CHAPTER 1

Making Sense of Edge Computing

V. K. Cody Bumgardner
Caylin Hickey

MANNING

Making Sense of Edge Computing

by Cody Bumgardner and Caylin Hickey

Chapter 1

Copyright 2020 Manning Publications
To pre-order or learn more about these books go to www.manning.com

For online information and ordering of these and other Manning books, please visit www.manning.com. The publisher offers discounts on these books when ordered in quantity.

For more information, please contact

Special Sales Department
Manning Publications Co.
20 Baldwin Road
PO Box 761
Shelter Island, NY 11964
Email: Erin Twohey, corp-sales@manning.com

©2020 by Manning Publications Co. All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by means electronic, mechanical, photocopying, or otherwise, without prior written permission of the publisher.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in the book, and Manning Publications was aware of a trademark claim, the designations have been printed in initial caps or all caps.

Recognizing the importance of preserving what has been written, it is Manning's policy to have the books we publish printed on acid-free paper, and we exert our best efforts to that end. Recognizing also our responsibility to conserve the resources of our planet, Manning books are printed on paper that is at least 15 percent recycled and processed without the use of elemental chlorine.

Manning Publications Co.
20 Baldwin Road Technical
PO Box 761
Shelter Island, NY 11964

Cover designer: Marija Tudor

ISBN: 9781617298622

We, the authors, want to thank you for purchasing the MEAP for *Making Sense of Edge Computing*.

Edge computing seeks to take full advantage of the explosion of connected devices and new sources of data. Technology has become ubiquitous, fueled by inexpensive, power efficient, yet powerful devices that can be deployed almost anywhere. More common than not, devices are connected together providing communications ranging from the smallest embedded device to the largest computational clouds. By leveraging this connected network of resources, we'll help you make sense of the computational paradigm of edge computing. In edge computing we ask the fundamental question of when should be move data and when should we move functionality. Functionally might range from distributed processing, event detection, to AI training, and inference. We'll show you how many problems might be solved more efficiently and effectively by distributing them over networks of resources; in some cases, pushing the work closer to the original source of the data, in others coordinating distributed data movement to computational clouds.

Due to its inherently network-centric nature, some very basic understanding of networking is needed to get the most benefit from the subject. We also find a fundamental understanding of graphs beneficial when it comes to conceptualized distributed resources and applications. Examples and demonstrations in this book are presented using a pre-built Docker image, so if you want to get hands-on, some limited understanding of Docker will prove useful as well. Finally, we provide opportunities in the book to modify or create new code. As our demonstration framework is in Java, if you want to code along with us, some familiarity with Java and Maven will be required.

We've been working in edge computing for a number of years, with some predating the community coalescing around the term itself. Dr. Bumgardner was so enamored with the paradigm, it even became the subject of his PhD dissertation. With coauthor and collaborator Mr. Hickey, work began with the theorizing and realizing of an edge computing framework, to its continual improvement, to its utilization in solving problems ranging from international network measurement and monitoring, to data

management and processing in clinical laboratories. We hope to distill what we've learned over these years into a book that can guide you to making use of edge computing in your own projects as well.

In *Making Sense of Edge Computing*, we start by giving both a definition and motivation for the edge computing paradigm. Then, after a brief introduction to, and demonstration of, an edge-computing framework, we build out some of the fundamental theory that has helped shape our work over the years; with demonstrations sprinkled throughout to help ground theory with experience. Finally, with the basics out of the way, we will walk you through some example implementations and applications to show how you can begin to leverage edge computing in similar projects.

From motivation to demonstration, theory to practice, we hope you'll walk away with a better understanding of edge computing and its uses in your project designs. While we aim to provide a useful tool, we've learned over the years that feedback is a crucial component of getting to a finished product. We look forward to your questions and comments, either in the forums or via email.

—Authors Cody Bumgardner, PhD
and Caylin Hickey

contents

1 Introducing edge computing

This chapter covers

- Defining edge computing
- Discussing the differences between the client-server model and edge computing
- Explaining the core components of edge computing
- Understanding where to use edge computing

Edge computing is the practice of moving data processing closer in proximity to the sources of data generation. The purpose of edge computing is to establish a decentralized and potentially hierarchical computational paradigm to support the development and operations of distributed systems. This practice can break down a centralized computing problem—usually one where you move all your data to a central server for processing—into a much more efficient, distributed solution by doing part of the processing on devices connected to your network. Instead of needing a large investment in the form of a data center or cloud infrastructure, you can use those same devices you use to gather the data to perform some or all of

your processing, saving you from having to send all your data back to a central server or data center and reducing overall costs to boot!

Maybe you're happy with your current setup, gathering all your data to a local data center or to the "cloud," where it can be easily dealt with. For smaller, less complex, workflows this works fine. Consider, though, if you need to process something fairly large and continuous, such as streams of video for unauthorized personnel. Can your centralized system handle 10 video streams, or hundreds, or thousands? What if those streams are in higher resolutions such as 4K? With data feeds that large, you may find yourself struggling—or, worse, paying handsomely—to even transfer it all back to your data lake; let alone storing and processing it. With edge computing you handle the processing of these streams either on the devices themselves—or on infrastructure that's close in proximity to the devices with more adequate resources—meaning scalability is much easier to achieve and the hardware requirements are much easier to meet on a per-device basis. Rather than processing all those video streams in your data center, you tell each device to look for unauthorized personnel and only send alerts when they find one. Now you only need to transfer video data when you're reviewing and confirming an alert you've received, thus saving time and money on data transfers and infrastructure.

Maybe the volume isn't so much of a concern, but rather the content of the data itself. What if your project contains private information, such as patient health data, that you may not be able to transfer off the source device? By processing the private data on the device, or on an adjacent device in the same location, and only transferring the results, you alleviate the need to transmit the original private or sensitive information unless absolutely required, ensuring security while still allowing utility out of potentially sensitive data.

These are only two examples of situations where you could immediately benefit from edge computing. We've only scratched the surface of the many ways in which edge computing can make your data-driven operations more manageable and efficient. Let's introduce you to edge computing more formally, see how it differs from the more traditional client-server model, and explore ways in which edge computing could improve your applications and projects.

1.1 What's this "edge" you speak of?

Conceptually, edge computing is concerned with when it's best to migrate computational functionally toward source of data and when it's best to move the data itself. This abstract concept of function versus data migration drives not only the fundamental motivations of edge computing, but also the broader field of distributed systems. The act of distributing processes makes even the simplest tasks more complicated.

As any programmer is painfully aware, developing and debugging multi-threaded applications compared with their single-threaded counterparts is challenging. The network engineer can attest to the complexity of a multi-homed autonomous router compared with a home access point. Likewise, the systems professional might give you

a laugh when asked to explain the differences between an application running on a laptop and geographically distributed cluster.

If distributing things is so hard then why do we do it? In most cases, computational and infrastructure distribution is driven out of necessity. It would be no more practical to run Google.com (the entire search engine) from my laptop than it would be to require a vehicular collision system to transfer data across a country to make critical sub-millisecond decisions. Although you might agree with the previous arguments, you might ask yourself what edge computing has to do with you? Perhaps at this second you're holding a smart phone (are dumb ones still sold?)—a device that at this point is a phone primarily in name only. If on June 28, 2007, the day before the iPhone was released, you were told that your phone would replace many functions of your computer, TV, and much more, you might not have believed it. Likewise, few of us who experienced the early days of the internet would have predicted the pervasive global impact, for better or worse, it has had on our lives. What events occurred between the first message being sent over ARPANET in 1969 and my thermostat consorting with my AI assistant as to when I might get home and what temperature it should be when I get there?

Ok, stay with us here as we journey a bit down the "rabbit hole". Those of you who have been around communications a while will know the difference between circuit- and packet-switched networks. For the unfamiliar, a *circuit-switched* network is what might come to mind when you see an old movie. When someone picks up a phone, they're connected to switchboard operator who manually plugs a cable into a socket physically connecting two telephones. You might be surprised to learn that although advancements have been made, public telephone and 2/3G cellular networks still use of circuit switching. Circuit switching is considered inefficient because it dedicates end-to-end resources for communications, regardless of link utilization.

In contrast, *packet switching* breaks data into units, which we call packets, allowing independent communications over shared links. If you have broadband internet or a 4G or newer phone, you use a packet-switched network. Why is this important? Consider all the network devices in your home, office, or even on your person. If these devices used circuit-switching technology, you'd have the equivalent of a physical phone line per device. Can you imagine ordering a new phone line to be installed for your internet-connected toaster and a separate line for your TV? Those involved in the early days of the Internet didn't conceive of connecting your home lighting to a remote cloud service. The cloud didn't exist, and there was no necessity for the average company, much less individual, to have access to generalized packet-switched networks, allowing access to and from many devices globally. What new computational paradigm will allow for and perhaps drive a new age of technology?

You could argue that the current practice of individual devices communicating directly with remote cloud services is as silly as installing a phone line for your toaster. If devices are to generate data that's acted upon by more than one system—perhaps AI in a cooperative manner—then not only must data democratization occur, but new

computational paradigms must be leveraged to support it. The leap to make here is that just as packet-switching technology fundamentally changes the approach to connectivity by allowing communications decisions to be made closer to devices, such as your TV and toaster sharing your connection to the Internet, edge computing can have the same impact on the computational level.

Imagine the constraints of transmitting data from a device across the country that needs to communicate to a device across the room, which then might need to communicate to a service on the other coast for decision-making. Now, consider the benefits of data processing taking place on your phone, in your house, in your neighborhood, city, or any other remotely connected location based on the global optimization of infrastructure, data, and application needs.

1.2 What makes edge computing different?

To understand the utility, and some might argue necessity, of edge computing, you must take a step back and think about the fundamental characteristics of various types of networks. In this context, we're not necessarily referring to communications networks, but the more fundamental theory behind networks of all types works. Figure 1.1 illustrates three types of networks was presented by Paul Baran in his paper "On Distributed Communication Networks" nearly 60 years ago. Although more modern conceptions of the distributed systems communications differ from this early work, we find it useful to explain where edge computing fits in the distributed space.

A *system* is defined as a set of connected components that form a complex whole. Systems are described as *centralized* if individual components are directly controlled by a central component. An example of a centralized system is a personal computer. Whereas a computer is comprised of many functional components, the overall system is useless without a central processing unit (CPU). Likewise, a mobile phone interacting with a remote website is also operating in a centralized paradigm. Examples of centralized systems exist outside of technology, such as a medical practice with a single physician. Similar to a computer without a CPU, a medical practice without a licensed physician cannot function.

Individual services provided by systems can be described in terms of workloads. Workloads consume resources to perform specific tasks. For example, calculating the first N Fibonacci numbers is a computer workload. In the medical example, performing a personal physical is a medical workload. The study of workload arrival, servicing, and departure is referred to as queueing theory. In theory, the duplication of centralized systems would double the potential aggregate output of that system.

A system is said to be *distributed* if individual components can process workload tasks while coordinating with each other to provide a common service. An example of a distributed system is a cluster of computers that can calculate the first N Fibonacci numbers in parallel, or in the healthcare example, a medical practice with multiple practicing partners. In the given centralized and decentralized examples, workloads are processed by distributed systems, but workload scheduling (workload arrival, sequencing, and resource assignment) in these systems is controlled centrally.

As distributed systems with central scheduling grow in both size and service complexity, scheduling functions must also be distributed. The process of dispersing the functional control of a central authority is defined as *decentralization*. Through decentralization, control functions are divided into interdependent layers. The arrangement of distributed control layers and workload-servicing components is defined as a hierarchy. Levels of hierarchy both provide and accept control from other levels, while maintaining a service-level autonomy. You can see a visual contrast between *centralized, decentralized,* and *distributed* systems in figure 1.1.

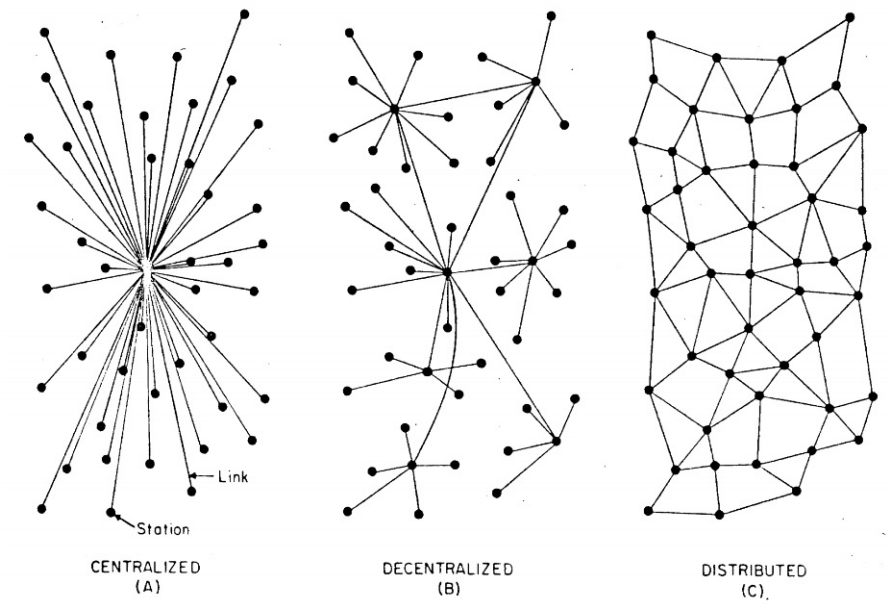

Figure 1.1 Centralized, decentralized, and distributed network types provide several of the historic motivations for the benefits of moving computation from a central location to the devices themselves, a basic principle of edge computing.

A university is an example of a decentralized (hierarchical) system. High-level administrators and advisory boards evaluate metrics and implement university-wide changes by communicating with colleges, colleges implement college-level changes and communicate with departments, and departments implement departmental changes by communicating with faculty and staff.

For example, suppose a university (high-level administration) wanted to increase the number of engineering graduates. A change in configuration (strategic direction, increased resource allocations, and so on) is communicated to the college of engineering. The college determines that based on metrics reported from all departments under their hierarchy, the computer science department can benefit most from a configuration change. Once the change is communicated to the department, resources can be committed, and metrics related to the configuration change and be reported back

through the system. Figure 1.2 shows how you can represent this problem in terms of a hierarchical distributed processing system that uses advanced message queuing protocol (AMQP) and representational state transfer (ReST) channels for communication.

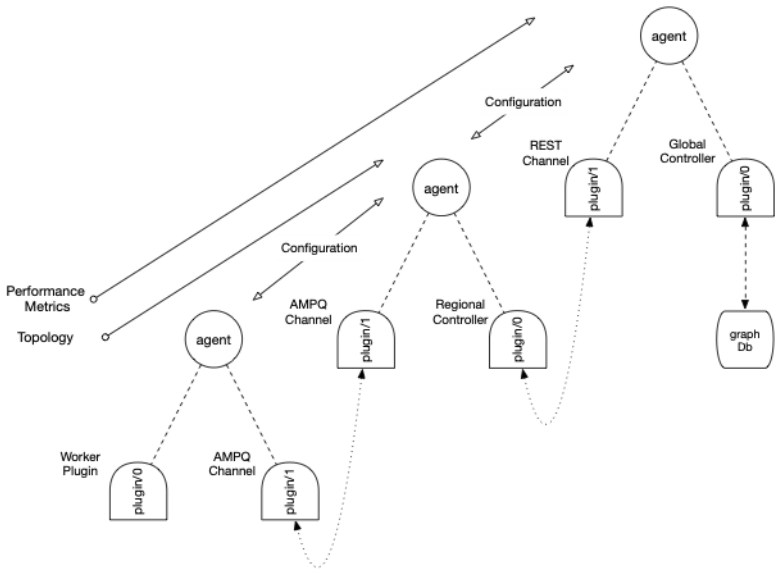

Figure 1.2 An example edge computing hierarchy in which backend agent data is propagated from lower layers to administrative agents while tasks can communicate with one another when necessary.

In the following subsections we discuss various aspects of edge computing, including its relation to existing computational models and motivations.

1.2.1 Client-server and cloud models

When you talk about a centralized computing, one of the most common computational examples is that of a "client-server model", such as the one in figure 1.3. This typically means a system in which remote clients contacts one or more central servers that provides resources in response to single or on-going requests. What you have is a central system (computer, server, or group of servers) in control of the storing, processing, and distributing a set of resources or data to clients who request this data. If you've ever used an email client or a web browser, this should be familiar. While you may compose your email or read a web page on your device, the server is what handles the background routing and formatting of your emails or the rendering of the web content, usually coupled with resources pulled from a database. Figure 1.3 illustrates data from many potentially distributed devices being aggregated by an internet of things (IoT) gateway for remote cloud processing by a single VM.

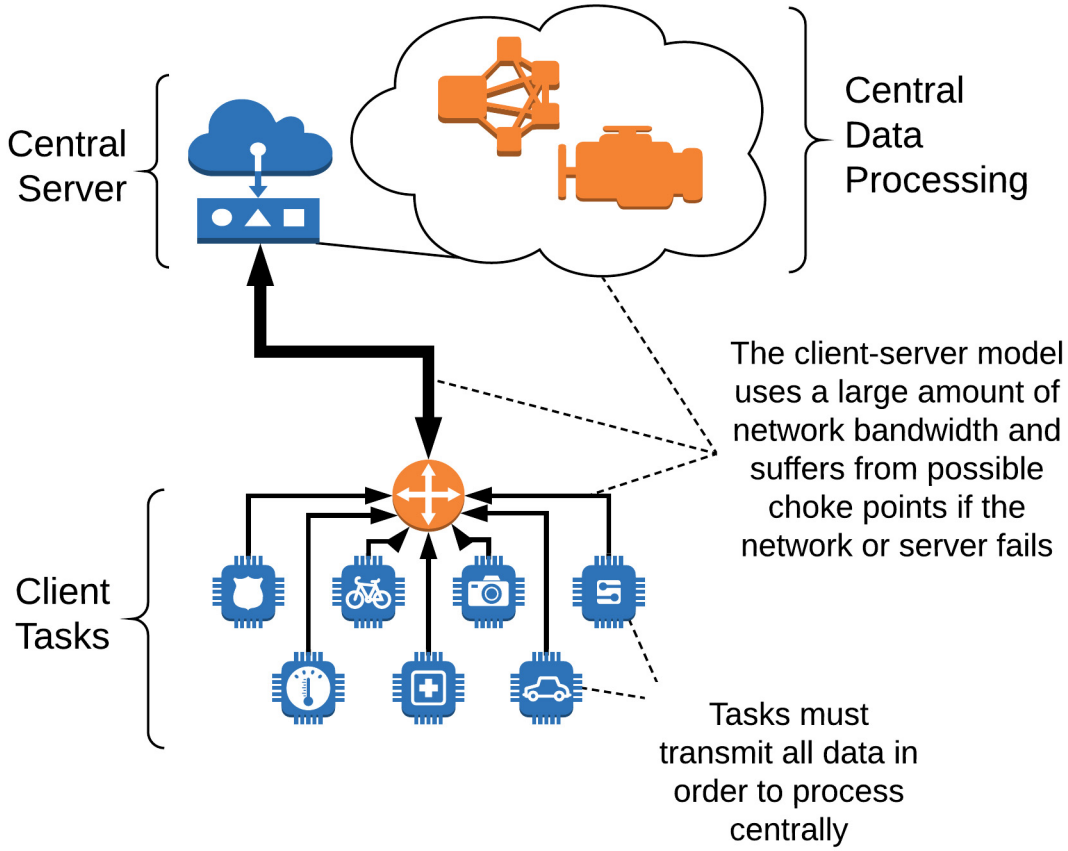

Figure 1.3 **Client-server-cloud model diagram with an aggregated device gateway and remote centralized processing.**

In these types of systems, most of the work is done by the central server and the clients only interact as needed. As we said, a world of workflows exists for which this is a perfectly suitable approach. However, as you'll see, as the size of the data increases, or the number or distance between clients grows, certain limitations start to determine how useful, or even practical, this approach can be. In our example, the IoT gateway and VM function as centralized choke points and single points of failure. If the IoT gateway or VM fails, capacity is exceeded, or communication is disrupted, and the entire system fails.

Let's assume that you replace your single VM with a globally distributed cloud service, or perhaps your entire back-end system based on a multi-region auto-scaling Kubernetes deployment. While you might have confidence in our backend system, the important part, namely data acquisition and response, is still at risk. The internet is inherently a decentralized system operating across antonymous nodes that route data based on changes in link topologies, utilization capacities, and traffic classifications. No one, not even Amazon or Google, can deterministically predict the end-to-end

path a packet will take from an end-user device across the public internet to arrive at a distant cloud service. Likewise, no end-to-end offering exist (this isn't a comment on net neutrality) to guarantee end-to-end service for network communications. If one can neither predict, avoid, or guarantee quality of service of communications between end-devices and the public cloud across the Internet, then it's a game of chance. This game of chance is played potentially hundreds of times for every interaction a device might have with a remote service. It's a testament to our network communication forbearers that it's more common these days than not to exclusively use of highly centralized clouds for nearly all system operations. However, you'll learn this isn't a tenable future strategy, which is where edge computing comes in.

1.2.2 How can edge computing help?

What if emergency medicine functioned like devices interacting with the public cloud? Suppose that if by consolidating local clinics and regional hospitals, free national healthcare was provided at a dozen state-of-the art medical facilities across the US. Now suppose you're in a traffic accident and are in need emergency medical treatment. Your medical card has the address of the closest facility, but you must find directions to this potentially remote and distant facility without further assistance.

Perhaps you can drive yourself, take an Uber, or book a flight. This may seem unreasonable, but how could the remote centralized medical facility possibly help you get there? They don't know the extent of your condition until you get there; they don't know the local landscape, your fear of flying, or to never take Champions Avenue before or after the big game. Medicine doesn't work this way in the real world. You rely on local emergency medical dispatchers to determine (monitor) the following:

- What resources are needed
- Notifying local emergency medical services to provide urgent local treatment (alerting and escalation)
- Assessing patient condition and vitals (data enrichment) and relaying information to the health care provider (data aggregation)
- Engaging the appropriate emergency physicians to treat (complex event processing) life-threating illnesses
- Referring patients to the appropriate specialist

Emergency medicine has evolved into a complex decentralized hierarchical ensemble of players with well-defined roles, protocols, and operational boundaries to ensure that patients receive the correct treatment in a timely manner by delegating resources—such as time, effort, and expertise—to where they are most efficiently utilized. This delegation of resources by imposing a decentralized hierarchical organization makes sense in the world of computing as well.

Edge computing aims to decentralize operational functions, including but not limited to computation and communications, allowing the overall system to continue to function, and if needed, dynamically scale, to tolerate both failures and capacity

fluctuations. Figure 1.4 illustrates a refactoring of the previously described client-server-cloud model to be deployed using the edge computing paradigm, pushing computation and related decision-making toward sources of data generation.

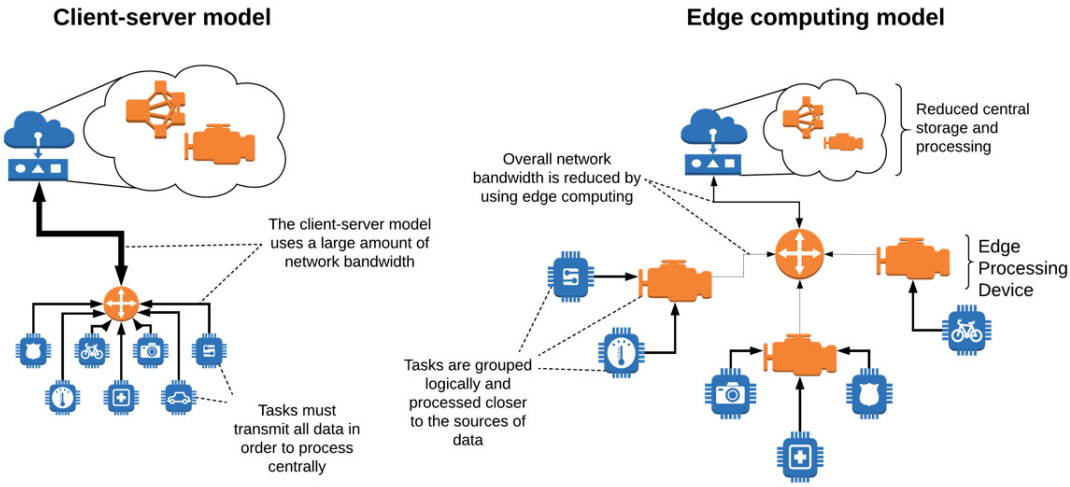

Figure 1.4 Edge computing—where computation takes place close to the source of data generation—reduces network congestion and logically organizes the location of task processing, reducing centralized resources and processing needs.

As shown in figure 1.4, the public cloud hasn't been removed from the system, but you're no longer dependent on the centralized functions provided by the IoT gateway or a remote and potentially distant clouds. Borrowing from our emergency medical example, our edge devices function as local dispatchers and EMS. Once data is processed locally and, if needed, prepared for transport, the edge-computing system determines the path of transport, which could be to an intermediate resource for additional processing, such as a regional hospital. Depending on the capabilities of the local dispatchers and EMS (application), enriched data might be further transported to a high-level trauma center (public cloud).

Having seen an example of how edge computing can improve operations in an emergency medical operation over a more traditional client-server approach, we'll next introduce several of the fundamental components in edge computing that allow for this decentralized approach.

1.3 The components of edge computing

Now that you're familiar with the general background and motivations of edge computing, we'll briefly describe what we consider the foundational components of edge computing. At a fundamental level, edge computing involves multiple layers of abstraction that organize tasks and resources to allow for heterogeneous, distributed tasks to

be concurrently processed on heterogenous, distributed hardware in a managed fashion. In figure 1.5 you can see a breakdown of the layers of abstraction. Because edge computing is a new field, these may vary from implementation to implementation as features are added and removed. Starting from the bottom layer, which provides an abstraction that organizes the underlying infrastructure and data the edge-computing framework operates on, each higher layer adds new functionality and further abstraction to the layer below it. Those of you with a background in networking will recognize the similarity of the edge computing component hierarchy to the OSI model. We'll next briefly describe the layers that compose an edge-computing system.

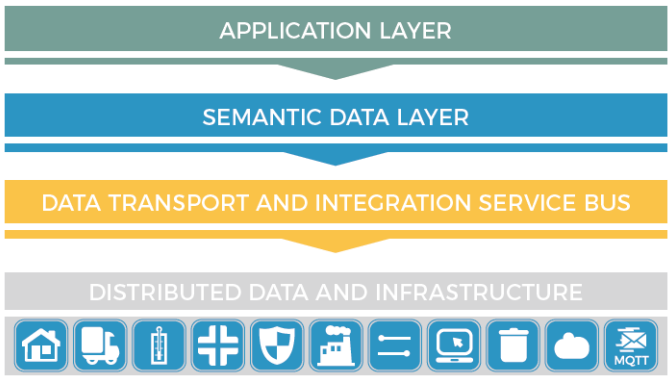

Figure 1.5 **General overview of edge computing components.**

1.3.1 Distributed data and infrastructure

At the bottom of the pile is the actual edge-computing framework code that runs on your embedded device, computer, or server. In this layer, devices and data sources are organized into networks and hierarchies that structure how they can and will be used by the layers above them. This abstraction can reorganize, for instance, devices into contained groups by device class, department of use, or ownership, with needing to relocate the physical device to allow for segmentation to take place at the network level. Typically, this layer runs directly on the host machine and facilitates all operations required by the framework itself. From start-up to building secure data connections, this layer contains the backbone of operations meaning when one of the above layers requires action, such as launching an application or building a new communications channel, this layer handles the bulk of the work.

1.3.2 Data transport and integration service bus

The data transport and information service bus, or data layer for short, handles direct communication paths between hosts in your distributed infrastructure. These paths can serve a couple of purposes: to handle framework control messages, as in the integration service bus, or to pass data messages, as in the data transport. These channels

can be either fixed, long-running messaging paths, or other external ad-hoc connections using other means of communication. This system will also handle path routing to the desired end host as well as any return messages such as in a remote procedure call (RPC).

1.3.3 Semantic data layer

While the data layer is mainly focused on where the data is going and how it needs to get there, the information layer is instead concerned with what kind of data is being transmitted. As this is a distributed infrastructure, with functions that may be spun up and down dynamically, using fixed data paths is impractical. To remedy this, edge computing frameworks will typically choose to look at data according to its semantics. For example, rather than saying, "I need a TCP connection to camera A with IP 1.1.1.1 onto which it will put its video feed," the framework may instead say, "I need video data tagged with camera A as a source." This subtle shift allows functions to request a type of data, and possibly a specific tag that may correspond to an identifier but ignore tracking down an exact location of where to grab the actual bits.

1.3.4 Application layer

The application layer is the most visible to the user of an edge computing framework. In this layer, one or more function(s), or small pieces of computation to be performed, are distributed over the framework, either statically, automatically, or dynamically (if efficient resource scheduling is provided). These functions perform predefined tasks, usually on incoming data, and, if necessary, provide the results of their computation to other functions. End users can either use existing functions compatible with their chosen edge computing framework or write their own as the software allows.

The quote "Dans les champs de l'observation le hasard ne favorise que les esprits prepares—Where observation is concerned, chance favors only the prepared mind" is attributed to Louis Pasteur. The purpose of this book is to prepare you to identify the challenges that edge computing intends to address across the continuum of computational layers. To that end, we'll next discuss several sources of these challenges.

1.4 The edge is inevitable

You might wonder if future technology advancements will mitigate the need for such a computational paradigm. As you're no doubt aware, companies have solved problems in distributed computing for as long as we've networked computers. Over the years these computers have made huge gains in memory, processing power, and networking speed in an attempt to keep up with the challenges general computing seeks to solve. But as the cost of these advances has come down, the number of devices in the wild has skyrocketed, so much so that we've had to update the dictionary to include a term for it—internet of things or IoT—which loosely defines as the system of interconnected computers, devices, and machines communicating, usually in an autonomous

or human-less fashion, heterogenous streams of data. This increasing pervasiveness of IoT devices and infrastructure will only continue to compound issues related to data ETL (extraction, transformation, and loading). In the following section, we discuss the so-called "Three Laws of IoT," which describe the ongoing need for edge services that use edge computing.

1.4.1 The three laws of IoT

More connected devices are generating more data than ever before, data which is increasing in value with each passing year. This data growth, and even the nature of the data itself, have natural consequences for which certain considerations must be made when designing distributed systems. These consequences can be summarized in the so-called three laws of IoT: Laws of Physics, Laws of Economics, and Laws of the Land.

LAWS OF PHYSICS

Data transfer is governed by the speed of light, so the overall speed of computational inference (decision making) will be governed, in part, by the speed in which data and be translated from the source of generation to computation. For example, a multi-node collision avoidance system can't transmit data across the US and be effective. Moving computational resources and functions closer to sources of data generation, as practiced in edge computing, addresses physics constraints.

LAWS OF ECONOMICS

Data transfer and storage cost money, and while all data might need to be processed, not all data needs to be transferred or stored. Battery-powered sensors and satellite communications used in high-risk monitoring systems are extreme cases of economic cost vs risk in distributed computing, but many other examples exist. Moving initial filtering and complex event processing closer to sources of data generation, while providing control of what data should be propagated, what resilience policies are being satisfied, and what data can be safely ignored, as address by edge computing, addresses many economic challenges.

LAWS OF THE LAND

In many cases, such as medical, defense, and other areas, data cannot be legally transmitted in the form in which it was generated. Compensating controls related to privacy, preservation, de-identification, and other functions might be required before data can be acquired from distributed system. Likewise, for regulatory purposes, such as in laboratory testing data streams might require enrichment to indicate the province of data relating to a specific instrument. Edge computing allows for complete application and data-layer control of information at sources of data generation satisfying law of the land constraints.

You can argue that as a consequence of the described laws, that edge computing will continue exist regardless of any future technical advancements. In fact, an argument can be made that edge computing techniques will be required to address not only the challenges of large numbers of distributed devices and their resulting vol-

umes of data, but also how to manage resulting information. To help illustrate several areas where you may run into limitations, we'll introduce a few brief examples of edge computing solutions.

1.5 Example uses of edge computing

Edge computing can be difficult. Distributing computation, storage, and other parts of a computer system introduces a host of challenges that must be overcome. Hardware fails, people press buttons and pull on cables, and disasters happen, so parts of the system could fail at any time. Since edge computing requires extra effort, it doesn't make sense to use it when it isn't useful unless you're into that sort of thing (no judgement here). But how do you know when it's most useful?

In short, you only need it when you need it. That situation often looks an awful lot like a problem you can almost solve. The Three Laws of Edge Computing can help us determine when edge computing techniques might be appropriate. Wherever existing techniques run afoul of physics, economics, or the long, wrinkly arm of the Law, edge computing should be considered. A simpler rule of thumb is to consider using edge computing when you need to handle lots of streaming data or have latency constraints. Like other rules of thumb, they aren't written in stone. Several of the leading cloud computing providers already have commercial edge-computing platforms, and as more people enter the field, they're bound to start "abstracting away" the complexity of edge computing in order to put these powerful tools in the hands of more people. In the meantime, I know of a great book on edge computing if you're interested!

We'll apply these rules of thumb to a few examples that represent typical uses of edge computing. You'll see concepts from three areas where edge computing can help: a smarter city, a hospital, and a small hobbyist app. As you walk through these examples, we hope to illustrate ways in which the existing techniques fall short.

1.5.1 Gunshot detection in a smarter city

Smarter cities use various sensors to monitor different kinds of activity within the city. These sensors are connected to computer networks that process the data and, in certain cases, take some kind of action. That could mean turning a switch on or off, contacting the police, changing traffic signals, or many other things.

For this example, consider a fictional smarter city with a large population. The city government worked with a local university to develop a gunshot detection system that uses a network of more than 9,000 microphones strategically placed throughout the city to detect gunshots. This system uses machine learning to determine when a gunshot has occurred and approximately where it came from. The models require a stream of uncompressed digital audio from as many microphones as possible, with fewer microphones causing reduced performance. When a gunshot is detected, the police are notified and provided with a video feed from traffic cameras near the putative shot location. The system also stores a copy of the data from all of the microphones near the shot for a ten-minute window before and after a positive detection.

This system must process large amounts of streaming data and respond quickly, which is an ideal problem for edge computing.

The machine learning models in our example don't require much computing power—the computationally hard work was done during training. However, they require a large amount of data to make accurate determinations, and this data must be handled in near real-time to be useful. Rather than ship all our audio data to a datacenter for processing, which might be impossible in certain cases due to limited infrastructure, you can feed it to the machine learning models on low-powered computers near where it's generated. Those computers only have to send something indicating a gunshot detection and the location, generating miniscule network traffic compared to the microphones. They can also communicate with other networked devices close by more quickly than they could with a distant server, allowing them to respond more quickly. Without edge computing, you'd run smack into the laws of physics and be forced to accept the latency and bandwidth possible any given moment.

Edge computing also helps make the system more economical to build and operate. If you need more bandwidth than you have, there are two options: live with less or pay for more bandwidth. For something at the scale of a large city, one doesn't simply call the internet company and ask for an upgrade. If the infrastructure isn't there, it has to be built, and there are also costs associated with maintenance of additional hardware. City governments aren't known for being awash with cash, so you can't always assume that it's possible to buy or build enough infrastructure. Edge computing can help by reducing the need for new infrastructure and making better use of what's there, thus saving us from the Law of Economics.

Privacy and security issues are particularly salient in our society at the moment. With numerous high-profile data breaches, people are rightfully cautious about what happens to their personal data. The idea of a giant network of microphones that are always on and connected to a data center somewhere is more than a little unnerving. There are legal and political ramifications to the actions of the city government, and indiscriminate recording of the citizenry may not go over well. Once again, edge computing can help. Pushing the processing to those small computers at the edge of the network lets us reduce how far the sensitive data must travel and thus limits opportunities for interception or interference. Also, you can avoid recording all of the audio by storing only the last 30 seconds received and overwriting old data with the new in a structure called a *ring buffer*. When a gunshot is detected, one could dump the contents of the buffer and begin recording on all the microphones that "heard" the shot, thus only saving data that's likely to be relevant. Take that, Law of the Land!

1.5.2 Managing patient alerts in a hospital

Modern medical practice generates massive amounts of data in various forms. Perhaps the best-known example of this is diagnostic imaging, such as CT, MRI, PET, and x-ray. Other forms of digitized medical data include the various sensors and machines to monitor patient conditions, whole-slide images (WSI), health records, patient charts,

physicians' notes, pathology reports, and lab results. A particularly important concern is the need to maintain patient privacy and control of data, often codified in laws like the Health Insurance Portability and Accountability Act, better known as HIPAA. Once again, you have large amounts of data (especially streaming data), privacy, ethical and legal concerns, and the need for rapid responses. The healthcare industry is fertile ground for the growth of edge computing, especially because edge computing helps to address the privacy concerns. Privacy concerns are often cited as one of the main barriers to innovation and exchange of information in the healthcare industry.

As an example, imagine a fictitious patient monitoring system that integrates with various hospital information systems to provide a picture of a patient's health status with links to the patient's records. When the system detects a condition that requires attention, it pages the appropriate personnel so they can respond immediately. The hospital is part of a network of hospitals that have agreed to use the same system to facilitate the secure and seamless transfer of patient records as needed by different healthcare providers from different institutions. Physicians often list excessive paperwork and frustration with electronic health records (EHR) as one of the things that most negatively impacts their practice, so this feature is an important part of the system.

Designing such a system presents numerous difficulties. Standard formats for EHR that are already well-suited for sharing between institutions, but securely transferring only what's needed when it's needed presents a tougher challenge because all the participating hospitals must coordinate in near real-time. Edge computing provides tools to help achieve that coordination without extra investment in hardware or infrastructure. As with the smarter city example, this system uses various machine learning models to detect health problems. But in this case, the models are personalized, so they're at least partly trained using the patient's own data. This allows for better, more personal care and could be easily and securely shared with other institutions. Rather than send all of the patient's data, only the details of the machine learning model are sent. Through *differential privacy*, or the concept of observing patterns about a dataset without revealing individual items of said dataset, it's difficult to reverse engineer the models to arrive at unencrypted patient data, thus keeping your patient data safe and secure.

Such a system would be expensive or downright impossible to build without edge computing. The political and legal factors at play would only serve to increase the difficulty. To make the application we've described work, you have to move large amounts of data as quickly as possible to a data center that continuously trains the models and runs them against the input data. Processing the huge amounts of data required in the short time permissible would take tons of hardware and infrastructure if you take a traditional client-server approach: hardware for obvious reasons and infrastructure to get the data and responses to their destinations quickly enough. As before, you can overcome these difficulties by moving processing closer to the point of data generation, greatly reducing the amount of data that needs to be sent through the interior portions of the network. Part of the monitoring equipment may be able to run the models generated for the patient to detect negative conditions, and the more

expensive training part could be done on a workstation or small server located near the patient's room instead of in a central hospital data center.

If distributed storage is also employed, the only thing you need to centralize is the "command and control" of the system, including the alerts, and an index of where each patient's records reside. Each hospital could then share their index with the others and provide a secure interface that returns an encrypted copy of a patient's records when queried. Existing techniques for achieving consensus in distributed systems could be employed to ensure that all institutions that need a copy of a patient's record have it and that it's kept up to date. Such an interface would also allow for authorized entities to revoke the right to store a patient's records, triggering an automatic deletion of the encryption key needed to decrypt the data and thus rendering it unreadable, even if backups are taken. The keys would have to be kept secure, and you'd need a way to make sure that the key couldn't be backed up, and a bunch of other things you haven't even thought of yet but know to be out there, where the wild things are.

The implementation details of the security system for our hypothetical app aren't needed to illustrate that edge computing could help overcome numerous security challenges, so we ask you to hold your questions and suspend your disbelief a little in the interest of brevity. The important ideas are that less data transmitted means less to be intercepted and edge techniques can be used with relatively inexpensive hardware to perform encryption/decryption at the point of generation and use.

1.5.3 Hobbyist app: personal fitness coach

At this point, you might think that edge computing is only useful or applicable for big institutions doing big things, but it can be as useful for smaller projects. For this example, suppose you need a personal fitness coaching app that uses heart rate, GPS, barometric pressure, and accelerometer data to deliver real-time coaching cues to runners and cyclists. There are free web-based services that have advanced analytics and tracking, but they require surrendering one's data to be analyzed. They also have the drawback of requiring an internet connection to function, which means they can't be used in remote areas where connectivity is limited or unavailable. A programmer with an interest in endurance sports or an athlete with an interest in programming could employ edge computing techniques to create an app that doesn't require sending off their personal data yet allows for cloud storage when it's available.

At the heart of this app is the analysis of historical data and the use of real-time data to gauge how hard an athlete is exerting himself and how hard he *should* be exerting himself according the goals he's defined. Cyclists already have this in the form of power meters and cycle computers that integrate power, cadence, speed, heart rate, and GPS data and can include the ability to store user-generated workouts. Power meters for runners exist but have not yet seen widespread adoption among the running public, so they must rely on other methods such as pace and heart rate. The proposed app differs because it not only "knows" your fitness level and can compare your current performance to a predefined standard, but it can guess what you're capa-

ble of on any day by using machine learning models. The machine learning models will be trained iteratively, with each new run or ride used to improve the models. These will be run against the data being recorded during a run or ride to predict how hard the athlete is working. Because GPS and barometric data is also available, the software could also dynamically adjust workout intensity based on upcoming terrain to keep the difficulty level of the workout at or below desired levels, which would be useful for exploring new areas.

While it may be possible to run an app like we've described in the cloud, it would mean you can't use it where there's no internet connection available. You could run it entirely on a mobile phone or similar small portable computer, but you'd run out of storage space fairly quickly. What if you could use both? You already have a small computer on hand in the form of a mobile phone or cycle computer, so you can run the most recent version of the machine learning model against the input data on those devices, which also record the data. After a workout, those devices can upload the data to a home or cloud-based server, where it's used to update the models and can be archived for further analysis. As long as a model has been downloaded and the various sensors are operational, the system will work without an internet connection.

We've only scratched the surface of edge computing with these examples. After a brief introduction to the edge-computing testbed you'll be using in future demonstrative examples, we'll dive deeper into the components of edge computing. Then, once you have a firmer grasp, we'll connect those concepts to the real world with fully workable demonstration applications in subsequent chapters to get your hands on real edge-computing solutions that you can hopefully draw inspiration from in your own applications and workflows.

Summary

- Edge computing is an approach to distributed computing in which data processing components are deployed in close proximity to the sources of data generation as a means of efficiently using computing and storage resources in your distributed infrastructure.
- The layers of a typical edge computing system help build an infrastructure that supports functional application deployment at almost any point of the network in which enough computing and storage resources exist. This yields efficient resource utilization and dynamic reallocation as natural benefits.
- Multiple areas exist in which existing and future systems can gain benefit from using an edge computing approach. From larger city-scale operations, critical clinical-facing applications, to the hobbyist who wants to keep better track of their health, there's something in it for everyone with edge computing.
- The three laws of IoT: physics, economics, and land, limit the effectiveness of centralized systems. One approach to limiting the consequences of these laws it to leverage functional distribution methods, with edge computing being a prime candidate.