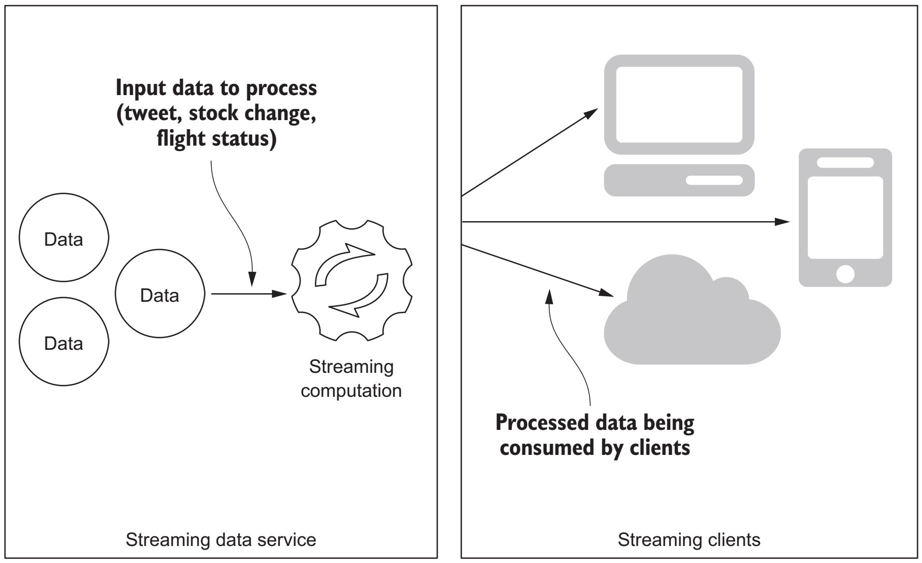
Input data to process (tweet, stock change, flight status)
Data
Data
Data
Streaming computation
Streaming data service
Processed data being consumed by clients
Streaming clients

Figure 1.4 A first view of a streaming data system

The concept of streaming data eliminates the confusion of soft versus near and real-time versus not real-time, allowing us to concentrate on designing systems that deliver the information a client requests at the moment it is needed. Let's use our examples from before, but this time think about them from the standpoint of streaming. See if you can split each one up and recognize the streaming data service and streaming client.

- Someone you are following on Twitter posts a tweet, and moments later you see the tweet in your Twitter client.
- You are tracking flights around New York using the real-time Live Flight Tracking service from FlightAware.
- You are using the NASDAQ Real Time Quotes application to track your favorite stocks.

How did you do? Here is how I thought about them:

- *Twitter*—A streaming system that processes tweets and allows clients to request the latest tweets at the moment they are needed; some may be seconds old, and others may be hours old.
- *FlightAware*—A streaming system that processes the most recent flight status data and allows a client to request the latest data for particular airports or flights.

- *NASDAQ Real Time Quotes*—A streaming system that processes the price quotes of all stocks and allows clients to request the latest quote for particular stocks.

Did you notice that doing this exercise allowed you to stop worrying about soft or near real-time? You got to think and focus on what and how a service makes its data available to clients at the moment they need it. Thinking about it this way, you can say that the system is an *in-the-moment* system—any system that delivers the data at the point in time when it is needed. Granted, we don't know how these systems work behind the scenes, but that's fine. Together we are going to learn to assemble systems that use open source technologies to consume, process, and present data streams.

1.3 The architectural blueprint

With an understanding of real-time and streaming systems in general under our belt, we can now turn our attention to the architectural blueprint we will use throughout this book. Throughout our journey we are going to follow an architectural blueprint that will enable us to talk about all streaming systems in a generic way—our pattern language. Figure 1.5 depicts the architecture we will follow. Take time to become familiar with it.

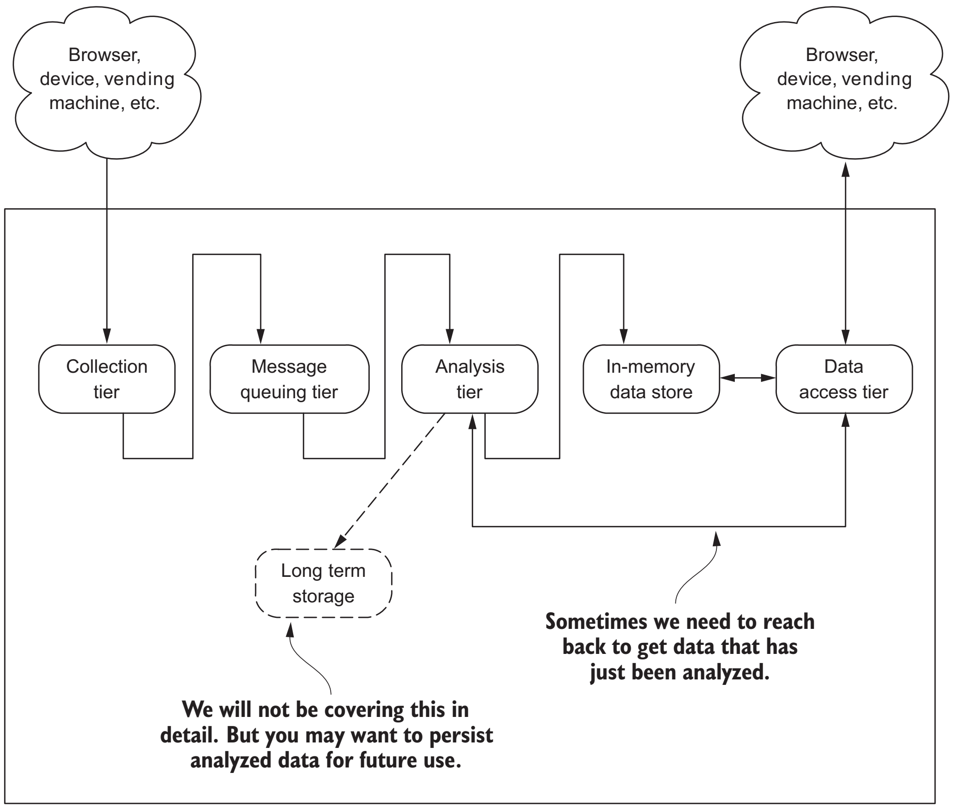
Browser, device, vending machine, etc.
Browser, device, vending machine, etc.
Collection tier
Message queuing tier
Analysis tier
In-memory data store
Data access tier
Long term storage
Sometimes we need to reach back to get data that has just been analyzed.
We will not be covering this in detail. But you may want to persist analyzed data for future use.

Figure 1.5 The streaming data architectural blueprint

As we progress, we will zoom in and focus on each of the tiers shown in figure 1.5 while also keeping the big picture in mind. Although our architecture calls out the different tiers, remember these tiers are not hard and rigid, as you may have seen in other architectures. We will call them tiers, but we will use them more like LEGO pieces, allowing us to design the correct solution for the problem at hand. Our tiers don't prescribe a deployment scenario. In fact, they are in many cases distributed across different physical locations.

Let's take our examples from before and walk through how Twitter's service maps to our architecture:

- *Collection tier*—When a user posts a tweet, it is collected by the Twitter services.
- *Message queuing tier* —Undoubtedly, Twitter runs data centers in locations across the globe, and conceivably the collection of a tweet doesn't happen in the same location as the analysis of the tweet.
- *Analysis tier*—Although I'm sure a lot of processing is done to those 140 characters, suffice it to say, at a minimum for our examples, Twitter needs to identify the followers of a tweet.
- *Long-term storage tier*—Even though we're not going to discuss this optional tier in depth in this book, you can probably guess that tweets going back in time imply that they're stored in a persistent data store.
- *In-memory data store tier*—The tweets that are mere seconds old are most likely held in an in-memory data store.
- *Data access*—All Twitter clients need to be connected to Twitter to access the service.

Walk yourself through the exercise of decomposing the other two examples and see how they fit our streaming architecture:

- *FlightAware*—A streaming system that processes the most recent flight status data and allows a client to request the latest data for particular airports or flights.
- *NASDAQ Real Time Quotes*—A streaming system that processes the price quotes of all stocks and allows clients to request the latest quote for particular stocks.

How did you do? Don't worry if this seemed foreign or hard to break down. You will see plenty more examples in the coming chapters. As we work through them together, we will delve deeper into each tier and discover ways that these LEGO pieces can be assembled to solve different business problems.

1.4 Security for streaming systems

As you reflect on our architectural blueprint, you may notice that it doesn't explicitly call out security. Security is important in many cases, but it can be overlaid on this architecture naturally. Figure 1.6 shows how security can be applied to this architecture.

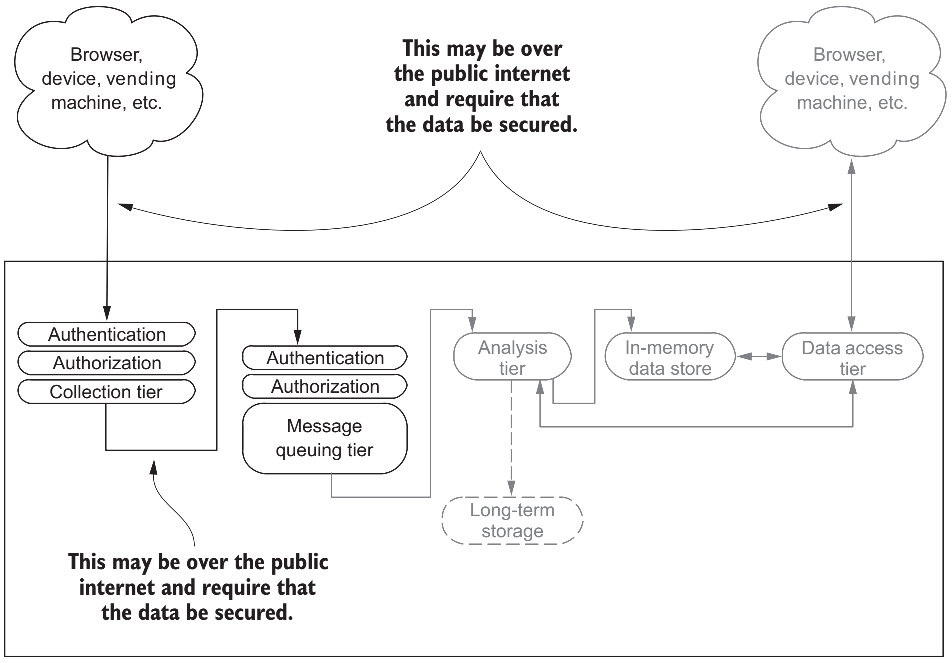
Browser, device, vending machine, etc.
This may be over the public internet and require that the data be secured.
Browser, device, vending machine, etc.
Authentication
Authorization
Collection tier
Authentication
Authorization
Message queuing tier
Analysis tier
In-memory data store
Data access tier
Long-term storage
This may be over the public internet and require that the data be secured.

Figure 1.6 The architectural blueprint with security identified

We won't be spending time discussing security in depth, but along the way I will call it out so you can see how it fits and think about what it may mean for the problems you're solving. If you're interested in taking a deeper look at security and distributed systems, see Ross Anderson's *Security Engineering: A Guide to Building Dependable Distributed Systems* (Wiley, 2008). This book also is available for free at www.cl.cam.ac.uk/~rja14/book.html.

1.5 How do we scale?

From a high level, there are two common ways of scaling a service: vertically and horizontally.

Vertical scaling lets you increase the capacity of your existing hardware (physical or virtual) or software by adding resources. A restaurant is a good example of the limitations of vertical scaling. When you enter a restaurant, you may see a sign that tells you the maximum occupancy. As more patrons come in, more tables may be set up and more chairs added to accommodate the crowd—this is scaling vertically. But when the maximum capacity is reached, you can't seat any more customers. In the end, the capacity is limited by the size of the restaurant. In the computing world, adding more memory, CPUs, or hard drives to your server are examples of vertical scaling. But as with the restaurant, you're limited by the maximum capacity of the system, physical or virtual.

Horizontal scaling approaches the problem from a different angle. Instead of continuing to add resources to a server, you add servers. A highway is a good example of horizontal scaling. Imagine a two-lane highway that was originally constructed to handle 2,000 vehicles an hour. Over time more homes and commercial buildings are built along the highway, resulting in a load of 8,000 vehicles per hour. As you might imagine (and perhaps have experienced), the results are terrible traffic jams during rush hour and overall unpleasant commutes. To alleviate these issues, more lanes are added to the highway—now it is horizontally scaled and can handle the traffic. But it would be even more efficient if it could expand (add lanes) and contract (remove lanes) based on traffic demands. At an airport security checkpoint, when there are few travelers TSA closes down screening lines, and when the volume increases they open lines up. If you're hosting your service with one of the major cloud providers (Google, AWS, Microsoft Azure), you may be able to take advantage of this elasticity—a feature they often call *auto-scaling*. The basic idea is that as demand for your service increases, servers are automatically added, and as demand decreases, servers are removed.

In modern-day system design, our goal is to have horizontal scaling—but that doesn't mean that we won't use vertical scaling too. Vertical scaling is often employed to determine an ideal resource configuration for a service, and then the service is scaled out. But in this book, when the topic of scaling comes up, the focus will be on horizontal, not vertical scaling.

1.6 Summary

Now that you have an idea of the architectural blueprint, let's see where we have been:

- We defined a real-time system.
- We explored the differences between real-time and streaming (in-the-moment) systems.
- We developed an understanding of why streaming is important.
- We laid out an architectural blueprint.
- We discussed where security for streaming systems fits in.

Don't worry if some of this is slightly fuzzy at this point, or if teasing apart the different business problems and applying the blueprint seems overwhelming. I will walk through this slowly over many different examples in the coming chapters. By the end, these concepts will seem much more natural.

We are now ready to dive into each of the tiers to find out what they're composed of and how to apply them in the building of a streaming data system. Which tier should we tackle first? Take a look at a slightly modified version of our architectural blueprint in figure 1.7.

We're going to take on the tiers one at a time, starting from the left with the collection tier. Don't let the lack of emphasis on the message queuing tier in figure 1.7

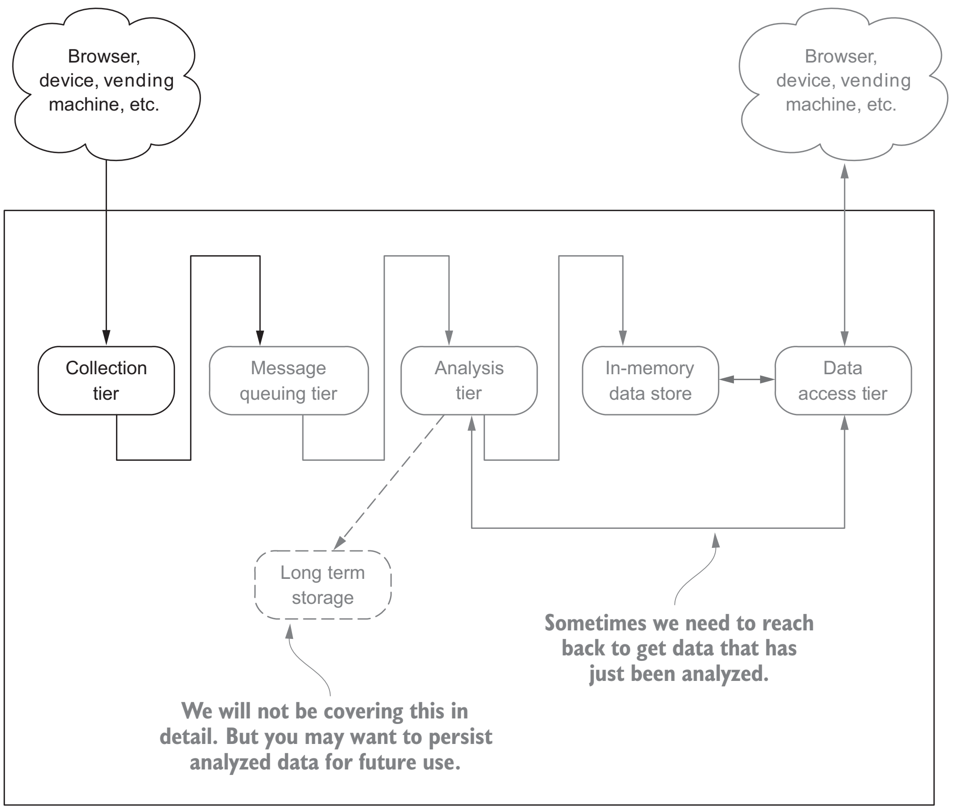
Browser, device, vending machine, etc.
Browser, device, vending machine, etc.
Collection tier
Message queuing tier
Analysis tier
In-memory data store
Data access tier
Long term storage
Sometimes we need to reach back to get data that has just been analyzed.
We will not be covering this in detail. But you may want to persist analyzed data for future use.

Figure 1.7 Architectural blueprint with emphasis on the first tier

bother you—in certain cases where it serves a collection role, I'll talk about it and clear up any confusion. Now, on to our first tier, the collection tier—our entry point for bringing data into our streaming, in-the-moment system.

DATA SCIENCE/PROGRAMMING

Streaming Data

Andrew G. Psaltis

As humans, we're constantly filtering and deciphering the information streaming toward us. In the same way, streaming data applications can accomplish amazing tasks like reading live location data to recommend nearby services, tracking faults with machinery in real time, and sending digital receipts before your customers leave the shop. Recent advances in streaming data technology and techniques make it possible for any developer to build these applications if they have the right mindset. This book will let you join them.

Streaming Data is an idea-rich tutorial that teaches you to think about efficiently interacting with fast-flowing data. Through relevant examples and illustrated use cases, you'll explore designs for applications that read, analyze, share, and store streaming data. Along the way, you'll discover the roles of key technologies like Spark, Storm, Kafka, Flink, RabbitMQ, and more. This book offers the perfect balance between big-picture thinking and implementation details.

What's Inside

- The right way to collect real-time data
- Architecting a streaming pipeline
- Analyzing the data
- Which technologies to use and when

Written for developers familiar with relational database concepts. No experience with streaming or real-time applications required.

Andrew Psaltis is a software engineer focused on massively scalable real-time analytics.

To download their free eBook in PDF, ePub, and Kindle formats, owners of this book should visit www.manning.com/books/streaming-data

Free eBook
SEE INSERT

"The definitive book if you want to master the architecture of an enterprise-grade streaming application."
—Sergio Fernández González
Accenture

"A thorough explanation and examination of the different systems, strategies, and tools for streaming data implementations."
—Kosmas Chatzimichalis, Mach 7x

"A well-structured way to learn about streaming data and how to put it into practice in modern real-time systems."
—Giuliano Araujo Bertoti, FATEC

"This book is all you need to really understand what streaming is all about!"
—Carlos Curotto, Globant

ISBN-13: 978-1-61729-228-6
ISBN-10: 1-61729-228-1
54999
9 781617 292286

MANNING $49.99 / Can $65.99 [INCLUDING eBOOK]