An in-depth guide to AWS

Amazon Web Services IN ACTION

THIRD EDITION

Andreas Wittig
Michael Wittig

MANNING

AWS Services Explained in the Book

Compute and Networking

Abbr.	Name	Description	Section
EC2	Amazon Elastic Compute Cloud	Virtual machines with Linux and Windows	3
	AWS Lambda	Run code without the need for virtual machines	6
	AWS App Runner	The simplest way to run containers on AWS	18.2
ECS	Amazon Elastic Container Service	Container orchestration layer	18.3
	AWS Fargate	Container execution layer	18.4
EIP	Elastic IP Address	Fixed public IP address for EC2 instances	3.6
ENI	Amazon EC2 Elastic Network Interface	Virtual network interface for EC2 instances	3.7
VPC	Amazon Virtual Private Cloud	Private network inside the cloud	5.5
	Amazon EC2 Security Group	Network firewall	5.4

Deployment and Management

Abbr.	Name	Description	Section
	AWS CodeDeploy	Deployment tool for in-place deployments	15.1
	Packer by HashiCorp	Deploying customized AMIs	15.3
	AWS CloudFormation	Infrastructure automation and deployment tool	4.4
IAM	AWS Identity and Access Management	Secure access to your cloud resources (authentication and authorization)	5.3
CLI	AWS command-line interface	AWS in your terminal	4.2
SDK	AWS software development kits	AWS in your applications	4.3

Praise for the second edition

Slices through the complexity of AWS using examples and visuals to cement knowledge in the minds of readers.

—From the foreword by Ben Whaley, AWS community hero and author

The authors' ability to explain complex concepts is the real strength of the book.

—Antonio Pessolano, Consoft Sistemi

Useful examples, figures, and sources to help you learn efficiently.

—Christof Marte, Daimler-Benz

Does a great job of explaining some of the key services in plain English so you have the knowledge necessary to dig deeper.

—Ryan Burrows, Rooster Park Consulting

This is a great book that covers all aspects of Amazon Web Services, from top to bottom.

—Ariel Gamino, Northwestern Medicine

A great way to learn AWS step by step, using the Free Tier.

—Jose San Leandro, DevOps, OSOCO.es

A perfect journey to the world of Amazon Web Services.

—Jean-Pol Landrain, Agile Partner

Amazon Web Services in Action

THIRD EDITION

AN IN-DEPTH GUIDE TO AWS

ANDREAS WITTIG
MICHAEL WITTIG

MANNING
SHELTER ISLAND

For online information and ordering of this and other Manning books, please visit www.manning.com. The publisher offers discounts on this book 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: orders@manning.com


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

Development editor: Frances Lefkowitz
Review editor: Aleksandar Dragosavljević
Production editor: Kathleen Rossland
Copy editor: Pamela Hunt
Proofreader: Keri Hales
Technical proofreader: Shawn Bolan
Typesetter: Dennis Dalinnik
Cover designer: Marija Tudor

ISBN: 9781633439160
Printed in the United States of America

brief contents

contents

preface

When we started our career as software developers in 2008, we didn't care about operations. We wrote code, and someone else was responsible for deployment and operations. A huge gap existed between software development and IT operations back then. On top of that, releasing new features was risky because it was impossible to test all the changes to software and infrastructure manually. Every six months, when new features needed to be deployed, we experienced a nightmare.

Then, in 2012, we became responsible for a product: an online banking platform. Our goal was to iterate quickly and to be able to release new features for the product every week. Our software was responsible for managing money, so the quality and security of the software and infrastructure were as important as the ability to innovate. But the inflexible on-premises infrastructure and the outdated process of deploying software made those goals impossible to reach. We started to look for a better way.

Our search led us to Amazon Web Services, which offered a flexible and reliable way to build and operate our applications. The possibility of automating every part of our infrastructure struck us as fascinating and innovative. Step by step, we dove into the different AWS services, from virtual machines to distributed message queues. Being able to outsource tasks like operating an SQL database or a load balancer saved us a lot of time. We invested this time in automating the testing and operations for our entire infrastructure.

The changes that took place during this transformation to the cloud went beyond the technical. After a while, the software architecture changed from a monolithic application to microservices, and the separation between software development and

operations got very blurry—and, in some cases, disappeared altogether. Instead, we built our organization around the core principle of DevOps: you build it, you run it.

Since 2015, we have worked as independent consultants, helping our clients get the most out of AWS. We have accompanied startups, midsized companies, and enterprise corporations on their journey to the cloud. Along the way, we have identified—and solved—the common challenges that confront companies of all sizes as they move to the cloud. In fact, we ended up turning some of our solutions into products to sell on the AWS Marketplace.

We enjoyed writing the first edition of our book in 2015. The astonishing support from Manning and our MEAP readers allowed us to finish the whole book in only nine months. Above all, it was a pleasure to observe you—our readers—using our book to get started with AWS or deepen your knowledge with the platform.

AWS is always innovating and constantly releasing new features or whole new services. So, in 2018, we released a second edition of the book, updated and revised based on the feedback of our readers. The second edition added three more chapters to cover newer developments—Lambda, EFS, and ElastiCache—and updated all the previous chapters.

Now, in 2023, it is time to update our book once again. In this third edition, we meticulously reviewed every chapter, updating the text and screenshots so they match the current way things work on the AWS platform. We've also added new content, including a chapter on containerized architectures as well as sections about CodeDeploy, Packer, and more.

We hope you enjoy the third edition of *Amazon Web Services in Action* as much as we do!

acknowledgments

Writing a book is time consuming. We invested our time, and other people did as well. Thank you to everyone involved!

We want to thank all the readers who bought the MEAP edition of this book. Thanks for overlooking the rough edges and focusing on learning about AWS instead. Your feedback helped us polish the final version of the book that you are now reading.

Thank you to all the people who posted comments in the book's liveBook forum and who provided excellent feedback, which improved the book.

Thanks to all the reviewers of the third, second, and first editions who provided detailed comments from the first to the last page. To all the reviewers of this edition: Adrian Rossi, Alessandro Campeis, Amitabh P. Cheekoth, Andres Sacco, Ashley Eatly, Bobby Lin, Brent Honadel, Chris Villanueva, Darnell Gadberry, Edin Kapic, Ernesto Cardenas Cangahuala, Floris Bouchot, Franklin Neves, Frans Oilinki, Ganesh Swaminathan, George Onofrei, Gilberto Taccari, Jeffrey Chu, Jeremy Chen, John Larsen, John Zoetebier, Jorge Bo, Kamesh Ganesan, Kent Spillner, Matteo Battista, Matteo Rossi, Mohammad Shahnawaz Akhter, Philip Patterson, Rahul Modpur, Roman Levchenko, Simeon Leyzerzon, Simone Sguazza, Uziel Linares, Venkatesh Rajagopal, and Vidhya Vinay—your feedback helped shape this book. We hope you like it as much as we do.

Special thanks to Michael Labib for his input and feedback on chapter 11 covering AWS ElastiCache.

Furthermore, we want to thank the technical editors, John Hyaduck and Jonathan Thoms. Your unbiased and technical view on Amazon Web Services helped to perfect our book.

Shawn P. Bolan made sure all the examples in this third edition work as expected. Thanks for proofing the technical parts of our book. Thanks to David Fombella Pombal and Doug Warren for proofing the technical parts in previous editions.

We also want to thank Manning Publications for placing their trust in us. Especially, we want to thank the following staff at Manning for their excellent work:

- Frances Lefkowitz, our development editor, who guided us through the process of writing the second and third editions. Her writing and teaching expertise is noticeable in every part of our book. Thanks for your support.
- Dan Maharry, our development editor for the first edition. Thanks for taking us by the hand from writing the first pages to finishing our first book.
- Aleksandar Dragosavljević, our review editor, who organized the reviews of our book. Thanks for making sure we got valuable feedback from our readers.
- Tiffany Taylor, our copyeditor, who perfected our English in the first two editions, and Pamela Hunt, who copyedited the third edition. We know you had a hard time with us, but our mother tongue is German, and we thank you for your efforts.
- Charlotte Harborne, Ana Romac, and Christopher Kaufmann, who helped us to promote this book.
- Ivan Martinović, who answered our many questions regarding the technical aspects of writing a book in Asciidoc.
- And thanks to the production staff, who worked behind the scenes to take our rough draft and turn it into a real book.

Last but not least, we want to thank the significant people in our lives who supported us as we worked on the book.

about this book

Our book guides you from creating an AWS account to building fault-tolerant and autoscaling applications. You will learn about services offering compute, network, and storage capacity. We get you started with everything you need to run web applications on AWS: load balancers, virtual machines, containers, file storage, database systems, and in-memory caches.

The first part of the book introduces you to the principles of Amazon Web Services and gives you a first impression of the possibilities in the cloud. Next, in part 2, you will learn about fundamental compute and network services. In part 3, we demonstrate six different ways to store your data. Finally, part 4 focuses on architecting on AWS: highly available or even fault-tolerant architectures using load balancers and queues, containerized applications, deployment options, and autoscaling strategies to scale your infrastructure dynamically as well.

Amazon offers a wide variety of services—more than 200 services in 25 categories at last count, with more added regularly. Unfortunately, the number of pages within a book is limited. Therefore, you will not find instructions for all AWS services in this book. What you *will* find is a collection of the most important and universally popular services. We consider these services the essential toolkit, the ones you need to get up and running and get your business done. You could operate fine with just these services, but once you have mastered them, we hope you will have the confidence and curiosity to explore what else is out there—for instance: Machine Learning as a Service, anyone?

Automation sneaks in throughout the book, so by the end, you'll be comfortable with using AWS CloudFormation, an Infrastructure as Code tool that allows you to

manage your cloud infrastructure in an automated way; this will be one of the most important things you will learn from our book.

Most of our examples use popular web applications to demonstrate important points. We use tools offered by AWS instead of third-party tools whenever possible, because we appreciate the quality and support offered by AWS. Our book focuses on the different aspects of security in the cloud, for example, by following the "least-privilege" principle when accessing cloud resources.

We focus on Linux as the operating system for virtual machines. Our examples are based on open source software.

Amazon operates data centers in various geographic regions around the world. To simplify the examples, we use the region US East (N. Virginia). You will also learn how to switch to another region to use resources in the region Asia Pacific (Sydney).

About the third edition

In this third edition, we have revised all of the previous 17 chapters. AWS has made significant progress since the second edition in 2018. As a result, we incorporated countless new features into the third edition. Of course, we also updated all the examples.

The most significant change is the addition of chapter 18, "Building modern architecture for the cloud: ECS, Fargate, and App Runner." The brand-new chapter discusses deploying a web application using containers. We start with a simple example based on App Runner and end the chapter with a cloud-native architecture based on ALB, ECS, Fargate, and S3. We also rewrote chapter 15, "Automating deployment: CloudFormation, CodeDeploy, and Packer," to provide you the tools to deploy your applications to AWS.

Who should read this book

Amazon Web Services is a toolbox. You can find tools to run a website that can sell goods and services to the general public, but you can also host private applications securely and economically, which a corporation with thousands of customers depends on. Tools are also available to crunch numbers or to train your ML models. The possibilities go on and on. Reading this book should help you get used to the most common tools. Once you are familiar with the common tools, you are equipped to explore the rest of the toolbox on your own.

You don't need much training to read, understand, and adapt the lessons from this book to your own needs. Familiarity with Linux computers, the markup language YAML, and an understanding of basic networking concepts are all you need to get started. You don't even need an AWS account—we'll show you how to sign up for one in chapter 1.

How this book is organized: A road map

Chapter 1 introduces cloud computing and Amazon Web Services. You'll learn about key concepts and basics, and you'll create and set up your AWS account.

Chapter 2 brings Amazon Web Services into action. You'll spin up and dive into a complex cloud infrastructure with ease.

Chapter 3 is about working with a virtual machine. You'll learn about the key concepts of the Elastic Compute Service (EC2) with the help of a handful of practical examples.

Chapter 4 presents different approaches for automating your infrastructure: the AWS Command Line Interface (CLI) from your terminal, the AWS SDKs to program in your favorite language, and AWS CloudFormation, an Infrastructure as Code tool.

Chapter 5 is about security. You'll learn how to secure your networking infrastructure with private networks and firewalls. You'll also learn how to protect your AWS account and your cloud resources.

Chapter 6 is about automating operational tasks with AWS Lambda. You will learn how to execute small code snippets in the cloud without needing to launch a virtual machine.

Chapter 7 introduces the Amazon Simple Storage Service (S3), a service offering object storage, and Amazon Glacier, a service offering long-term storage. You'll learn how to integrate object storage into your applications to implement a stateless server by creating an image gallery.

Chapter 8 is about storing data from your virtual machines on hard drives with Amazon Elastic Block Storage (EBS) and instance storage. To get an idea of the different options available, you'll take some performance measurements.

Chapter 9 explains how to use a networking filesystem to share data among multiple machines. Therefore, we introduce the Amazon Elastic File System (EFS).

Chapter 10 introduces Amazon Relational Database Service (RDS), offering managed relational database systems like MySQL, PostgreSQL, Oracle, and Microsoft SQL Server. You will learn how to connect an application to an RDS database instance, for example.

Chapter 11 is about adding a cache to your infrastructure to speed up your application and save costs due to minimizing load on the database layer. Specifically, you will learn about Amazon ElastiCache, which provides Redis or Memcached as a service, as well as Amazon MemoryDB for Redis.

Chapter 12 introduces Amazon DynamoDB, a NoSQL database offered by AWS. DynamoDB is typically not compatible with legacy applications. You need to rework your applications to use DynamoDB. You'll implement a to-do application in this chapter.

Chapter 13 explains what's needed to make your infrastructure highly available. You'll learn how to recover automatically from a failed virtual machine or even a whole data center.

Chapter 14 introduces the concept of decoupling your system to increase reliability. You'll learn how to use synchronous decoupling with the help of Elastic Load Balancing (ELB). Asynchronous decoupling is also part of this chapter; we explain how to use the Amazon Simple Queue Service (SQS), a distributed queuing service, to build a fault-tolerant system.

Chapter 15 introduces three different ways to deploy software to AWS. You'll use each of the tools to deploy an application to AWS in an automated fashion.

Chapter 16 dives into building fault-tolerant applications based on the concepts explained in chapters 13 and 14. You'll create a fault-tolerant image-processing web service within this chapter.

Chapter 17 is all about flexibility. You'll learn how to scale the capacity of your infrastructure based on a schedule or based on the current load of your system.

Chapter 18 explains ways to deploy containers on AWS. You'll learn to use ECS with Fargate and App Runner to run your containerized application.

AWS costs

AWS offers a Free Tier, which allows you to experiment with a number of services for at least a full year at no charge. Most of the projects we walk you through in this book can be done within the Free Tier. For the few processes we teach that do go beyond the Free Tier, we provide a clear warning for you, so you can opt out if you do not want to incur charges. In chapter 1, you'll learn much more about how AWS charges for services, what's covered in the Free Tier, and how to set budgets and alerts so you don't receive any unexpected bills from AWS.

About the code

You'll find four types of code listings in this book: bash, YAML, Python, and Node.js/JavaScript. We use bash to create tiny scripts to interact with AWS in an automated way. YAML is used to describe infrastructure in a way that AWS CloudFormation can understand. In addition, we use Python to manage our cloud infrastructure. Also, we use the Node.js platform to create small applications in JavaScript to build cloud-native applications.

All source code in listings or in text is in `a fixed-width font like this` to separate it from ordinary text. Code annotations accompany many of the listings, highlighting important concepts. In some cases, numbered bullets link to explanations that follow the listing, and sometimes we needed to break a line into two or more to fit on the page. In our bash code, we used the continuation backslash. The $ at the beginning indicates that the following line was an input. If you are using Windows, you have to make the following adjustments: the leading $ can be ignored. In PowerShell: replace the continuation backslash \ with a `. At the command prompt: replace \ with a ^. An artificial line break is indicated by this symbol: ➥.

You can get executable snippets of code from the liveBook (online) version of this book at https://livebook.manning.com/book/amazon-web-services-in-action-third-edition. The complete code for the examples in the book is available for download from the Manning website at https://www.manning.com/books/amazon-web-services-in-action-third-edition, and from GitHub at https://github.com/AWSinAction/code3/.

liveBook discussion forum

Purchase of *Amazon Web Services in Action, Third Edition*, includes free access to liveBook, Manning's online reading platform. Using liveBook's exclusive discussion features, you can attach comments to the book globally or to specific sections or paragraphs. It's a snap to make notes for yourself, ask and answer technical questions, and receive help from the author and other users. To access the forum, go to https://livebook.manning.com/book/amazon-web-services-in-action-third-edition/discussion. You can also learn more about Manning's forums and the rules of conduct at https://livebook.manning.com/discussion.

Manning's commitment to our readers is to provide a venue where a meaningful dialogue between individual readers and between readers and the author can take place. It is not a commitment to any specific amount of participation on the part of the authors, whose contribution to the forum remains voluntary (and unpaid). We suggest you try asking them some challenging questions lest their interest stray! The forum and the archives of previous discussions will be accessible from the publisher's website as long as the book is in print.

about the authors

ANDREAS WITTIG and **MICHAEL WITTIG** are software engineers and consultants, focusing on Amazon Web Services. The brothers started building on AWS in 2013 when migrating the IT infrastructure of a German bank to AWS—the first bank in Germany to do so. Since 2015, Andreas and Michael have worked as consultants, helping their clients migrate and run their workloads on AWS. They focus on Infrastructure as Code, continuous deployment, serverless applications based on AWS Lambda, containers, and security. Andreas and Michael are building SaaS products on top of Amazon's cloud as well. On top of that, Andreas and Michael love to share their knowledge and teach others how to use Amazon Web Services through their book, *Amazon Web Services in Action*, as well as their blog, podcast, and YouTube channel at cloudonaut.io.

about the cover illustration

The figure on the cover of *Amazon Web Services in Action, Third Edition*, is "Paysan du Canton de Lucerne," or "A Peasant from the Canton of Lucerne," taken from a collection by Jacques Grasset de Saint-Sauveur, published in 1797. Each illustration is finely drawn and colored by hand.

In those days, it was easy to identify where people lived and what their trade or station in life was just by their dress. Manning celebrates the inventiveness and initiative of the computer business with book covers based on the rich diversity of regional culture centuries ago, brought back to life by pictures from collections such as this one.

Part 1

Getting started

Have you watched a blockbuster on Netflix, bought a gadget on Amazon.com, or booked a room on Airbnb today? If so, you have used Amazon Web Services (AWS) in the background. Because Netflix, Amazon.com, and Airbnb all use AWS for their business.

AWS is the biggest player in the cloud computing markets. According to analysts, AWS maintains a market share of more than 30%.[1] Another impressive number: AWS accounts for net sales of $20.5 billion year-over-year (a 27% increase).[2] AWS data centers are distributed worldwide in North America, South America, Europe, Africa, Asia, and Australia. But the cloud does not consist of hardware and computing power alone. Software is part of every cloud platform and makes the difference for you, as a customer who aims to provide a valuable experience to your service's users. The research firm Gartner has yet again classified AWS as a leader in their Magic Quadrant for Cloud Infrastructure & Platform Services in 2022. Gartner's Magic Quadrant groups vendors into four quadrants—niche players, challengers, visionaries, and leaders—and provides a quick overview of the cloud computing market.[3] Being recognized as a leader attests to AWS's high speed and high quality of innovation.

---

1 Statista, "Global Cloud Infrastructure Market Share 2022," http://mng.bz/Popv.

2 Amazon, "Amazon.com Announces Third Quarter Results 2022," http://mng.bz/JVXa.

3 AWS Blog, "AWS Named as a Leader in the 2022 Gartner Cloud Infrastructure & Platform Services (CIPS) Magic Quadrant for the 12th Consecutive Year," http://mng.bz/wy7a.

The first part of this book will guide you through your initial steps with AWS. You will get an impression of how you can use AWS to improve your IT infrastructure.

Chapter 1 introduces cloud computing and AWS. This will get you familiar with the big-picture basics of how AWS is structured.

Chapter 2 brings Amazon Web Service into action. Here, you will spin up and dive into a complex cloud infrastructure with ease.

1 What is Amazon Web Services?

This chapter covers

- Overview of Amazon Web Services
- The benefits of using Amazon Web Services
- What you can do with Amazon Web Services
- Creating and setting up an AWS account

Almost every IT solution gets labeled with the term *cloud computing* or even just *cloud* nowadays. Buzzwords like these may help sales, but they're hard to work with when trying to teach—or learn—how to work with these technologies. So, for the sake of clarity, let's start this book by defining some terms.

Cloud computing, or the cloud, is a metaphor for supply and consumption of IT resources. The IT resources in the cloud aren't directly visible to the user; layers of abstraction exist in between. The level of abstraction offered by the cloud varies, from offering virtual machines (VMs) to providing Software as a Service (SaaS) based on complex distributed systems. Resources are available on demand in enormous quantities, and you pay for what you use.

The official definition from the National Institute of Standards and Technology follows:

> *Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (networks, virtual machines, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.*
>
> —National Institute of Standards and Technology

Also, NIST defines the following five essential characteristics for cloud computing:

- *On-demand self-service*—The cloud enables us to provision resources ad hoc with the click of a button or an API call.
- *Broad network access*—Capabilities are available over the network.
- *Resource pooling*—The cloud assigns resources based on a multitenant model, which means consumers share the same physical and virtual resources.
- *Rapid elasticity*—The cloud allows us to expand and shrink the provisioned capacity on demand.
- *Measured service*—The cloud provides metrics allowing consumers to gain insights into the utilization of their resources.

Besides that, offerings are often divided into the following three types:

- *Public*—A cloud managed by an organization and open to use by the general public
- *Private*—A cloud that virtualizes and distributes the IT infrastructure for a single organization
- *Hybrid*—A mixture of a public and a private cloud

Amazon Web Services (AWS) is a public cloud. By combining your on-premises data center with AWS, you are building a hybrid cloud.

Cloud computing services also have several classifications, described here:

- *Infrastructure as a Service (IaaS)*—Offers fundamental resources like computing, storage, and networking capabilities, using virtual machines such as Amazon EC2, Google Compute Engine, and Microsoft Azure Virtual Machines.
- *Platform as a Service (PaaS)*—Provides platforms to deploy custom applications to the cloud, such as AWS Lambda, AWS App Runner, Google App Engine, and Heroku.
- *Software as a Service (SaaS)*—Combines infrastructure and software running in the cloud, including office applications like Amazon WorkSpaces, Google WorkSpace, and Microsoft 365.

AWS is a cloud-computing provider with a wide variety of IaaS, PaaS, and SaaS offerings. Let's go into a bit more detail about what AWS is and does.

1.1 What is Amazon Web Services (AWS)?

Amazon Web Services (AWS) is a platform of web services that offers solutions for computing, storing, and networking, at different layers of abstraction. For example, you can attach volumes to a virtual machine—a low level of abstraction—or store and retrieve data via a REST API—a high level of abstraction. Use the services provided by AWS to host websites, run enterprise applications, and mine tremendous amounts of data. *Web services* are accessible via the internet by using typical web protocols (such as HTTP) and are used by machines or by humans through a UI. The most prominent services provided by AWS are *EC2*, which offers virtual machines, and *S3*, which offers storage capacity. Services on AWS work well together: you can use them to migrate existing on-premises infrastructures or build from scratch. The pricing model for services is pay-per-use.

As an AWS customer, you can choose among different *data centers*. AWS data centers are distributed worldwide. For example, you can start a virtual machine in Japan in exactly the same way as you would start one in Ireland. This enables you to serve customers worldwide.

The map in figure 1.1 shows AWS's data centers. Access to some of them is limited: some data centers are accessible for US government organizations only, and special conditions apply for the data centers in China. Additional data centers have been announced for Canada, Spain, Switzerland, Israel, UAE, India, Australia, and New Zealand.

* Limited access

Figure 1.1 AWS data center locations

Now that we have defined the most important terms, the question is: what can you do with AWS?

1.2 What can you do with AWS?

You can run all sorts of application on AWS by using one or a combination of services. The examples in this section will give you an idea of what you can do.

1.2.1 Hosting a web shop

John is CIO of a medium-sized e-commerce business. He wants to develop a fast, reliable, and scalable web shop. He initially decided to host the web shop on-premises, and three years ago, he rented machines in a data center. A web server handles requests from customers, and a database stores product information and orders. John is evaluating how his company can take advantage of AWS by running the same setup on AWS, as shown in figure 1.2.

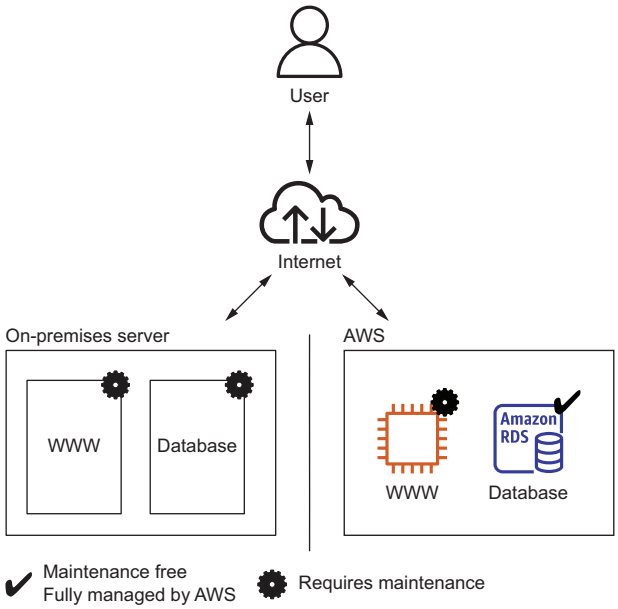


Figure 1.2 Running a web shop on-premises vs. on AWS

John not only wants to lift-and-shift his current on-premises infrastructure to AWS, he wants to get the most out of the advantages the cloud is offering. Additional AWS services allow John to improve his setup as follows:

- The web shop consists of dynamic content (such as products and their prices) and static content (such as the company logo). Splitting these up would reduce the load on the web servers and improve performance by delivering the static content over a content delivery network (CDN).
- Switching to maintenance-free services, including a database, an object store, and a DNS system, would free John from having to manage these parts of the system, decreasing operational costs and improving quality.
- The application running the web shop can be installed on virtual machines. Using AWS, John can run the same amount of resources he was using on his on-premises machine but split them into multiple, smaller virtual machines at no extra cost. If one of these virtual machines fails, the load balancer will send customer requests to the other virtual machines. This setup improves the web shop's reliability.

Figure 1.3 shows how John enhanced his web shop setup with AWS.

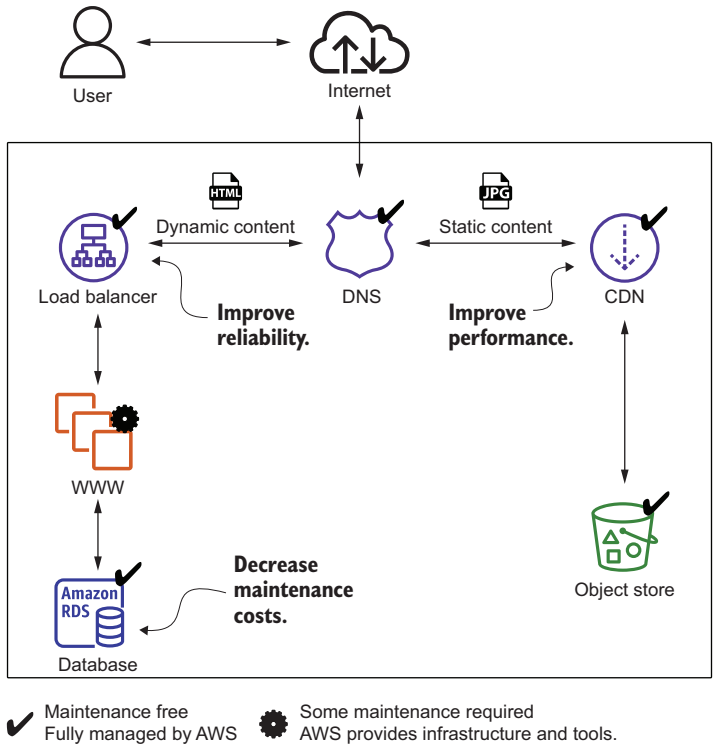


Figure 1.3 Running a web shop on AWS with CDN for better performance, a load balancer for high availability, and a managed database to decrease maintenance costs

John is happy with running his web shop on AWS. By migrating his company's infrastructure to the cloud, he was able to increase the reliability and performance of the web shop.

1.2.2 Running a Java EE application in your private network

Maureen is a senior system architect in a global corporation. She wants to move parts of her company's business applications to AWS when the data center contract expires in a few months, to reduce costs and gain flexibility. She would like to run enterprise applications (such as Java Enterprise Edition [EE] applications) consisting of an application server and an SQL database on AWS. To do so, she defines a virtual network in the cloud and connects it to the corporate network through a virtual private network (VPN) connection. She installs application servers on virtual machines to run the Java EE application. Maureen also wants to store data in an SQL database service (such as Oracle Database EE or Microsoft SQL Server EE).

For security, Maureen uses subnets to separate systems with different security levels from each other. By using access-control lists, she can control ingoing and outgoing traffic for each subnet. For example, the database is accessible only from the Java EE server's subnet, which helps to protect mission-critical data. Maureen controls traffic to the internet by using network address translation (NAT) and firewall rules as well. Figure 1.4 illustrates Maureen's architecture.

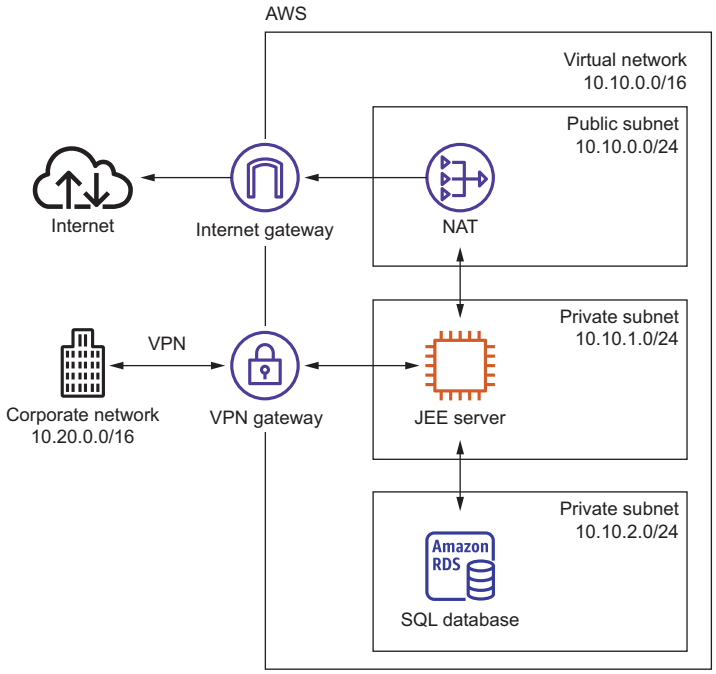


Figure 1.4 Running a Java EE application with enterprise networking on AWS improves flexibility and lowers costs.

Maureen has managed to connect the local data center with a private network running remotely on AWS to enable clients to access the Java EE server. To get started, Maureen uses a VPN connection between the local data center and AWS, but she is already thinking about setting up a dedicated network connection to reduce network costs and increase network throughput in the future.

The project was a great success for Maureen. She was able to reduce the time needed to set up an enterprise application from months to hours because AWS provides virtual machines, databases, and even the networking infrastructure on demand within a few minutes. Maureen's project also benefits from lower infrastructure costs on AWS, compared to using its own infrastructure on-premises.

1.2.3 Implementing a highly available system

Alexa is a software engineer working for a fast-growing startup. She knows that Murphy's Law applies to IT infrastructure: anything that can go wrong *will* go wrong. Alexa is working hard to build a highly available system to prevent outages from ruining the business. All services on AWS are either highly available or can be used in a highly available way. So, Alexa builds a system like the one shown in figure 1.5 with a high availability architecture. The database service is offered with replication and fail-over handling. In case the primary database instance fails, the standby database is promoted as the new primary database automatically. Alexa uses virtual machines acting as web servers. These virtual machines aren't highly available by default, but Alexa launches multiple virtual machines in different data centers to achieve high availability. A load balancer checks the health of the web servers and forwards requests to healthy machines.

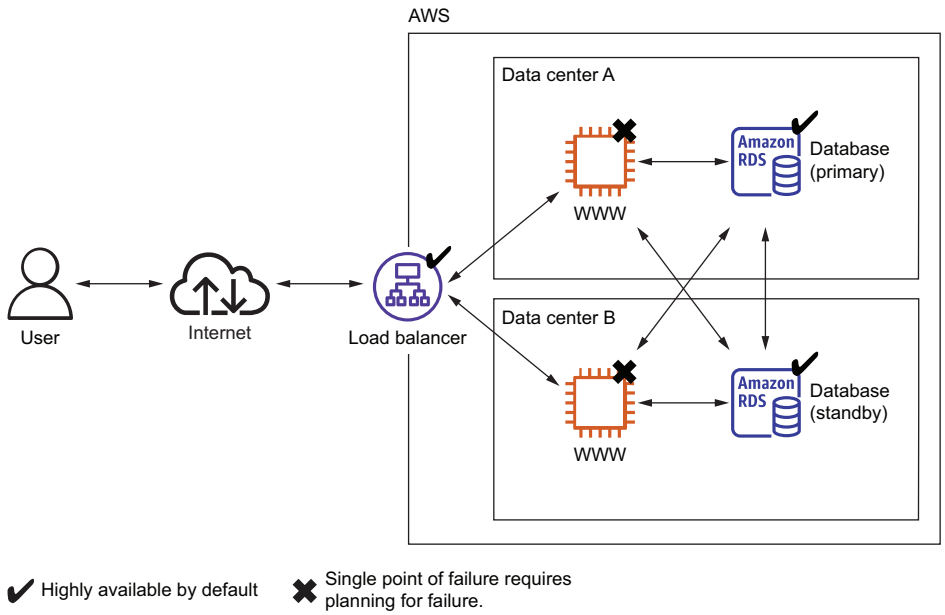


Figure 1.5 Building a highly available system on AWS by using a load balancer, multiple virtual machines, and a database with primary-standby replication

So far, Alexa has protected the startup from major outages. Nevertheless, she and her team are always planning for failure and are constantly improving the resilience of their systems.

1.2.4 Profiting from low costs for batch processing infrastructure

Nick is a data scientist who needs to process massive amounts of measurement data collected from gas turbines. He needs to generate a daily report containing the maintenance condition of hundreds of turbines. Therefore, his team needs a computing

infrastructure to analyze the newly arrived data once a day. Batch jobs are run on a schedule and store aggregated results in a database. A business intelligence (BI) tool is used to generate reports based on the data stored in the database.

Because the budget for computing infrastructure is very small, Nick and his team have been looking for a cost effective solution to analyze their data. He finds the following ways to make clever use of AWS's price model:

- *AWS bills virtual machines per second with a minimum of 60 seconds.* So Nick launches a virtual machine when starting a batch job and terminates it immediately after the job finishes. Doing so allows him to pay for computing infrastructure only when actually using it. This is a big game changer compared to the traditional data center where Nick had to pay a monthly fee for each machine, no matter how much it was used.
- *AWS offers spare capacity in their data centers at a substantial discount.* It is not important for Nick to run a batch job at a specific time. He can wait to execute a batch job until there is enough spare capacity available, so AWS offers him a virtual machine with a discount of 75%.

Figure 1.6 illustrates how Nick benefits from the pay-per-use price model for virtual machines.

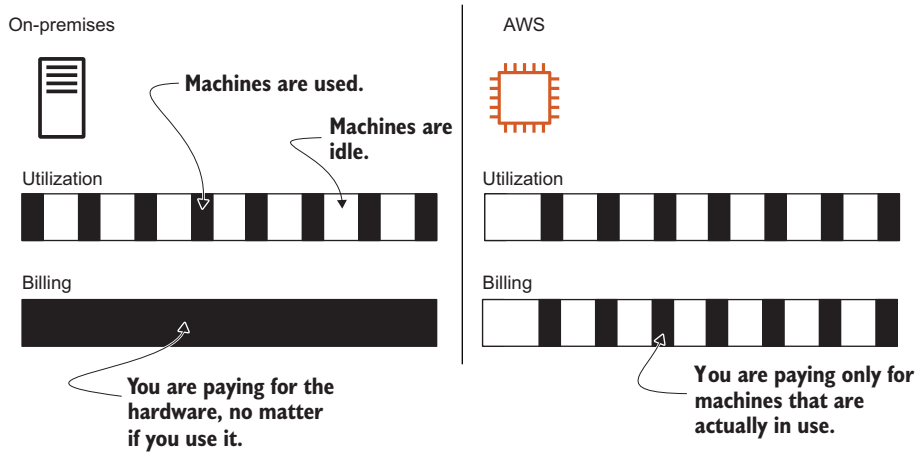


Figure 1.6 Making use of the pay-per-use price model of virtual machines

Nick is happy to have access to a computing infrastructure that allows his team to analyze data at low costs. You now have a broad idea of what you can do with AWS. Generally speaking, you can host any application on AWS. The next section explains the nine most important benefits AWS has to offer.

1.3 How you can benefit from using AWS

What's the most important advantage of using AWS? Cost savings, you might say. But saving money isn't the only advantage. Let's see how else you can benefit from using AWS by looking at some of its key features.

1.3.1 Innovative and fast-growing platform

AWS is announcing new services, features, and improvements constantly. Go to https://aws.amazon.com/about-aws/whats-new/ to get an impression of the speed of innovation. We counted 2,080 announcements in 2021. Making use of the innovative technologies provided by AWS helps you to generate valuable solutions for your customers and thus achieve a competitive advantage.

Amazon reported net sales of $62 billion for 2021. See http://mng.bz/lRqB if you are interested in the full report. We expect AWS to expand the size and extent of its platform in the upcoming years, for example, by adding additional services and data centers.

1.3.2 Services solve common problems

As you've learned, AWS is a platform of services. Common problems such as load balancing, queuing, sending email, and storing files are solved for you by services. You don't need to reinvent the wheel. It's your job to pick the right services to build complex systems. Let AWS manage those services while you focus on your customers.

1.3.3 Enabling automation

Because AWS is API driven, you can automate everything: write code to create networks, start virtual machine clusters, or deploy a relational database. Automation increases reliability and improves efficiency.

The more dependencies your system has, the more complex it gets. A human can quickly lose perspective, whereas a computer can cope with interconnected systems of any size. You should concentrate on tasks humans are good at—such as describing a system—while the computer figures out how to resolve all those dependencies to create the system. Setting up an environment in the cloud based on your blueprints can be automated with the help of infrastructure as code, covered in chapter 4.

1.3.4 Flexible capacity (scalability)

Flexible capacity reduces overcapacity. You can scale from one virtual machine to thousands of virtual machines. Your storage can grow from gigabytes to petabytes. You no longer need to predict your future capacity needs for the coming months and years to purchase hardware.

If you run a web shop, you have seasonal traffic patterns, as shown in figure 1.7. Think about day versus night, and weekday versus weekend or holiday. Wouldn't it be nice if you could add capacity when traffic grows and remove capacity when traffic

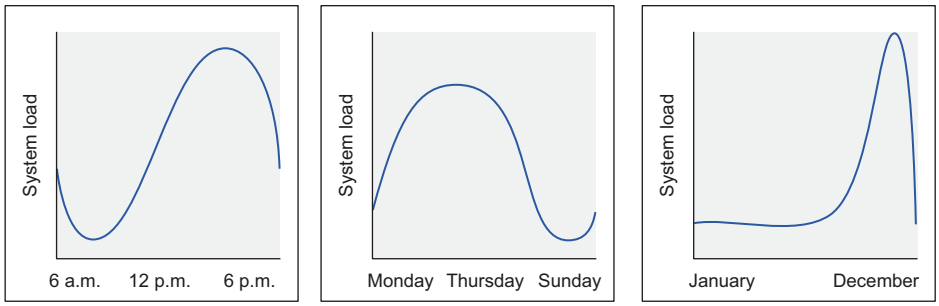


Figure 1.7 Seasonal traffic patterns for a web shop

shrinks? That's exactly what flexible capacity is about. You can start new virtual machines within minutes and throw them away a few hours after that.

The cloud has almost no capacity constraints. You no longer need to think about rack space, switches, and power supplies—you can add as many virtual machines as you like. If your data volume grows, you can always add new storage capacity.

Flexible capacity also means you can shut down unused systems. In one of our last projects, the test environment ran only from 7 a.m. to 8 p.m. on weekdays, allowing us to save 60%.

1.3.5 Built for failure (reliability)

Most AWS services are highly available or fault tolerant by default. If you use those services, you get reliability for free. Also, AWS provides tooling allowing you to build systems in a reliable way. It provides everything you need to create your own highly available or even fault-tolerant systems.

1.3.6 Reducing time to market

In AWS, you request a new virtual machine, and a few minutes later, that virtual machine is booted and ready to use. The same is true with any other AWS service available. You can use them all on demand.

Your development process will be faster because of the shorter feedback loops. You can eliminate constraints such as the number of test environments available; if you need another test environment, you can create it for a few hours.

1.3.7 Benefiting from economies of scale

AWS is increasing its global infrastructure constantly, and, therefore, AWS benefits from an economy of scale. As a customer, you will benefit partially from these effects.

AWS reduces prices for their cloud services every now and then. A few examples follow:

- In January 2019, AWS reduced the price for running containers on Fargate by 20% for vCPU and 65% for memory.

- In November 2020, AWS reduced prices for EBS volumes of type Cold HDD by 40%.
- In November 2021, AWS reduced prices for S3 storage by up to 31% in three storage classes.
- In April 2022, AWS removed additional costs for network traffic between data centers when using AWS PrivateLink, AWS Transit Gateway, and AWS Client VPN.

1.3.8 Global infrastructure

Are you serving customers worldwide? Making use of AWS's global infrastructure has the following advantages: having low network latencies between your customers and your infrastructure, being able to comply with regional data protection requirements, and benefiting from different infrastructure prices in different regions. AWS offers data centers in North America, South America, Europe, Africa, Asia, and Australia, so you can deploy your applications worldwide with little extra effort.

1.3.9 Professional partner

When you use AWS services, you can be sure that their quality and security follow the latest standards and certifications, such as the following:

- *ISO 27001*—A worldwide information security standard certified by an independent and accredited certification body.
- *ISO 9001*—A standardized quality management approach used worldwide and certified by an independent and accredited certification body.
- *PCI DSS Level 1*—A data security standard (DSS) for the payment card industry (PCI) to protect cardholders data.

Go to https://aws.amazon.com/compliance/ if you want to dive into the details. If you're still not convinced that AWS is a professional partner, you should know that Expedia, Volkswagen, FINRA, Airbnb, Slack, and many more are running serious workloads on AWS. To read about AWS customer success, go to https://aws.amazon.com/solutions/case-studies/.

We have discussed a lot of reasons to run your workloads on AWS. But what does AWS cost? You will learn more about the pricing models in the next section.

1.4 How much does it cost?

A bill from AWS is similar to an electric bill. Services are billed based on use. You pay for the time a virtual machine was running, the used storage from the object store, or the number of running load balancers. Services are invoiced on a monthly basis. The pricing for each service is publicly available; if you want to calculate the monthly cost of a planned setup, you can use the AWS Pricing Calculator (https://calculator.aws/).

1.4.1 Free Tier

You can use some AWS services for free within the first 12 months of signing up. The idea behind the Free Tier is to enable you to experiment with AWS and get some experience using its services. Here is a taste of what's included in the Free Tier:

- 750 hours (roughly a month) of a small virtual machine running Linux or Windows. This means you can run one virtual machine for a whole month, or you can run 750 virtual machines for one hour.
- 750 hours (or roughly a month) of a classic or application load balancer.
- Object store with 5 GB of storage.
- Small relational database with 20 GB of storage, including backup.
- 25 GB of data stored on NoSQL database.

If you exceed the limits of the Free Tier, you start paying for the resources you consume without further notice. You'll receive a bill at the end of the month. We'll show you how to monitor your costs before you begin using AWS.

After your one-year trial period ends, you pay for all resources you use. But some resources are free forever. For example, the first 25 GB of the NoSQL database are free forever.

You get additional benefits, as detailed at http://aws.amazon.com/free. This book will use the Free Tier as much as possible and will clearly state when additional resources are required that aren't covered by the Free Tier.

1.4.2 Billing example

As mentioned earlier, you can be billed in the following ways:

- *Based on time of use*—A virtual machine is billed per second. A load balancer is billed per hour.
- *Based on traffic*—Traffic is measured in gigabytes or in number of requests, for example.
- *Based on storage usage*—Usage can be measured by capacity (e.g., 50 GB volume no matter how much you use) or real usage (such as 2.3 GB used).

Remember the web shop example from section 1.2? Figure 1.8 shows the web shop and adds information about how each part is billed.

Let's assume your web shop started successfully in January, and you ran a marketing campaign to increase sales for the next month. Lucky you: you were able to increase the number of visitors to your web shop fivefold in February. As you already know, you have to pay for AWS based on usage. Table 1.1 shows your bill for February. The number of visitors increased from 100,000 to 500,000, and your monthly bill increased from $112 to $473, which is a 4.2-fold increase. Because your web shop had to handle more traffic, you had to pay more for services, such as the CDN, the web servers, and the database. Other services, like the amount of storage needed for static files, didn't change, so the price stayed the same.

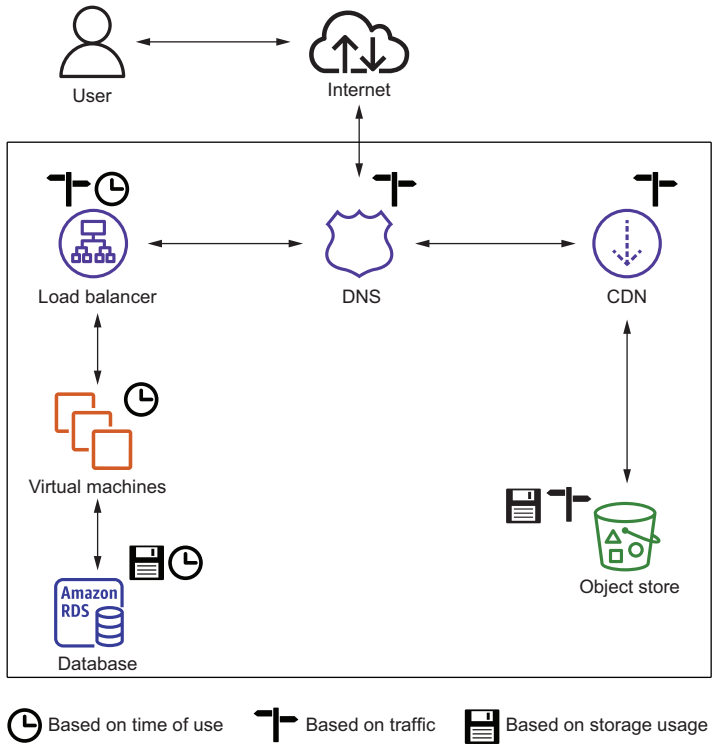


Figure 1.8 Some services are billed based on time of use, others by throughput or consumed storage.

Table 1.1 How an AWS bill changes if the number of web shop visitors increases

Service	January usage	February usage	February charge	Increase
Visits to website	100,000	500,000		
CDN	25 M requests + 25 GB traffic	125 M requests + 125 GB traffic	$115.00	$100.00
Static files	50 GB used storage	50 GB used storage	$1.15	$0.00
Load balancer	748 hours + 50 GB traffic	748 hours + 250 GB traffic	$19.07	$1.83
Web servers	1 virtual machine = 748 hours	4 virtual machines = 2,992 hours	$200.46	$150.35
Database (748 hours)	Small virtual machine + 20 GB storage	Large virtual machine + 20 GB storage	$133.20	$105.47
DNS	2 M requests	10 M requests	$4.00	$3.20
Total cost			***$472.88***	***$360.85***

With AWS, you can achieve a linear relationship between traffic and costs. And other opportunities await you with this pricing model.

1.4.3 Pay-per-use opportunities

The AWS pay-per-use pricing model creates new opportunities. For example, the barrier for starting a new project is lowered, because you no longer need to invest in infrastructure up front. You can start virtual machines on demand and pay only per second of usage. You can stop using those virtual machines whenever you like, and you no longer have to pay for them. You don't need to make an upfront commitment regarding how much storage you'll use.

Another example: a big virtual machine costs exactly as much as two smaller ones with the same capacity. Thus, you can divide your systems into smaller parts, because the cost is the same. This makes fault tolerance affordable not only for big companies but also for smaller budgets.

1.5 Comparing alternatives

AWS isn't the only cloud computing provider. Microsoft Azure and Google Cloud Platform (GCP) are major players as well. The three major cloud providers share a lot in common. They all have the following:

- A worldwide infrastructure that provides computing, networking, and storage capabilities
- An IaaS offering that provides virtual machines on-demand: Amazon EC2, Azure Virtual Machines, Google Compute Engine
- Highly distributed storage systems able to scale storage and I/O capacity without limits: Amazon S3, Azure Blob Storage, Google Cloud Storage
- A pay-as-you-go pricing model

But what are the differences between the cloud providers?

AWS is the market leader in cloud computing, offering an extensive product portfolio. Although AWS has expanded into the enterprise sector during recent years, it is still obvious that AWS started with services to solve internet-scale problems. Overall, AWS is building great services based on innovative, mostly open source, technologies. AWS offers complicated but rock-solid ways to restrict access to your cloud infrastructure.

Microsoft Azure provides Microsoft's technology stack in the cloud, recently expanding into web-centric and open source technologies as well. It seems like Microsoft is putting a lot of effort into catching up with Amazon's market share in cloud computing.

The Google Cloud Platform (GCP) is focused on developers looking to build sophisticated distributed systems. Google combines their worldwide infrastructure to offer scalable and fault-tolerant services (such as Google Cloud Load Balancing). The GCP seems more focused on cloud-native applications than on migrating your locally hosted applications to the cloud, in our opinion.

There are no shortcuts to making an informed decision about which cloud provider to choose. Each use case and project is different—the devil is in the details. Also don't forget where you are coming from. (Are you using Microsoft technology heavily? Do you have a big team consisting of system administrators or are you a developer-centric company?) Overall, in our opinion, AWS is the most mature and powerful cloud platform available at the moment.

1.6 Exploring AWS services

In this section, you will get an idea of the range of services that AWS offers. We'll also construct a mental model, with the help of some diagrams, to give you a high-level look at where those services sit in relation to the AWS setup as a whole.

Let's start with the mental model overview. Hardware for computing, storing, and networking is the foundation of the AWS cloud. AWS runs services on this hardware, as shown in figure 1.9.

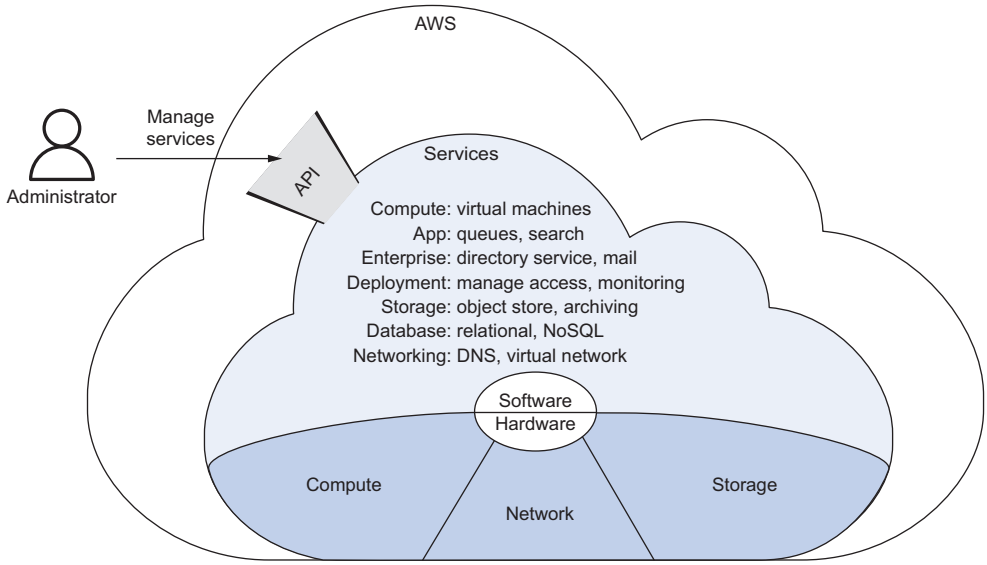


Figure 1.9 The AWS cloud is composed of hardware and software services accessible via an API.

You can manage services by sending requests to the API manually via a web-based GUI like the Management Console, a command-line interface (CLI), or programmatically via an SDK. Virtual machines have a special feature: you can connect to virtual machines through SSH, for example, and gain administrator access. This means you can install any software you like on a virtual machine. Other services, like the NoSQL database service, offer their features through an API and hide everything that's going

on behind the scenes. Figure 1.10 shows an administrator installing a custom PHP web application on a virtual machine and managing dependent services such as a NoSQL database used by the application.

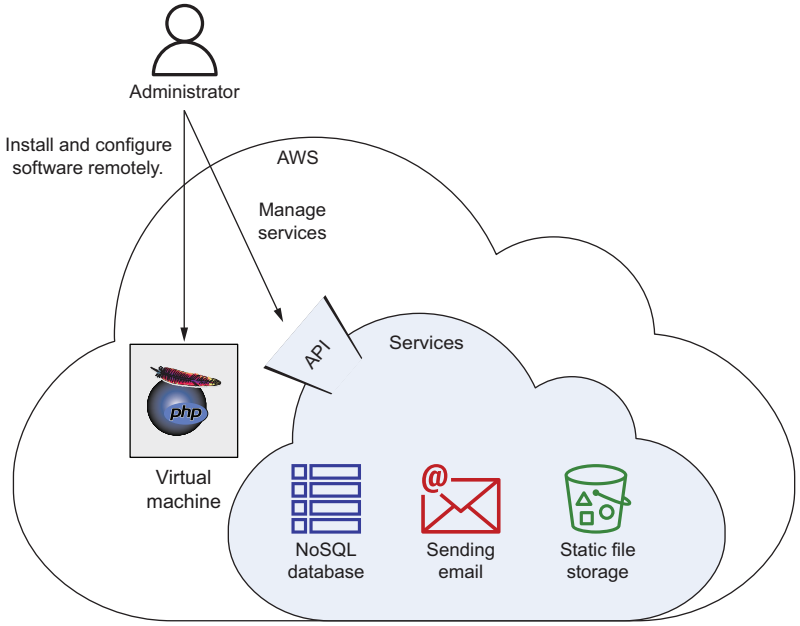


Figure 1.10 Managing a custom application running on a virtual machine and cloud-native services

Users send HTTP requests to a virtual machine. This virtual machine is running a web server along with a custom PHP web application. The web application needs to talk to AWS services to answer HTTP requests from users. For example, the application might need to query data from a NoSQL database, store static files, and send email. Communication between the web application and AWS services is handled by the API, as figure 1.11 shows.

The number of services available can be scary at the outset. When logging into AWS's web interface, you are presented with an overview listing around 200 services in 25 categories. On top of that, new services are announced constantly during the year and at the big conference in Las Vegas, AWS re:Invent, which takes place annually in November.

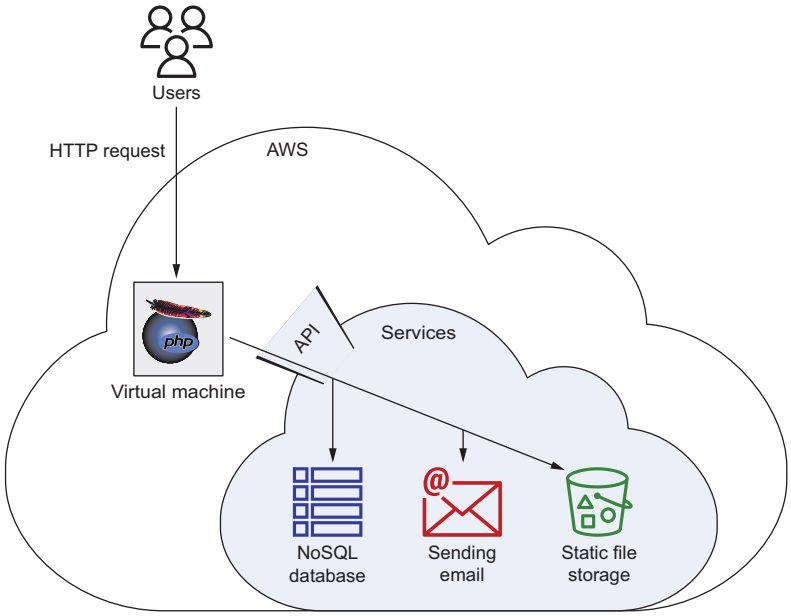


Figure 1.11 Handling an HTTP request with a custom web application using additional AWS services

AWS offers services in the following categories:

■ Analytics	■ Application integration	■ AR and VR
■ AWS cost management	■ Blockchain	■ Business applications
■ Compute	■ Containers	■ Customer enablement
■ Database	■ Developer tools	■ End-user computing
■ Frontend web and mobile	■ Game Development	■ Internet of Things
■ Machine learning	■ Management and governance	■ Media services
■ Migration and transfer	■ Networking and content delivery	■ Quantum technologies
■ Robotics	■ Satellite	■ Security, identity, and compliance
■ Storage		

Obviously, it is impossible for us to cover all the services offered by AWS in one book. Therefore, in this book, we have selected for you the services that will help you get started quickly to build a fully capable, responsive, and dependable system, and then

to grow and maintain that system. These are the most widely used services and will address most of your needs. After you've become more adept at working with AWS on these must-have services, feel free to investigate the nice-to-have services available to you.

The following services are covered in detail in our book:

- *EC2*—Virtual machines
- *ECS and Fargate*—Running and managing containers
- *Lambda*—Executing functions
- *S3*—Object store
- *Glacier*—Archiving data
- *EBS*—Block storage for virtual machines
- *EFS*—Network filesystem
- *RDS*—SQL databases
- *DynamoDB*—NoSQL database
- *ElastiCache*—In-memory key-value store
- *VPC*—Virtual network
- *ELB*—Load balancers
- *Simple Queue Service*—Distributed queues
- *CodeDeploy*—Automating code deployments
- *CloudWatch*—Monitoring and logging
- *CloudFormation*—Automating your infrastructure
- *IAM*—Restricting access to your cloud resources

We are missing at least three important topics that could fill their own books: continuous delivery, machine learning, and analytics. In fact, Manning has a whole book on doing machine learning and data analysis on AWS: *AI as a Service: Serverless Machine Learning with AWS* by Peter Elger and Eóin Shanaghy (https://www.manning.com/books/ai-as-a-service). You can read the first chapter free by following this link: http://mng.bz/BZvr. We suggest you do so after you've finished our book, because we provide the foundational knowledge you need to work with any of these more advanced services.

Let's return to a more immediate question: how exactly do you interact with an AWS service? The next section explains how to use the web interface, the CLI, and SDKs to manage and access AWS resources.

1.7 Interacting with AWS

When you interact with AWS to configure or use services, you make calls to the API. The API is the entry point to AWS, as figure 1.12 demonstrates.

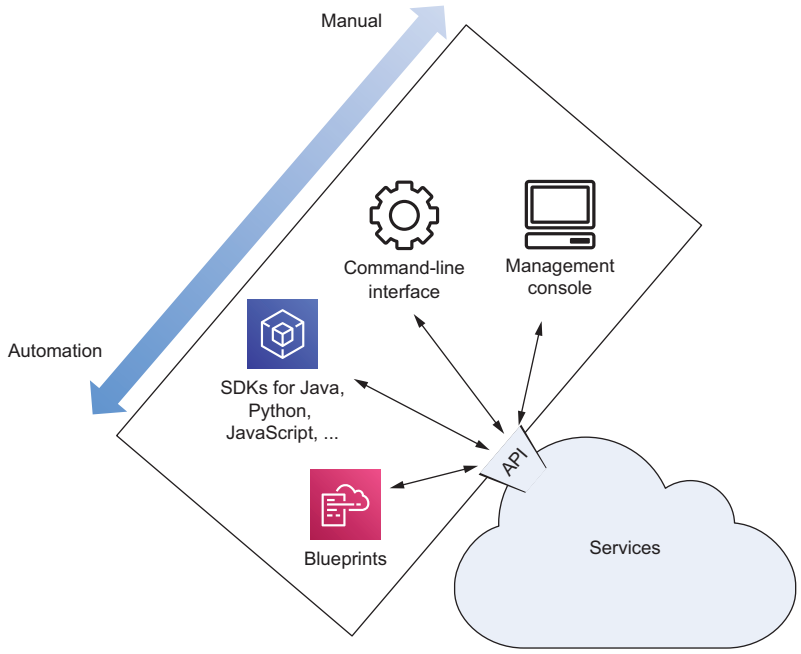


Figure 1.12 Different ways to access the AWS API, allowing you to manage and access AWS services

Next, we'll give you an overview of the tools available for communicating with AWS's APIs: the Management Console, the command-line interface, the SDKs, and infrastructure blueprints. We will compare the different tools, and you will learn how to use all of them while working your way through the book.

1.7.1 Management Console

The AWS Management Console allows you to manage and access AWS services through a graphical user interface (GUI), which works with modern browsers on desktop computers, laptops, and tablets. See figure 1.13.

When getting started or experimenting with AWS, the Management Console is the best place to start. It helps you to gain a quick overview of the different services. The Management Console is also a good way to set up a cloud infrastructure for development and testing.

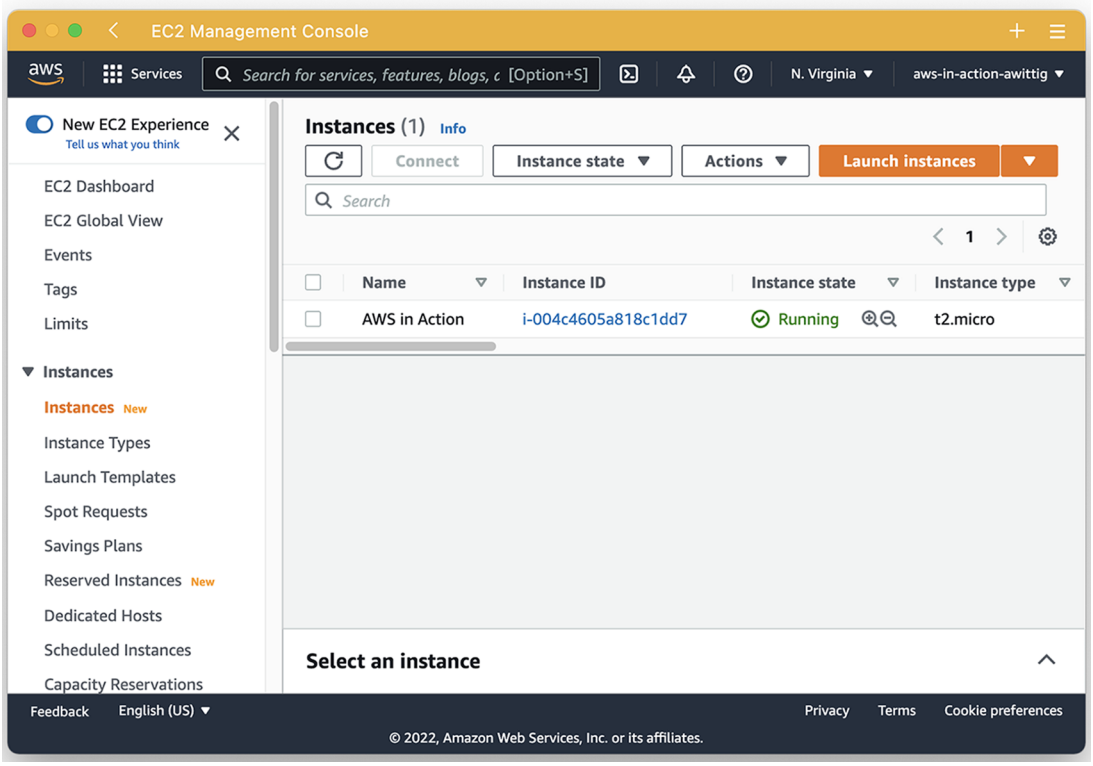


Figure 1.13 The AWS Management Console offers a GUI to manage and access AWS services.

1.7.2 Command-line interface

The command-line interface (CLI) allows you to manage and access AWS services within your terminal. Because you can use your terminal to automate or semi-automate recurring tasks, the CLI is a valuable tool. You can use the terminal to create new cloud infrastructures based on blueprints, upload files to the object store, or get the details of your infrastructure's networking configuration regularly. Figure 1.14 shows the CLI in action.

If you want to automate parts of your infrastructure with the help of a continuous integration server, like Jenkins, the CLI is the right tool for the job. The CLI offers a convenient way to access the API and combine multiple calls into a script.

You can even begin to automate your infrastructure with scripts by chaining multiple CLI calls together. The CLI is available for Windows, Mac, and Linux, as well as PowerShell.

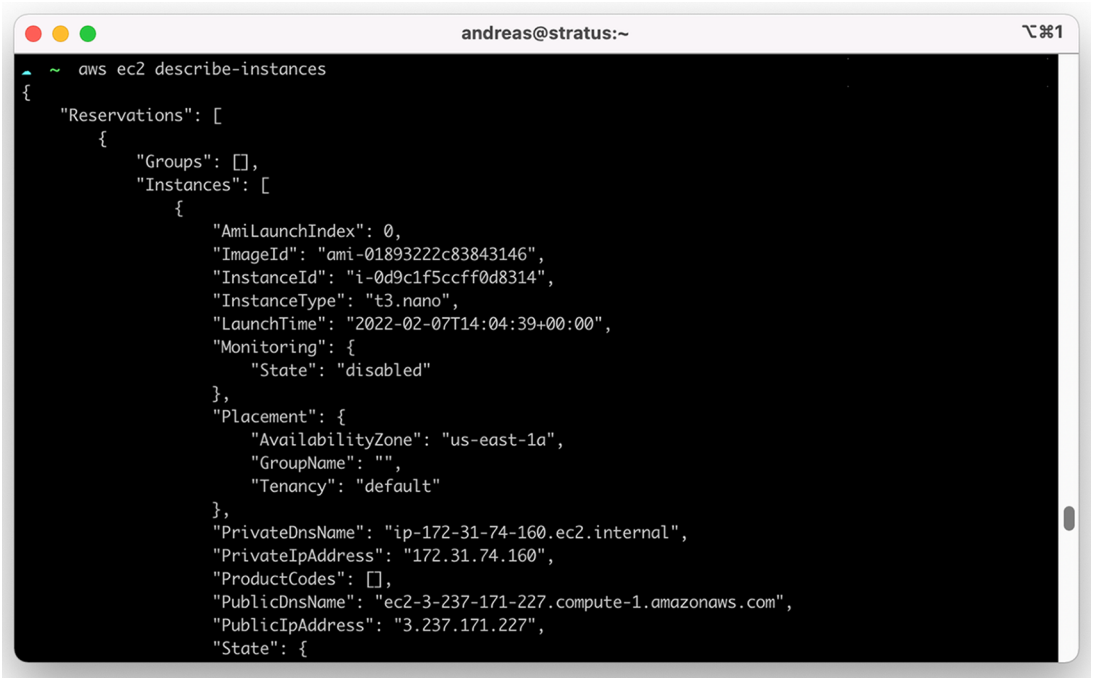


Figure 1.14 The CLI allows you to manage and access AWS services from your terminal.

1.7.3 SDKs

Use your favorite programming language to interact with the AWS API. AWS offers SDKs for the following platforms and languages:

■ JavaScript	■ Python	■ PHP
■ .NET	■ Ruby	■ Java
■ Go	■ Node.js	■ C++

SDKs are typically used to integrate AWS services into applications. If you're doing software development and want to integrate an AWS service like a NoSQL database or a push-notification service, an SDK is the right choice for the job. Some services, such as queues and topics, must be used with an SDK.

1.7.4 Blueprints

A *blueprint* is a description of your system containing all resources and their dependencies. An Infrastructure as Code tool compares your blueprint with the current

system and calculates the steps to create, update, or delete your cloud infrastructure. Figure 1.15 shows how a blueprint is transferred into a running system.

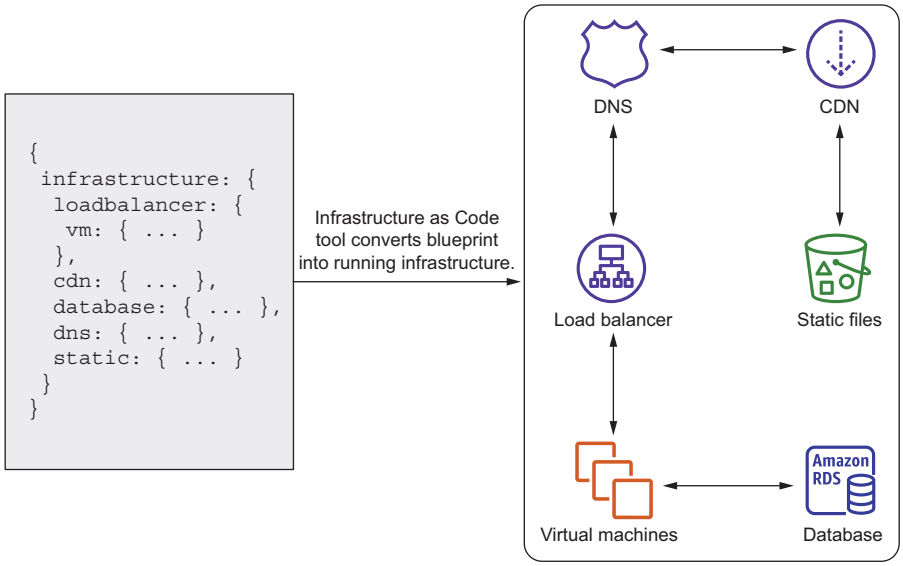


Figure 1.15 Infrastructure automation with blueprints

Consider using blueprints if you have to control many or complex environments. Blueprints will help you to automate the configuration of your infrastructure in the cloud. You can use them to set up a network and launch virtual machines, for example.

Automating your infrastructure is also possible by writing your own source code with the help of the CLI or the SDKs. Doing so, however, requires you to resolve dependencies, to make sure you are able to update different versions of your infrastructure, and to handle errors yourself. As you will see in chapter 4, using a blueprint and an Infrastructure-as-Code tool solves these challenges for you. It's time to get started creating your AWS account and exploring AWS practice after all that theory.

1.8 Creating an AWS account

Before you can start using AWS, you need to create an account, which is a basket for all your cloud resources. You can attach multiple users to an account if multiple people need access to it; by default, your account will have one AWS account root user. To create an account, you need the following:

- A telephone number to validate your identity
- A credit card to pay your bills

USING AN OLD ACCOUNT? It is possible to use your existing AWS account while working through this book. In this case, your usage might not be covered by the Free Tier, so you might have to pay for the use.

Also, if you created your existing AWS account before December 4, 2013, please create a new one, because some legacy problems might cause trouble when following our examples.

MULTIPLE AWS ACCOUNTS? It is fine to create more than one AWS account. AWS even encourages you to do so, to isolate different workloads.

1.8.1 Signing up

The sign-up process consists of five steps:

1. Providing login credentials
2. Providing contact information
3. Providing payment details
4. Verifying your identity
5. Choosing a support plan

Point your favorite web browser to https://aws.amazon.com, and click the Create an AWS Account button.

1. PROVIDING LOGIN CREDENTIALS

Creating an AWS account starts with defining a unique AWS account name, as shown in figure 1.16. The AWS account name has to be globally unique among all AWS customers. Try `aws-in-action-$yourname` and replace `$yourname` with your name. In addition

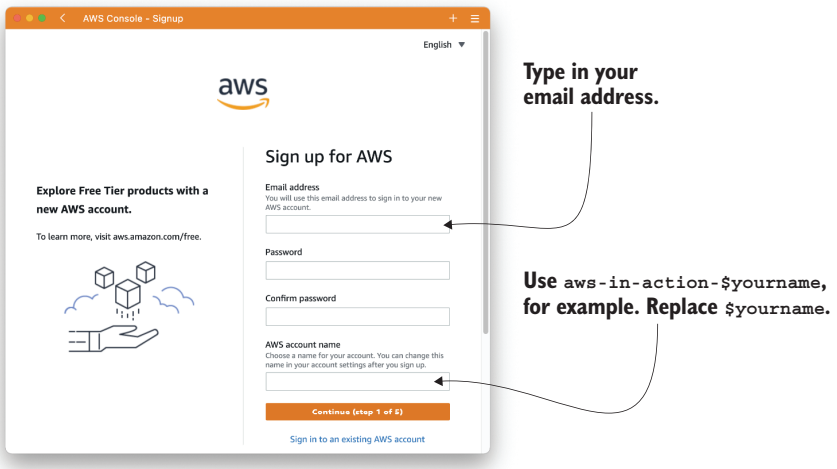


Figure 1.16 First step of creating an AWS account: account name and password

to the account name, you have to specify an email address and a password used to authenticate the root user of your AWS account.

We advise you to choose a strong password to prevent misuse of your account. *Use a password consisting of at least 20 characters.* Protecting your AWS account from unwanted access is crucial to avoid data breaches, data loss, or unwanted resource usage on your behalf.

2. PROVIDING CONTACT INFORMATION

The next step, as shown in figure 1.17, is adding your contact information. Fill in all the required fields, and continue.

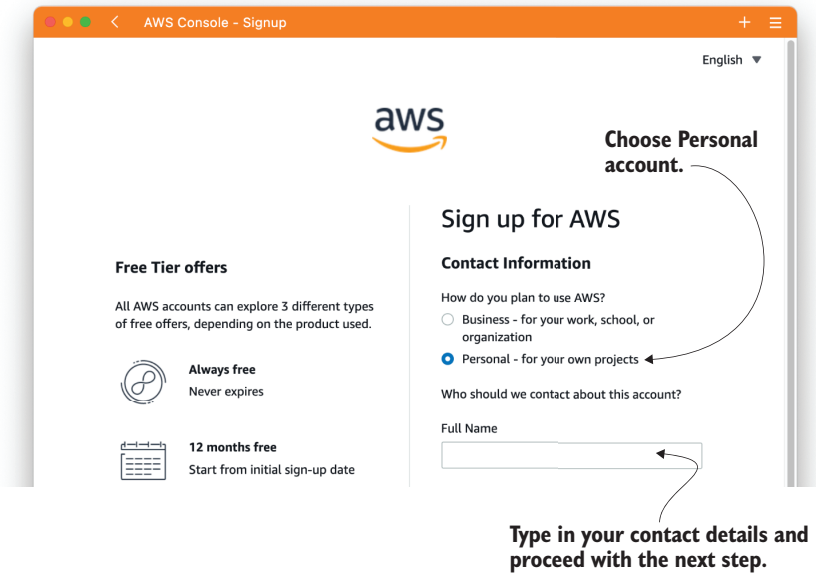


Figure 1.17 Second step of creating an AWS account: contact details

3. PROVIDING PAYMENT DETAILS

Next, the screen shown in figure 1.18 asks for your payment information. Provide your credit card information. There's an option to change the currency setting from USD to AUD, BRL, CAD, CHF, CNY, DKK, EUR, GBP, HKD, JPY, KRW, NOK, NZD, SEK, SGD, or ZAR later if that's more convenient for you. If you choose this option, the amount in USD is converted into your local currency at the end of the month.

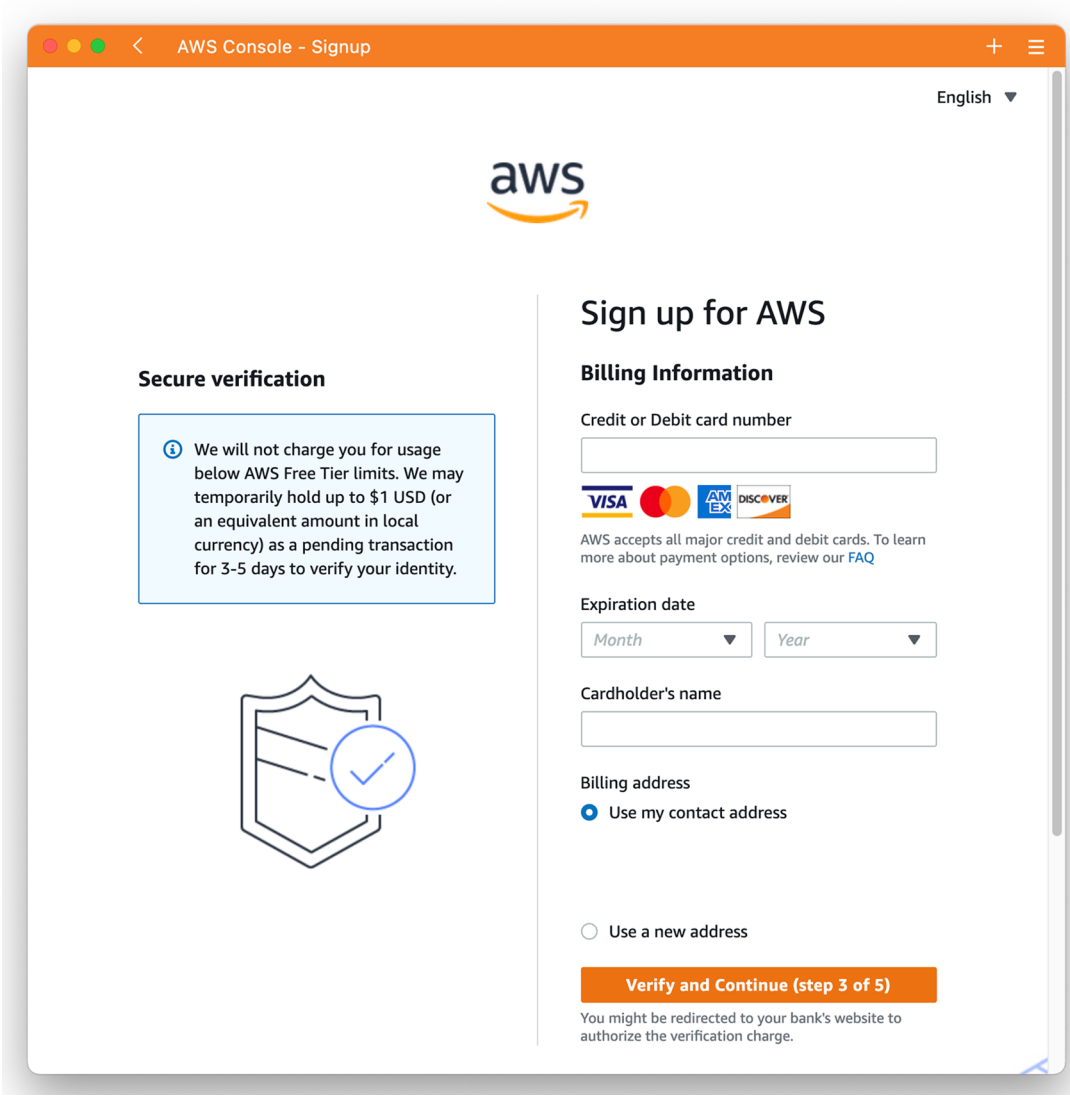


Figure 1.18 Third step of creating an AWS account: payment details

4. VERIFYING YOUR IDENTITY

The next step is to verify your identity. Figure 1.19 shows the first step of the process.

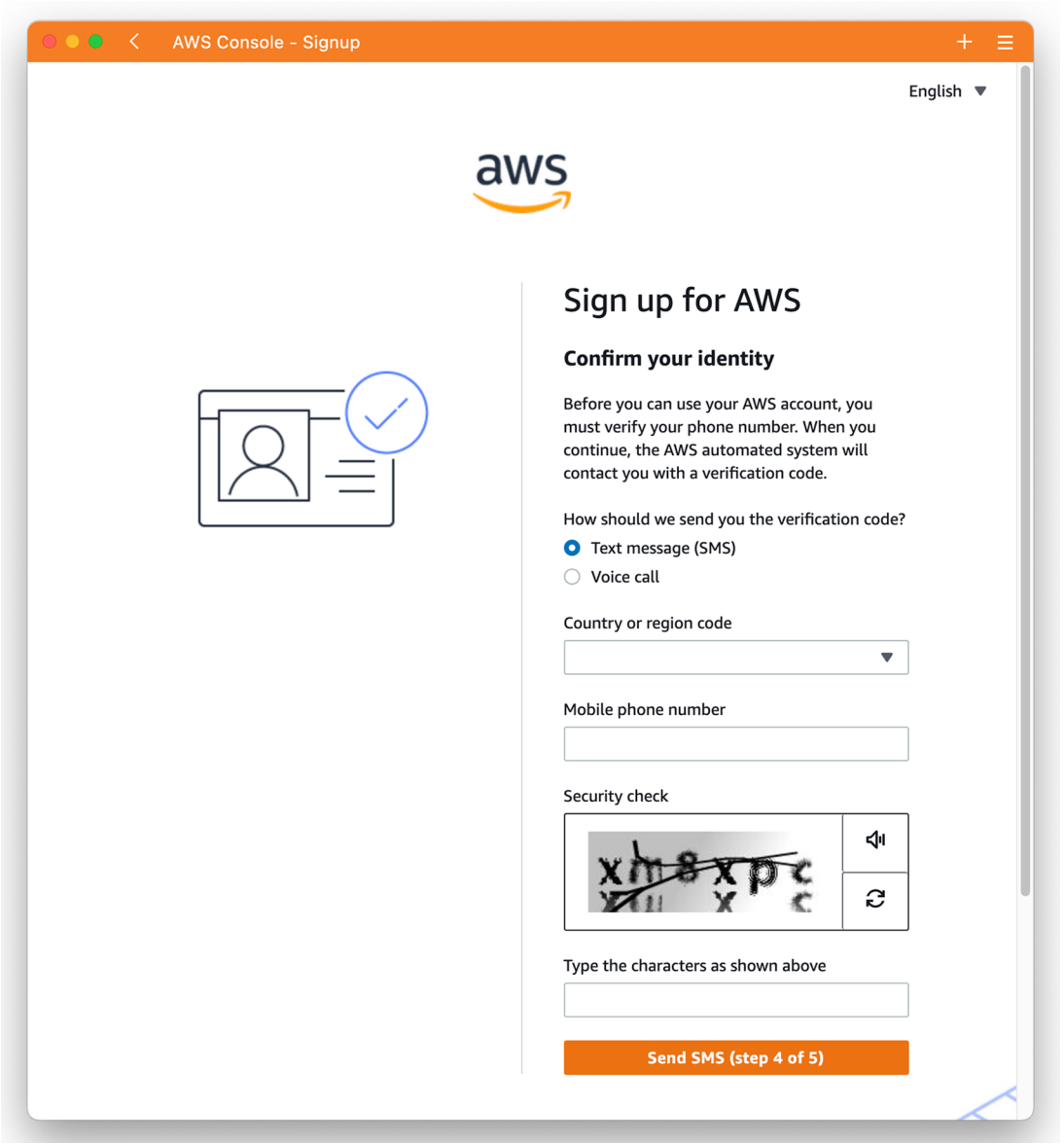


Figure 1.19 Fourth step of creating an AWS account: verify identity

After you complete the first part of the form, you'll receive a text message or call from AWS. All you need to do is to type in the verification code.

5. Choosing a support plan

The last step is to choose a support plan; see figure 1.20. For now, select the Basic plan, which is free. When running a production workload an AWS, we recommend at least a Developer plan to be able to ask questions about upcoming issues.

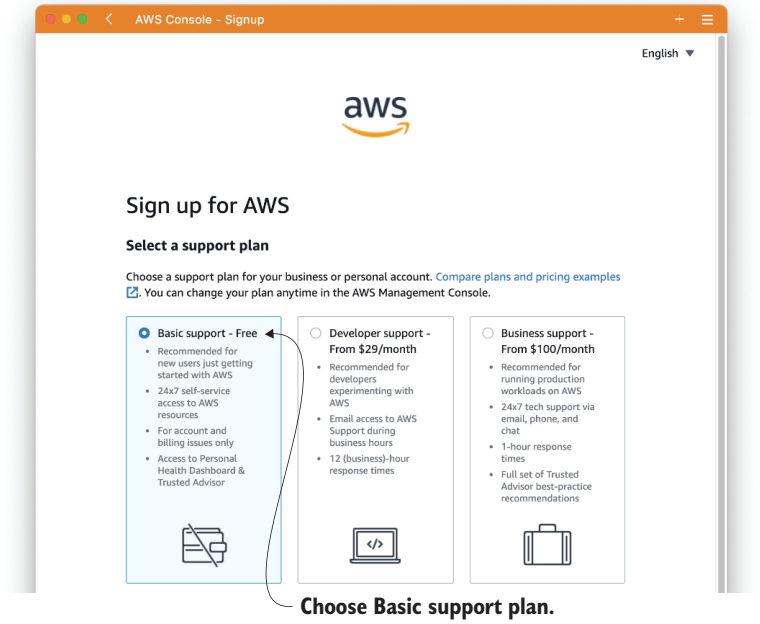


Figure 1.20 Fifth step of creating an AWS account: choose a support plan

High five! You're done. Click Go to the AWS Management Console, as shown in figure 1.21, to sign in to your AWS account for the first time.

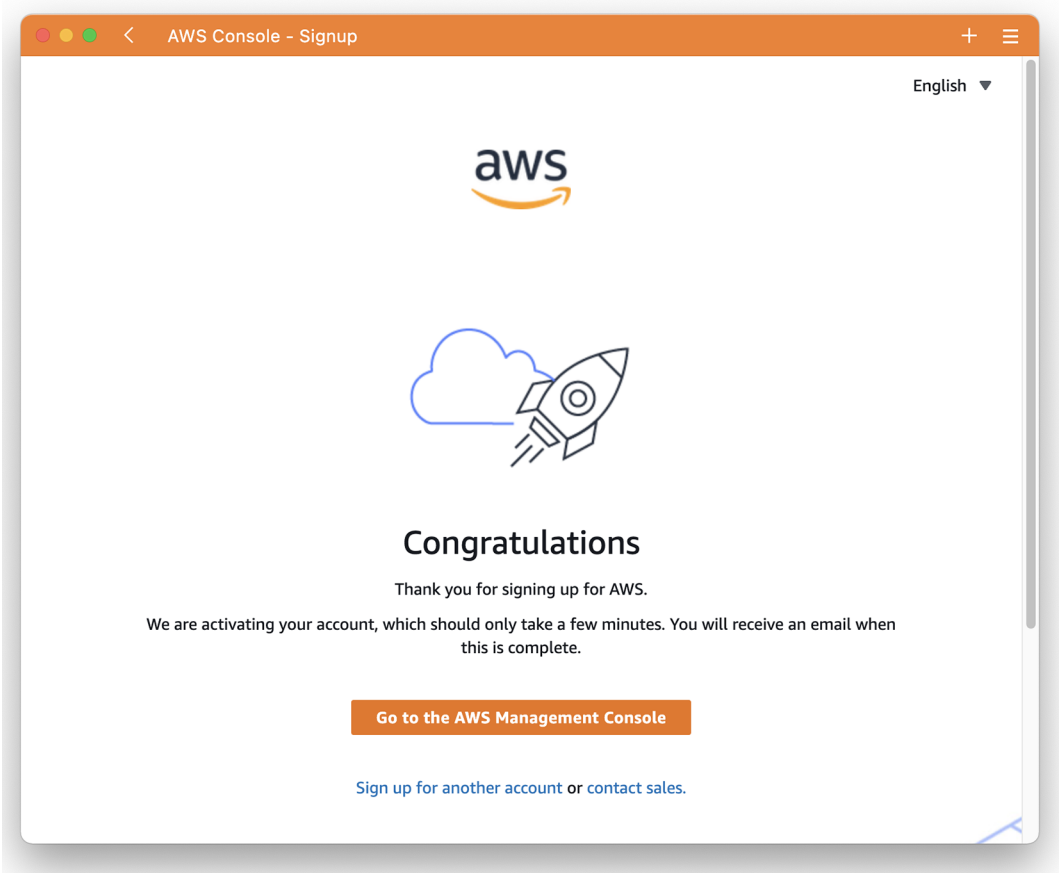


Figure 1.21 Fifth step of creating an AWS account: success!

1.8.2 Signing in

You now have an AWS account and are ready to sign in to the AWS Management Console. As mentioned earlier, the Management Console is a web-based tool you can use to control AWS resources; it makes most of the functionality of the AWS API available to you. Figure 1.22 shows the sign-in form at https://console.aws.amazon.com. Choose Root User and enter your email address, click Next, and then enter your password to sign in.

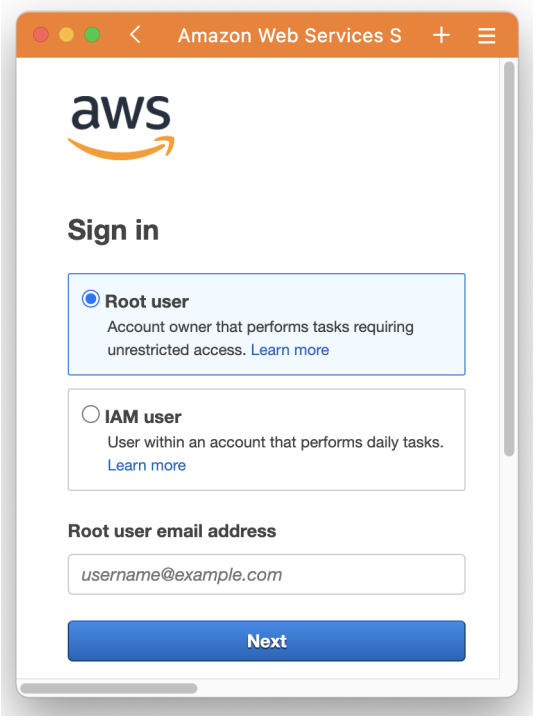


Figure 1.22 Sign in with the AWS account root user.

After you have signed in successfully, you are forwarded to the start page of the Management Console, as shown in figure 1.23.

The most important part is the navigation bar at the top, shown in figure 1.24. It consists of the following eight sections:

- *AWS*—The dashboard of the Management Console shows an overview of your AWS account.
- *Services*—Provides access to all AWS services.
- *Search*—Allows you to search for services, features, and more.
- *Terminal*—Spin up a terminal with access to your cloud resources in the browser.
- *Notifications*—View alerts and notifications by AWS, for example, planned downtimes or outages.
- *Help*—Get support by the community, experts, or AWS support.
- *Region*—Select the region you want to manage.
- *Account*—Manage your AWS account.

Next, you'll make sure to avoid unwanted AWS costs.

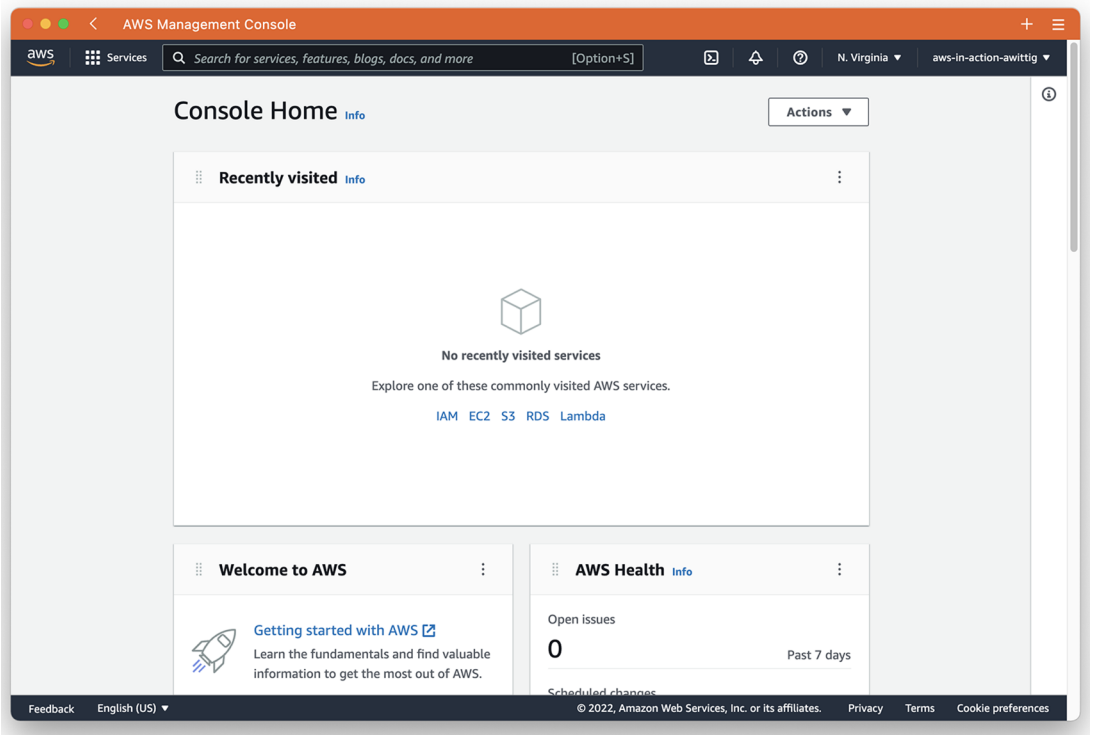


Figure 1.23 The dashboard after logging in for the first time

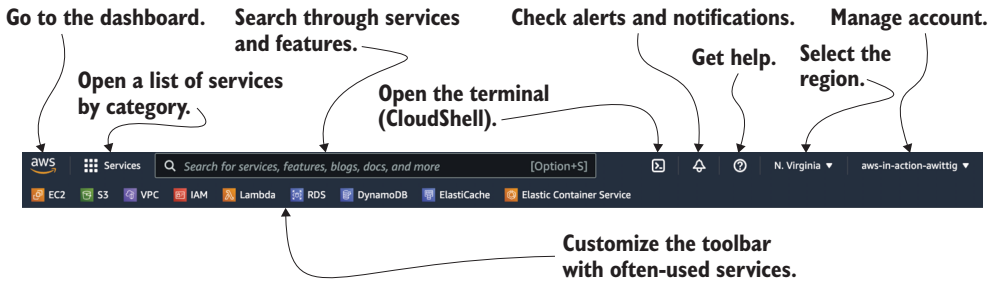


Figure 1.24 AWS Management Console navigation bar

1.9 Creating a budget alert to keep track of your AWS bill

At first, the pay-per-use pricing model of AWS might feel unfamiliar to you, because it is not 100% foreseeable what your bill will look like at the end of the month. Most of the examples in this book are covered by the Free Tier, so AWS won't charge you

anything. Exceptions are clearly marked. To provide you with the peace of mind needed to learn about AWS in a comfortable environment, you will create a budget next. The budget will notify you via email in case your AWS bill will exceed $5 so you can react quickly.

Before creating a budget, configure a Free Tier usage alert as follows:

1. Open the billing preferences of your AWS account at http://mng.bz/5m8D.
2. Enable Receive Free Tier Usage Alerts, and type in your email address as shown in figure 1.25.
3. Press the Save Preferences button.

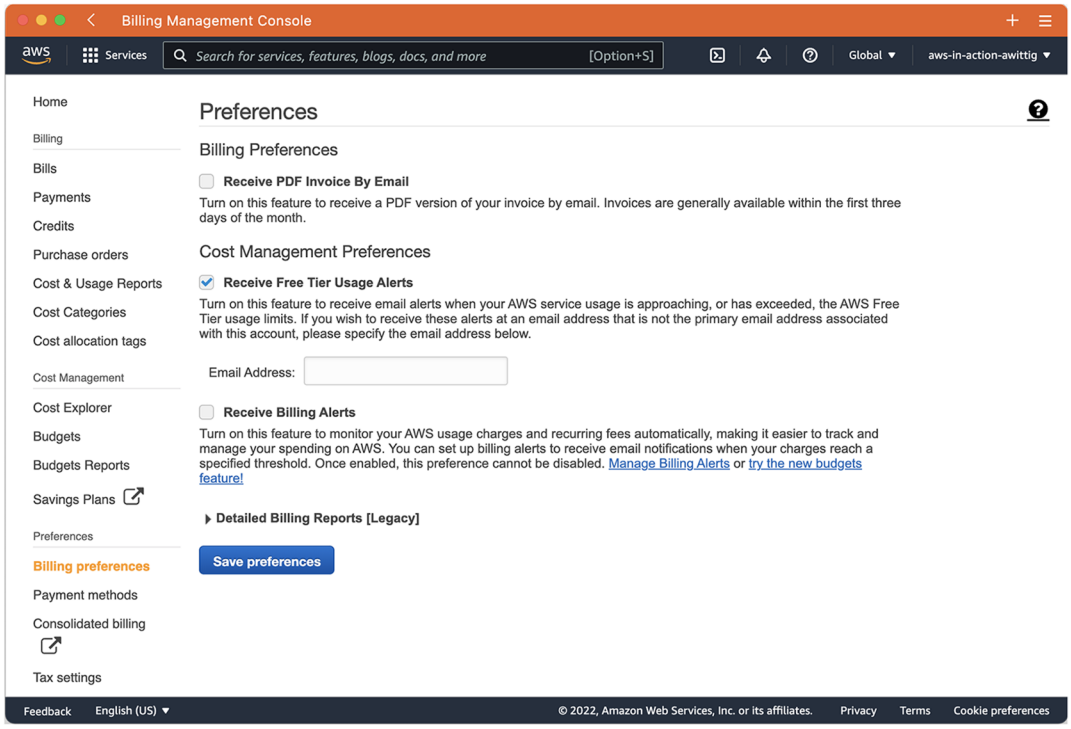


Figure 1.25 Creating a Free Tier usage alert to avoid unwanted costs

Next, you will create a budget that monitors costs incurred and forecasts costs to the end of the month, as described here:

1. Search and open Budgets in the Management Console's navigation bar.
2. Click Create Budget.
3. Select Cost Budget, as shown in figure 1.26.
4. Click Next.

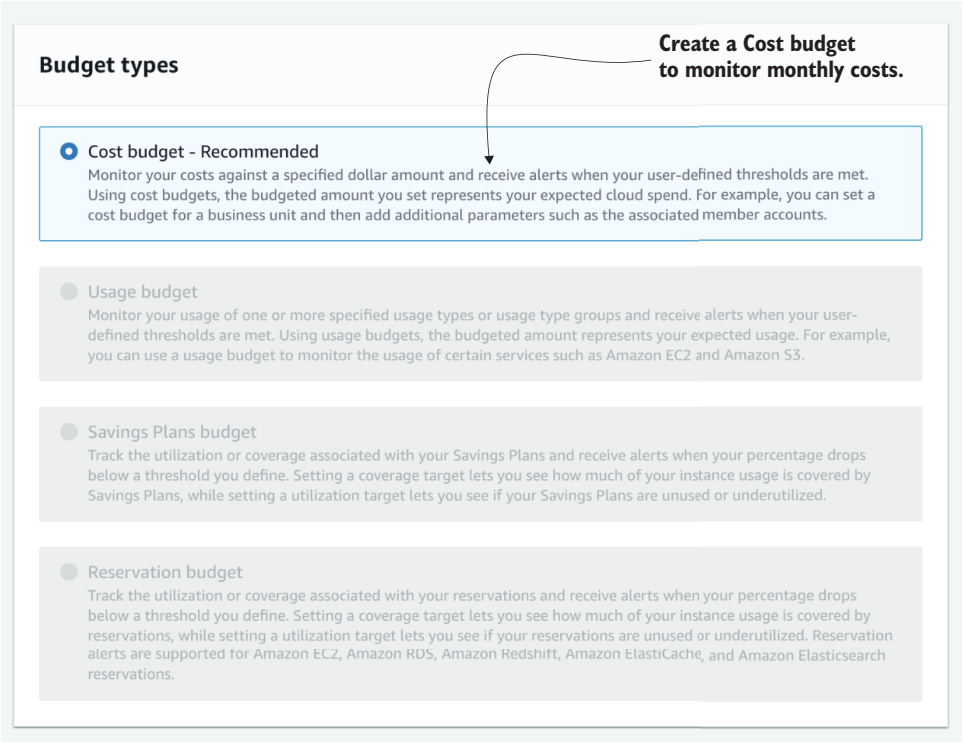


Figure 1.26 Creating a cost budget to monitor incurred and forecasted costs

Next, configure the cost budget as illustrated in figure 1.27 and described next:

1. Choose period Monthly.
2. Select Recurring Budget.
3. Choose the current month and year as start month.
4. Select Fixed to use the same budget for every month of the year.
5. Type in 5 to set the budget to $5 per month.
6. Click Next.

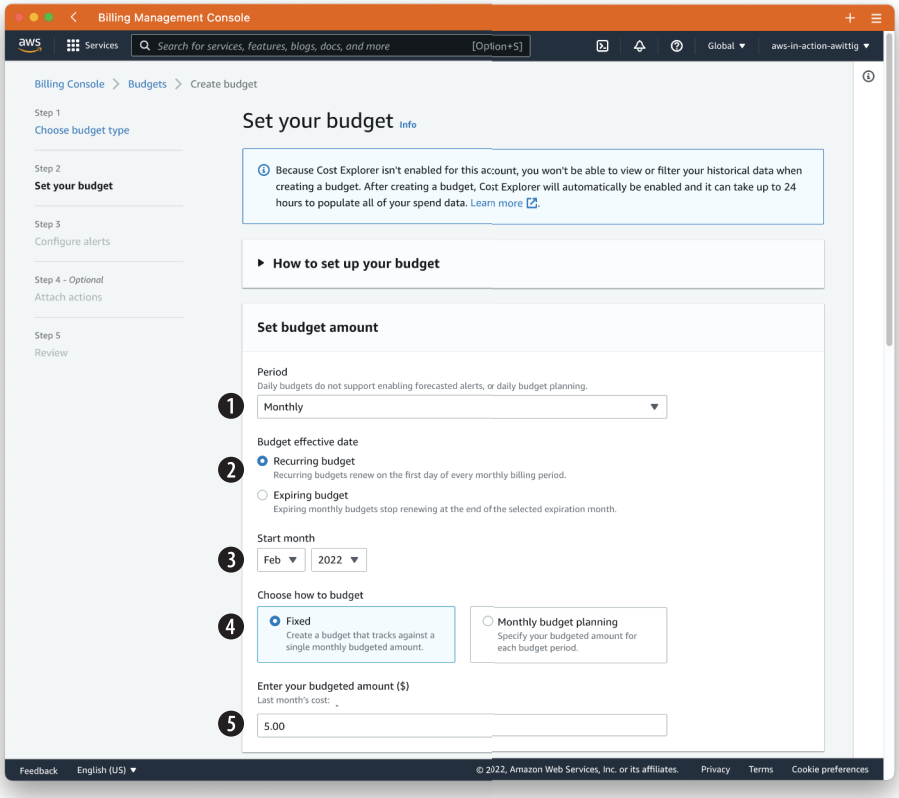


Figure 1.27 Creating a cost budget to monitor incurred and forecasted costs

After defining the budget, it is time to create alerts that will notify you via email as follows (figure 1.28):

1. Click Add Alert Threshold.
2. Type in 100% of budgeted amount.
3. Select trigger Actual to get notified when the incurred costs increase the budget.
4. Type in your email address.
5. Click Add Alert Threshold.
6. Type in 100% of budgeted amount.
7. Select trigger Forecasted to get notified when the forecasted costs increase the budget.
8. Type in your email address.
9. Click Next to proceed.
10. Review the budget, and click Create Budget.

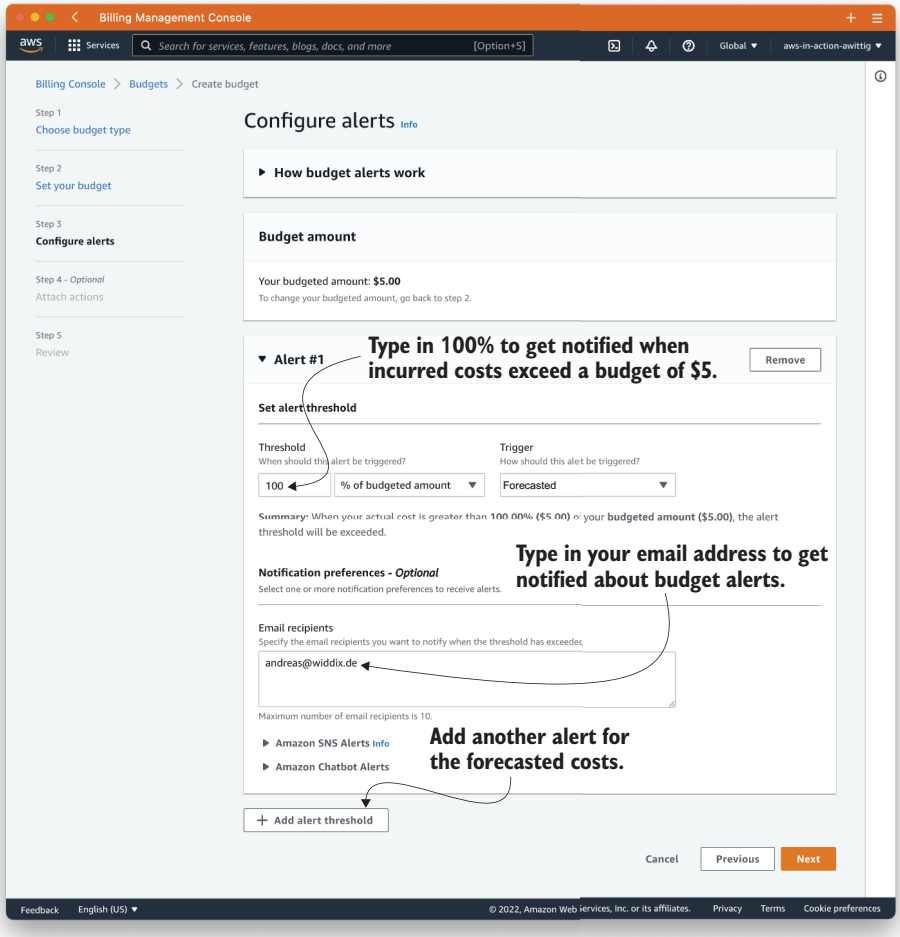


Figure 1.28 Define alerts to get notified when incurred or forecasted costs exceed your monthly budget.

That's it—you are ready to get started learning all the services and approaches we cover in the rest of this book. When you forget to shut down or delete resources when following our examples, AWS will notify you in case your monthly bill will exceed $5.

Summary

- Cloud computing, or the cloud, is a metaphor for supply and consumption of IT resources.
- Amazon Web Services (AWS) offers Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS).

- AWS is a platform of web services for computing, storing, and networking that work well together.
- Cost savings aren't the only benefit of using AWS. You'll also profit from an innovative and fast-growing platform with flexible capacity, fault-tolerant services, and a worldwide infrastructure.
- Almost any use case can be implemented on AWS, whether it's a widely used web application or a specialized enterprise application with an advanced networking setup.
- You can interact with AWS in many different ways. Control the different services by using the web-based user interface, use code to manage AWS programmatically from the command line or SDKs, or use blueprints to set up, modify, or delete your infrastructure on AWS.
- Pay-per-use is the pricing model for AWS services. Computing power, storage, and networking services are billed similarly to electricity.
- To create an AWS account, all you need is a telephone number and a credit card.
- Creating budget alerts allows you to keep track of your AWS bill and get notified whenever you exceed the Free Tier.

2 A simple example: WordPress in 15 minutes

This chapter covers

- Creating a cloud infrastructure for WordPress
- Exploring a cloud infrastructure consisting of a load balancer, virtual machines, a database, and a network filesystem
- Estimating costs of a cloud infrastructure
- Shutting down a cloud infrastructure

Having looked at why AWS is such a great choice to run web applications in the cloud, in this chapter, you'll explore migrating a simple web application to AWS by setting up a sample cloud infrastructure within 15 minutes. Over the course of the book, we will revisit the WordPress example to understand the concepts in more detail. For example, we will take a look at the relational database in chapter 10 and learn how to add and remove virtual machines based on the current load in chapter 17.

NOTE The example in this chapter is totally covered by the Free Tier (see section 1.4.1 for details). As long as you don't run this example longer than a few days, you won't pay anything for it. Keep in mind that this applies only if you created a fresh AWS account for this book and there is

nothing else going on in your AWS account. Try to complete the chapter within a few days, because you'll clean up your account at the end of the chapter.

Imagine you work for a mid-sized company that runs a blog to attract new software and operations engineers and uses WordPress as the content management system. Around 1,000 people visit the blog daily. You are paying $250 per month for the on-premises infrastructure. This seems expensive to you, particularly because at the moment, the blog suffers from several outages per month.

To leave a good impression on potential candidates, the infrastructure should be highly available, which is defined as an uptime of 99.99%. Therefore, you are evaluating new options to operate WordPress reliably. AWS seems to be a good fit. As a proof of concept, you want to evaluate whether a migration is possible, so you need to do the following:

- Set up a highly available infrastructure for WordPress.
- Estimate the monthly costs of the infrastructure.
- Come to a decision and delete the infrastructure afterward.

WordPress is written in PHP and uses a MySQL database to store data. Besides that, data like user uploads is stored on disk. Apache is used as the web server to serve the pages. With this information in mind, it's time to map your requirements to AWS services.

2.1 Creating your infrastructure

You'll use the following five AWS services to copy the old infrastructure to AWS:

- *Elastic Load Balancing (ELB)*—AWS offers a load balancer as a service. The load balancer distributes traffic to a bunch of virtual machines and is highly available by default. Requests are routed to virtual machines as long as their health check succeeds. You'll use the Application Load Balancer (ALB), which operates on layer 7 (HTTP and HTTPS).
- *Elastic Compute Cloud (EC2)*—The EC2 service provides virtual machines. You'll use a Linux machine with an optimized distribution called Amazon Linux to install Apache, PHP, and WordPress. You aren't limited to Amazon Linux; you could also choose Ubuntu, Debian, Red Hat, or Windows. Virtual machines can fail, so you need at least two of them. The load balancer will distribute the traffic between them. In case a virtual machine fails, the load balancer will stop sending traffic to the failed VM, and the remaining VM will need to handle all requests until the failed VM is replaced.
- *Relational Database Service (RDS) for MySQL*—WordPress relies on the popular MySQL database. AWS provides MySQL with its RDS. You choose the database size (storage, CPU, RAM), and RDS takes over operating tasks like creating backups and installing patches and updates. RDS can also provide a highly available MySQL database using replication.

- *Elastic File System (EFS)*—WordPress itself consists of PHP and other application files. User uploads—for example, images added to an article—are stored as files as well. By using a network filesystem, your virtual machines can access these files. EFS provides a scalable, highly available, and durable network filesystem using the NFSv4.1 protocol.
- *Security groups*—Control incoming and outgoing traffic to your virtual machine, your database, or your load balancer with a firewall. For example, use a security group allowing incoming HTTP traffic from the internet to port 80 of the load balancer. Or restrict network access to your database on port 3306 to the virtual machines running your web servers.

Figure 2.1 shows all the parts of the infrastructure in action. Sounds like a lot of stuff to set up, so let's get started!

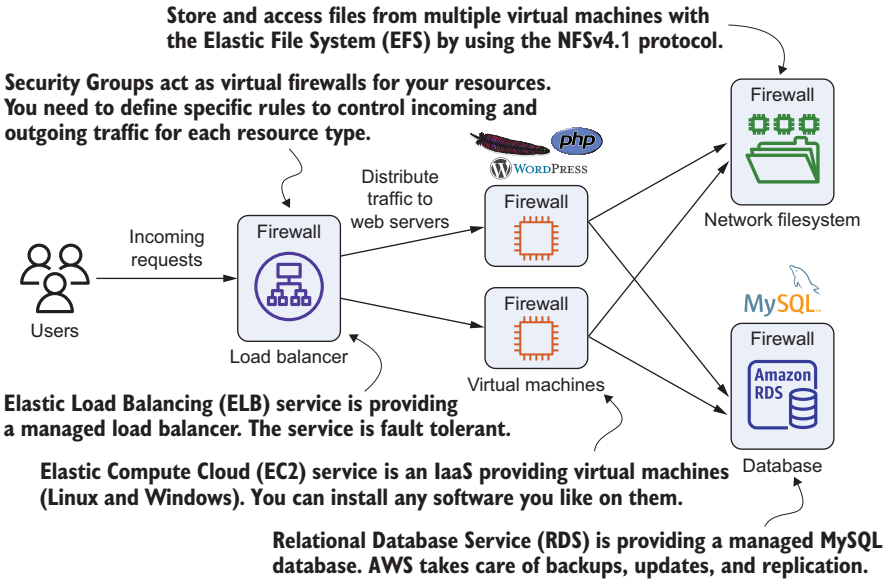


Figure 2.1 The company's blogging infrastructure consists of two load-balanced web servers running WordPress, a network filesystem, and a MySQL database server.

If you expect pages of instructions, you'll be happy to know that you can create all of this with just a few clicks using a service called AWS CloudFormation, which you will learn about in detail in chapter 4. AWS CloudFormation does all of the following automatically in the background:

1. Creates a load balancer (ELB)
2. Creates a MySQL database (RDS)
3. Creates a network filesystem (EFS)
4. Creates and attaches firewall rules (security groups)

5 Creates two virtual machines running web servers:
   a Creates two virtual machines (EC2)
   b Mounts the network filesystem (EFS)
   c Installs Apache and PHP
   d Downloads and extracts the 4.8 release of WordPress
   e Configures WordPress to use the created MySQL database (RDS)
   f Starts the Apache web server

To create the infrastructure for your proof of concept, open the AWS Management Console at https://console.aws.amazon.com. Click Services in the navigation bar, and select CloudFormation. You can use the search function to find CloudFormation more easily.

DEFAULT REGION FOR EXAMPLES All examples in this book use N. Virginia (also called us-east-1) as the default region. Exceptions are indicated. Please make sure you switch to the region N. Virginia before starting to work on an example. When using the AWS Management Console, you can check and switch the region on the right side of the main navigation bar at the top.

Click Create Stack, and choose With New Resources (Standard) to start the four-step wizard, as shown in figure 2.2. Choose Template Is Ready, and enter the Amazon S3

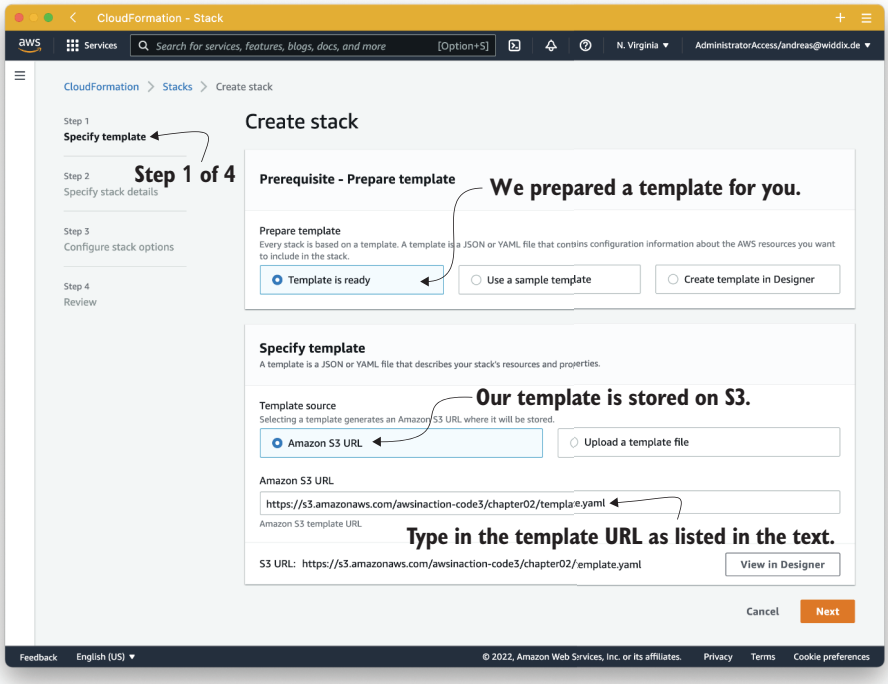


Figure 2.2 Creating the stack for the proof of concept: Step 1 of 4

URL https://s3.amazonaws.com/awsinaction-code3/chapter02/template.yaml to use the template prepared for this chapter. Proceed with the next step of the wizard.

Specify wordpress as the Stack name and, in the Parameters section, set the WordpressAdminPassword to a password of your choice that you are not using somewhere else, as shown in figure 2.3.

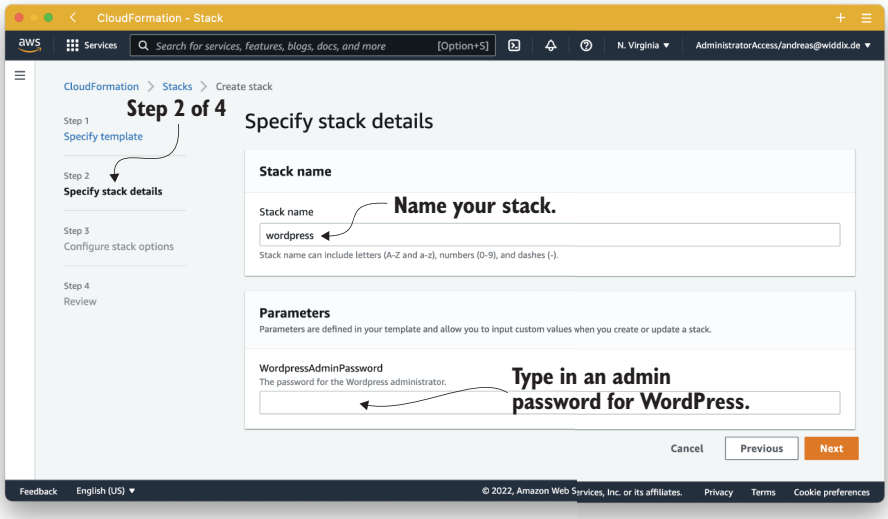


Figure 2.3 Creating the stack for the proof of concept: Step 2 of 4

The next step is to specify tags for your infrastructure, as illustrated in figure 2.4. A *tag* consists of a key and a value and can be used to add metadata to all parts of your infrastructure. You can use tags to differentiate between testing and production resources, add a cost center to easily track costs in your organization, or mark resources that belong to a certain application if you host multiple applications in the same AWS account.

Figure 2.4 shows how to configure the tag. In this example, you'll use a tag to mark all resources that belong to the wordpress system. This will help you to easily find all the parts of your infrastructure later. Use a custom tag consisting of the key—system—and the value—wordpress. Afterward, press the Next button to proceed to the next step.

You can define your own tags as long as the key name is less than 128 characters and the value is less than 256 characters.

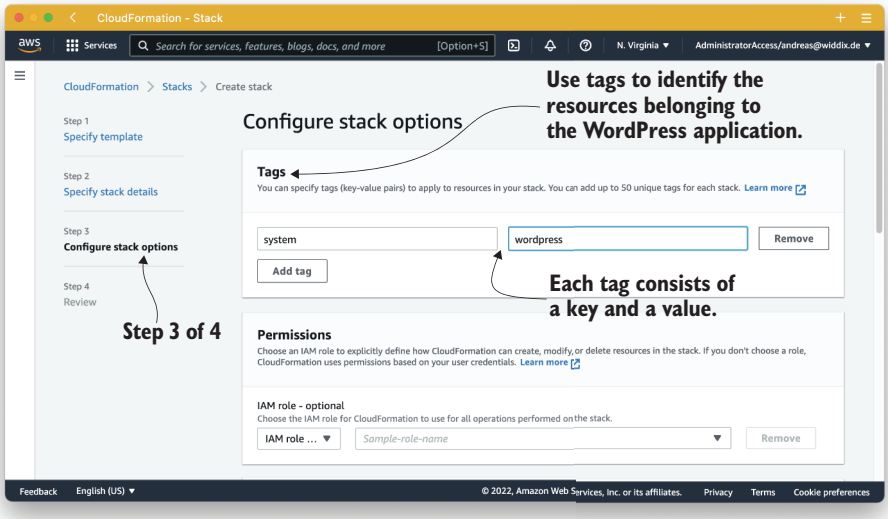


Figure 2.4 Creating the stack for the proof of concept: Step 3 of 4

Additional CloudFormation stack options

It is possible to define specific permissions used to manage resources, as well as to set up notifications and other advanced options. You won't need these options for 99% of the use cases, so we don't cover them in our book. Have a look at the CloudFormation User Guide (http://mng.bz/deqv) if you're interested in the details.

Figure 2.5 illustrates that all you need to do is to acknowledge that CloudFormation will create IAM resources and click Create Stack.

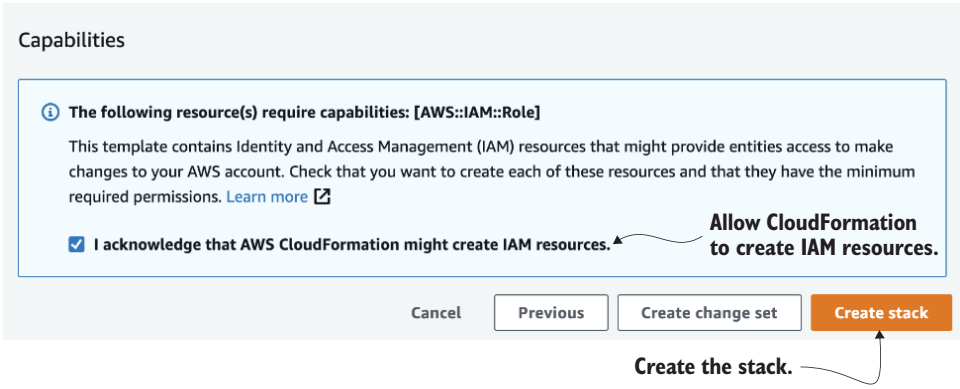


Figure 2.5 Creating the stack for the proof of concept: Step 4 of 4

Your infrastructure will now be created. Figure 2.6 shows that wordpress is in the state of `CREATE_IN_PROGRESS`. It's a good time to take a break; come back in 15 minutes, and you'll be surprised.

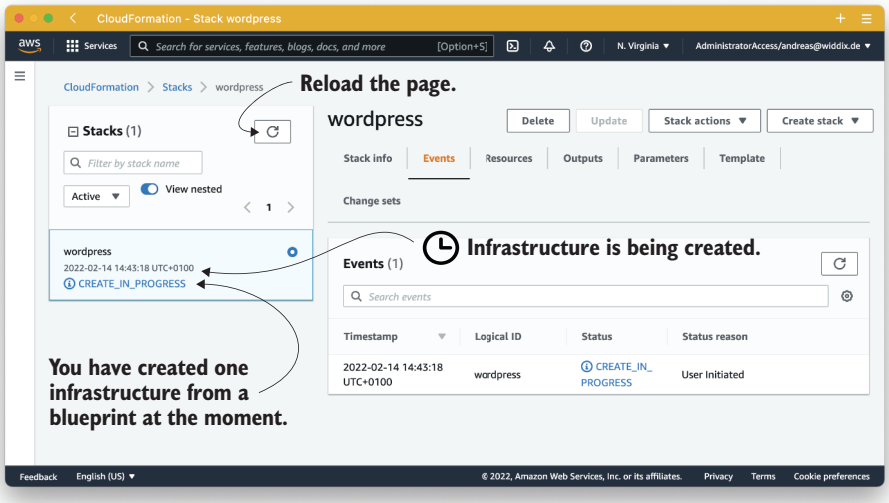


Figure 2.6 CloudFormation is creating the resources needed for WordPress.

After all the needed resources have been created, the status will change to `CREATE_COMPLETE`. Be patient and click the refresh button from time to time if your status continues to show `CREATE_IN_PROGRESS`.

Select the check box at the beginning of the row containing your wordpress stack. Switch to the Outputs tab, as shown in figure 2.7. There you'll find the URL to your WordPress installation; click the link to open it in your browser.

You may ask yourself, how does this work? The answer is *automation*.

Automation references

One of the key concepts of AWS is automation. You can automate everything. In the background, your cloud infrastructure was created based on a blueprint. You'll learn more about blueprints and the concept of programming your infrastructure in chapter 4. You'll learn to automate the deployment of software in chapter 15.

You'll explore the blogging infrastructure in the next section to get a better understanding of the services you're using.

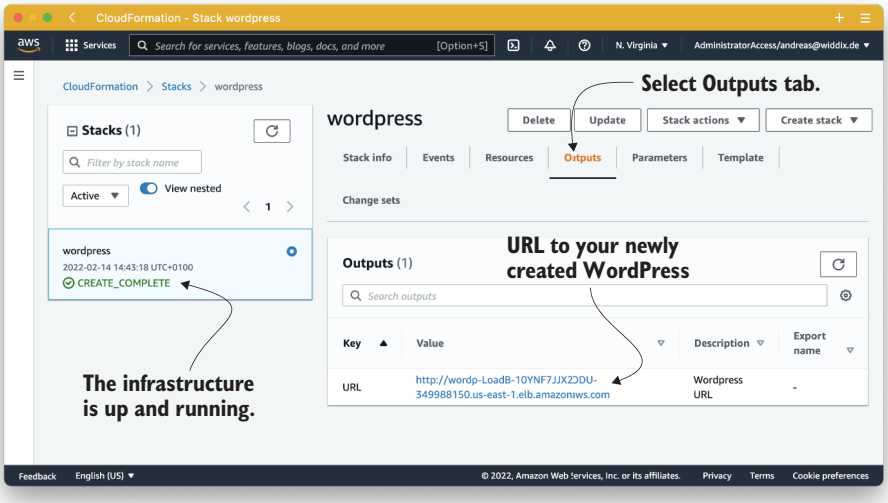


Figure 2.7 The blogging infrastructure has been created successfully.

2.2 Exploring your infrastructure

Now that you've created your blogging infrastructure, let's take a closer look at it. Your infrastructure consists of the following:

- Web servers running on virtual machines
- Load balancer
- MySQL database
- Network filesystem

2.2.1 Virtual machines

First, use the navigation bar to open the EC2 service as shown in figure 2.8. Next, select Instances from the subnavigation options. A list showing two virtual machines named wordpress shows up. When you select one of those instances, details about the virtual machine appear below.

You're now looking at the details of your virtual machine, also called an EC2 instance. Some interesting details follow:

- *Instance ID*—The ID of the virtual machine.
- *Instance type*—The size of the virtual machine (CPU and memory).
- *IPv4 Public IP*—The IP address that is reachable over the internet.
- *AMI ID*—Remember that you used the Amazon Linux OS. If you click the AMI ID, you'll see the version number of the OS, among other things.

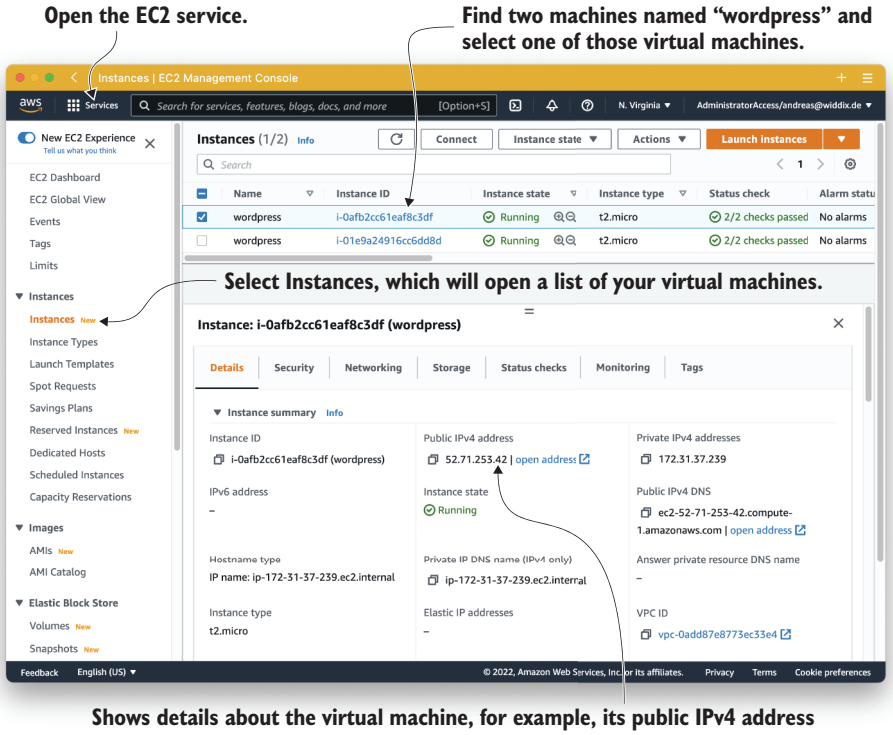


Figure 2.8 The virtual machines are running a web server to deliver WordPress.

Select the Monitoring tab to see how your virtual machine is used. This tab is essential if you really want to know how your infrastructure is doing. AWS collects some metrics and shows them here. For example, if the CPU is using more than 80% of its capacity, it might be a good time to add another virtual machine to prevent increasing response times. You will learn more about monitoring virtual machines in section 3.2.

2.2.2 Load balancer

Next, have a look at the load balancer, shown in figure 2.9, which is part of the EC2 service as well. There is no need to switch to a different service in the Management Console—just click Load Balancer in the subnavigation options on the left-hand side of the screen. Select your load balancer from the list to show more details. Your *internet-facing* load balancer is accessible from the internet via an automatically generated DNS name.

The load balancer forwards an incoming request to one of your virtual machines. A target group is used to define the targets for a load balancer. You'll find your target

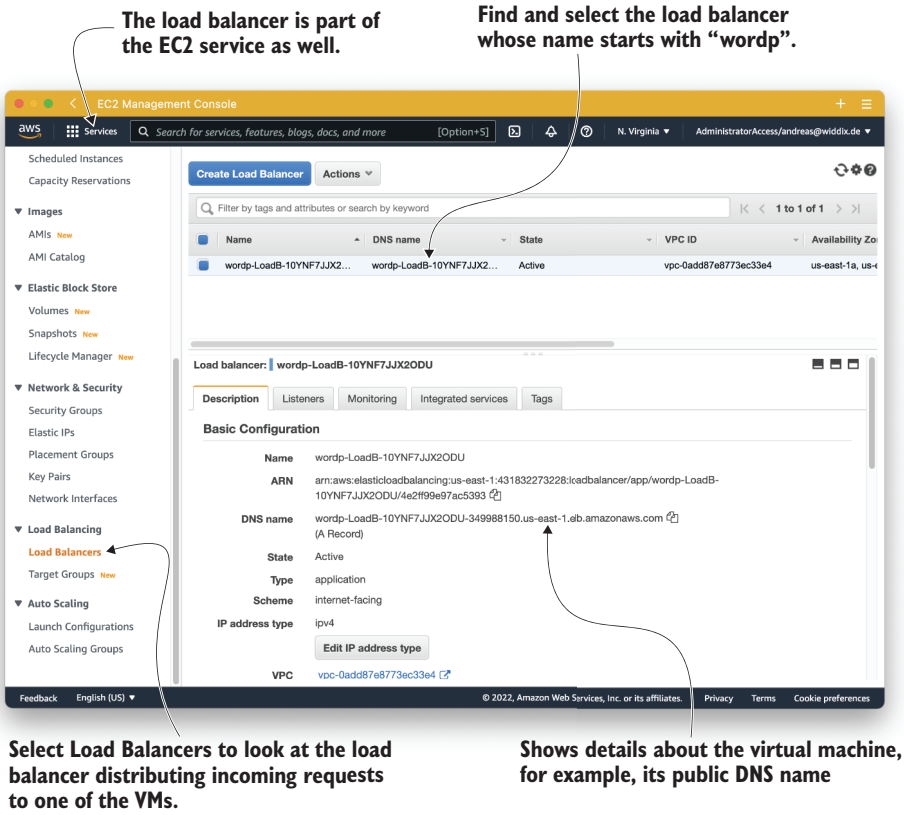


Figure 2.9 Get to know the load balancer.

group after switching to Target Groups through the subnavigation options of the EC2 service, as shown in figure 2.10.

The load balancer performs health checks to ensure requests are routed only to healthy targets. Two virtual machines are listed as targets for the target group. As you can see in the figure, the status of both virtual machines is healthy.

As before, on the Monitoring tab you can find interesting metrics that you should watch in production. If the traffic pattern changes suddenly, this indicates a potential problem with your system. You'll also find metrics indicating the number of HTTP errors, which will help you to monitor and debug your system.

2.2.3 MySQL database

The MySQL database is an important part of your infrastructure; you'll look at it next. To do so, open the Relational Database Service (RDS) via the main navigation. Afterward, select Databases from the subnavigation options. Select the database using the engine called MySQL community, as illustrated in figure 2.11.

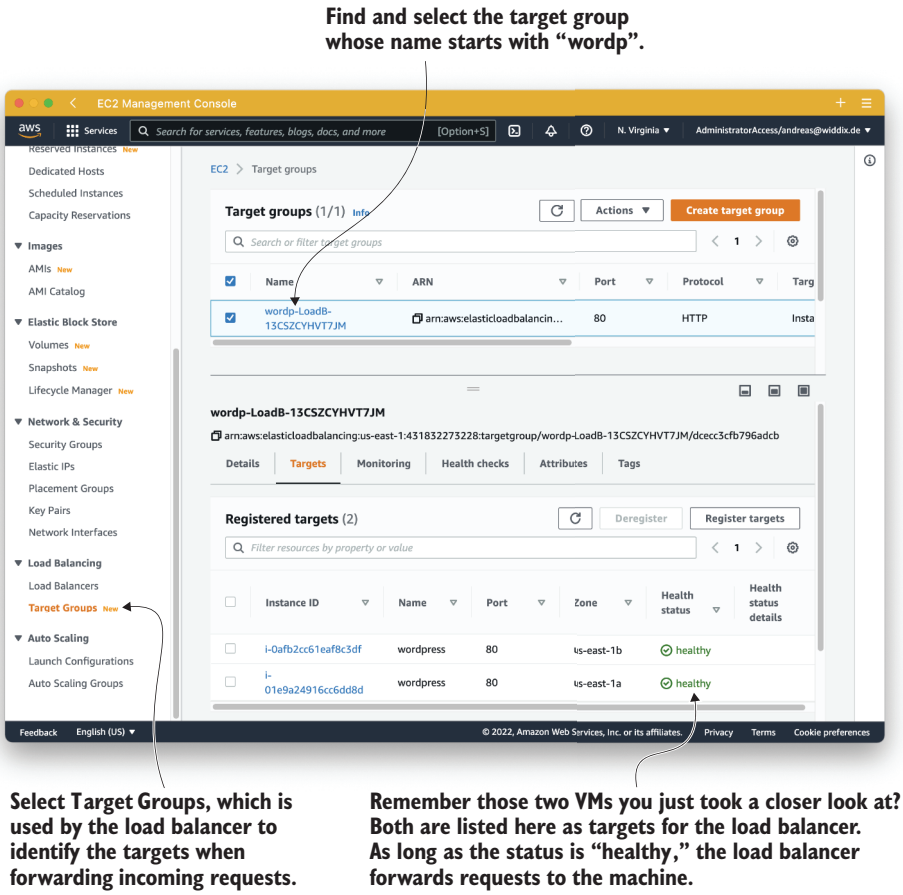


Figure 2.10 Details of target group belonging to the load balancer

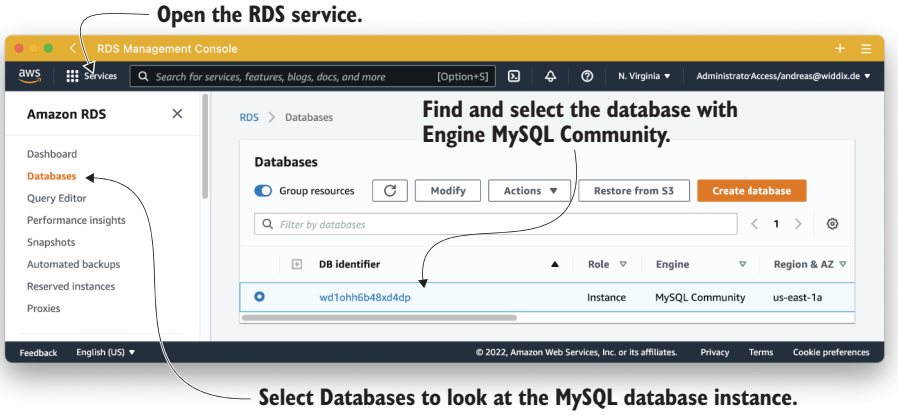


Figure 2.11 Finding the MySQL database

WordPress requires a MySQL database, so you have launched a database instance with the MySQL engine as noted in figure 2.12. Your blog receives a low amount of traffic, so the database doesn't need to be very powerful. A small instance class with a single virtual CPU and 1 GB memory is sufficient. Instead of using SSD storage, you are using magnetic disks, which is cheaper and sufficient for a web application with around 1,000 visitors per day.

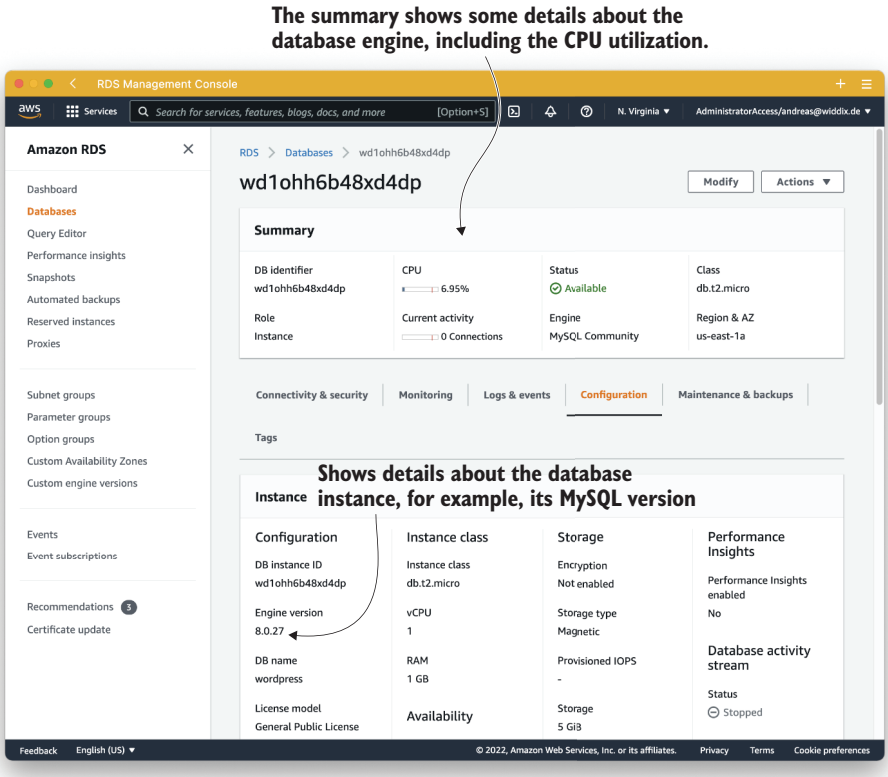


Figure 2.12 Details of the MySQL database storing data for the blogging infrastructure

As you'll see in chapter 10, other database engines, such as PostgreSQL or Oracle Database, are available, as well as more powerful instance classes, offering up to 96 cores with 768 GB memory, for example.

Common web applications use a database to store and query data. That is true for WordPress as well. The content management system (CMS) stores blog posts, comments, and more within a MySQL database.

WordPress also stores data outside the database on disk. For example, if an author uploads an image for their blog post, the file is stored on disk. The same is true when you are installing plug-ins and themes as an administrator.

2.2.4 Network filesystem

The Elastic File System (EFS) is used to store files and access them from multiple virtual machines. EFS is a storage service accessible through the NFS protocol. To keep things simple, all files that belong to WordPress, including PHP, HTML, CSS, and PNG files, are stored on EFS so they can be accessed from all virtual machines.

Open the EFS service via the main navigation as shown in figure 2.13. Next, select the filesystem whose name starts with "wordpress".

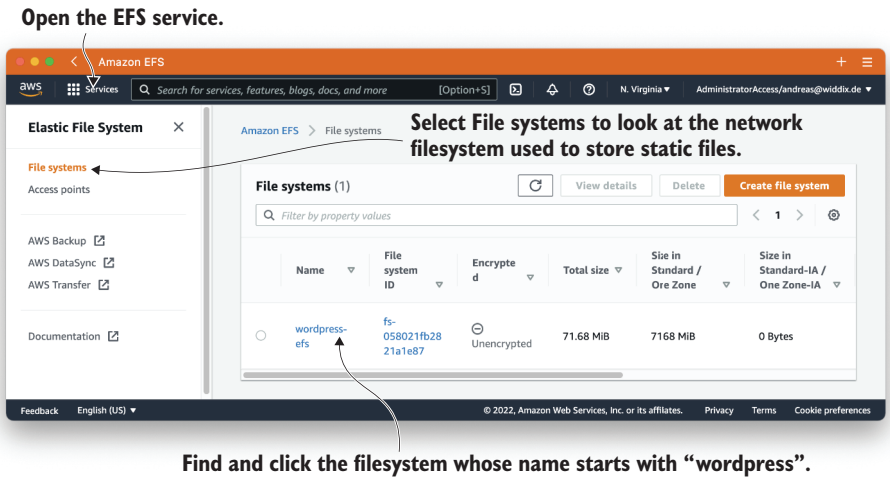


Figure 2.13 The NFS used to store the WordPress application and user uploads

Figure 2.14 shows the details of the filesystem. For example, the throughput mode *bursting* is used, to allow high I/O throughput for small periods of time during the day at low cost.

To mount the Elastic File System from a virtual machine, mount targets are needed. You should use two mount targets for fault tolerance. The network filesystem is accessible using a DNS name for the virtual machines.

Now it's time to evaluate costs. You'll analyze the costs of your blogging infrastructure in the next section.

2.3 How much does it cost?

Part of evaluating AWS is estimating costs. We recommend using the AWS Pricing Calculator to do so. We created a cost estimation including the load balancer, the virtual machines, the database, and the network file system. Go to http://mng.bz/VyEW to check out the results, which are also shown in figure 2.15. We estimate costs of about $75 for hosting WordPress on AWS in a highly available manner, which means all components are distributed among at least two different data centers. Don't worry—the

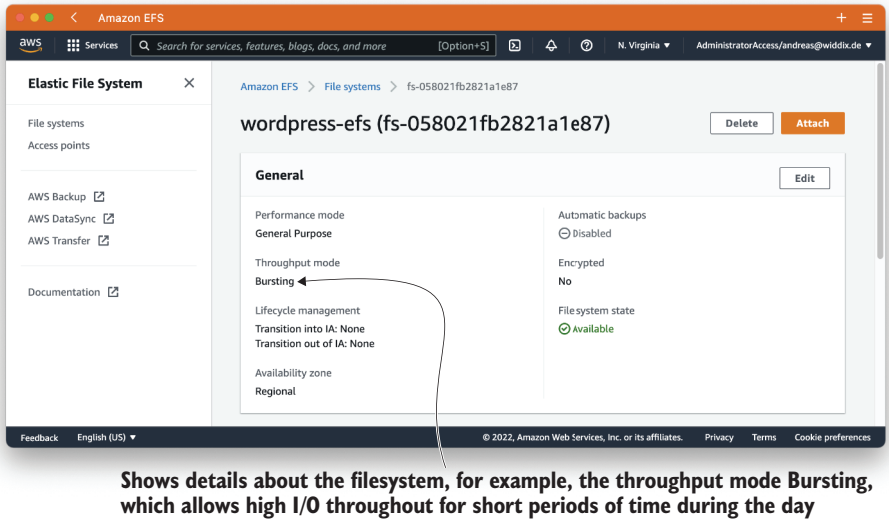


Figure 2.14 The details of the EFS

example in this chapter is covered by the Free Tier. Our cost estimation does not consider the Free Tier, because it applies for only the first 12 months.

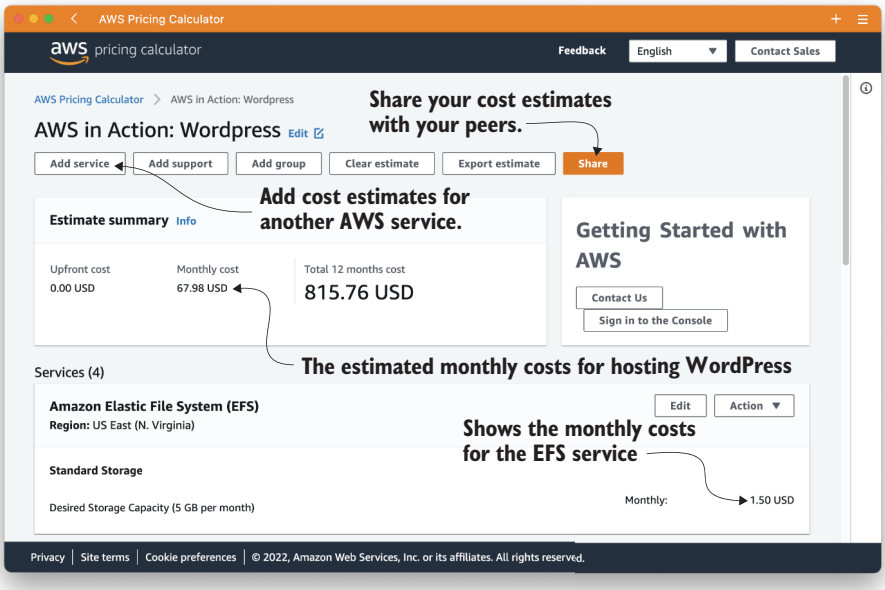


Figure 2.15 Blogging infrastructure cost calculation

Table 2.1 summarizes the results from the AWS Pricing Calculator.

Table 2.1 More detailed cost calculation for blogging infrastructure

AWS service	Infrastructure	Pricing	Monthly cost
EC2	Virtual machines	2 * 730 hours * $0.0116 (t2.micro)	$16.94
EC2	Storage	2 * 8 GB * $0.10 per month	$1.60
Application Load Balancer	Load balancer	730 hours * $0.0225 (load balancer hour) 200 GB per month	$18.03
Application Load Balancer	Outgoing traffic	100 GB * $0.00 (first 100 GB) 100 GB * $0.09 (up to 10 TB)	$9.00
RDS	MySQL database instance (primary + standby)	732.5 hours * $0.017 * 2	$24.82
RDS	Storage	5 GB * $0.115 * 2	$2.30
EFS	Storage	5 GB * $0.3	$1.50
			$74.19

Keep in mind that this is only an estimate. You're billed based on actual use at the end of the month. Everything is on demand and usually billed by seconds or gigabyte of usage. The following factors might influence how much you actually use this infrastructure:

- *Traffic processed by the load balancer*—Expect costs to go down in December and in the summer when people are on vacation and not looking at your blog.
- *Storage needed for the database*—If your company increases the amount of content in your blog, the database will grow, so the cost of storage will increase.
- *Storage needed on the NFS*—User uploads, plug-ins, and themes increase the amount of storage needed on the NFS, which will also increase the cost.
- *Number of virtual machines needed*—Virtual machines are billed by seconds of usage. If two virtual machines aren't enough to handle all the traffic during the day, you may need a third machine. In that case, you'll consume more seconds of virtual machines.

Estimating the cost of your infrastructure is a complicated task, but that is also true if your infrastructure doesn't run in AWS. The benefit of using AWS is that it's flexible. If your estimated number of virtual machines is too high, you can get rid of a machine and stop paying for it. You will learn more about the pricing model of the different AWS services throughout the course of this book.

You have completed the proof of concept for migrating your company's blog to AWS. It's time to shut down the infrastructure and complete your migration evaluation.

2.4 Deleting your infrastructure

Your evaluation has confirmed that you can migrate the infrastructure needed for the company's blog to AWS from a technical standpoint. You have estimated that a load balancer, virtual machines, and a MySQL database, as well as a NFS capable of serving 1,000 people visiting the blog per day, will cost you around $75 per month on AWS. That is all you need to come to a decision.

Because the infrastructure does not contain any important data and you have finished your evaluation, you can delete all the resources and stop paying for them.

Go to the CloudFormation service in the Management Console, and take the following steps, as shown in figure 2.16:

1. Select your wordpress stack.
2. Click the Delete button.

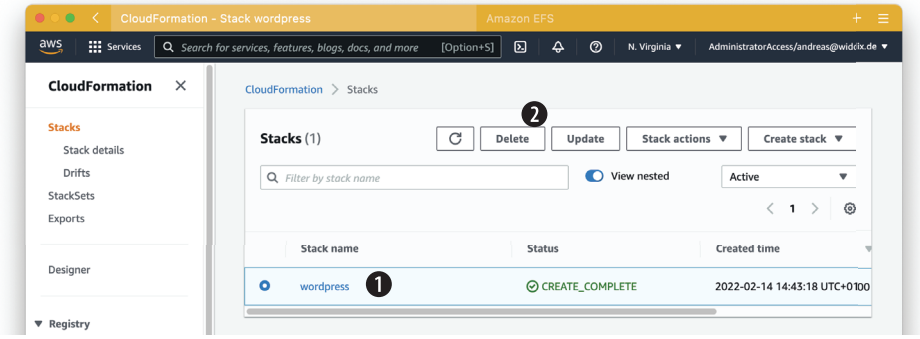


Figure 2.16 Deleting your blogging infrastructure

After you confirm the deletion of the infrastructure, as shown in figure 2.17, it takes a few minutes for AWS to delete all of the infrastructure's dependencies.

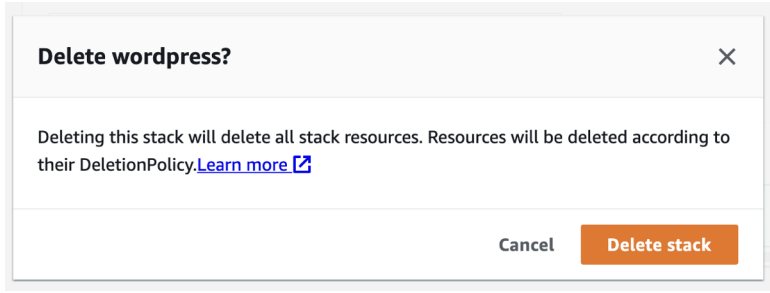


Figure 2.17 Confirming deletion of your blogging infrastructure

This is an efficient way to manage your infrastructure. Just as the infrastructure's creation was automated, its deletion is also. You can create and delete infrastructure on demand whenever you like. You pay for infrastructure only when you create and run it.

Summary

- Creating a cloud infrastructure for WordPress and any other application can be fully automated.
- AWS CloudFormation is a tool provided by AWS for free. It allows you to automate the managing of your cloud infrastructure.
- The infrastructure for a web application like WordPress can be created at any time on demand, without any up-front commitment for how long you'll use it.
- You pay for your infrastructure based on usage. For example, you pay for a virtual machine per second of usage.
- The infrastructure required to run WordPress consists of several parts, such as virtual machines, load balancers, databases, and network filesystems.
- The whole infrastructure can be deleted with one click. The process is powered by automation.

Part 2

Building virtual infrastructure consisting of computers and networking

Computing power and network connectivity have become a basic need for private households, medium-sized enterprises, and big corporations. Operating hardware in data centers that are in-house or outsourced has covered these needs in the past. Now the cloud is revolutionizing the way you can access computing power.

Virtual machines can be started and stopped on demand to fulfill your computing needs within minutes. Being able to install software on virtual machines enables you to execute your computing tasks without needing to buy or rent hardware.

If you want to understand AWS, you have to dive into the possibilities of the API working behind the scenes. You can control every single service on AWS by sending requests to a REST API. Based on this, a variety of solutions can help you automate your overall infrastructure. Infrastructure automation is a big advantage of the cloud compared to hosting on-premises. This part will guide you into infrastructure orchestration and the automated deployment of applications.

Creating virtual networks allows you to build closed and secure network environments on AWS and to connect these networks with your home or corporate network.

Chapter 3 covers working with virtual machines. You will learn about the key concepts of the EC2 service.

Chapter 4 contains different approaches to automate your infrastructure. You will learn how to make use of Infrastructure as Code.

Chapter 5 is about networking. You will learn how to secure your system with a virtual private network and firewalls.

Chapter 6 is about a new way of computing: functions. You will learn how to automate operational tasks with AWS Lambda.

3 Using virtual machines: EC2

This chapter covers

- Launching a virtual machine with Linux
- Controlling a virtual machine remotely via the Session Manager
- Monitoring and debugging a virtual machine
- Saving costs for virtual machines

It's impressive what you can achieve with the computing power of the smartphone in your pocket or the laptop in your bag. But if your task requires massive computing power or high network traffic, or needs to run reliably 24/7, a virtual machine is a better fit. With a virtual machine, you get access to a slice of a physical machine located in a data center. On AWS, virtual machines are offered by the service called Elastic Compute Cloud (EC2).

In this chapter, you will learn how to launch and manage a virtual machine on AWS. Also, we will show you how to connect to a virtual machine to install or configure applications. On top of that, you will learn how to monitor a virtual machine. Last but not least, we will introduce the different pricing options of the Elastic Compute Cloud (EC2) to make sure you get the most computing power for your money.

Not all examples are covered by Free Tier

The examples in this chapter are not all covered by the Free Tier. A special warning message appears when an example incurs costs. As for the other examples, as long as you don't run them longer than a few days, you won't pay anything for them. Keep in mind that this applies only if you created a fresh AWS account for this book and nothing else is going on in your AWS account. Try to complete the chapter within a few days; you'll clean up your account at the end.

3.1 Exploring a virtual machine

A virtual machine (VM) runs on a physical machine isolated from other virtual machines by the hypervisor; it consists of CPUs, memory, networking interfaces, and storage. The physical machine is called the *host machine*, and the VMs running on it are called *guests*. A *hypervisor* is responsible for isolating the guests from each other and for scheduling requests to the hardware by providing a virtual hardware platform to the guest system. Figure 3.1 shows these layers of virtualization.

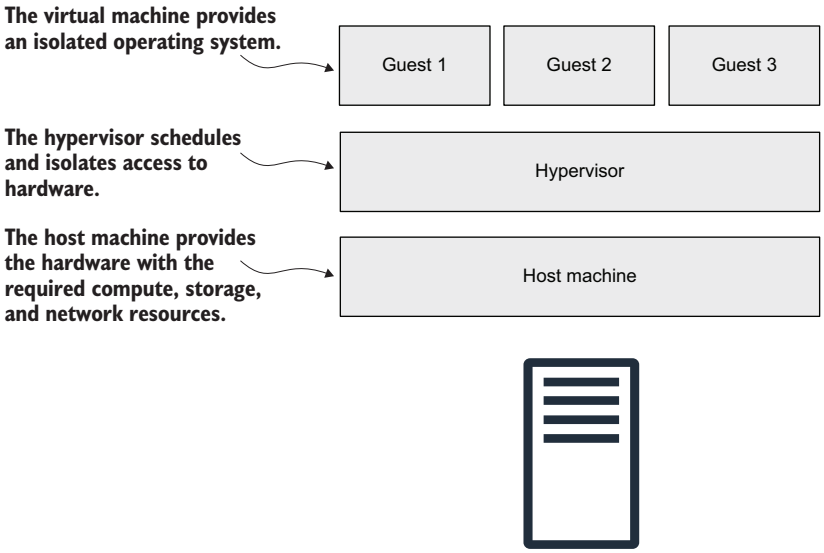


Figure 3.1 Layers of virtualization

Typical use cases for a virtual machine follow:

- Hosting a web application such as WordPress
- Operating an enterprise application, such as an ERP (enterprise resource planning) application
- Transforming or analyzing data, such as encoding video files

3.1.1 Launching a virtual machine

In the following example, you will launch a virtual machine to run a tool called *LinkChecker* that checks a website for broken links. Checking for links resulting in "404 Not Found" errors improves the usability and SEO score of your website. You could run LinkChecker on your local machine as well, but an EC2 instance in Amazon's data center offers more compute and networking capacities. As shown here, it takes only a few clicks to launch a virtual machine, which AWS calls an EC2 instance:

1. Open the AWS Management Console at https://console.aws.amazon.com.
2. Make sure you're in the N. Virginia (US East) region (see figure 3.2), because we optimized our examples for this region.
3. Click Services and search for EC2.
4. Click Launch Instance to start the wizard for launching a virtual machine, as shown in figure 3.2.

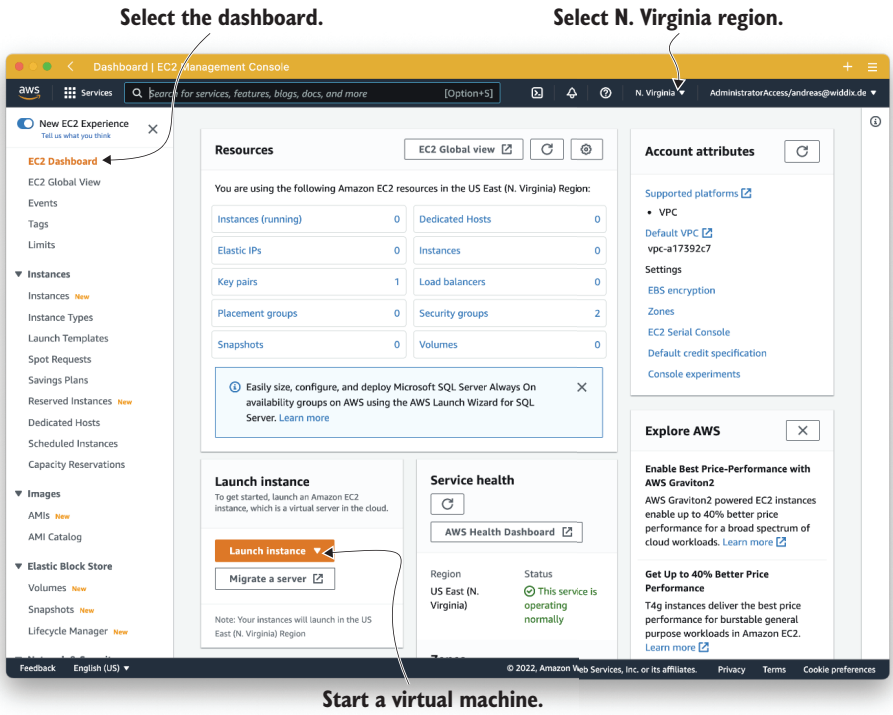


Figure 3.2 The EC2 dashboard gives you an overview of all parts of the service.

A form will appear, guiding you through the following details needed to create a virtual machine:

1. Naming the virtual machine
2. Selecting the operating system (OS)
3. Choosing the size of your virtual machine

4 Configuring details
5 Adding storage
6 Configuring a firewall
7 Granting the virtual machine permissions to access other AWS services

NAMING THE VIRTUAL MACHINE

To make it easy to find your virtual machine later, it is recommended you assign a name to it. That's especially important when other people have access to the same AWS account. Figure 3.3 shows the details.

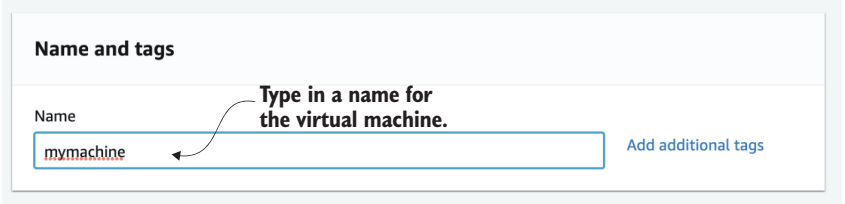


Figure 3.3 Naming the virtual machine

SELECTING THE OPERATING SYSTEM

Next, you need to choose an operating system (OS). In AWS, the OS comes bundled with preinstalled software for your virtual machine; this bundle is called an *Amazon Machine Image* (AMI). Select Amazon Linux 2 AMI (HVM), as shown in figure 3.4.

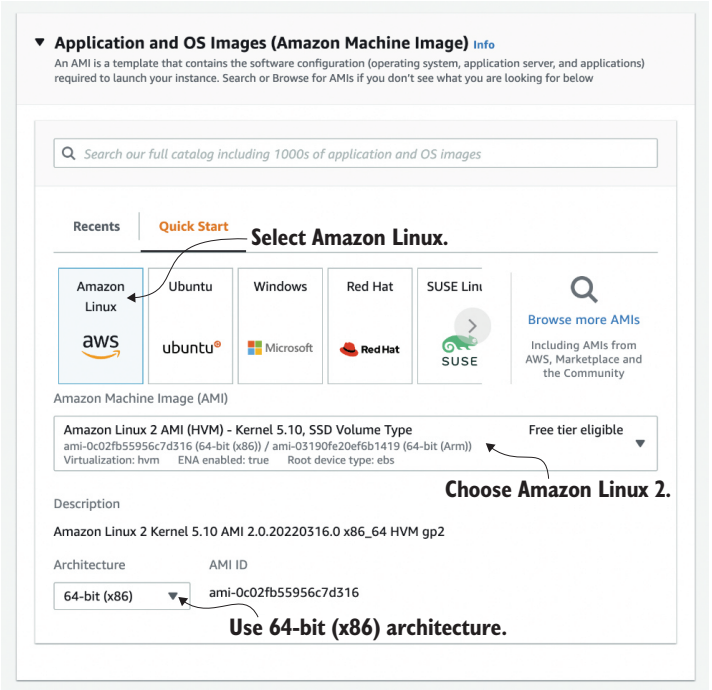


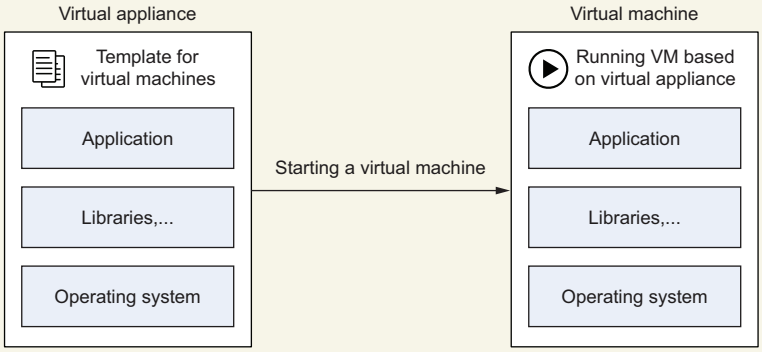
Figure 3.4 Choose the OS for your virtual machine.

The AMI is the basis for your virtual machine starts. AMIs are offered by AWS, third-party providers, and by the community. AWS offers the Amazon Linux AMI, which is based on Red Hat Enterprise Linux and optimized for use with EC2. You'll also find popular Linux distributions and AMIs with Microsoft Windows Server as well as more AMIs with preinstalled third-party software in the AWS Marketplace.

When choosing an AMI, start by thinking about the requirements of the application you want to run on the VM. Your knowledge and experience with a specific operating system are other important factors when deciding which AMI to start with. It's also important that you trust the AMI's publisher. We prefer working with Amazon Linux, because it's maintained and optimized by AWS.

Virtual appliances on AWS

As shown in the following figure, a virtual appliance is an image of a virtual machine containing an OS and preconfigured software. Virtual appliances are used when the hypervisor starts a new VM. Because a virtual appliance contains a fixed state, every time you start a VM based on a virtual appliance, you'll get exactly the same result. You can reproduce virtual appliances as often as needed, so you can use them to eliminate the cost of installing and configuring complex stacks of software. Virtual appliances are used by virtualization tools from VMware, Microsoft, and Oracle, and for Infrastructure as a Service (IaaS) offerings in the cloud.



A virtual appliance contains a template for a virtual machine.

The AMI is a special type of virtual appliance for use with the EC2 service. An AMI technically consists of a read-only filesystem including the OS, additional software, and configuration. You can also use AMIs for deploying software on AWS, but the AMI does not include the kernel of the OS. The kernel is loaded from an Amazon Kernel Image (AKI).

Nowadays, AWS uses a virtualization called Nitro. Nitro combines a KVM-based hypervisor with customized hardware (ASICs), aiming to provide a performance that is indistinguishable from bare metal machines.

(continued)

The current generations of VMs on AWS use hardware-assisted virtualization. The technology is called a hardware virtual machine (HVM). A virtual machine run by an AMI based on HVM uses a fully virtualized set of hardware and can take advantage of extensions that provide fast access to the underlying hardware.

CHOOSING THE SIZE OF YOUR VIRTUAL MACHINE

It's now time to choose the computing power needed for your virtual machine. AWS classifies computing power into instance types. An instance type primarily describes the number of virtual CPUs and the amount of memory.

Table 3.1 shows examples of instance types for different use cases. The prices represent the actual prices in USD for a Linux VM in the US East (N. Virginia) region, as recorded April 5, 2022.

Table 3.1 Examples of instance families and instance types

Instance type	Virtual CPUs	Memory	Description	Typical use case	Monthly cost (USD)
t2.nano	1	0.5 GiB	Small and cheap instance type, with moderate baseline performance and the ability to burst CPU performance above the baseline	Testing and development environments and applications with very low traffic	$4
m6i.large	2	8 GiB	Has a balanced ratio of CPU, memory, and networking performance	All kinds of applications, such as medium databases, web servers, and enterprise applications	$69
r6i.large	2	16 GiB	Optimized for memory-intensive applications with extra memory	In-memory caches and enterprise application servers	$90

Instance families are optimized for different kinds of use cases, as described next:

- *T family*—Cheap, moderate baseline performance with the ability to burst to higher performance for short periods of time
- *M family*—General purpose, with a balanced ration of CPU and memory
- *C family*—Computing optimized, high CPU performance
- *R family*—Memory optimized, with more memory than CPU power compared to the M family
- *X family*—Extensive capacity with a focus on memory, up to 1952 GB memory and 128 virtual cores

- *D family*—Storage optimized, offering huge HDD capacity
- *I family*—Storage optimized, offering huge SSD capacity
- *P, G, and CG family*—Accelerated computing based on GPUs (graphics processing units)
- *F family*—Accelerated computing based on FPGAs (field-programmable gate arrays)

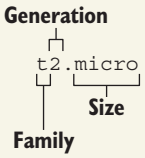
Additional instance families are available for niche workloads like high-performance computing, in-memory databases, MacOS workloads, and more. See https://aws.amazon.com/ec2/instance-types/ for a full list of instance types and families.

Our experience indicates that you'll overestimate the resource requirements for your applications. We recommend that you try to start your application with a smaller instance type than you think you need at first—you can change the instance family and type later if needed.

Instance types and families

The names for different instance types are all structured in the same way. The instance family groups instance types with similar characteristics. AWS releases new instance types and families from time to time; the different versions are called generations. The instance size defines the capacity of CPU, memory, storage, and networking. For example, the instance type `t2.micro` tells you the following:

- The instance family is called `t`. It groups small, cheap virtual machines with low-baseline CPU performance but the ability to burst significantly over baseline CPU performance for a short time.
- You're using generation 2 of this instance family.
- The size is micro, indicating that the EC2 instance is very small.



Disassembling the t2.micro instance type

Computer hardware is getting faster and more specialized, so AWS is constantly introducing new instance types and families. Some of them are improvements of existing instance families, and others are focused on specific workloads. For example, the instance family `R6i`, introduced in November 2021, provides instances for memory-intensive workloads and replaces the `R5` instance types.

One of the smallest and cheapest VMs will be enough for your first experiments. Choose the instance type `t2.micro`, as shown in figure 3.5, which is eligible for the Free Tier.

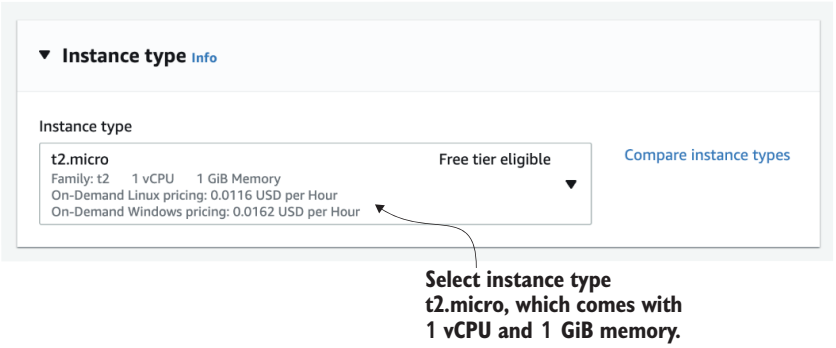


Figure 3.5 Choosing the size of your virtual machine

You might have already heard about Apple switching from Intel processors to ARM processors. The reason for this is that custom-built ARM processors achieve higher performance with lower energy consumption. This is, of course, exciting not only for laptops but also for servers in the data center.

AWS offers machines based on custom-built ARM processors called *Graviton* as well. As a customer, you will notice similar performance at lower costs. However, you need to make sure that the software you want to run is compiled for the ARM64 architecture. We migrated workloads from EC2 instances with Intel processors to virtual machines with ARM processors a few times already, typically within one to four hours. We would have liked to use Graviton instances for the third edition of this book, but, unfortunately, these were not yet part of the Free Tier at that time. We highly recommend you check out the following Graviton instance types offered by AWS:

- `T4g`—Burstable and cost-effective instance types
- `M6g`, `M6gd`—General-purpose instance types
- `C6g`, `C6gd`, `C6gn`, `C7g`—Compute-optimized instance types
- `R6g`, `R6gd`, `X2gd`—Memory-optimized instance types
- `Im4gn`, `Is4gen`—Instance types providing built-in storage
- `G5g`—A special instance type optimized for Android game streaming

CONFIGURING THE KEY PAIR FOR LOGIN

As an administrator of a Linux machine, you used a username and password or username and a public/private key pair to authenticate yourself in the past. By default, AWS uses a username and a key pair for authentication. That's why the next section of the wizard asks you about defining a key pair for the EC2 instance you are going to launch. We try to avoid this approach, because it works only for a single user, and it is not possible to change the key pair externally after launching an EC2 instance.

Therefore, we recommend a different approach to authenticate to an EC2 instance that we will introduce shortly. There is no need to configure the key pair now, so please select Proceed without a Key Pair, as shown in figure 3.6.

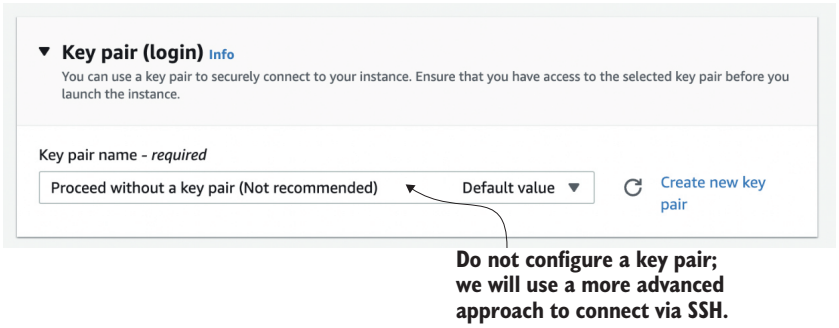


Figure 3.6 Proceeding without a key pair

Defining network and firewall settings

In the next section of the setup process, you can configure the network and firewall settings for the EC2 instance. The default settings are fine for now. You will learn more about networking on AWS in chapter 5. The only thing you should change is to deselect the Allow SSH Traffic option. As promised before, you will learn about a new approach to connect to EC2 instances that does not require inbound SSH connectivity. With the configuration shown in figure 3.7, the firewall does not allow any incoming connections at all.

Attaching storage

Next, attach some storage to your virtual machine for the root filesystem. It is fine to keep the defaults and attach a volume with 8 GB of type `gp2`, which consists of network-attached SSDs, as illustrated in figure 3.8.

Setting advanced details

Last but not least, you need to configure an advanced detail for the EC2 instance you are going to launch: an IAM role. You will learn more about IAM in chapter 5. For now, all you need to know is that an IAM role grants processes running on the virtual machine access to other AWS services. This is needed because you will use AWS services called Systems Manager and EC2 Instance Connect to establish an SSH connection with your virtual machine later.

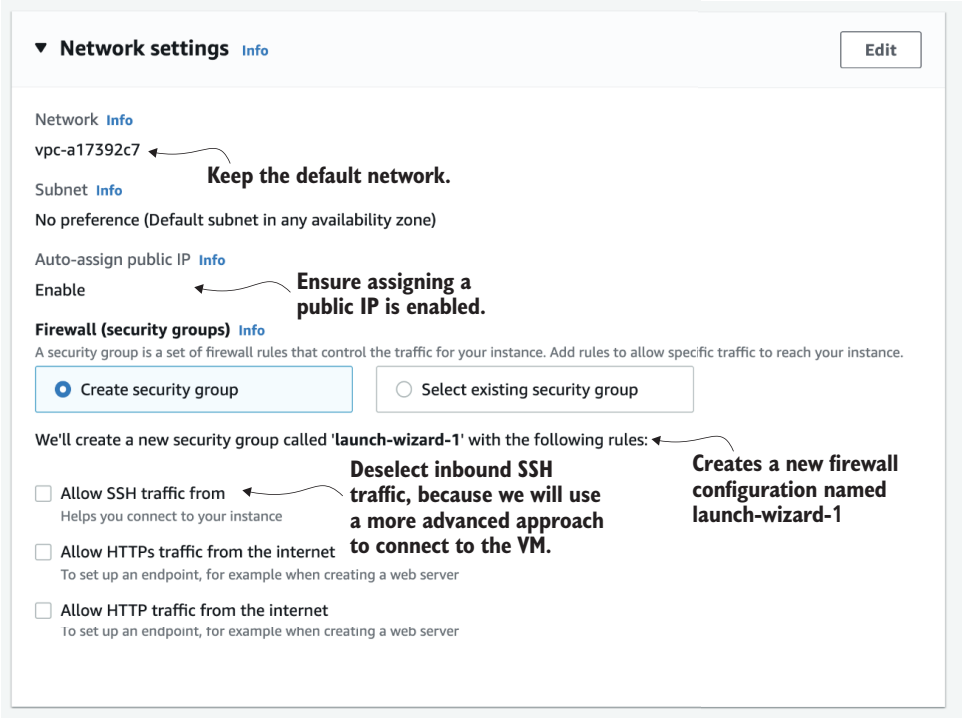


Figure 3.7 Configuring the network and firewall settings for the EC2 instance

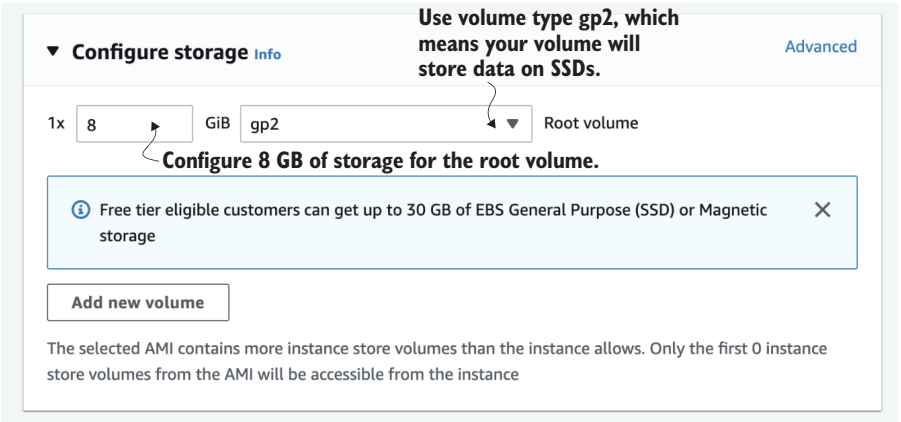


Figure 3.8 Adding storage to the virtual machine

Creating an IAM role

Before you proceed configuring your EC2 instance, you need to create an IAM role. To do so open https://console.aws.amazon.com/iam/ in a new tab in your browser and do the following:

1. Select Roles from the subnavigation options.
2. Click the Create Role button.
3. Select the trusted entity type AWS Service and EC2, as shown in figure 3.9.
4. Proceed to the next step.

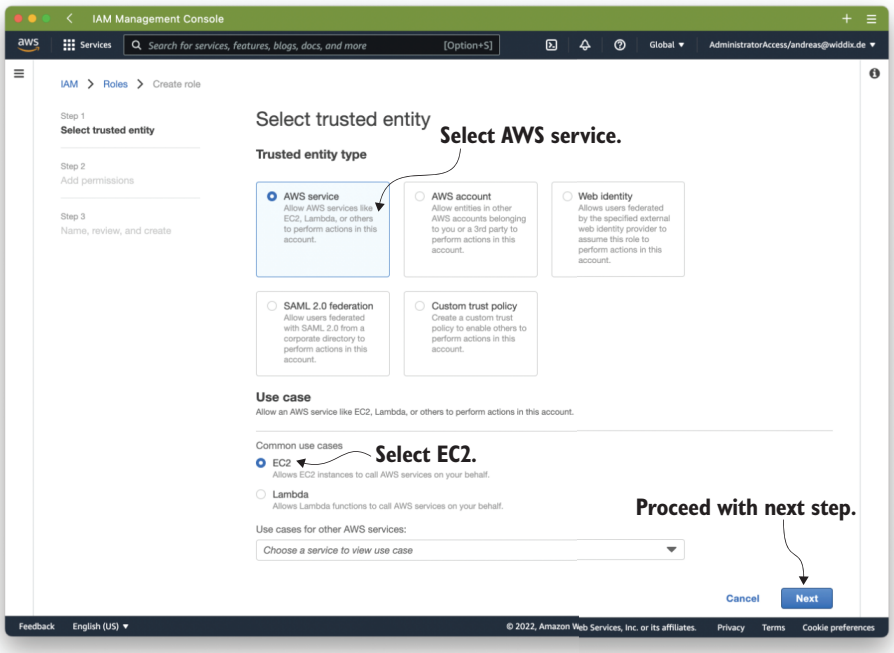


Figure 3.9 Creating an IAM role: Select trusted entity

Add permissions to the IAM role by filtering and selecting the policies named `AmazonSSMManagedInstanceCore`, as demonstrated in figure 3.10. Doing so is required so that the Systems Manager agent running on the EC2 instance, which you will use to connect to the EC2 instance later, works properly. Afterward, proceed to the next step.

To create the IAM role, type the name `ec2-ssm-core`—please use this exact name because later chapters depend on it—and a description, as shown in figure 3.11. After doing so, click the Create Role button at the bottom of the page.

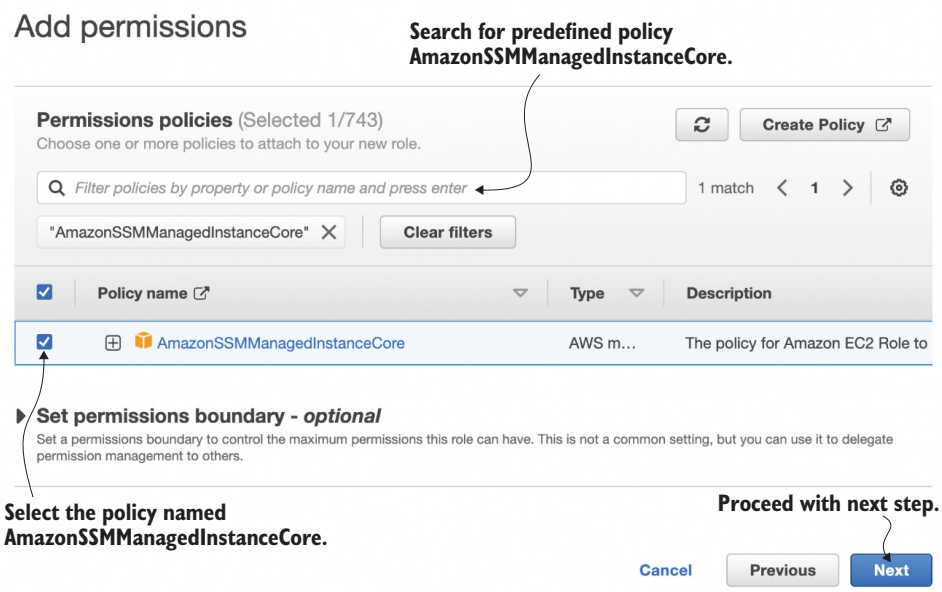


Figure 3.10 Creating an IAM role: Add permissions

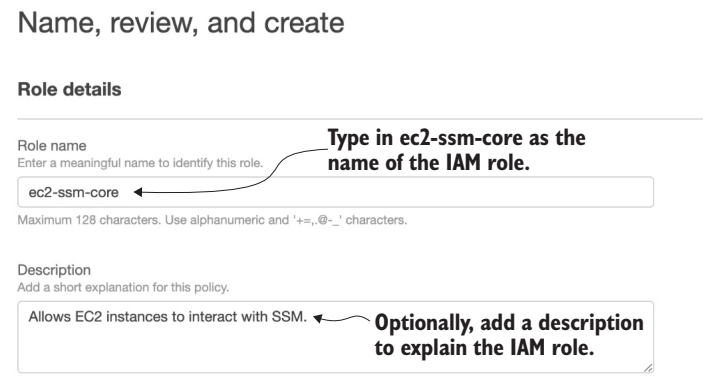


Figure 3.11 Creating an IAM role: Role details

Switch back to the EC2 browser tab. We have a lot to configure in the Advanced Details section. Keep the default setting for everything except the IAM instance profile. Click the Reload button next to the dropdown list and select the IAM instance profile named `ec2-ssm-core` (see figure 3.12).

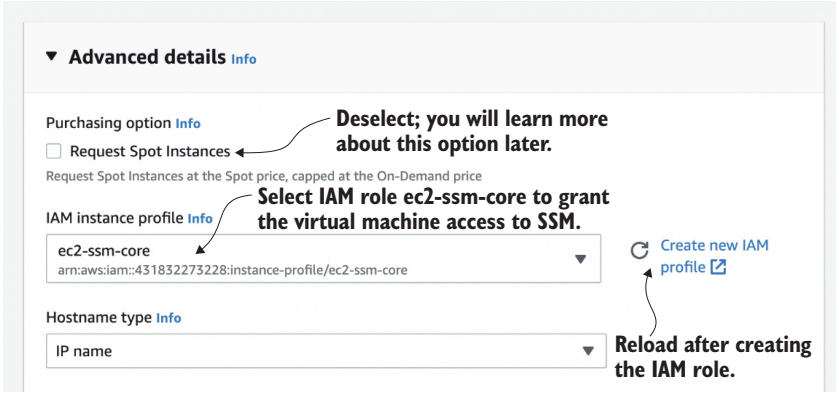


Figure 3.12 Configuring the IAM role for the EC2 instance

Launching the EC2 instance

You are now ready to launch your virtual machine. To do so, just click the Launch Instance button, as illustrated in figure 3.13.

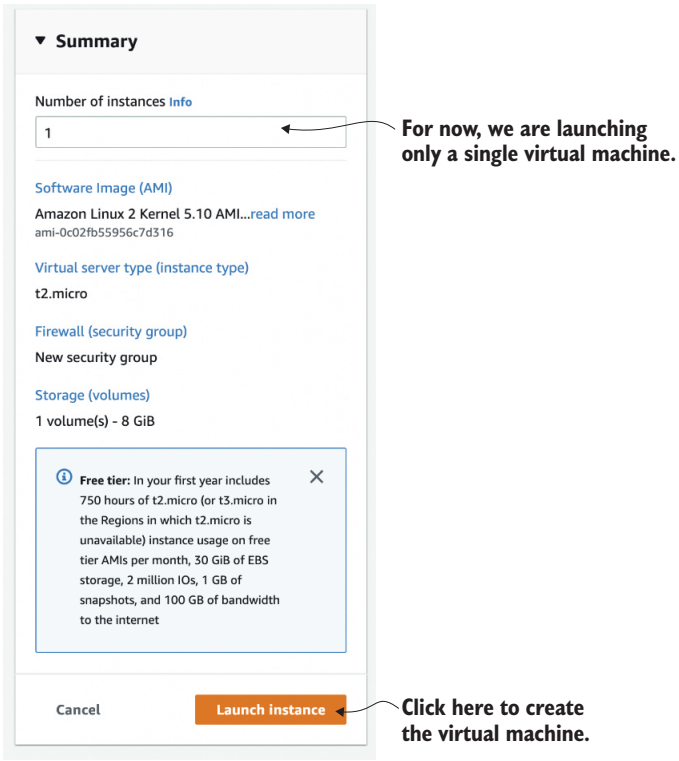


Figure 3.13 Launching the EC2 instance

Mark this day on your calendar: you've just launched your first EC2 instance. Many more will follow!

3.1.2 Connecting to your virtual machine

To be able to do something with the running virtual machines, you have to log in next. After connecting to your E2 instance, you will install and run LinkChecker to check a website for broken links. Of course, this exercise is just an example. When you get administrator access to the virtual machine, you have full control and are able to install the application and configure the operating system as needed.

You will learn how to connect to an EC2 instance by using the AWS Systems Manager Session Manager. The advantages of this approach follow:

- You do not need to configure key pairs upfront but use temporary key pairs instead.
- You don't need to allow inbound SSH or RDP connectivity, which limits the attack surface.

Using AWS Systems Manager Session Manager comes with the following requirements:

- Works with virtual machines based on Amazon Linux 2, CentOS, Oracle Linux, Red Hat Enterprise Linux, SUSE Linux Enterprise Server, macOS, and Windows Server.
- Requires the SSM agent, which is preinstalled on Amazon Linux 2, macOS >10.14, SUSE Linux Enterprise Server 12/15, Ubuntu Server >16.04, and Windows Server 2008–2012/2016/2019/2022.
- Requires an IAM role that grants permissions to the AWS Systems Manager service; see the previous section.

The following instructions guide you through the steps necessary to connect to the EC2 instance you launched in the previous section, which fulfills all of the following requirements:

1. Choose Instances from the subnavigation options of the EC2 service, in case you are not looking at the list of EC2 instances already, as shown in figure 3.14.
2. Select the instance named `mymachine`.
3. Wait until the instance reaches the status `Running`.
4. Click the Connect button.
5. Select the Session Manager tab.
6. Press the Connect button as shown in figure 3.15.

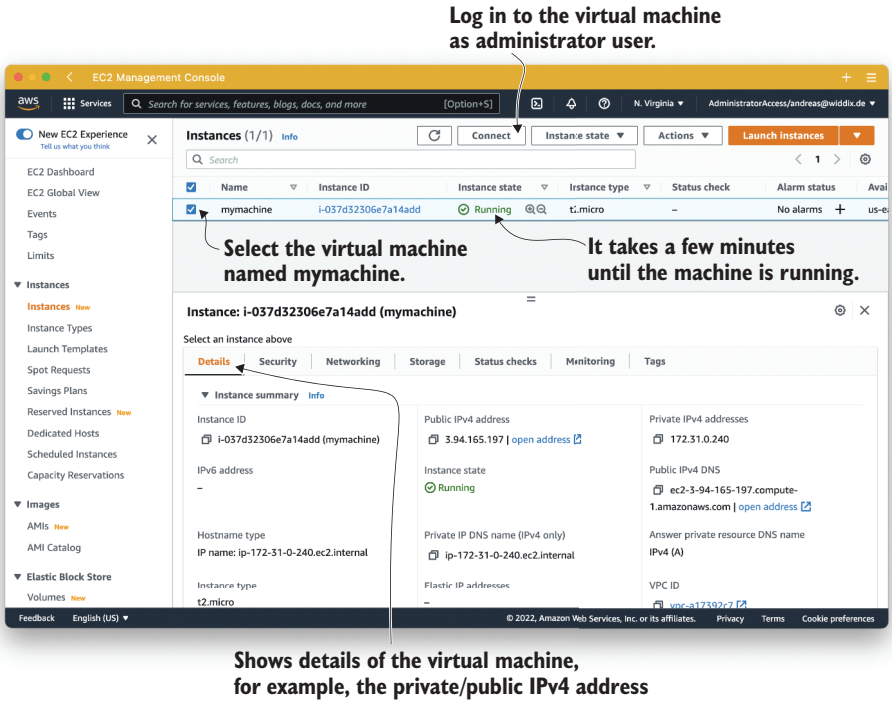


Figure 3.14 Listing all EC2 instances running in the region N. Virginia

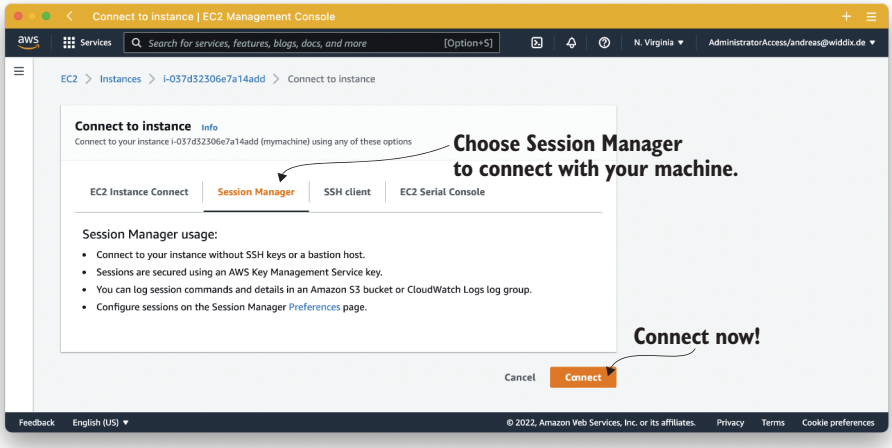


Figure 3.15 Connecting to an EC2 instance via Session Manager

After a few seconds, a terminal appears in your browser window, as shown in figure 3.16.

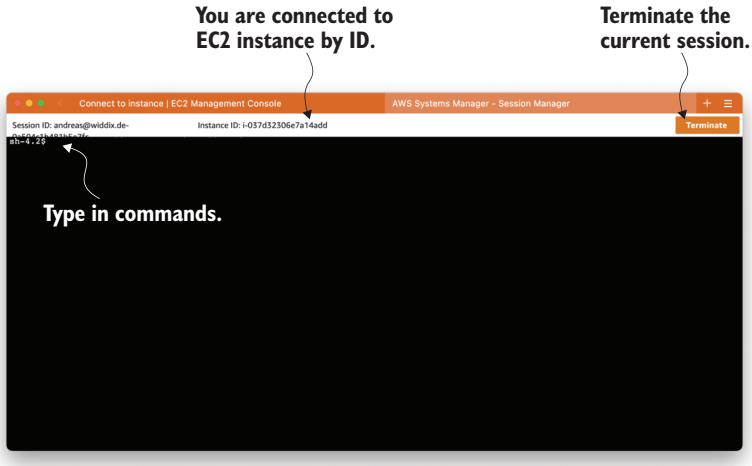


Figure 3.16 The terminal connected with your EC2 instance is waiting for input.

SSH and SCP

Accessing an EC2 instance directly from the browser is great, but sometimes you might prefer a good old SSH connection from your local machine, for example, to copy files with the help of SCP (secure file copy via SSH). Doing so is possible as well. If you're interested in the details, see http://mng.bz/JleK and http://mng.bz/wPw2.

You are now ready to enter the first commands into the terminal of your virtual machine.

3.1.3 *Installing and running software manually*

Back to our example: you launched a virtual machine to run LinkChecker to find broken links on a website. First, you need to install LinkChecker. Also, the tool requires a Python runtime environment.

In general, Amazon Linux 2 comes with the package manager `yum`, which allows you to install additional software. Besides that, Amazon Linux 2 comes with an extras library, covering additional software packages. Run the following command to install Python 3.8. Press y when prompted to acknowledge the changes:

```
$ sudo amazon-linux-extras install python3.8
```

Next, execute the following command to install the LinkChecker tool, which allows you to find broken links on a website:

```
$ pip3 install linkchecker
```

Now you're ready to check for links pointing to websites that no longer exist. To do so, choose a website and run the following command. The `-r` option limits the recursion level that the tool will crawl through:

```
$ ~/.local/bin/linkchecker -r 2 https://
```

The output of checking the links looks something like this:

```
[...]
URL        `/images/2022/02/terminal.png'
Name       `Connect to your EC2 instance using SSH the modern way'
Parent URL https://cloudonaut.io, line 379, col 1165
Real URL   https://cloudonaut.io/images/2022/02/terminal.png
Check time 2.959 seconds
Size       0B
Result     Error: 404 Not Found
10 threads active,     5 links queued,   72 links in  87 URLs checked, ...
 1 thread active,      0 links queued,   86 links in  87 URLs checked, ...

Statistics:
Downloaded: 66.01KB.
Content types: 26 image, 29 text, 0 video, 0 audio, 3 application, ...
URL lengths: min=21, max=160, avg=56.

That's it. 87 links in 87 URLs checked. 0 warnings found. 1 error found.
Stopped checking at 2022-04-04 09:02:55+000 (22 seconds)
[...]
```

Depending on the number of web pages, the crawler may need some time to check all of them for broken links. At the end, it lists the broken links and gives you the chance to find and fix them.

3.2 Monitoring and debugging a virtual machine

If you need to find the reason for an error or determine why your application isn't behaving as you expect, it's important to have access to tools that can help with monitoring and debugging. AWS provides tools that let you monitor and debug your virtual machines. One approach is to examine the virtual machine's logs.

3.2.1 Showing logs from a virtual machine

If you need to find out what your virtual machine was doing during and after startup, you have a simple solution. AWS allows you to see the EC2 instance's logs with the help of the Management Console (the web interface you use to start and stop virtual machines). Follow these steps to open your VM's logs:

1. Go to EC2 in the AWS Management Console: https://console.aws.amazon.com/ec2/.
2. Open the list of all virtual machines by selecting Instances from the subnavigation options.

3 Select the running virtual machine by clicking the row in the table.
4 In the Actions menu, select Monitor and Troubleshoot > Get System Log.
5 A screen showing the system logs from your VM that would normally be displayed on a physical monitor during startup, as shown in figure 3.17, appears.

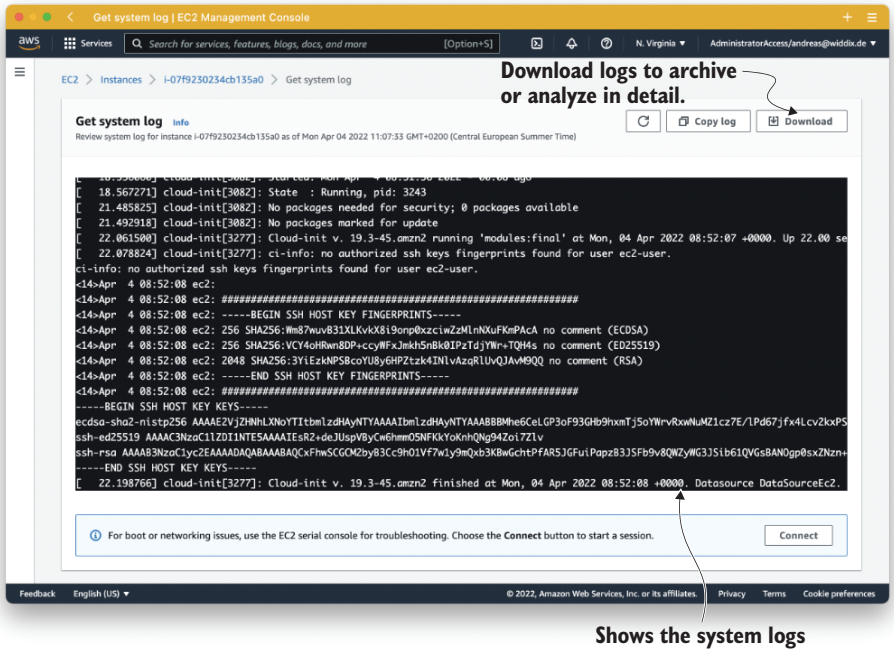


Figure 3.17 Debugging a virtual machine with the help of system logs

The log contains all log messages that would be displayed on the monitor of your machine if you were running it on-premises. Watch for any log messages stating that an error occurred during startup. If the error message is not obvious, you should contact the vendor of the AMI or AWS Support, or post your question to the official AWS community at https://repost.aws.

This is a simple and efficient way to access your system logs without needing an SSH connection. Note that it will take several minutes for a log message to appear in the log viewer.

3.2.2 Monitoring the load of a virtual machine

AWS can help you answer another question: is your virtual machine close to its maximum capacity? Follow these steps to open the EC2 instance's metrics:

1 Go to EC2 in the AWS Management Console: https://console.aws.amazon.com/ec2/.

2. Open the list of all virtual machines by choosing Instances from the subnavigation options.
3. Select the running virtual machine by clicking the row in the table.
4. Select the Monitoring tab in the lower-right corner.
5. Click the three dots at the upper-right corner of the Network in (Bytes) metric and choose Enlarge.

You'll see a graph that shows the virtual machine's use of incoming networking traffic, similar to figure 3.18, with metrics for CPU, network, and disk usage. As AWS is looking at your VM from the outside, there is no metric indicating the memory usage. You can publish a memory metric yourself, if needed. The metrics are updated every five minutes if you use basic monitoring, or every minute if you enable detailed monitoring of your virtual machine, which costs extra.

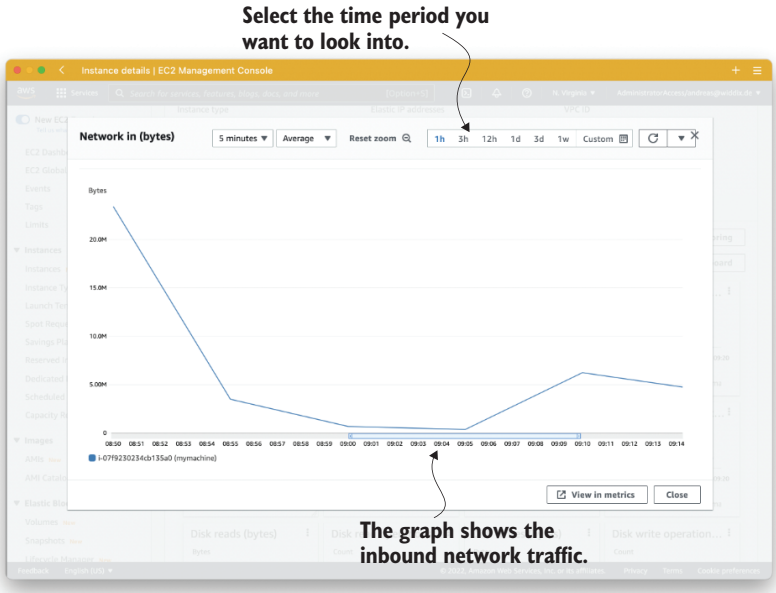


Figure 3.18 Gaining insight into a virtual machine's incoming network traffic with the CloudWatch metric

Checking the metrics of your EC2 instance is helpful when debugging performance problems. You will also learn how to increase or decrease your infrastructure based on these metrics in chapter 17.

Metrics and logs help you monitor and debug your virtual machines. Both tools can help ensure that you're providing high-quality services in a cost-efficient manner. Look at "Monitor Amazon EC2" in the AWS documentation at http://mng.bz/xMqg for more detailed information about monitoring your virtual machines.

3.3 Shutting down a virtual machine

To avoid incurring charges, you should always turn off virtual machines when you're not using them. You can use the following four actions to control a virtual machine's state:

- *Start*—You can always start a stopped virtual machine. If you want to create a completely new machine, you'll need to launch another virtual machine.
- *Stop*—You can always stop a running virtual machine. A stopped virtual machine doesn't incur charges, except for attached resources like network-attached storage. A stopped virtual machine can be started again but likely on a different host. If you're using network-attached storage, your data persists.
- *Reboot*—Have you tried turning off your virtual machine, then turning it on again? If you need to reboot your virtual machine, this action is what you want. You won't lose any persistent data when rebooting a virtual machine because it stays on the same host.
- *Terminate*—Terminating a virtual machine means deleting it. You can't start a virtual machine that you've terminated. The virtual machine is deleted, usually together with its dependencies, like network-attached storage and public and private IP addresses. A terminated virtual machine doesn't incur charges.

WARNING The difference between *stopping* and *terminating* a virtual machine is important. You can start a stopped virtual machine. This isn't possible with a terminated virtual machine. If you terminate a virtual machine, you delete it.

Figure 3.19 illustrates the difference between stopping and terminating an EC2 instance, with the help of a flowchart.

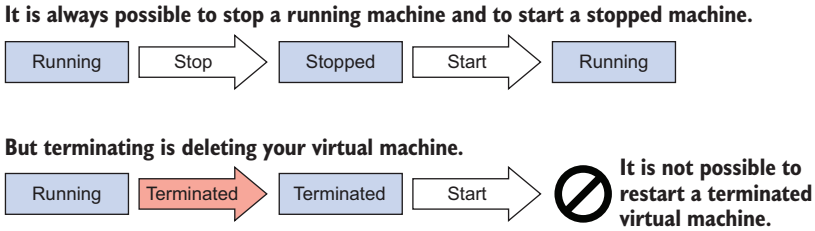


Figure 3.19 The difference between stopping and terminating a virtual machine

Stopping or terminating unused virtual machines saves costs and prevents you from being surprised by an unexpected bill from AWS. You may want to stop or terminate unused virtual machines when the following situations arise:

- You have launched virtual machines to implement a proof of concept. After finishing the project, the virtual machines are no longer needed. Therefore, you can terminate them.

- You are using a virtual machine to test a web application. Because no one else uses the virtual machine, you can stop it before you knock off work and start it back up again the following day.
- One of your customers canceled their contract. After backing up the relevant data, you can terminate the virtual machines that had been used for your former customer.

After you terminate a virtual machine, it's no longer available and eventually disappears from the list of virtual machines.

Cleaning up

Terminate the virtual machine named `mymachine` that you started at the beginning of this chapter by doing the following:

1. Go to EC2 in the AWS Management Console: https://console.aws.amazon.com/ec2/.
2. Open the list of all virtual machines and select Instances from the subnavigation options.
3. Select the running virtual machine by clicking the row in the table.
4. Click the Instance State button and select Terminate Instance.

3.4 Changing the size of a virtual machine

It is always possible to change the size of a virtual machine. This is one of the advantages of using the cloud, and it gives you the ability to scale vertically. If you need more computing power, increase the size of the EC2 instance, or vice versa.

In this section, you'll learn how to change the size of a running virtual machine. To begin, follow these steps to start a small virtual machine:

1. Go to EC2 in the AWS Management Console: https://console.aws.amazon.com/ec2/.
2. Click the Launch Instances button.
3. Enter growingup as the name of the virtual machine.
4. Choose the Amazon Linux 2 AMI.
5. Select instance type `t2.micro`.
6. Select the option Proceed without a Key Pair.
7. Keep the default for network and storage.
8. Select the IAM instance profile `ec2-ssm-core` under Advanced Details.
9. Launch the instance.

You've now started an EC2 instance of type `t2.micro`. This is one of the smallest virtual machines available on AWS.

Use the Session Manager to connect to the instance as demonstrated in the previous section, and execute `cat /proc/cpuinfo` and `free -m` to see information about the machine's capabilities. The output should look similar to this:

```
$ cat /proc/cpuinfo
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 63
model name      : Intel(R) Xeon(R) CPU E5-2676 v3 @ 2.40GHz
stepping        : 2
microcode       : 0x46
cpu MHz         : 2399.915
cache size      : 30720 KB
[...]

$ free -m
              total        used        free      shared  buff/cache   available
Mem:            965          93         379           0         492         739
Swap:             0           0           0
```

Your virtual machine provides a single CPU core and 965 MB of memory. If your application is having performance problems, increasing the instance size can solve this. Use your machine's metrics as described in section 3.2 to find out whether you are running out of CPU or networking capacity. Would your application benefit from additional memory? If so, increasing the instance size will improve the application's performance as well.

If you need more CPUs, more memory, or more networking capacity, you can choose from many other sizes. You can even change the virtual machine's instance family and generation. To increase the size of your VM, you first need to stop it as follows:

1. Go to EC2 in the AWS Management Console: https://console.aws.amazon.com/ec2/.
2. Click Instances in the submenu to jump to an overview of your virtual machines.
3. Select your running VM from the list by clicking it.
4. Click Instance State > Stop Instance.

WARNING Starting a virtual machine with instance type `m5.large` incurs charges. Go to http://aws.amazon.com/ec2/pricing if you want to see the current on-demand hourly price for an `m5.large` virtual machine.

After waiting for the virtual machine to stop, you can change the instance type as follows:

1. Click the Actions button and select Instance Settings.
2. Click Change Instance Type.

3 Select `m5.large` as the new instance type and click the Apply button, as shown in figure 3.20.

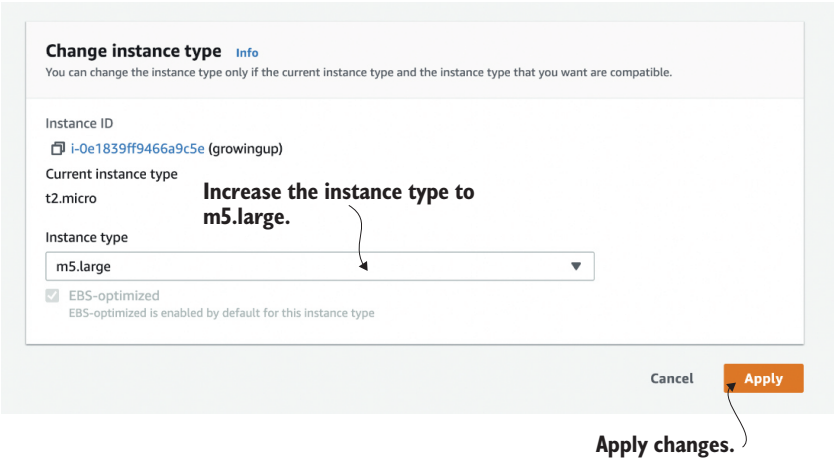


Figure 3.20 Increasing the size of your virtual machine by selecting `m4.large` as the instance type

You've now changed the size of your virtual machine and are ready to start it again. To do so, select your EC2 instance and click Start Instance under Instance State. Your VM will start with more CPUs, more memory, and increased networking capabilities.

Use the Session Manager to connect to your EC2 instance, and execute `cat /proc/cpuinfo` and `free -m` to see information about its CPU and memory again. The output should look similar to this:

```
$ cat /proc/cpuinfo
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 85
model name      : Intel(R) Xeon(R) Platinum 8259CL CPU @ 2.50GHz
stepping        : 7
microcode       : 0x500320a
cpu MHz         : 3117.531
cache size      : 36608 KB
[...]

processor       : 1
vendor_id       : GenuineIntel
cpu family      : 6
model           : 85
model name      : Intel(R) Xeon(R) Platinum 8259CL CPU @ 2.50GHz
stepping        : 7
microcode       : 0x500320a
```

```
cpu MHz         : 3100.884
cache size      : 36608 KB
[...]

$ free -m
              total        used        free      shared  buff/cache   available
Mem:           7737         108        7427           0         202        7406
Swap:             0           0           0
```

Your virtual machine can use two CPU cores and offers 7,737 MB of memory, compared to a single CPU core and 965 MB of memory before you increased the VM's size.

Cleaning up

Terminate the EC2 instance named `growingup` of type `m5.large` to stop paying for it as follows:

1. Go to EC2 in the AWS Management Console: https://console.aws.amazon.com/ec2/.
2. Open the list of all virtual machines by selecting Instances from the subnavigation options.
3. Select the running virtual machine by clicking the row in the table.
4. In the Actions menu, select Instance State > Terminate Instance.

3.5 Starting a virtual machine in another data center

AWS offers data centers all over the world. Take the following criteria into account when deciding which region to choose for your cloud infrastructure:

- *Latency*—Which region offers the shortest distance between your users and your infrastructure?
- *Compliance*—Are you allowed to store and process data in that country?
- *Service availability*—AWS does not offer all services in all regions. Are the services you are planning to use available in the region? Check out the service availability region table at https://aws.amazon.com/about-aws/global-infrastructure/regional-product-services/ or https://awsservices.info.
- *Costs*—Service costs vary by region. Which region is the most cost-effective region for your infrastructure?

Let's assume you have customers not just in the United States but in Australia as well. At the moment you are operating EC2 instances only in N. Virginia (US). Customers from Australia complain about long loading times when accessing your website. To make your Australian customers happy, you decide to launch an additional VM in Australia.

Changing a data center is simple. The Management Console always shows the current data center you're working in on the right side of the main navigation menu. So far, you've worked in the data centers located in N. Virginia (US), called `us-east-1`.

To change the data center, click N. Virginia and select Asia Pacific (Sydney) from the menu. Figure 3.21 shows how to jump to the data center in Sydney, also called `ap-southeast-2`.

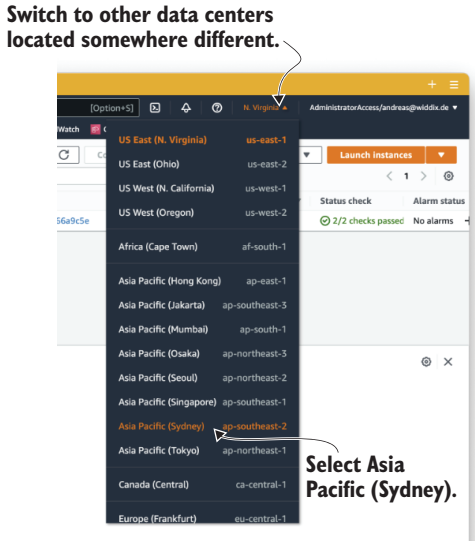


Figure 3.21 Changing the data center's location from N. Virginia to Sydney in the Management Console

AWS groups its data centers into these regions:

- US East, N. Virginia (us-east-1)
- US East, Ohio (us-east-2)
- US West, N. California, us-west-1
- US West, Oregon (us-west-2)
- Africa, Cape Town (af-south-1)
- Asia Pacific, Hong Kong (ap-east-1)
- Asia Pacific, Jakarta (ap-southeast-3)
- Asia Pacific, Mumbai (ap-south-1)
- Asia Pacific, Osaka (ap-northeast-3)
- Asia Pacific, Seoul (ap-northeast-2)
- Asia Pacific, Singapore (ap-southeast-1)
- Asia Pacific, Sydney (ap-southeast-2)
- Asia Pacific, Tokyo (ap-northeast-1)
- Canada, Central (ca-central-1)
- Europe, Frankfurt (eu-central-1)
- Europe, Ireland (eu-west-1)
- Europe, London (eu-west-2)
- Europe, Milan (eu-south-1)
- Europe, Paris (eu-west-3)
- Europe, Stockholm (eu-north-1)
- Middle East, Bahrain (me-south-1)
- South America, São Paulo (sa-east-1)

You can specify the region for most AWS services. The regions are independent of each other; data isn't transferred between regions. Typically, a region is a collection of

three or more data centers located in the same area. Those data centers are well connected to each other and offer the ability to build a highly available infrastructure, as you'll discover later in this book. Some AWS services—like IAM, where you created the `ec2-ssm-core` role, or the CDN and the Domain Name System (DNS) service—act globally on top of these regions and even on top of some additional data centers.

Next, start a virtual machine in a data center in Sydney following these steps:

1. Go to EC2 in the AWS Management Console: https://console.aws.amazon.com/ec2/.
2. Click the Launch Instances button.
3. Enter sydney as the name of the virtual machine.
4. Choose the Amazon Linux 2 AMI.
5. Select the instance type `t2.micro`.
6. Select the option Proceed without a Key Pair.
7. Select the Allow HTTP Traffic from the Internet option.
8. Keep the defaults for storage.
9. Select the IAM instance profile `ec2-ssm-core` under Advanced Details.
10. Launch the instance.

You did it! Your virtual machine is now running in a data center in Sydney. Let's proceed with installing a web server on it. To do so, you have to connect to your virtual machine via Session Manager as you did in the previous sections.

Use the following commands to install and start an Apache web server on your virtual machine:

```
$ sudo yum install httpd -y
```

Next, start the web server and make sure it will get started whenever the machine starts automatically.

```
$ sudo systemctl start httpd
$ sudo systemctl enable httpd
```

To access the default website served by Apache, you need to know the public IPv4 address of your EC2 instance. Get this information by selecting your virtual machine and looking into the details via the Management Console. You can also execute the following command in the terminal of your EC2 instance:

```
$ curl http:/ /169.254.169.254/latest/meta-data/public-ipv4
```

Open http://$PublicIp in your browser. Don't forget to replace `$PublicIp` with the public IPv4 address of your EC2 instance, for example, http://52.54.202.9. A demo website appears.

The public IPv4 address assigned to your EC2 instance is subject to change. For example, when you stop and start your instance, AWS assigns a new public IPv4

address. Therefore, you will learn how to attach a fixed public IP address to the virtual machine in the following section.

3.6 Allocating a public IP address

You've already launched some virtual machines while reading this book. Each VM was connected to a public IP address automatically. But every time you launched or stopped a VM, the public IP address changed. If you want to host an application under a fixed IP address, this won't work. AWS offers a service called *Elastic IPs* for allocating fixed public IP addresses.

Using a fixed public IP address is useful, in case clients aren't able to resolve a DNS name, a firewall rule based on IP addresses is required, or you don't want to update DNS records to avoid the delay until all clients resolve to the new IP address. Therefore, allocate a public IP address and associate with your EC2 instance named `sydney` as follows:

1. Open the Management Console, and go to the EC2 service.
2. Choose Elastic IPs from the submenu. You'll see an overview of public IP addresses allocated by you.
3. Allocate a public IP address by clicking Allocate Elastic IP Address, as shown in figure 3.22.

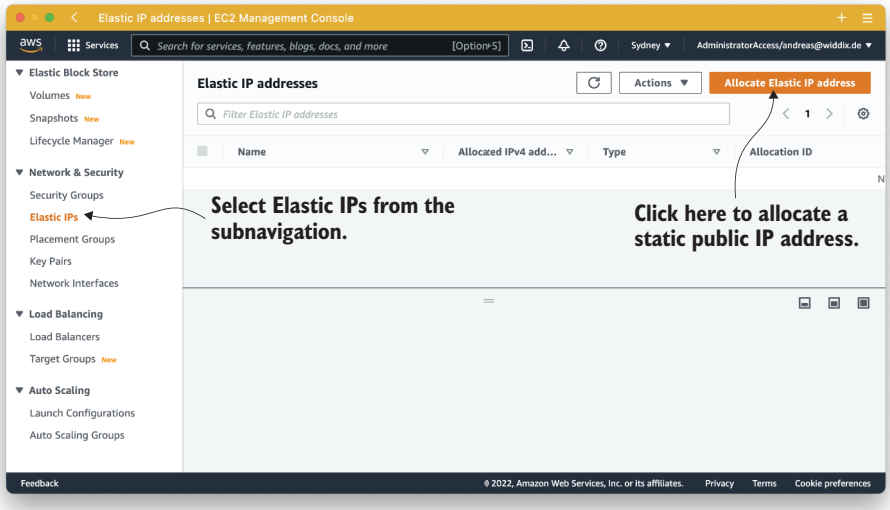


Figure 3.22 An overview of public IP addresses connected to your account in the current region

4. Select Amazon's Pool of IPv4 Addresses, and click the Allocate button, as shown in figure 3.23.
5. To associate the Elastic IP with your EC2 instance, select the public IP address you just allocated, click the Actions button, and select Associate Elastic IP Address.

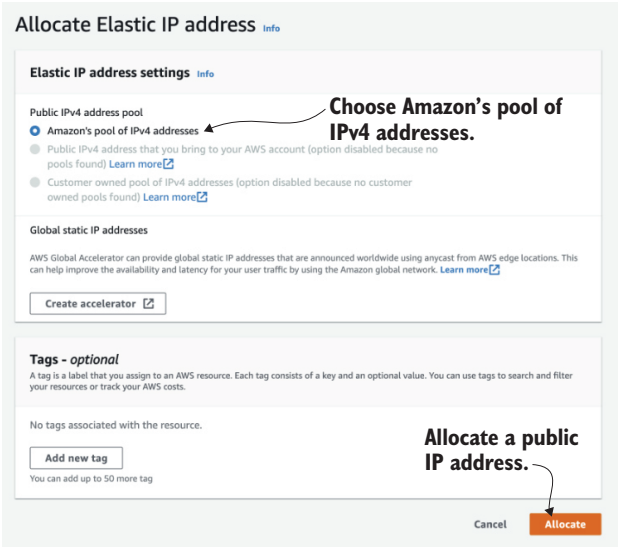


Figure 3.23 Allocating a public IPv4 address

6 Select the resource type Instance, and select your EC2 instance named `sydney` from the dropdown list, as shown in figure 3.24.

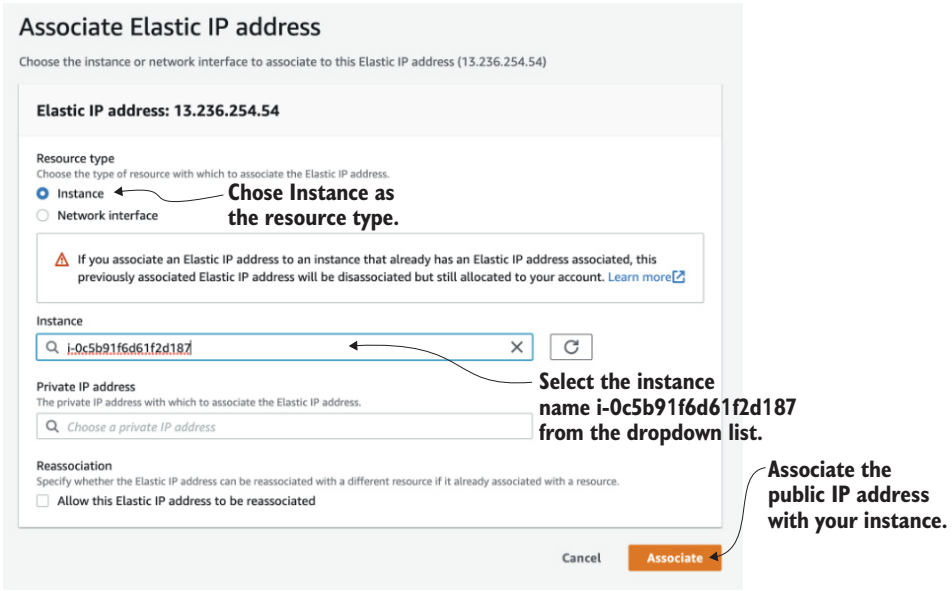


Figure 3.24 Associating an Elastic IP address with an EC2 instance

7 Click the Associate button.

Hurray! Your virtual machine is now accessible through the public IP address you allocated at the beginning of this section. Point your browser to this IP address, and you should see the placeholder page as you did in section 3.5.

Allocating a static public IP address can be useful if you want to make sure the endpoint to your application doesn't change, even if you have to replace the virtual machine behind the scenes. For example, assume that virtual machine A is running and has an associated Elastic IP. The following steps let you replace the virtual machine with a new one without changing the public IP address:

1. Start a new virtual machine B to replace the running virtual machine A.
2. Install and start applications as well as all dependencies on virtual machine B.
3. Disassociate the Elastic IP from virtual machine A, and associate it with virtual machine B.

Requests using the Elastic IP address will now be routed to virtual machine B, with a short interruption while moving the Elastic IP. You can also connect multiple public IP addresses with a virtual machine by using multiple network interfaces, as described in the next section. This method can be useful if you need to host different applications running on the same port, or if you want to use a unique fixed public IP address for different websites.

WARNING IPv4 addresses are scarce. To prevent stockpiling Elastic IP addresses, AWS will charge you for Elastic IP addresses that aren't associated with a virtual machine. You'll clean up the allocated IP address at the end of the next section.

3.7 Adding an additional network interface to a virtual machine

In addition to managing public IP addresses, you can control your virtual machine's network interfaces. It is possible to add multiple network interfaces to a VM and control the private and public IP addresses associated with those network interfaces. Here are some typical use cases for EC2 instances with multiple network interfaces:

- Your web server needs to answer requests by using multiple TLS/SSL certificates, and you can't use the Server Name Indication (SNI) extension due to legacy clients.
- You want to create a management network separated from the application network, and, therefore, your EC2 instance needs to be accessible from two networks. Figure 3.25 illustrates an example.
- Your application requires or recommends the use of multiple network interfaces (e.g., network and security appliances).

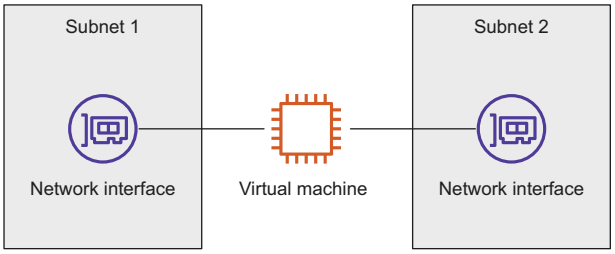


Figure 3.25 A virtual machine with two network interfaces in two different subnets

In the next procedure, you will use an additional network interface to connect a second public IP address to your EC2 instance. Follow these steps to create an additional networking interface for your virtual machine:

1. Open the Management Console, and go to the EC2 service.
2. Select Network Interfaces from the submenu. The default network interface of your virtual machine is shown in the list. Note the subnet ID of this network interface.
3. Click Create Network Interface.
4. Enter 2nd interface as the description, as shown in figure 3.26.
5. Choose the subnet you noted in step 3.
6. Select Auto-Assign Private IPv4 Address.
7. Chose the security group named `launch-wizard-1`.
8. Click Create Network Interface.

After the new network interface's state changes to `Available`, you can attach it to your virtual machine. Select the new 2nd interface network interface, and select Attach from the Actions menu. A dialog opens like the one shown in figure 3.27. Choose the only available Instance ID, and click Attach.

You've attached an additional networking interface to your virtual machine. Next, you'll associate an additional public IP address to the additional networking interface. To do so, note the network interface ID of the 2nd interface shown in the overview–`eni-0865886f80fcc31a9` in our example—and follow these steps:

1. Open the AWS Management Console, and go to the EC2 service.
2. Choose Elastic IPs from the submenu.
3. Click Allocate Elastic IP Address as you did in section 3.6.
4. Select the newly created public IP address, and select Associate Elastic IP Address from the Actions menu.

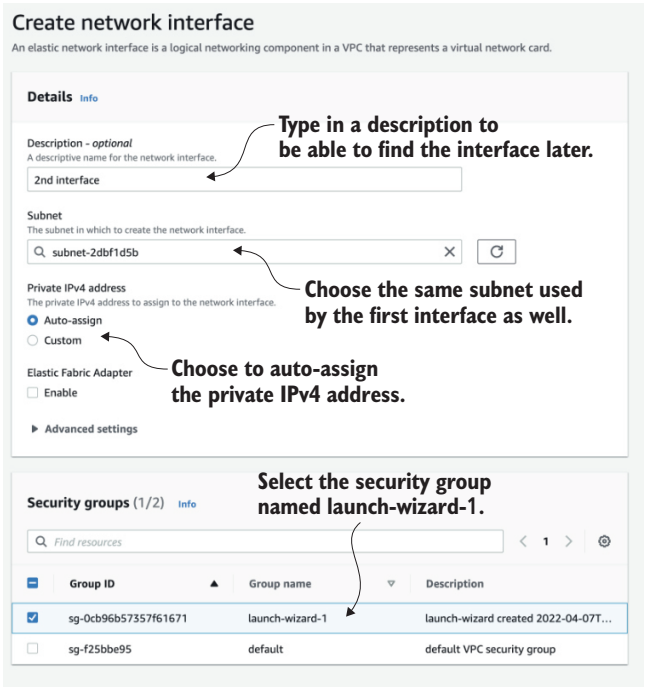


Figure 3.26 Creating an additional networking interface for your virtual machine

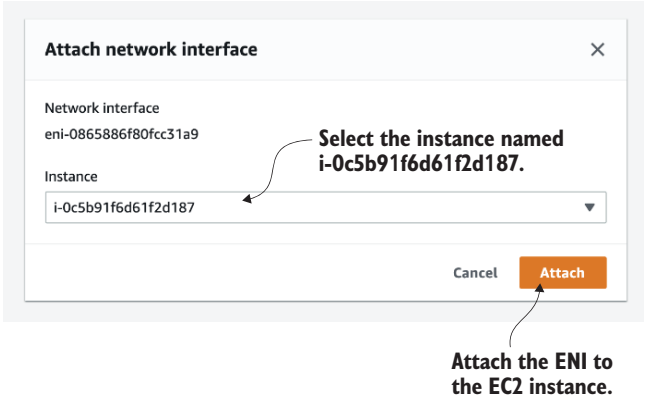


Figure 3.27 Attaching an additional networking interface to your virtual machine

5 Select Network Interface as the resource type, as shown in figure 3.28.
6 Select your 2nd interface's ID.
7 Select the only available private IP of your network interface.
8 Click Associate to finish the process.

Your virtual machine is now reachable under two different public IP addresses. This enables you to serve two different websites, depending on the public IP address. You

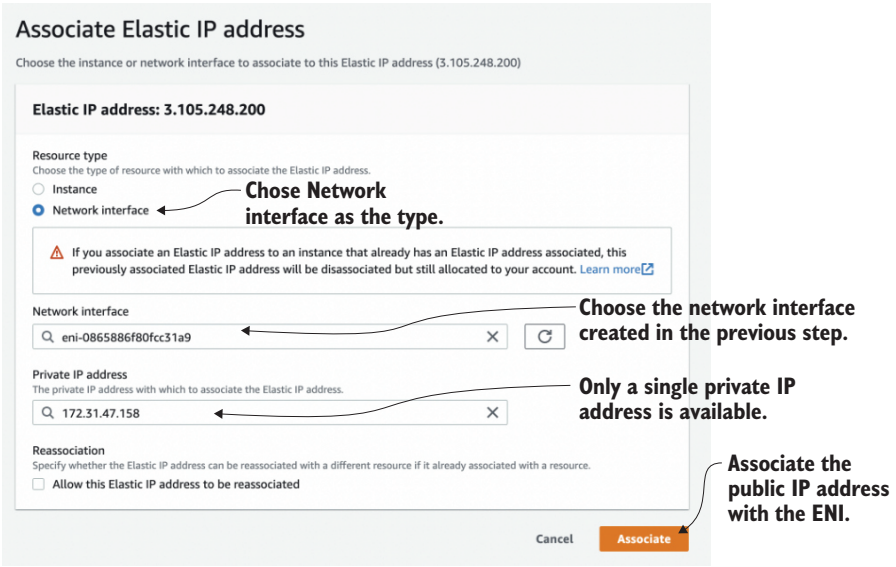


Figure 3.28 Associating a public IP address with the additional networking interface

need to configure the web server to answer requests depending on the public IP address. Use the Session Manager to connect to your EC2 instance named `sydney` and execute `ifconfig` in the terminal, which will output the network configuration of your virtual machine, shown here:

```
$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 9001
   inet 172.31.33.219  netmask 255.255.240.0  broadcast 172.31.47.255
   inet6 fe80::495:5fff:fea6:abde  prefixlen 64  scopeid 0x20<link>
   ether 06:95:5f:a6:ab:de  txqueuelen 1000  (Ethernet)
   RX packets 68382  bytes 80442006 (76.7 MiB)
   RX errors 0  dropped 0  overruns 0  frame 0
   TX packets 35228  bytes 4219870 (4.0 MiB)
   TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 9001
   inet 172.31.47.158  netmask 255.255.240.0  broadcast 172.31.47.255
   inet6 fe80::4a2:8fff:feea:bbba  prefixlen 64  scopeid 0x20<link>
   ether 06:a2:8f:ea:bb:ba  txqueuelen 1000  (Ethernet)
   RX packets 22  bytes 1641 (1.6 KiB)
   RX errors 0  dropped 0  overruns 0  frame 0
   TX packets 33  bytes 2971 (2.9 KiB)
   TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
 [...]
```

The primary network interface uses the private IP address 172.31.33.219.

The secondary network interface uses the private IP address 172.31.47.158.

Each network interface is connected to a private and a public IP address. You'll need to configure the web server to deliver different websites, depending on the IP address.

Your virtual machine doesn't know anything about its public IP address, but you can distinguish the requests based on the private IP addresses.

First you need two websites. Run the following commands on your virtual machine in Sydney via the Session Manager to download two simple placeholder websites:

```
$ sudo -s
$ mkdir /var/www/html/a
$ wget -P /var/www/html/a \
➥ https://raw.githubusercontent.com/AWSinAction/code3/main/chapter03
➥ /a/index.html
$ mkdir /var/www/html/b
$ wget -P /var/www/html/b \
➥ https://raw.githubusercontent.com/AWSinAction/code3/main/chapter03
➥ /b/index.html
```

Next, you need to configure the web server to deliver the websites, depending on which IP address is called. To do so, add a file named a.conf under /etc/httpd/conf.d. The following example uses the editor `nano`:

```
$ nano /etc/httpd/conf.d/a.conf
```

Copy and paste the following file content. Change the IP address from 172.31.x.x to the IP address from the `ifconfig` output for the networking interface `eth0`:

```
<VirtualHost 172.31.x.x:80>
  DocumentRoot /var/www/html/a
</VirtualHost>
```

Press CTRL + X and select y to save the file.

Repeat the same process for a configuration file named b.conf under /etc/httpd/conf.d with the following content. Change the IP address from 172.31.y.y to the IP address from the `ifconfig` output for the networking interface `eth1`:

```
<VirtualHost 172.31.y.y:80>
  DocumentRoot /var/www/html/b
</VirtualHost>
```

To activate the new web server configuration, execute `systemctl restart httpd`.

Next, go to the Elastic IP overview in the Management Console. Copy both public IP addresses, and open them with your web browser. You should get the answer "Hello A!" or "Hello B!," depending on the public IP address you're calling. Thus, you can deliver two different websites, depending on which public IP address the user is calling. Congrats—you're finished!

NOTE You switched to the AWS region in Sydney earlier. Now you need to switch back to the region US East (N. Virginia). You can do so by selecting US East (N. Virginia) from the region chooser in the main navigation menu of the Management Console.

Cleaning up

It's time to clean up your setup:

1. Terminate the virtual machine, and wait until it is terminated.
2. Go to Networking Interfaces, and select and delete the 2nd networking interface.
3. Change to Elastic IPs, and select and release the two public IP addresses by clicking Release Elastic IP Address from the Actions menu.
4. Go to Security Groups, and delete the `launch-wizard-1` security group you created.

That's it. Everything is cleaned up, and you're ready for the next section.

3.8 Optimizing costs for virtual machines

Usually you launch virtual machines *on demand* in the cloud to gain maximum flexibility. AWS calls them on-demand instances, because you can start and stop VMs on demand, whenever you like, and you're billed for every second or hour the machine is running.

Billing unit: Seconds

Most EC2 instances running Windows or Linux (such as Amazon Linux or Ubuntu) are billed per second. The minimum charge per instance is 60 seconds. For example, if you terminate a newly launched instance after 30 seconds, you have to pay for 60 seconds. But if you terminate an instance after 61 seconds, you pay exactly for 61 seconds.

Other instances are billed per hour. See https://aws.amazon.com/ec2/pricing/ for details.

Besides stopping or downsizing EC2 instances, you have two options to reduce costs: *Spot Instances* and *Savings Plans*. Both help to reduce costs but decrease your flexibility. With a Spot Instance, you take advantage of unused capacity in an AWS data center. A Spot Instance comes with a discount of up to 90% compared to on-demand instances. However, AWS might terminate a Spot Instance at anytime when the resources are needed for someone else, so this type of instance is for stateless and fault-tolerant workloads only. With Savings Plans, you commit to a certain amount of resource consumption for one or three years and get a discount in turn. Therefore, Savings Plans are a good fit if you are running workloads with planning security. Table 3.2 summarizes the differences between the pricing options.

Table 3.2 Differences between on-demand instances, Savings Plans, and Spot Instances

	On-demand instances	Savings Plans	Spot Instances
Price	High	Medium	Low
Flexibility	High	Low	Medium
Reliability	High	High	Low
Scenarios	Dynamic workloads (e.g., for a news site) or proof of concept	Predictable and static workloads (e.g., for a business application)	Batch workloads (e.g., for data analytics, media encoding, etc.)

3.8.1 Commit to usage, get a discount

AWS offers the following two types of Savings Plans for EC2:

- Compute Savings Plans do not apply only to EC2 but also to Fargate (Container) and Lambda (Serverless) as well.
- EC2 Instance Savings Plans apply to EC2 instances only.

When purchasing a Compute Savings Plan, you need to specify the following details:

- *Term*—One year or three years
- *Hourly commitment*—In USD
- *Payment option*—All/partial/no upfront

For example, when committing to $1 per hour for one year and paying $8,760.00 upfront, you will get an `m5.large` EC2 instance at a discount of 31% in US East (N. Virginia). As you might have guessed already, the discount between on-demand and Savings Plans differs based on term, payment option, and even region. Find more details at https://aws.amazon.com/savingsplans/compute-pricing/.

An EC2 Instance Savings Plan applies to EC2 instances only. Therefore, it does not provide the flexibility to migrate a workload from virtual machines (EC2) to containers (Fargate). However, EC2 Instance Savings Plans offer a higher discount compared to Compute Savings Plans.

When purchasing an EC2 Instance Savings Plan, you need to specify the following details:

- *Term*—One year or three years
- *Region*—US East (N. Virginia), for example.
- *Instance family*—`m5`, for example
- *Hourly commitment*—In USD
- *Payment Option*—All/partial/no upfront

So the Savings Plan applies only to EC2 instances of a certain instance family in a certain region. Note that you are able to modify the instance family of a Savings Plan later, if needed.

Let's look at the earlier example again. When committing to $1 per hour for one year of `m5` instances running in `us-east-1` and paying $8,760.00 upfront, you will get

an `m5.large` EC2 instance at a discount of 42% in US East (N. Virginia). Compare that to the 31% discount when purchasing a Compute Savings Plan instead.

WARNING Buying a reservation will incur costs for one or three years. That's why we did not add an example for this section.

Think of Savings Plans as a way to optimize your AWS bill. Buying a Savings Plan does not have any effect on your running EC2 instances. Also, an on-demand instance gets billed under the conditions of a Savings Plan automatically. There is no need to restart or modify an EC2 instance.

Capacity Reservations

As mentioned, Savings Plans do not have any effect on EC2 instances, only on billing. In contrast, EC2 Capacity Reservations allow you to reserve capacities in AWS's data centers. A Capacity Reservation comes with an hourly fee. For example, when you reserve the capacity for an `m5.large` instance, you will pay the typical on-demand fee of $0.096 per hour, no matter whether or not an EC2 instance of that type is running. In return, AWS guarantees the capacity to launch an `m5.large` instance at anytime.

With on-demand instances, you might run into the case that AWS cannot fulfill your request of spinning up a virtual machine. This type of situation might happen for rare instance types, during peak hours, or in special cases like outages causing many AWS customers to replace their failed instances.

If you need to guarantee that you are able to spin up an EC2 instance at anytime, consider EC2 Capacity Reservations.

In summary, we highly recommend purchasing Savings Plans for workloads, where predicting the resource consumption for the next year is possible. It is worth noting that it is not necessary to cover 100% of your usage with Savings Plans. Reducing costs is also possible by committing to a smaller fraction of your workload.

3.8.2 Taking advantage of spare compute capacity

AWS is operating data centers at large scale, which results in spare capacity because it has to build and provision data centers and machines in advance to be able to fulfill future needs for on-demand capacity. But spare capacity does not generate revenue. That's why AWS tries to reduce spare capacity within its data centers. One way of doing so is offering Spot Instances.

Here is the deal. With Spot Instances, you get a significant discount on the on-demand price without the need to commit to using capacity in advance. In turn, a Spot Instance will start only when AWS decides that there is enough spare capacity available. In addition, a Spot Instance might be terminated by AWS at any time on short notice.

For example, when writing this on April 8, 2022, the price for an `m5.large` Spot Instance in US East (N. Virginia) is $0.039 per hour. That's a discount of about 60% compared to the $0.096 on-demand price. The spot price for EC2 instances used to be very volatile. Nowadays, it changes much more slowly.

But who is crazy enough to use virtual machines that might be terminated by AWS at any time with notice of only two minutes before the machine gets interrupted? Here are a few scenarios:

- Scanning objects stored on S3 for viruses and malware, by processing tasks stored in a queue
- Converting media files into different formats, where the process orchestrator will restart failed jobs automatically
- Processing parts of the requests for a web application, when the system is designed for fault tolerance

On top of that, we use Spot Instances for test systems where dealing with short outages is worth the cost savings.

As discussed in the previous section, using Savings Plans does not require any changes to your EC2 instances. But to use Spot Instances, you have to launch new EC2 instances and also plan for interrupted virtual machines.

Next, you will launch your first spot instance as follows:

1. Go to EC2 in the AWS Management Console: https://console.aws.amazon.com/ec2/.
2. Select Spot Requests from the subnavigation options.
3. Click the Request Spot Instances button.
4. Select Manually Configure Launch Parameters, as shown in figure 3.29.

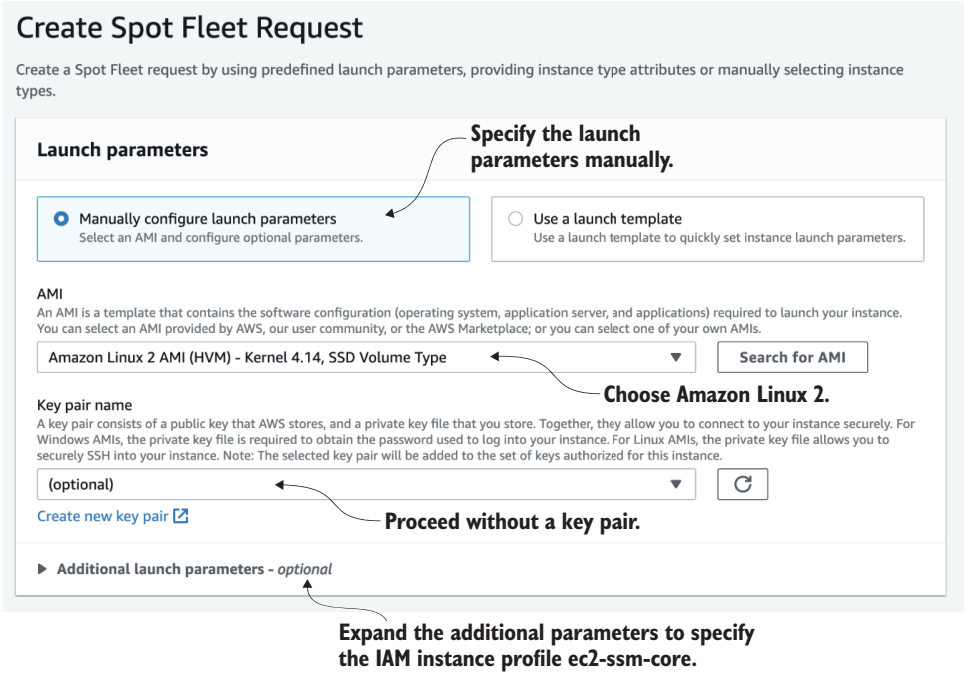


Figure 3.29 Step 1 of creating a spot fleet request

5 Choose an Amazon Linux 2 AMI.
6 Do not configure a key pair; select (optional) instead.
7 Expand the Additional Launch Parameters section.
8 Select the IAM instance profile `ec2-ssm-core` to be able to connect to the Spot Instance using the Session Manager.
9 Keep the defaults for Addition Request Details.
10 Set the total target capacity to one instance, as demonstrated in figure 3.30.

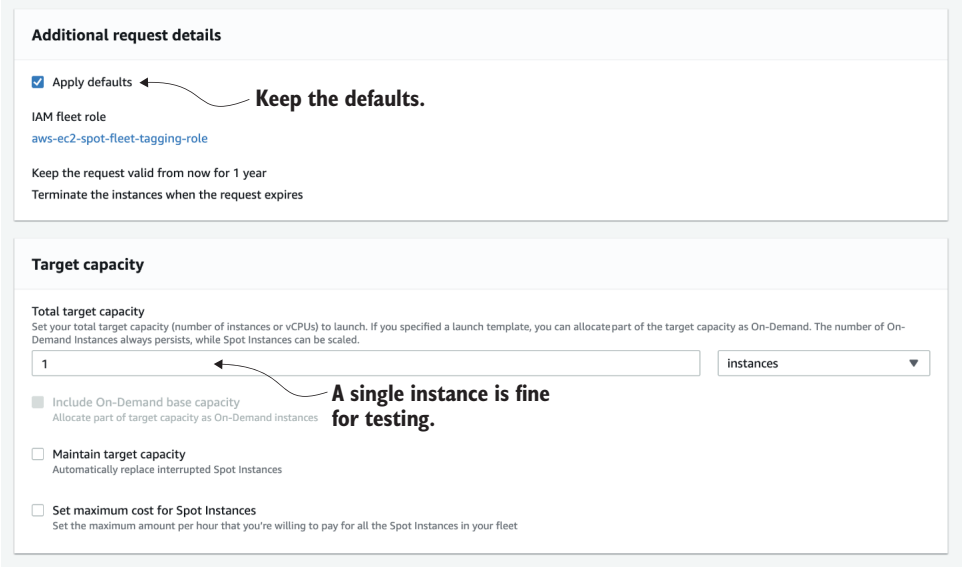


Figure 3.30 Step 2 of creating a spot fleet request

11 Chose Manually Select Instance Types, as shown in figure 3.31.
12 Empty the list of prepopulated instance types by selecting all instance types and clicking the Delete button.
13 Click the Add Instance Types button and select `t2.micro` from the list.
14 Choose the allocation strategy Capacity Optimized to increase the availability of a spot instance, as shown in figure 3.32.
15 Press the Launch button to create a spot fleet request.

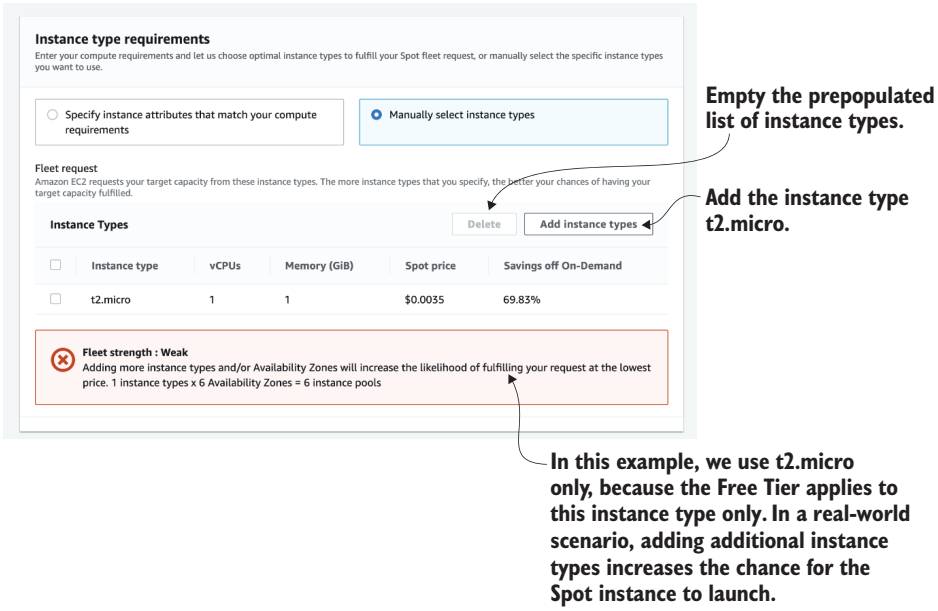


Figure 3.31 Step 3 of creating a spot fleet request

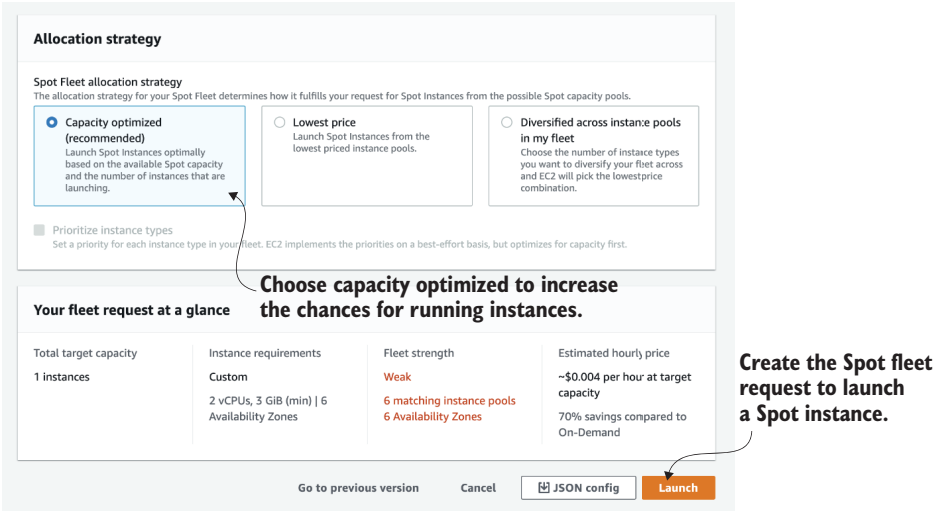


Figure 3.32 Step 4 of creating a spot fleet request

16 Two items appear in the list of spot requests. Wait until both the instance and fleet request reach status `Fulfilled`, as shown in figure 3.33.

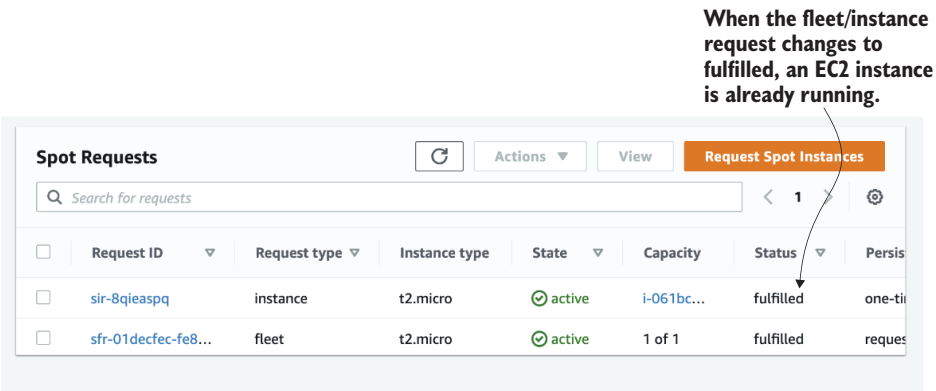


Figure 3.33 Step 5 of creating a spot fleet request

17 Select Instances from the subnavigation options. The list of EC2 instances includes your first Spot Instance.

The Spot Instance is ready for your workload. But be aware that AWS might terminate the Spot Instance at any time to free capacity for other workloads. AWS notifies you two minutes before terminating a Spot Instance.

One way to get notified about an interruption is to ask the EC2 metadata service about planned instance actions. Use the Session Manager to connect with your EC2 instance and execute the following command to send an HTTP request to the EC2 metadata service, which is accessible only from your virtual machine. Most likely, the HTTP request will result in a 404 error, which is a good sign, because that means AWS did not mark your Spot Instance for termination:

```
$ curl http://169.254.169.254/latest/meta-data/spot/instance-action
<?xml version="1.0" encoding="iso-8859-1"?><!DOCTYPE html PUBLIC "-//W3C//DTD
     XHTML 1.0 Transitional//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
 <head>
  <title>404 - Not Found</title>
 </head>
 <body>
  <h1>404 - Not Found</h1>
 </body>
</html>
sh-4.2$
```

If the HTTP request results into something like that shown in the following snippet, your instance will be terminated within two minutes:

```
{"action": "stop", "time": "2022-04-08T12:12:00Z"}
```

In summary, Spot Instances help AWS to reduce spare capacity in their data centers and save us costs. However, you need to make sure that your application tolerates interruptions of Spot Instances, which might cause increased engineering effort.

Cleaning up

Terminate the Spot Instance as follows:

1. Go to the list of Spot Requests.
2. Select the fleet request.
3. Click the Action button and then the Cancel Request button.
4. Make sure to select the Terminate Instances check box and then click the Confirm button.

Summary

- When launching a virtual machine on AWS, you chose between a wide variety of operating systems: Amazon Linux, Ubuntu, Windows, and many more.
- Modifying the size of a virtual machine is simple: stop the virtual machine, modify the instance type—which defines the number of CPUs as well as the amount of memory and storage—and start the virtual machine.
- Using logs and metrics can help you to monitor and debug your virtual machine.
- AWS offers data centers all over the world. Starting VMs in Sydney, Australia, works the same as starting a machine in northern Virginia.
- Choose a data center by considering network latency, legal requirements, and costs, as well as available features.
- Allocating and associating a public IP address to your virtual machine gives you the flexibility to replace a VM without changing the public IP address.
- Committing to a certain compute usage for one or three years reduces the cost of virtual machines through buying Savings Plans.
- Use spare capacity at significant discount but with the risk of AWS terminating your virtual machine in case the capacity is needed elsewhere.

4 Programming your infrastructure: The command line, SDKs, and CloudFormation

This chapter covers

- Starting a virtual machine with the command-line interface (CLI)
- Starting a virtual machine with JavaScript SDK for Node.js
- Understanding the idea of infrastructure as code
- Using CloudFormation to start a virtual machine

Imagine that we want to provide room lighting as a service. To switch off the lights in a room using software, we need a hardware device like a relay connected to the light circuit. This hardware device must have some kind of interface that lets us send commands via software. With a relay and an interface, we can offer room lighting as a service.

To run a virtual machine, we need a lot of hardware and software—power supply, networking gear, host machine, operating system, virtualization layer, and much more. Luckily, AWS runs the hardware and software for us. Even better, we can control all of that with software. AWS provides an *application programming interface* (API) that we can use to control every part of AWS with HTTPS requests. In the end, you can write software that spins up VMs on AWS as well as in-memory caches, data warehouses, and much more.

Calling the HTTP API is a low level task and requires a lot of repetitive work, like authentication, data (de)serialization, and so on. That's why AWS offers tools on top of the HTTP API that are easier to use. Those tools follow:

- *Command-line interface (CLI)*—Use the CLI to call the AWS API from your terminal.
- *Software development kit (SDK)*—SDKs, available for most programming languages, make it easy to call the AWS API from your programming language of choice.
- *AWS CloudFormation*—Templates are used to describe the state of the infrastructure. AWS CloudFormation translates these templates into API calls.

Not all examples are covered by the Free Tier

The examples in this chapter are not all covered by the Free Tier. A special warning message appears when an example incurs costs. As for the other examples, as long as you don't run them longer than a few days, you won't pay anything. Keep in mind that this applies only if you created a fresh AWS account for this book and nothing else is going on in your AWS account. Try to complete the chapter within a few days; you'll clean up your account at the end.

On AWS, you control everything via an API. You interact with AWS by making calls to the REST API using the HTTPS protocol, as figure 4.1 illustrates. Everything is available through the API. You can start a virtual machine with a single API call, create 1 TB of storage, or start a Hadoop cluster over the API. By everything, we really mean *everything*.

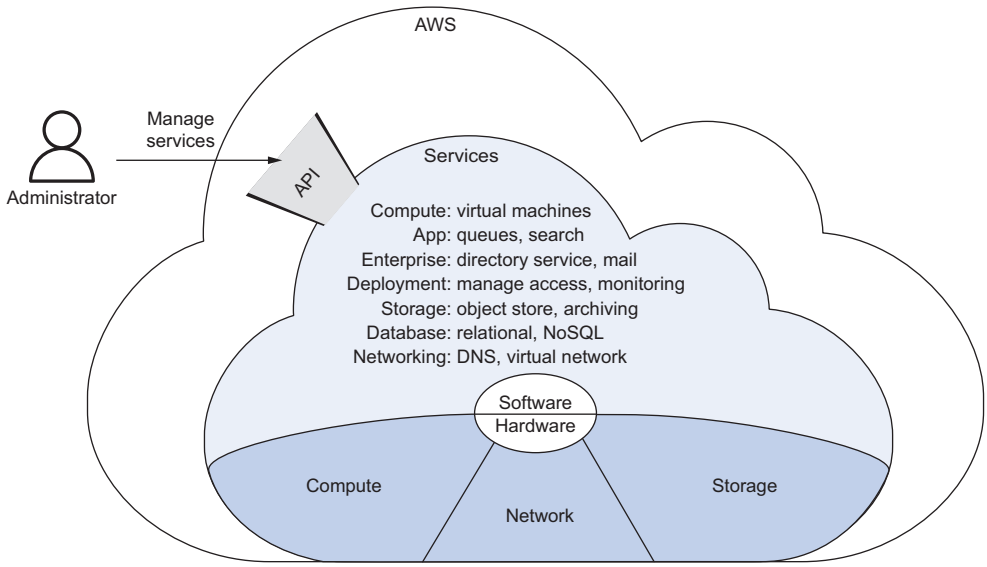
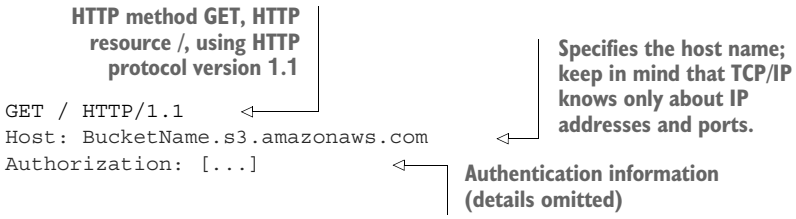


Figure 4.1 The AWS cloud is composed of hardware and software services accessible via an API.

You'll need some time to understand the consequences. By the time you finish this book, you'll ask why the world wasn't always this easy.

Let's look at how the API works. Imagine you uploaded a few files to the object store S3 (you will learn about S3 in chapter 7). Now you want to list all the files in the S3 object store to check whether the upload was successful. Using the raw HTTP API, you send a `GET` request to the API endpoint using the following HTTP protocol:




```
GET / HTTP/1.1
Host: BucketName.s3.amazonaws.com
Authorization: [...]
```

The HTTP response will look like this:


```
HTTP/1.1 200 OK
 x-amz-id-2: [...]
 x-amz-request-id: [...]
 Date: Mon, 09 Feb 2015 10:32:16 GMT
 Content-Type: application/xml

 <?xml version="1.0" encoding="UTF-8"?>
 <ListBucketResult xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
 [...]
 </ListBucketResult>
```

Calling the API directly using plain HTTPS requests is inconvenient. The easy way to talk to AWS is by using the CLI or SDKs, as you'll learn in this chapter. The API, however, is the foundation of all these tools.

4.1 Automation and the DevOps movement

The *DevOps movement* aims to bring software development and operations together. This usually is accomplished in one of two ways:

- Using mixed teams with members from both operations and development. Developers become responsible for operational tasks like being on call. Operators are involved from the beginning of the software development cycle, which helps make the software easier to operate.
- Introducing a new role that closes the gap between developers and operators. This role communicates a lot with both developers and operators and cares about all topics that touch both worlds.

The goal is to develop and deliver software to the customer rapidly without negatively affecting quality. Communication and collaboration between development and operations are, therefore, necessary.

The trend toward automation has helped DevOps culture bloom, because it codifies the cooperation between development and operations. You can do multiple deployments per day only if you automate the whole process. If you commit changes to the repository, the source code is automatically built and tested against your automated tests. If the build passes the tests, it's automatically installed in your testing environment, which triggers some acceptance tests. After those tests have passed, the change is propagated into production. But this isn't the end of the process: now you need to carefully monitor your system and analyze the logs in real time to ensure that the change was successful.

If your infrastructure is automated, you can spawn a new system for every change introduced to the code repository and run the acceptance tests isolated from other changes that were pushed to the repository at the same time. Whenever a change is made to the code, a new system is created (virtual machine, databases, networks, and so on) to run the change in isolation.

4.1.1 Why should you automate?

Why should you automate instead of using the graphical AWS Management Console? A script or a blueprint can be reused and will save you time in the long run. You can build new infrastructures quickly with ready-to-use modules from your former projects, or automate tasks that you will have to do regularly. Automating your infrastructure also enhances your software development process, for example, by using a deployment pipeline.

Another benefit is that a script or blueprint is the most detailed documentation you can imagine (even a computer understands it). If you want to reproduce on Monday what you did last Friday, a script is worth its weight in gold. If you're sick and a coworker needs to take care of your tasks, they will appreciate your blueprints.

You're now going to install and configure the CLI. After that, you can get your hands dirty and start scripting. The AWS CLI is one tool for automating AWS. Read on to learn how it works.

4.2 Using the command-line interface

The AWS CLI is a convenient way to interact with AWS from your terminal. It runs on Linux, macOS, and Windows and provides a unified interface for all AWS services. Unless otherwise specified, the output is by default in JSON format.

4.2.1 Installing the CLI

How you proceed depends on your OS. If you're having difficulty installing the CLI, consult http://mng.bz/AVng for a detailed description of many installation options.

LINUX X86 (64-BIT)

In your terminal, execute the following commands:

```
$ curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" \
➥ -o "awscliv2.zip"
$ unzip awscliv2.zip
$ sudo ./aws/install
```

Verify your AWS CLI installation by running `aws --version` in your terminal. The version should be at least 2.4.0.

LINUX ARM

In your terminal, execute the following commands:

```
$ curl "https://awscli.amazonaws.com/awscli-exe-linux-aarch64.zip" \
➥ -o "awscliv2.zip"
$ unzip awscliv2.zip
$ sudo ./aws/install
```

Verify your AWS CLI installation by running `aws --version` in your terminal. The version should be at least 2.4.0.

MACOS

The following steps guide you through installing the AWS CLI on macOS using the installer:

1. Download the AWS CLI installer at https://awscli.amazonaws.com/AWSCLIV2.pkg.
2. Run the downloaded installer, and install the CLI by going through the installation wizard for **all users**.
3. Verify your AWS CLI installation by running `aws --version` in your terminal. The version should be at least 2.4.0.

WINDOWS

The following steps guide you through installing the AWS CLI on Windows using the MSI Installer:

1. Download the AWS CLI installer at https://awscli.amazonaws.com/AWSCLIV2.msi.
2. Run the downloaded installer, and install the CLI by going through the installation wizard.
3. Run PowerShell as administrator by searching for "PowerShell" in the Start menu and choosing Run as Administrator from its context menu.
4. Type `Set-ExecutionPolicy Unrestricted` into PowerShell, and press Enter to execute the command. This allows you to execute the unsigned PowerShell scripts from our examples.

5 Close the PowerShell window; you no longer need to work as administrator.
6 Run PowerShell by choosing PowerShell from the Start menu.
7 Verify whether the CLI is working by executing `aws --version` in PowerShell. The version should be at least 2.4.0.

WARNING Setting the PowerShell execution policy to `Unrestricted` allows you to run unsigned scripts. There is a risk of running malicious scripts. Use it to run the scripts provided in our examples only. Check about Execution Policies (http://mng.bz/1MK1) to learn more.

4.2.2 Configuring the CLI

To use the CLI, you need to authenticate. Until now, you've been using the root AWS account. This account can do everything, good and bad. It's strongly recommended that you not use the AWS root account (you'll learn more about security in chapter 5), so let's create a new user. To create a new user, use the following steps, which are illustrated in figure 4.2:

1 Open the AWS Management Console at https://console.aws.amazon.com.
2 Click Services and search for IAM.
3 Open the IAM service.

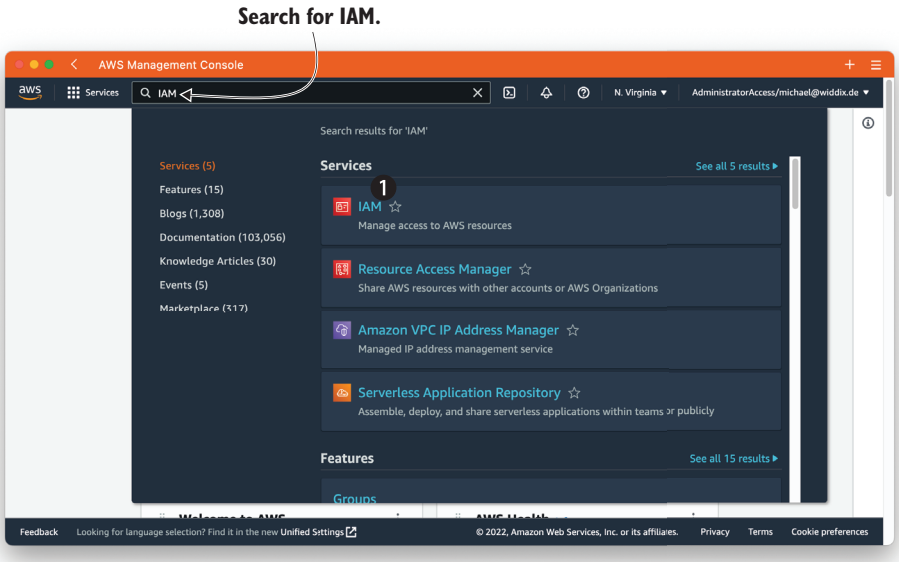


Figure 4.2 Open the IAM service

A page opens as shown in figure 4.3; select Users at the left.

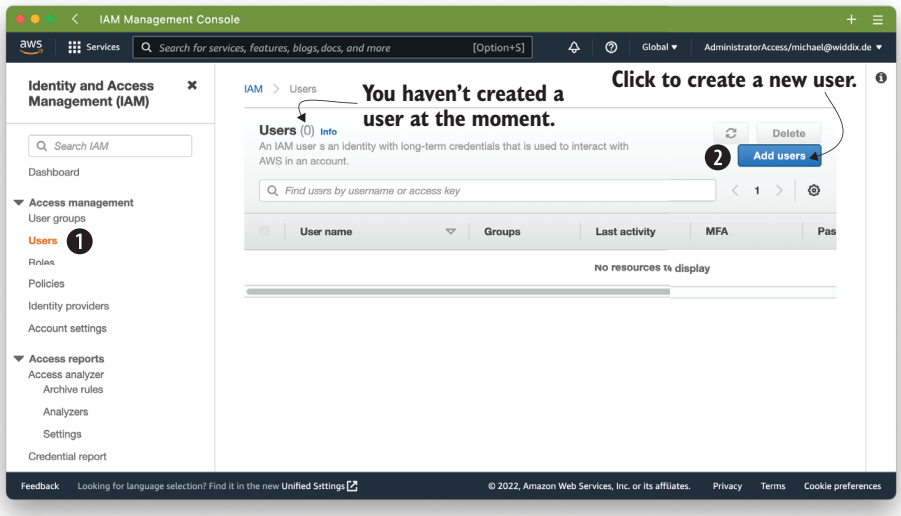


Figure 4.3 IAM users (empty)

Follow these steps to create a user:

1. Click Add Users to open the page shown in figure 4.4.
2. Enter mycli as the user name.

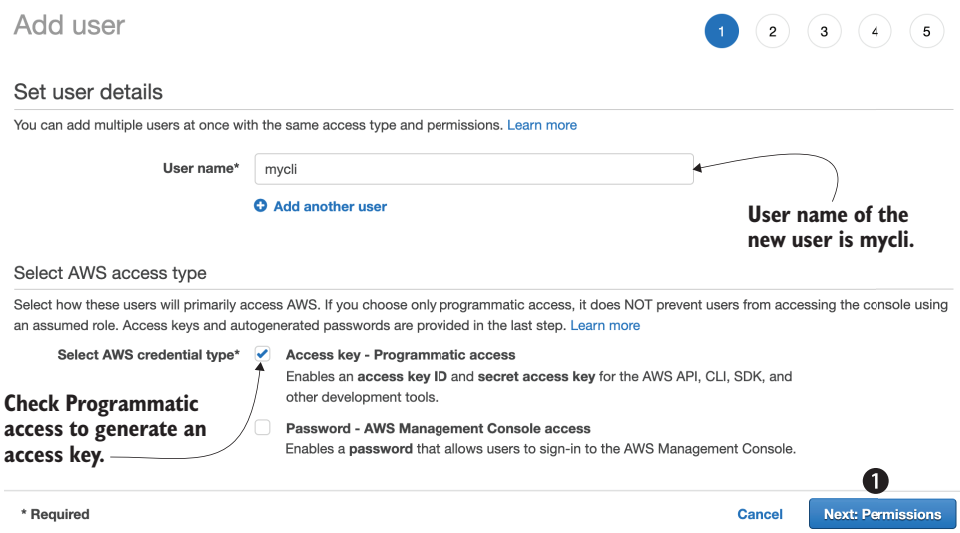


Figure 4.4 Creating an IAM user

3 Under AWS credential type, select Access Key—Programmatic Access.
4 Click the Next: Permissions button.

In the next step, you have to define the permissions for the new user, as shown in figure 4.5:

1 Click Attach Existing Policies Directly.
2 Select the AdministratorAccess policy.
3 Click the Next: Tags button.

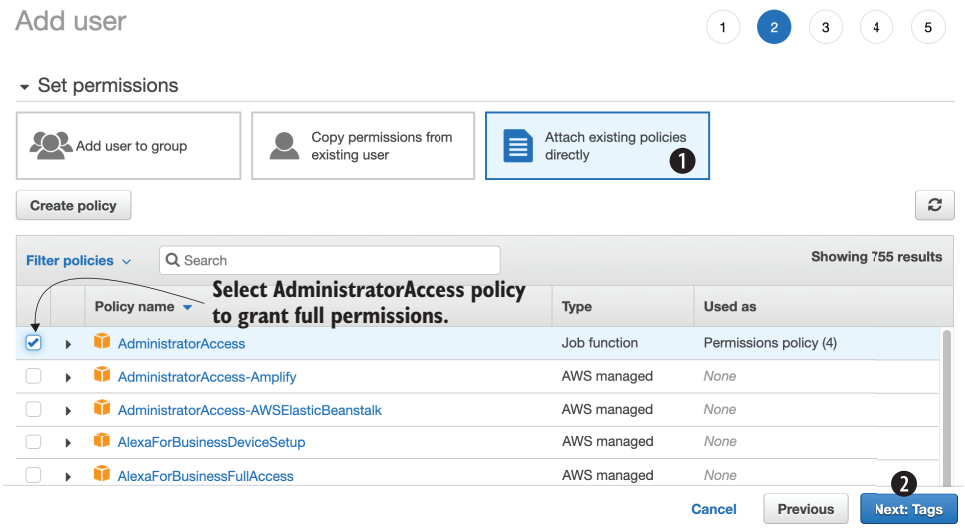


Figure 4.5 Setting permissions for an IAM user

No tags are needed (you learned about tags in chapter 2), so click the Next: Review button.

The review page sums up what you have configured. Click the Create User button to save. Finally, you will see the page shown in figure 4.6. Click the Show link to display the secret value. You now need to copy the credentials to your CLI configuration. Read on to learn how this works.

WARNING Treat the access key ID and secret access key as top secret. Anyone who gets access to these credentials will have administrator access to your AWS account.

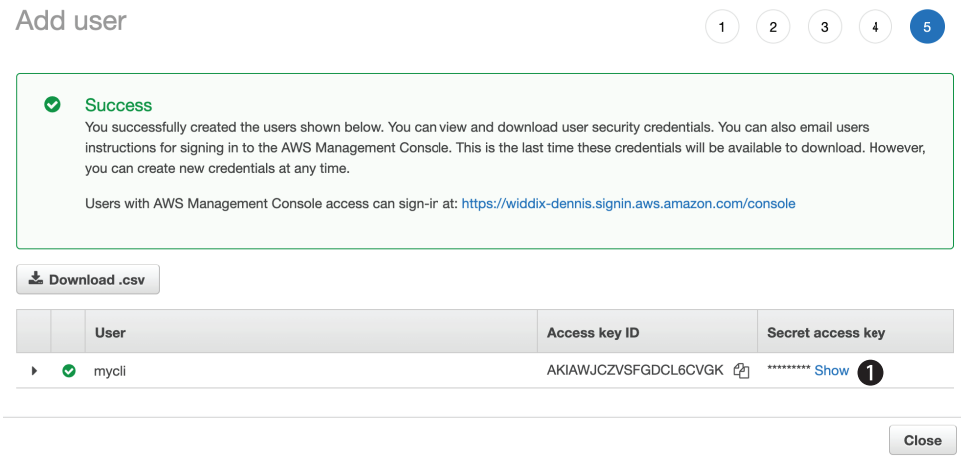


Figure 4.6 Showing the access key of an IAM user

Open the terminal on your computer (PowerShell on Windows or a shell on Linux and macOS; not the AWS Management Console), and run `aws configure`. You're asked for the next four pieces of information:

1. *AWS access key ID*—Copy and paste this value from the Access key ID column (your browser window).
2. *AWS secret access key*—Copy and paste this value from the Secret access key column (your browser window).
3. *Default region name*—Enter us-east-1.
4. *Default output format*—Enter json.

In the end, the terminal should look similar to this:

```
$ aws configure
 AWS Access Key ID [None]:  AKIAIRUR3YLPOSVD7ZCA
 AWS Secret Access Key [None]:
➥ SSKIng7jkAKERpcT3YphX4cD87sBYgWVw2enqBj7
 Default region name [None]: us-east-1
 Default output format [None]: json
```

Your value will be different! Copy it from your browser window.

Your value will be different! Copy it from your browser window.

The CLI is now configured to authenticate as the user `mycli`. Switch back to the browser window, and click Close to finish the user-creation wizard.

It's time to test whether the CLI works. Switch to the terminal window, and enter `aws ec2 describe-regions` to get a list of all available regions, as shown here:

```
$ aws ec2 describe-regions
{
  "Regions": [
    {
      "Endpoint": "ec2.eu-north-1.amazonaws.com",
```

```
      "RegionName": "eu-north-1",
      "OptInStatus": "opt-in-not-required"
    },
    [...]
    {
      "Endpoint": "ec2.us-west-2.amazonaws.com",
      "RegionName": "us-west-2",
      "OptInStatus": "opt-in-not-required"
    }
  ]
}
```

It works! You can now begin to use the CLI.

4.2.3 Using the CLI

Suppose you want to get a list of all running EC2 instances of type `t2.micro` so you can see what is running in your AWS account. Execute the following command in your terminal:

```
$ aws ec2 describe-instances --filters "Name=instance-type,Values=t2.micro"
{
  "Reservations": []
}
```

The list is empty because you haven't created an EC2 instance.

> **Dealing with long output**
>
> By default, the AWS CLI returns all output through your operating system's default pager program (on my system this is `less`). This method is helpful to avoid massive amounts of data being printed to your terminal.
>
> There is one thing to know: **to quit less**, press q and you will be taken back to where you issued the AWS CLI command.

To use the AWS CLI, you need to specify a service and an action. In the previous example, the service is `ec2` and the action is `describe-instances`. You can add options with `--name value` as follows:

```
$ aws <service> <action> [--name value ...]
```

One important feature of the CLI is the help keyword. You can get help at the following three levels of detail:

- `aws help`—Shows all available services
- `aws <service> help`—Shows all actions available for a certain service
- `aws <service> <action> help`—Shows all options available for the particular service action

4.2.4 Automating with the CLI

> **IAM role ec2-ssm-core**
>
> The following script requires an IAM role named `ec2-ssm-core`. You created the role in the section "Creating an IAM role" in chapter 3.

Sometimes you need temporary computing power, like a Linux machine to test something. To do this, you can write a script that creates a virtual machine for you. The script will run on your local computer and does the following:

1. Starts a virtual machine.
2. Helps you to connect to the VM via the Session Manager.
3. Waits for you to finish your temporary usage of the VM.
4. Terminates the virtual machine.

The script is used like this:

```
$ ./virtualmachine.sh
waiting for i-08f21510e8c4f4441 ...
i-08f21510e8c4f4441 is up and running
connect to the instance using Session Manager
https://console.aws.amazon.com/systems-manager/ses[...]441
Press [Enter] key to terminate i-08f21510e8c4f4441 ...
terminating i-08f21510e8c4f4441 ...
done.
```

Waits until started

Waits until terminated

Your virtual machine runs until you press the Enter key. When you press Enter, the virtual machine is terminated. The CLI solution solves the following use cases:

- Creating a virtual machine
- Getting the ID of a virtual machine to connect via the Session Manager
- Terminating the virtual machine if it's no longer needed

Depending on your OS, you'll use either Bash (Linux and macOS) or PowerShell (Windows) to script.

One important feature of the CLI needs explanation before you can begin. The `--query` option uses JMESPath syntax, which is a query language for JSON, to extract data from the result. Doing this can be useful because usually you need only a specific field from the result. The following CLI command gets a list of all AMIs in JSON format:

```
$ aws ec2 describe-images --filters \
➥  "Name=name,Values=amzn2-ami-hvm-2.0.202*-x86_64-gp2"
{
  "Images": [
    {
      "ImageId": "ami-0ce1e3f77cd41957e",
      "State": "available"
```

The filter returns only Amazon Linux 2 images for AMD/Intel.

```
      [...]
    },
    [...]
    {
      "ImageId": "ami-08754599965c30981",
      "State": "available"
    }
  ]
}
```

The output is overwhelming. To start an EC2 instance, you need the `ImageId` without all the other information. With JMESPath, you can extract just that information, like so:

```
$ aws ec2 describe-images --filters \
➥ "Name=name,Values=amzn2-ami-hvm-2.0.202*-x86_64-gp2" \
➥ --query "Images[0].ImageId"
"ami-146e2a7c"
```

Returns the first image ID from the list

The output is wrapped in quotes. This is caused by the default setting of the AWS CLI to output all data in JSON format and JSON strings are enclosed in quotes. To change that, use the `--output text` option as follows to format the output as multiple lines of tab-separated string values. This setting can be useful to pass the output to a text processor, like `grep`, `sed`, or `awk`:

```
$ aws ec2 describe-images --filters \
➥ "Name=name,Values=amzn2-ami-hvm-2.0.202*-x86_64-gp2" \
➥ --query "Images[0].ImageId" --output text
ami-146e2a7c
```

Sets the output format to plain text

With this short introduction to JMESPath, you're well equipped to extract the data you need.

> **Where is the code located?**
>
> All code can be found in the book's code repository on GitHub: https://github.com/AWSinAction/code3. You can download a snapshot of the repository at https://github.com/AWSinAction/code3/archive/main.zip.

Linux and macOS can interpret Bash scripts, whereas Windows prefers PowerShell scripts. So, we've created two versions of the same script.

Linux and macOS

You can find the following listing in /chapter04/virtualmachine.sh in the book's code folder. You can run it either by copying and pasting each line into your terminal or by executing the entire script via the following:

```
chmod +x virtualmachine.sh
./virtualmachine.sh
```

Ensures that the script is executable (only required once)

Listing 4.1 Creating and terminating a virtual machine from the CLI (Bash)

```
#!/bin/bash -e
AMIID="$(aws ec2 describe-images --filters \
➥ "Name=name,Values=amzn2-ami-hvm-2.0.202*-x86_64-gp2" \
➥ --query "Images[0].ImageId" --output text)"
VPCID="$(aws ec2 describe-vpcs --filter "Name=isDefault,
➥ Values=true" \
➥ --query "Vpcs[0].VpcId" --output text)"
SUBNETID="$(aws ec2 describe-subnets --filters \
➥ "Name=vpc-id, Values=$VPCID" --query "Subnets[0].SubnetId" \
➥ --output text)"
INSTANCEID="$(aws ec2 run-instances --image-id "$AMIID" \
➥ --instance-type t2.micro --subnet-id "$SUBNETID" \
➥ --iam-instance-profile "Name=ec2-ssm-core" \
➥ --query "Instances[0].InstanceId" --output text)"
echo "waiting for $INSTANCEID ..."
aws ec2 wait instance-running --instance-ids "$INSTANCEID"
echo "$INSTANCEID is up and running"
echo "connect to the instance using Session Manager"
echo "https://console.aws.amazon.com/systems-manager/
➥ session-manager/$INSTANCEID"
read -r -p "Press [Enter] key to terminate $INSTANCEID ..."
aws ec2 terminate-instances --instance-ids
➥ "$INSTANCEID" > /dev/null
echo "terminating $INSTANCEID ..."
aws ec2 wait instance-terminated --instance-ids
➥ "$INSTANCEID"
echo "done."
```

-e makes Bash abort if a command fails.

Gets the ID of Amazon Linux AMI

Gets the default VPC ID

Gets the default subnet ID

Creates and start the virtual machine

Waits until the virtual machine is started

Terminates the virtual machine

Waits until the virtual machine is terminated

Cleaning up

Make sure you terminate the virtual machine before you go on by pressing the Enter key!

WINDOWS

You can find the following listing in /chapter04/virtualmachine.ps1 in the book's code folder. Right-click the virtualmachine.ps1 file, and select Run with PowerShell to execute the script.

Listing 4.2 Creating and terminating a virtual machine from the CLI (PowerShell)

```
$ErrorActionPreference = "Stop"

$AMIID=aws ec2 describe-images --filters \
➥ "Name=name,Values=amzn2-ami-hvm-2.0.202*-x86_64-gp2" \
➥ --query "Images[0].ImageId" --output text
$VPCID=aws ec2 describe-vpcs --filter
➥ "Name=isDefault, Values=true" \
```

Aborts if the command fails

Gets the ID of Amazon Linux AMI

Gets the default VPC ID

```
➥ --query "Vpcs[0].VpcId" --output text
$SUBNETID=aws ec2 describe-subnets \
➥ --filters "Name=vpc-id, Values=$VPCID" --query "Subnets[0].SubnetId" \
➥ --output text
$INSTANCEID=aws ec2 run-instances --image-id $AMIID \
➥ --instance-type t2.micro --subnet-id $SUBNETID \
➥ --iam-instance-profile "Name=ec2-ssm-core" \
➥ --query "Instances[0].InstanceId" --output text
Write-Host "waiting for $INSTANCEID ..."
aws ec2 wait instance-running --instance-ids $INSTANCEID
Write-Host "$INSTANCEID is up and running"
Write-Host "connect to the instance using Session Manager"
Write-Host "https://console.aws.amazon.com/systems-manager
➥ /session-manager/$INSTANCEID"
Write-Host "Press [Enter] key to terminate $INSTANCEID ..."
Read-Host
aws ec2 terminate-instances --instance-ids $INSTANCEID
Write-Host "terminating $INSTANCEID ..."
aws ec2 wait instance-terminated --instance-ids $INSTANCEID
Write-Host "done."
```

Gets the default subnet ID

Creates and starts the virtual machine

Waits until the virtual machine is started

Terminates the virtual machine

Waits until the virtual machine is terminated

Cleaning up

Make sure you terminate the virtual machine before you go on.

The limitations of the CLI solution follow:

- It can handle only one virtual machine at a time.
- There is a different version for Windows than for Linux and macOS.
- It's a command-line application, not a graphical one.

Next, learn how to improve the CLI solution using the AWS SDK.

4.3 Programming with the SDK

AWS offers SDKs for the following programming languages and platforms:

- JavaScript/Node.js
- Java
- .NET
- PHP
- Python
- Ruby
- Go
- C++

An AWS SDK is a convenient way to make calls to the AWS API from your favorite programming language. The SDK takes care of things like authentication, retry on error,

HTTPS communication, and XML or JSON (de)serialization. You're free to choose the SDK for your favorite language, but in this book, most examples are written in JavaScript and run in the Node.js runtime environment.

> **Installing and getting started with Node.js**
>
> Node.js is a platform for executing JavaScript in an event-driven environment so you can easily build network applications. To install Node.js, visit https://nodejs.org and download the package that fits your OS. All examples in this book are tested with Node.js 14.
>
> After Node.js is installed, you can verify that everything works by typing node –version into your terminal. Your terminal should respond with something similar to `v14.*`. Now you're ready to run JavaScript examples, like the Node Control Center for AWS. Your Node.js installation comes with a important tool called npm, which is the package manager for Node.js. Verify the installation by running `npm --version` in your terminal.
>
> To run a JavaScript script in Node.js, enter `node script.js` in your terminal, where `script.js` is the name of the script file. We use Node.js in this book because it's easy to install, it requires no IDE, and the syntax is familiar to most programmers.
>
> Don't be confused by the terms *JavaScript* and *Node.js*. If you want to be precise, JavaScript is the language and Node.js is the runtime environment. But don't expect anybody to make that distinction. Node.js is also called node.
>
> Do you want to get started with Node.js? We recommend *Node.js in Action* (second edition) by Alex Young et al. (Manning, 2017), or the video course *Node.js in Motion* by PJ Evans (Manning, 2018).

To understand how the AWS SDK for Node.js (JavaScript) works, let's create a Node.js (JavaScript) application that controls EC2 instances via the AWS SDK. The name of the application is *Node Control Center for AWS* or *nodecc* for short.

4.3.1 Controlling virtual machines with SDK: nodecc

The *Node Control Center for AWS* (nodecc) is for managing multiple temporary EC2 instances using a text UI written in JavaScript. nodecc offers the following features:

- It can handle multiple virtual machines.
- It's written in JavaScript and runs in Node.js, so it's portable across platforms.
- It uses a textual UI.

Figure 4.7 shows what nodecc looks like.

> **IAM role ec2-ssm-core**
>
> nodecc requires an IAM role named `ec2-ssm-core`. You created the role in the "Creating an IAM Role" section in chapter 3.

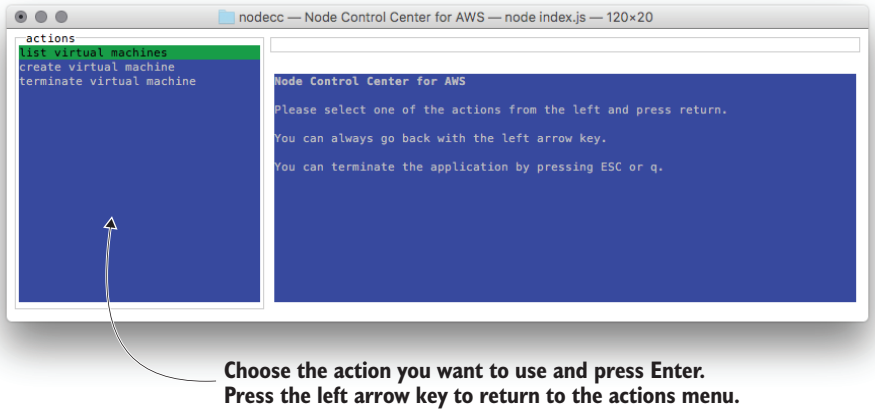


Figure 4.7 nodecc: Start screen

You can find the nodecc application at /chapter04/nodecc/ in the book's code folder. Switch to that directory, and run `npm install` in your terminal to install all needed dependencies. To start nodecc, run `node index.js`. You can always go back by pressing the left arrow key. You can quit the application by pressing Esc or q. The SDK uses the same settings you created for the CLI, so you're using the `mycli` user when running nodecc.

4.3.2 How nodecc creates a virtual machine

Before you can do anything with nodecc, you need at least one virtual machine. To start a virtual machine, choose the AMI you want, as figure 4.8 shows.

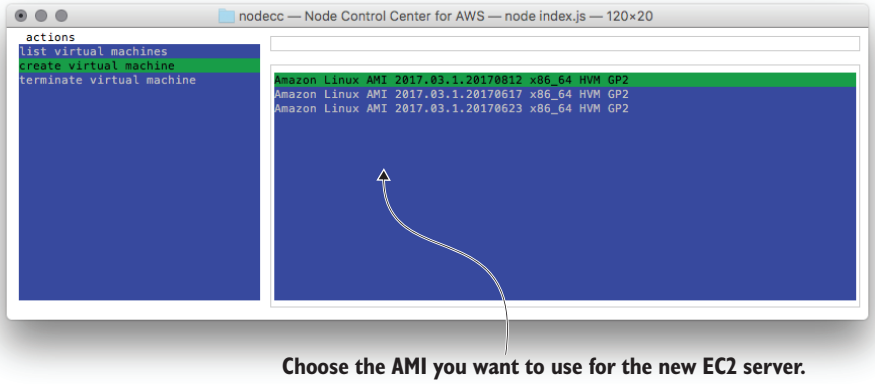


Figure 4.8 nodecc: Creating a virtual machine (step 1 of 2)

The code shown in the next listing, which fetches the list of the available AMIs, is located at lib/listAMIs.js.

Listing 4.3 Fetching the list of available AMIs: /lib/listAMIs.js

```
const AWS = require('aws-sdk');
const ec2 = new AWS.EC2({region: 'us-east-1'});

module.exports = (cb) => {
  ec2.describeImages({
    Filters: [{
      Name: 'name',
      Values: ['amzn2-ami-hvm-2.0.202*-x86_64-gp2']
    }]
  }, (err, data) => {
    if (err) {
      cb(err);
    } else {
      const amiIds = data.Images.map(image => image.ImageId);
      const descriptions = data.Images.map(image => image.Description);
      cb(null, {amiIds: amiIds, descriptions: descriptions});
    }
  });
};
```

API call to list AMIs

require is used to load modules.

Creates an EC2 client

module.exports makes this function available to users of the listAMIs module.

In case of failure, err is set.

Otherwise, data contains all AMIs.

The code is structured in such a way that each action is implemented in the lib folder. The next step to create a virtual machine is to choose which subnet the virtual machine should be started in. You haven't learned about subnets yet, so for now, select one randomly; see figure 4.9.

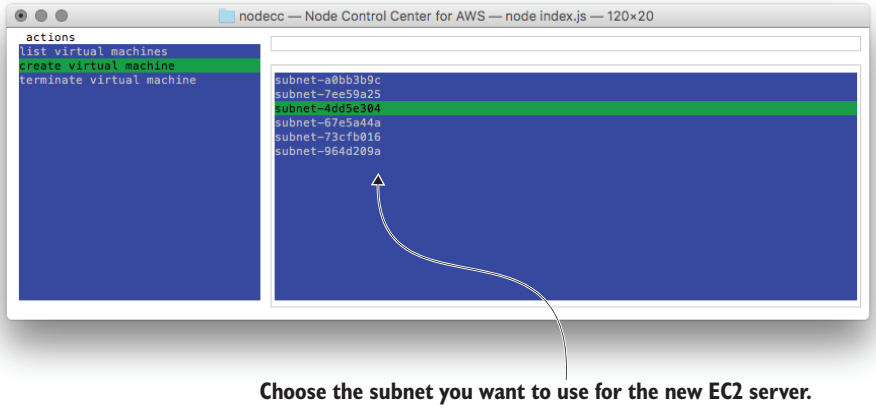


Figure 4.9 nodecc: Creating a virtual machine (step 2 of 2)

The corresponding script, shown in the next listing, is located at lib/listSubnets.js.

Listing 4.4 Fetching the list of available default subnets: /lib/listSubnets.js

```
const AWS = /* ... */;
const ec2 = /* ... */;

module.exports = (cb) => {
  ec2.describeVpcs({
    Filters: [{
      Name: 'isDefault',
      Values: ['true']
    }]
  }, (err, data) => {
    if (err) {
      cb(err);
    } else {
      const vpcId = data.Vpcs[0].VpcId;
      ec2.describeSubnets({
        Filters: [{
          Name: 'vpc-id',
          Values: [vpcId]
        }]
      }, (err, data) => {
        if (err) {
          cb(err);
        } else {
          const subnetIds = data.Subnets.map(subnet => subnet.SubnetId);
          cb(null, subnetIds);
        }
      });
    }
  });
};
```

API call to list VPCs

The filter selects default VPCs only.

There can be only one default VPC.

API call to list subnets

The filter selects subnets from the default VPC only.

After you select the subnet, the virtual machine is created by lib/createVM.js, and you see a starting screen, as shown in the following listing.

Listing 4.5 Launching an EC2 instance: /lib/createVM.js

```
module.exports = (amiId, subnetId, cb) => {
  ec2.runInstances({
    IamInstanceProfile: {
      Name: 'ec2-ssm-core'
    },
    ImageId: amiId,
    MinCount: 1,
    MaxCount: 1,
    InstanceType: 't2.micro',
    SubnetId: subnetId
  }, cb);
};
```

The IAM role and instance profile ec2-ssm-core was created in chapter 3.

API call to launch an EC2 instance

Passes the selected AMI

Launches one instance…

…of type t2.micro to stay in the Free Tier

Passes the selected subnet

Now it's time to find out some details of the newly created virtual machine. Use the left arrow key to switch to the navigation section.

4.3.3 How nodecc lists virtual machines and shows virtual machine details

One important use case that nodecc must support is showing the details of a VM that you can use to connect via the Session Manager. Because nodecc handles multiple virtual machines, the first step is to select a VM, as shown in figure 4.10.

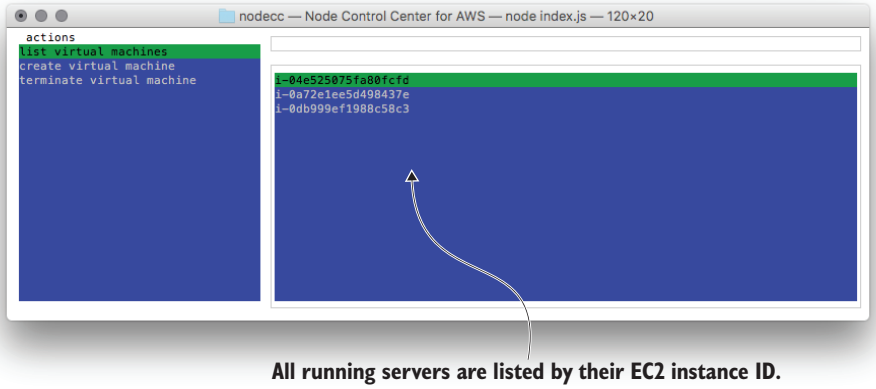


Figure 4.10 nodecc: Listing virtual machines

Look at lib/listVMs.js, shown in the next listing, to see how a list of virtual machines can be retrieved with the AWS SDK.

Listing 4.6 Listing EC2 instances: /lib/listVMs.js

```
module.exports = (cb) => {
  ec2.describeInstances({
    Filters: [{
      Name: 'instance-state-name',
      Values: ['pending', 'running']
    }],
    MaxResults: 10
  }, (err, data) => {
    if (err) {
      cb(err);
    } else {
      const instanceIds = data.Reservations
        .map(r => r.Instances.map(i => i.InstanceId))
        .flat();
      cb(null, instanceIds);
    }
  });
};
```

API call to list EC2 instances

The filter selects starting and running instances only.

Shows at most 10 instances

After you select the VM, you can display its details; see figure 4.11. You could use the `LaunchTime` to find old EC2 instances. You can also connect to an instance using the Session Manager. Press the left arrow key to switch back to the navigation section.

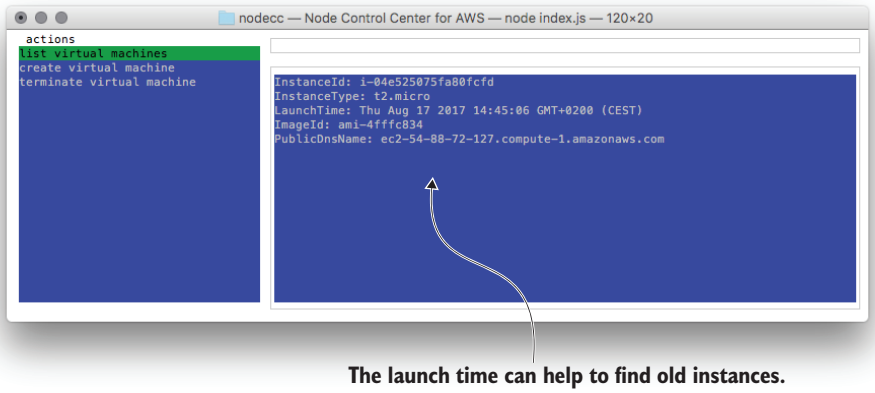


Figure 4.11 nodecc: Showing virtual machine details

4.3.4 How nodecc terminates a virtual machine

To terminate a virtual machine, you first have to select it. To list the virtual machines, use lib/listVMs.js again. After you select the VM, lib/terminateVM.js takes care of termination, as shown in the following code snippet.

Listing 4.7 Terminating an EC2 instance: /lib/terminateVM.js

```
module.exports = (instanceId, cb) => {
  ec2.terminateInstances({
    InstanceIds: [instanceId]
  }, cb);
};
```

API call to list EC2 instances

Passes the selected instance ID

That's nodecc: a text UI program for controlling temporary EC2 instances. Take some time to think about what you could create by using your favorite language and the AWS SDK. Chances are high that you might come up with a new business idea!

Cleaning up

Make sure you terminate all started virtual machines before you go on! You learned to use nodecc to terminate instances in the previous section.

The hard parts about using the SDK follow:

- The SDK (or, better, Node.js) follows an imperative approach. You provide all instructions, step by step, in the right order, to get the job done.

- You have to deal with dependencies (e.g., wait for the instance to be running before connecting to it).
- There is no easy way to update the instances that are running with new settings (e.g., change the instance type).

It's time to enter a new world by leaving the imperative world and entering the declarative world.

4.4 Infrastructure as Code

Infrastructure as Code is the idea of using a high-level programming language to control infrastructures. Infrastructure can be any AWS resource, like a network topology, a load balancer, a DNS entry, and so on. In software development, tools like automated tests, code repositories, and build servers increase the quality of software engineering. If your infrastructure is defined as code, then you can apply these types of software development tools to your infrastructure and improve its quality.

> **WARNING** Don't mix up the terms Infrastructure as Code and *Infrastructure as a Service* (IaaS)! IaaS means renting virtual machines, storage, and network with a pay-per-use pricing model.

4.4.1 Inventing an infrastructure language: JIML

For the purposes of learning the concepts behind Infrastructure as Code, let's invent a new language to describe infrastructure: JSON Infrastructure Markup Language (JIML). Figure 4.12 shows the infrastructure that will be created in the end.

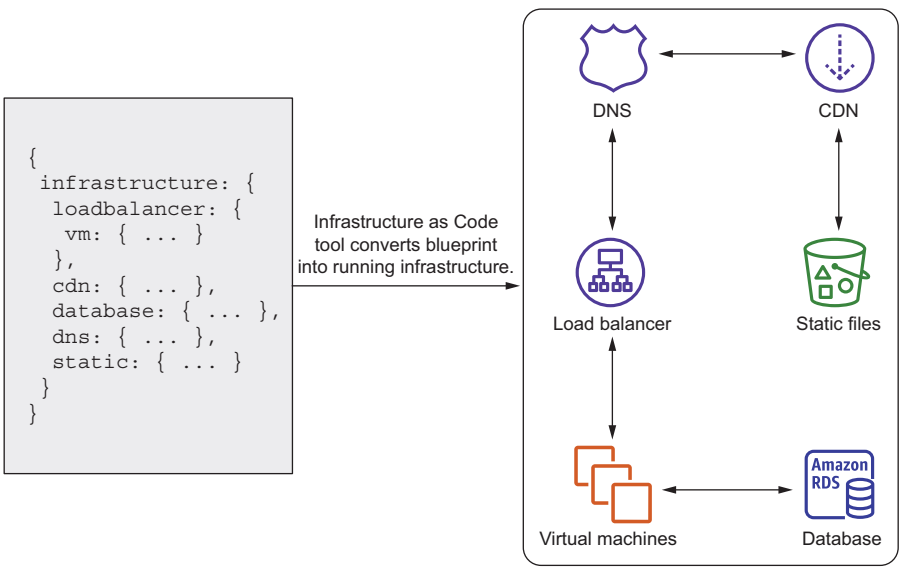


Figure 4.12 From JIML blueprint to infrastructure: Infrastructure automation

The infrastructure consists of the following:

- Load balancer (LB)
- Virtual machines (VMs)
- Database (DB)
- DNS entry
- Content delivery network (CDN)
- Storage for static files

To reduce problems with syntax, let's say JIML is based on JSON. The following JIML code creates the infrastructure shown in figure 4.12. The `$` indicates a reference to an ID.

Listing 4.8 Infrastructure description in JIML

```
{
  "region": "us-east-1",
  "resources": [{
    "type": "loadbalancer",
    "id": "LB",
    "config": {
      "virtualmachines": 2,
      "virtualmachine": {
        "cpu": 2,
        "ram": 4,
        "os": "ubuntu"
      }
    },
    "waitFor": "$DB"
  }, {
    "type": "cdn",
    "id": "CDN",
    "config": {
      "defaultSource": "$LB",
      "sources": [{
        "path": "/static/*",
        "source": "$BUCKET"
      }]
    }
  }, {
    "type": "database",
    "id": "DB",
    "config": {
      "password": "***",
      "engine": "MySQL"
    }
  }, {
    "type": "dns",
    "config": {
      "from": "www.mydomain.com",
      "to": "$CDN"
    }
  }, {
    "type": "bucket",
```

A load balancer is needed.

Needs two VMs

VMs are Ubuntu Linux (4 GB memory, 2 cores).

LB can be created only if the database is ready.

A CDN is used that caches requests to the LB or fetches static assets (images, CSS files, ...) from a bucket.

Data is stored within a MySQL database.

A DNS entry points to the CDN.

A bucket is used to store static assets (images, CSS files, ...).

```
    "id": "BUCKET"
  }]
}
```

How can we turn this JSON into AWS API calls?

1. Parse the JSON input.
2. The JIML tool creates a dependency graph by connecting the resources with their dependencies.
3. The JIML tool traverses the dependency graph from the bottom (leaves) to the top (root) and a linear flow of commands. The commands are expressed in a pseudo language.
4. The commands in pseudo language are translated into AWS API calls by the JIML runtime.

The AWS API calls have to be made based on the resources defined in the blueprint. In particular, you must send the AWS API calls in the correct order. Let's look at the dependency graph created by the JIML tool, shown in figure 4.13.

You traverse the dependency graph in figure 4.13 from bottom to top and from left to right. The nodes at the bottom have no children and therefore no dependencies: `DB`, `VM`, and `bucket`.

The `LB` node depends on the `DB` node and the two `VM` nodes. The `CDN` node depends on the `LB` and the `bucket` node. Finally, the `DNS` node depends on the `CDN` node.

The JIML tool turns the dependency graph into a linear flow of commands using pseudo language, as shown in the next listing. The pseudo language represents the steps needed to create all the resources in the correct order. The nodes are easy to create because they have no dependencies, so they're created first.

Listing 4.9 Linear flow of commands in pseudo language

```
$DB = database create {"password": "***",
➥ "engine": "MySQL"}
$VM1 = virtualmachine create {"cpu": 2, "ram": 4,
➥ "os": "ubuntu"}
$VM2 = virtualmachine create {"cpu": 2, "ram": 4,
➥ "os": "ubuntu"}
$BUCKET = bucket create {}

await [$DB, $VM1, $VM2]
$LB = loadbalancer create {"virtualmachines":
➥ [$VM1, $VM2]}

await [$LB, $BUCKET]
$CDN = cdn create {...}

await $CDN
$DNS = dns create {...}

await $DNS
```

Creates the database

Creates the virtual machine

Creates the bucket

Waits for the dependencies

Creates the load balancer

Creates the CDN

Creates the DNS entry

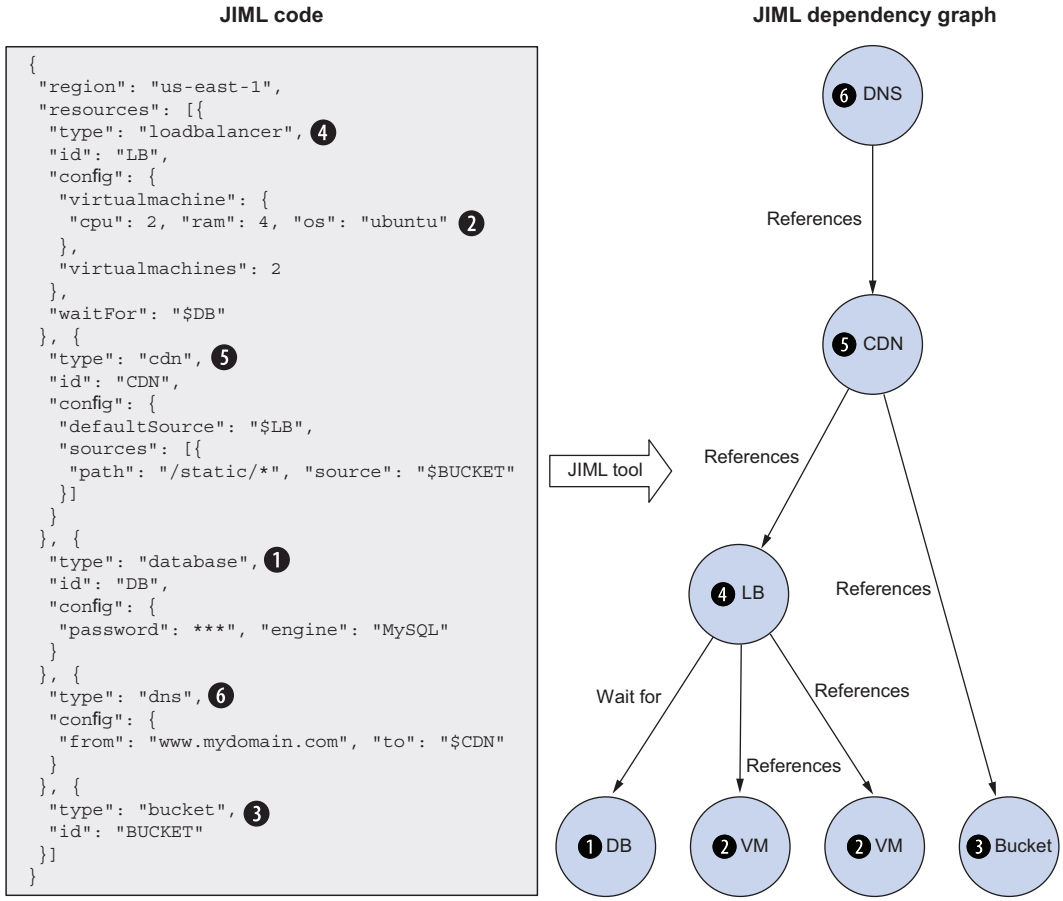


Figure 4.13 The JIML tool figures out the order in which resources need to be created.

We'll skip the last step—translating the commands from the pseudo language into AWS API calls. You've already learned everything you need to know about infrastructure as code: it's all about dependencies. Let's apply those newly learned ideas in practice.

4.5 Using AWS CloudFormation to start a virtual machine

In the previous section, we created JIML to introduce the concept of infrastructure as code. Luckily, AWS already offers a tool that does much better than our JIML: *AWS CloudFormation*.

> **NOTE** When using CloudFormation, what we have been referring to as a blueprint so far is actually referred to as a CloudFormation template.

A *template* is a description of your infrastructure, written in JSON or YAML, that can be interpreted by CloudFormation. The idea of describing something rather than

listing the necessary actions is called a *declarative approach*. Declarative means you tell CloudFormation how your infrastructure should look. You aren't telling CloudFormation what actions are needed to create that infrastructure, and you don't specify the sequence in which the actions need to be executed. The benefits of CloudFormation follow:

- *It's a consistent way to describe infrastructure on AWS*. If you use scripts to create your infrastructure, everyone will solve the same problem differently. This is a hurdle for new developers and operators trying to understand what the code is doing. CloudFormation templates provide a clear language for defining infrastructure.
- *It handles dependencies*. Ever tried to register a web server with a load balancer that wasn't yet available? When you first start trying to automate infrastructure creation, you'll miss a lot of dependencies. Trust us: never try to set up complex infrastructure using scripts. You'll end up in dependency hell!
- *It's reproducible*. Is your test environment an exact copy of your production environment? Using CloudFormation, you can create two identical infrastructures. It is also possible to apply changes to both the test and production environment.
- *It's customizable*. You can insert custom parameters into CloudFormation to customize your templates as you wish.
- *It's testable*. If you can create your architecture from a template, it's testable. Just start a new infrastructure, run your tests, and shut it down again.
- *It's updatable*. CloudFormation supports updates to your infrastructure. It will figure out the parts of the template that have changed and apply those changes as smoothly as possible to your infrastructure.
- *It minimizes human failure*. CloudFormation doesn't get tired—even at 3 a.m.
- *It's the documentation for your infrastructure*. A CloudFormation template is a JSON or YAML document. You can treat it as code and use a version control system like Git to keep track of the changes.
- *It's free*. Using CloudFormation comes at no additional charge. If you subscribe to an AWS support plan, you also get support for CloudFormation.

We think CloudFormation is the most powerful tool available for managing infrastructure on AWS.

4.5.1 Anatomy of a CloudFormation template

A basic CloudFormation template consists of the following five parts:

1. *Format version*—The latest template format version is 2010-09-09, and this is currently the only valid value. Specify this version; the default is to use the latest version, which will cause problems if new versions are introduced in the future.
2. *Description*—What is this template about?

3. *Parameters*—Parameters are used to customize a template with values, for example, domain name, customer ID, and database password.
4. *Resources*—A resource is the smallest block you can describe. Examples are a virtual machine, a load balancer, or an Elastic IP address.
5. *Outputs*—An output is comparable to a parameter, but the other way around. An output returns details about a resource created by the template, for example, the public name of an EC2 instance.

A basic template looks like the following listing.

Listing 4.10 CloudFormation template structure

```
---                                                    Start of a document
AWSTemplateFormatVersion: '2010-09-09'                 The only valid version
Description: 'CloudFormation template structure'       What is this template about?
Parameters:
  # [...]                                              Defines the parameters
Resources
  # [...]                                              Defines the resources
Outputs:
  # [...]                                              Defines the outputs
```

Let's take a closer look at parameters, resources, and outputs.

Format version and description

The only valid `AWSTemplateFormatVersion` value at the moment is 2010-09-09. Always specify the format version. If you don't, CloudFormation will use whatever version is the latest one. As mentioned earlier, this means if a new format version is introduced in the future, you'll end up using the wrong format and get into serious trouble.

`Description` isn't mandatory, but we encourage you to take some time to document what the template is about. A meaningful description will help you in the future to remember what the template is for. It will also help your coworkers.

Parameters

A parameter has at least a name and a type. We encourage you to add a description as well, as shown in the next listing.

Listing 4.11 CloudFormation parameter structure

```
Parameters:
  Demo:                                                You can choose the name of the parameter.
    Type: Number                                       This parameter represents a number.
    Description: 'This parameter is for demonstration' Description of the parameter
```

Valid types are listed in table 4.1.

Table 4.1 CloudFormation parameter types

Type	Description
`String` `CommaDelimitedList`	A string or a list of strings separated by commas
`Number` `List<Number>`	An integer or float, or a list of integers or floats
`AWS::EC2::AvailabilityZone::Name` `List<AWS::EC2::AvailabilityZone::Name>`	An Availability Zone, such as us-west-2a, or a list of Availability Zones
`AWS::EC2::Image::Id` `List<AWS::EC2::Image::Id>`	An AMI ID or a list of AMIs
`AWS::EC2::Instance::Id` `List<AWS::EC2::Instance::Id>`	An EC2 instance ID or a list of EC2 instance IDs
`AWS::EC2::KeyPair::KeyName`	An Amazon EC2 key-pair name
`AWS::EC2::SecurityGroup::Id` `List<AWS::EC2::SecurityGroup::Id>`	A security group ID or a list of security group IDs
`AWS::EC2::Subnet::Id` `List<AWS::EC2::Subnet::Id>`	A subnet ID or a list of subnet IDs
`AWS::EC2::Volume::Id` `List<AWS::EC2::Volume::Id>`	An EBS volume ID (network attached storage) or a list of EBS volume IDs
`AWS::EC2::VPC::Id` `List<AWS::EC2::VPC::Id>`	A VPC ID (virtual private cloud) or a list of VPC IDs
`AWS::Route53::HostedZone::Id` `List<AWS::Route53::HostedZone::Id>`	A DNS zone ID or a list of DNS zone IDs

In addition to using the `Type` and `Description` properties, you can enhance a parameter with the properties listed in table 4.2.

Table 4.2 CloudFormation parameter properties

Property	Description	Example
`Default`	A default value for the parameter	`Default: 'm5.large'`
`NoEcho`	Hides the parameter value in all graphical tools (useful for secrets)	`NoEcho: true`
`AllowedValues`	Specifies possible values for the parameter	`AllowedValues: [1, 2, 3]`
`AllowedPattern`	More generic than `AllowedValues` because it uses a regular expression	`AllowedPattern: '[a-zA-Z0-9]*'` allows only a–z, A–Z, and 0–9 with any length
`MinLength, MaxLength`	Defines how long a parameter can be	`MinLength: 12`

Table 4.2 CloudFormation parameter properties *(continued)*

Property	Description	Example
`MinValue, MaxValue`	Used in combination with the `Number` type to define lower and upper bounds	`MaxValue: 10`
`ConstraintDescription`	A string that explains the constraint when the constraint is violated	`ConstraintDescription: 'Maximum value is 10.'`

A parameter section of a CloudFormation template could look like this:

```
Parameters:
   KeyName:
     Description: 'Key Pair name'
     Type: 'AWS::EC2::KeyPair::KeyName'
   NumberOfVirtualMachines:
     Description: 'How many virtual machine do you like?'
     Type: Number
     Default: 1
     MinValue: 1
     MaxValue: 5
   WordPressVersion:
     Description: 'Which version of WordPress do you want?'
     Type: String
     AllowedValues: ['4.1.1', '4.0.1']
```

Only key-pair names are allowed.

The default is one virtual machine.

Prevents massive costs with an upper bound

Restricted to certain versions

Now you should have a better feel for parameters. If you want to know everything about them, see the documentation at http://mng.bz/ZpB5, or follow along in the book and learn by doing.

Resources

A resource has at least a name, a type, and some properties, as shown in the next listing.

Listing 4.12 CloudFormation resources structure

```
Resources:
  VM:
    Type: 'AWS::EC2::Instance'
    Properties:
      # [...]
```

Name or logical ID of the resource that you can choose

The resource of type AWS::EC2::Instances defines a virtual machine.

Properties needed for the type of resource

When defining resources, you need to know about the type and that type's properties. In this book, you'll get to know a lot of resource types and their respective properties. An example of a single EC2 instance appears in the following code snippet.

Listing 4.13 CloudFormation EC2 instance resource

```
Resources:
  VM:
    Type: 'AWS::EC2::Instance'
```

Name or logical ID of the resource that you can choose

The resource of type AWS::EC2::Instances defines a virtual machine.

```
Properties:
  ImageId: 'ami-6057e21a'
  InstanceType: 't2.micro'
  SecurityGroupIds:
  - 'sg-123456'
  SubnetId: 'subnet-123456'
```

The AMI defines the operating system of the vm.

The instance type defines the number of vCPUs, memory, and more.

You'll learn about this in chapter 5.

Now you've described the virtual machine, but how can you output its public name?

OUTPUTS

A CloudFormation template's output includes at least a name (like parameters and resources) and a value, but we encourage you to add a description as well, as illustrated in the next listing. You can use outputs to pass data from within your template to the outside.

Listing 4.14 CloudFormation outputs structure

```
Outputs:
  NameOfOutput:
    Value: '1'
    Description: 'This output is always 1'
```

Name of the output that you can choose

Value of the output

Static outputs like this one aren't very useful. You'll mostly use values that reference the name of a resource or an attribute of a resource, like its public name. If you see `!Ref NameOfSomething`, think of it as a placeholder for what is referenced by the name. A `!GetAtt 'NameOfSomething.AttributeOfSomething'`, shown in the next code, is similar to a ref but you select a specific attribute of the referenced resource.

Listing 4.15 CloudFormation outputs example

```
Outputs:
  ID:
    Value: !Ref Server
    Description: 'ID of the EC2 instance'
  PublicName:
    Value: !GetAtt 'Server.PublicDnsName'
    Description: 'Public name of the EC2 instance'
```

References the EC2 instance

Gets the attribute PublicDnsName of the EC2 instance

You'll get to know the most important attributes of `!GetAtt` while reading this book.

Now that we've taken a brief look at the core parts of a CloudFormation template, it's time to make one of your own.

4.5.2 Creating your first template

How do you create a CloudFormation template? Different options are available, as shown here:

- Use a text editor or IDE to write a template from scratch.
- Start with a template from a public repository that offers a default implementation, and adapt it to your needs.
- Use a template provided by your vendor.

AWS and its partners offer CloudFormation templates for deploying popular solutions: AWS Partner Solutions at https://aws.amazon.com/quickstart/. Furthermore, we have open sourced the templates we are using in our day-to-day work on GitHub: https://github.com/widdix/aws-cf-templates.

Suppose you've been asked to provide a VM for a developer team. After a few months, the team realizes the VM needs more CPU power, because the usage pattern has changed. You can handle that request with the CLI and the SDK, but before the instance type can be changed, you must stop the EC2 instance. The process follows:

1. Stop the instance.
2. Wait for the instance to stop.
3. Change the instance type.
4. Start the instance.
5. Wait for the instance to start.

A declarative approach like that used by CloudFormation is simpler: just change the `InstanceType` property and update the template. `InstanceType` can be passed to the template via a parameter, as shown in the next listing. That's it! You can begin creating the template.

Listing 4.16 A template to create an EC2 instance with CloudFormation

```
---
AWSTemplateFormatVersion: '2010-09-09'
Description: 'AWS in Action: chapter 4'
Parameters:
  VPC:
    Type: 'AWS::EC2::VPC::Id'
  Subnet:
    Type: 'AWS::EC2::Subnet::Id'
  InstanceType:
    Description: 'Select one of the possible instance types'
    Type: String
    Default: 't2.micro'
    AllowedValues: ['t2.micro', 't2.small', 't2.medium']
Resources:
  SecurityGroup:
    Type: 'AWS::EC2::SecurityGroup'
    Properties:
      # [...]
  VM:
    Type: 'AWS::EC2::Instance'
    Properties:
      ImageId: 'ami-061ac2e015473fbe2'
      InstanceType: !Ref InstanceType
      IamInstanceProfile: 'ec2-ssm-core'
      SecurityGroupIds: [!Ref SecurityGroup]
      SubnetId: !Ref Subnet
Outputs:
  InstanceId:
    Value: !Ref VM
    Description: 'Instance id (connect via Session Manager)'
```

You'll learn about this in chapter 5.

The user defines the instance type.

Defines a minimal EC2 instance

By referencing the security group, an implicit dependency is declared.

Returns the ID of the EC2 instance

You can find the full code for the template at /chapter04/virtualmachine.yaml in the book's code folder. Please don't worry about VPC, subnets, and security groups at the moment; you'll get to know them in chapter 5.

> **Where is the template located?**
> You can find the template on GitHub. You can download a snapshot of the repository at https://github.com/AWSinAction/code3/archive/main.zip. The file we're talking about is named chapter04/virtualmachine.yaml. On S3, the same file is located at https://s3.amazonaws.com/awsinaction-code3/chapter04/virtualmachine.yaml.

If you create an infrastructure from a template, CloudFormation calls it a *stack*. You can think of *template* versus *stack* much like *class* versus *object*. The template exists only once, whereas many stacks can be created from the same template. The following steps will guide you through creating your stack:

1. Open the AWS Management Console at https://console.aws.amazon.com.
2. Click Services and search for CloudFormation.
3. Open the CloudFormation service.
4. Select Stacks at the left.

Figure 4.14 shows the empty list of CloudFormation stacks.

1. Click Create Stack and select With New Resources (Standard) to start a four-step wizard.
2. Select Template Is Ready.

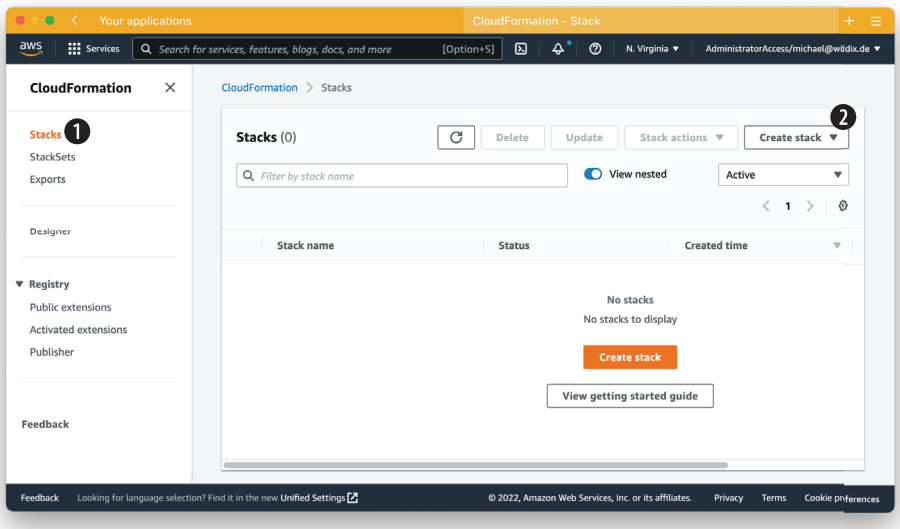


Figure 4.14 Overview of CloudFormation stacks

3. Select Amazon S3 URL as your template source, and enter the value https://s3.amazonaws.com/awsinaction-code3/chapter04/virtualmachine.yaml, as shown in figure 4.15.
4. Continue by clicking Next.

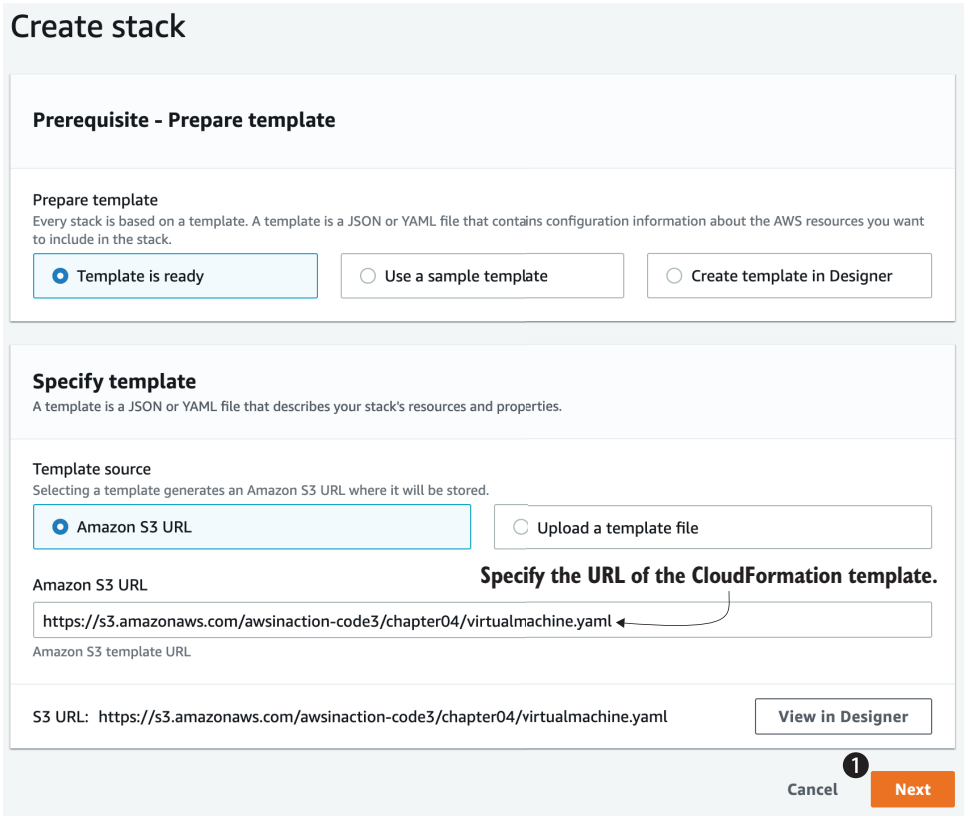


Figure 4.15 Creating a CloudFormation stack: Selecting a template (step 1 of 4)

In the second step, you define the stack name and parameters. Give the stack a name like myvm, and fill out the parameter values as follows:

1. `InstanceType`—Select t2.micro.
2. `Subnet`—Select the first value in the drop-down list. You'll learn about subnets later.
3. `VPC`—Select the first value in the drop-down list. You'll learn about VPCs later.

Figure 4.16 shows the second step. Click Next after you've chosen a value for the stack name and every parameter, to proceed with the next step.

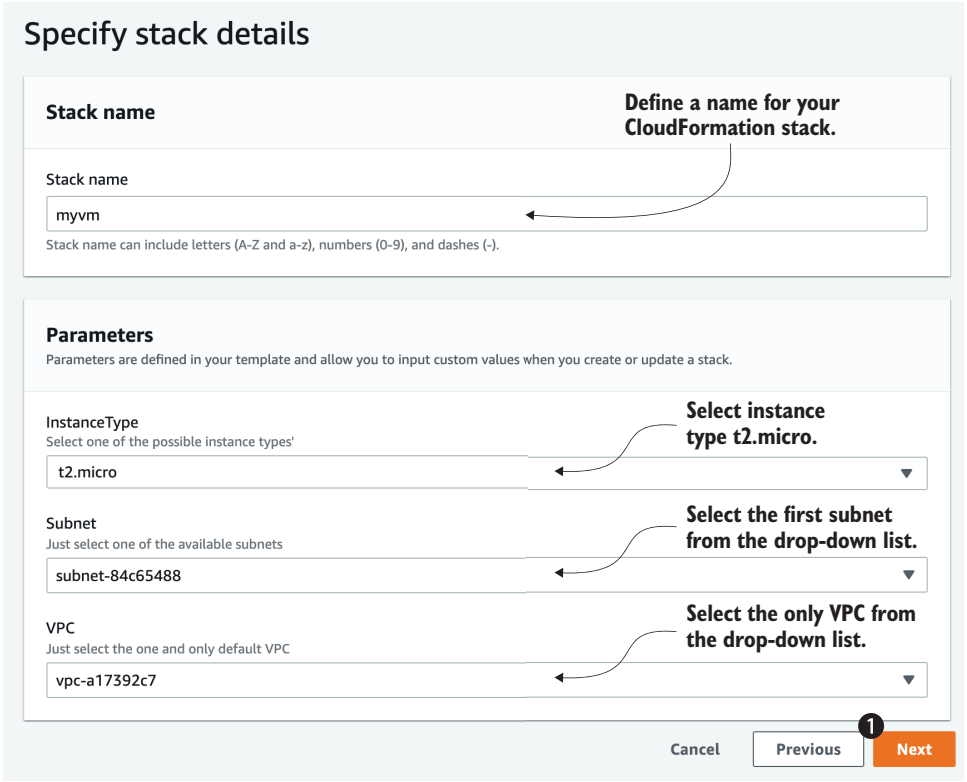


Figure 4.16 Creating a CloudFormation stack: Defining parameters (step 2 of 4)

In the third step, you can define optional tags for the stack and advanced configuration. You can skip this step at this point in the book, because you will not use any advanced features for now. All resources created by the stack will be tagged by CloudFormation by default. Click Next to go to the last step.

The fourth step displays a summary of the stack. At the bottom of the page, you are asked to Acknowledge the Creation of IAM Resources as figure 4.17 shows. You can safely allow CloudFormation to create IAM resources for now. You will learn more about them in chapter 5.

Click Create Stack. CloudFormation now starts to create the stack. If the process is successful, you'll see the screen shown in figure 4.18. As long as status is `CREATE_IN_PROGRESS`, you need to be patient and click the reload button from time to time. When the status is `CREATE_COMPLETE`, click the Outputs tab to see the ID of the EC2 instance. Double-check the instance type in the EC2 Management Console.

Your stack is now created. But that's not the end of the story. CloudFormation supports updating and deleting stacks as well.

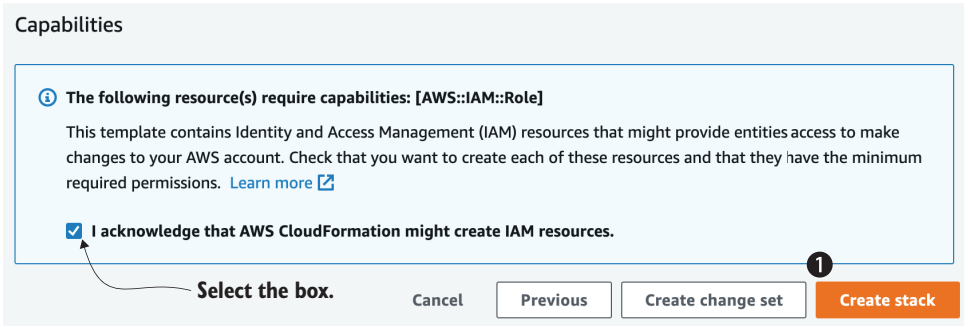


Figure 4.17 Creating a CloudFormation stack: Summary (step 4 of 4)

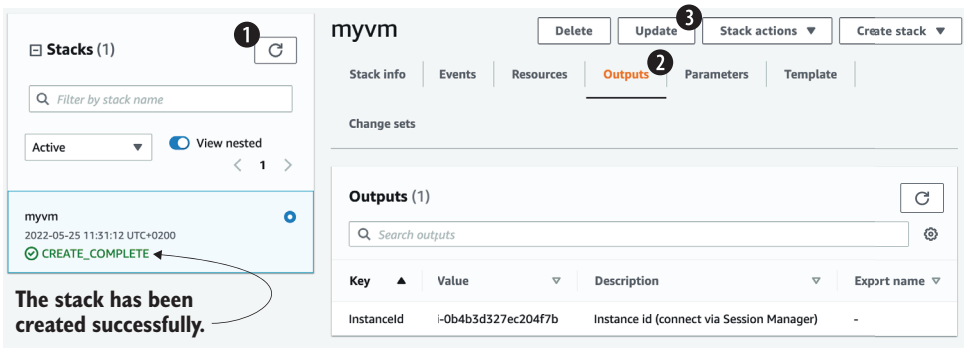


Figure 4.18 The CloudFormation stack has been created

4.5.3 Updating infrastructure using CloudFormation

It's time to test whether we can modify the instance type. Click the Update button. The wizard that starts is similar to the one you used during stack creation. Figure 4.19 shows the first step of the wizard. Select Use Current Template, and proceed with the next step by clicking the Next button.

In step 2, you need to change the `InstanceType` parameter value: choose `t2.small` to double or `t2.medium` to quadruple the computing power of your EC2 instance.

> **WARNING** Starting a virtual machine with instance type t2.small or t2.medium will incur charges. See https://aws.amazon.com/ec2/pricing/ to find out the current hourly price.

Step 3 is about sophisticated options during the update of the stack. You don't need any of these features now, so skip the step by clicking Next. Step 4 is a summary; acknowledge the creation of IAM resources and click Update Stack. The stack now has the status `UPDATE_IN_PROGRESS`. If you are quickly jumping to the EC2 Management

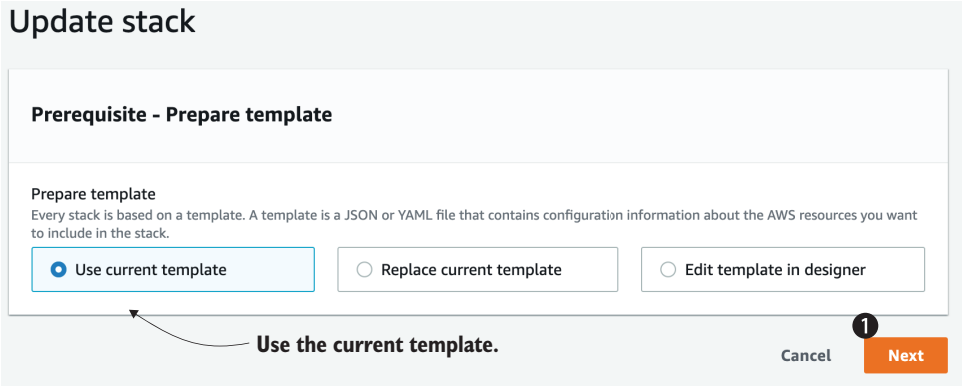


Figure 4.19 Updating the CloudFormation stack: Summary (step 1 of 4)

Console, you can see the instance to be stopped and started again with the new instance type. After a few minutes, the status should change to `UPDATE_COMPLETE`.

Alternatives to CloudFormation

If you don't want to write plain JSON or YAML to create templates for your infrastructure, a few alternatives to CloudFormation exist.

The AWS Cloud Development Kit (CDK) (https://aws.amazon.com/cdk/) allows you to use a general-purpose programming language to define your infrastructure. Under the hood, the CDK generates CloudFormation templates.

Another popular option is Terraform (https://www.terraform.io), which supports AWS as well as other cloud and service providers.

When you changed the parameter, CloudFormation figured out what needed to be done to achieve the end result. That's the power of a declarative approach: you say what the end result should look like, not how the end result should be achieved.

Cleaning up

Delete the stack by selecting it and clicking the Delete button.

Summary

- Use the CLI, one of the SDKs, or CloudFormation to automate your infrastructure on AWS.
- Infrastructure as Code describes the approach of programming the creation and modification of your infrastructure, including virtual machines, networking, storage, and more.

- You can use the CLI to automate complex processes in AWS with scripts (Bash and PowerShell).
- You can use SDKs for nine programming languages and platforms to embed AWS into your applications and create applications like nodecc.
- CloudFormation uses a declarative approach in JSON or YAML: you define only the end state of your infrastructure, and CloudFormation figures out how this state can be achieved. The major parts of a CloudFormation template are parameters, resources, and outputs.

5 Securing your system: IAM, security groups, and VPC

This chapter covers

- Who is responsible for security?
- Keeping your software up-to-date
- Controlling access to your AWS account with users and roles
- Keeping your traffic under control with security groups
- Using CloudFormation to create a private network

If security is a wall, you'll need a lot of bricks to build that wall, as shown in figure 5.1. This chapter focuses on the following four most important bricks to secure your systems on AWS:

1. *Installing software updates*—New security vulnerabilities are found in software every day. Software vendors release updates to fix those vulnerabilities, and it's your job to install those updates as quickly as possible after they're released on your systems. Otherwise, your systems will be an easy victim for hackers.
2. *Restricting access to your AWS account*—This becomes even more important if you aren't the only one accessing your AWS account, such as when coworkers and automated processes need access to your AWS account as well. A buggy

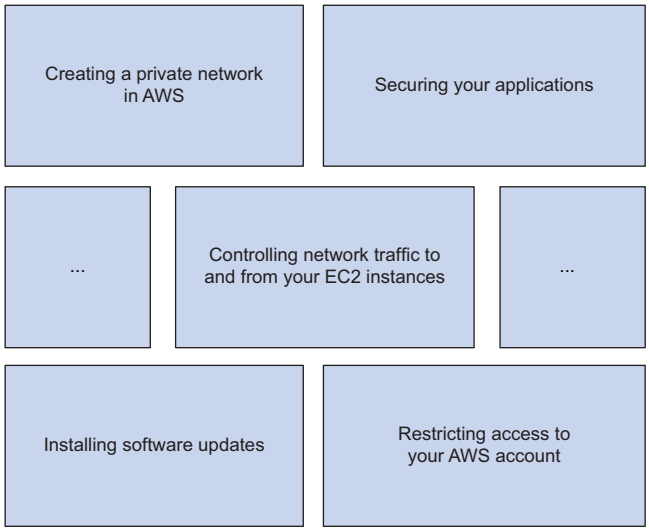


Figure 5.1 To achieve the security of your cloud infrastructure and application, all security building blocks have to be in place.

script could easily terminate all your EC2 instances instead of only the one you intended. Granting only the permissions needed is the key to securing your AWS resources from accidental or intended disastrous actions.

3. *Controlling network traffic to and from your EC2 instances*—You want ports to be accessible only if they must be. If you run a web server, the only ports you need to open to the outside world are ports 80 for HTTP traffic and 443 for HTTPS traffic. Do not open any other ports for external access.
4. *Creating a private network in AWS*—Control network traffic by defining subnets and routing tables. Doing so allows you to specify private networks that are not reachable from the outside.

One important brick is missing: securing your applications. We do not cover application security in our book. When buying or developing applications, you should follow security standards. For example, you need to check user input and allow only the necessary characters, don't save passwords in plain text, and use TLS/SSL to encrypt traffic between your virtual machines and your users.

This is going to be a long chapter—security is such an important topic, there's a lot to cover. But don't worry, we'll take it step by step.

Not all examples are covered by Free Tier

The examples in this chapter are not all covered by the Free Tier. A special warning message appears when an example incurs costs. As for the other examples, as long as you don't run them longer than a few days, you won't pay anything for them. Keep in mind that this applies only if you created a fresh AWS account for this book and nothing else is going on in your AWS account. Try to complete the chapter within a few days; you'll clean up your account at the end.

Chapter requirements

To fully understand this chapter, you should be familiar with the following networking concepts:

- Subnet
- Route tables
- Access control lists (ACLs)
- Gateway
- Firewall
- Port
- Access management
- Basics of the Internet Protocol (IP), including IP addresses

Before we look at the four bricks, let's talk about how responsibility is divided between you and AWS.

5.1 Who's responsible for security?

The cloud is a shared-responsibility environment, meaning responsibility is shared between you and AWS. AWS is responsible for the following:

- Protecting the network through automated monitoring systems and robust internet access, to prevent distributed denial of service (DDoS) attacks
- Performing background checks on employees who have access to sensitive areas
- Decommissioning storage devices by physically destroying them after end of life
- Ensuring the physical and environmental security of data centers, including fire protection and security staff

The security standards are reviewed by third parties; you can find an up-to-date overview at https://aws.amazon.com/compliance/.

What are your responsibilities? See the following:

- Configuring access management that restricts access to AWS resources like S3 and EC2 to a minimum, using AWS IAM
- Encrypting network traffic to prevent attackers from reading or manipulating data (e.g., using HTTPS)
- Configuring a firewall for your virtual network that controls incoming and outgoing traffic with security groups and NACLs
- Encrypting data at rest. For example, enable data encryption for your database or other storage systems
- Managing patches for the OS and additional software on virtual machines

Security involves an interaction between AWS and you, the customer. If you play by the rules, you can achieve high security standards in the cloud. Want to dive into more details? Check out https://aws.amazon.com/compliance/shared-responsibility-model/.

5.2 Keeping the operating system up-to-date

Not a week goes by without the release of an important update to fix security vulnerabilities in some piece of software or another. Sometimes the kernel is affected or libraries, like OpenSSL. Other times, it's affecting an environment like Java, Apache, and PHP, or an application like WordPress. If a security update is released, you must install it quickly, because the exploit may have already been released, or because unscrupulous people could look at the source code to reconstruct the vulnerability. You should have a working plan for how to apply updates to all running virtual machines as quickly as possible.

Amazon Linux 2 installs critical or important security updates automatically on startup while `cloud-init` is running. We highly recommend you install all the other updates as well. The following options are available:

- *Install all updates at the end of the boot process*—Include `yum -y update` in your user-data script. `yum` is the package manager used by Amazon Linux 2.
- *Install security updates at the end of the boot process only*—Include the `yum -y --security update` in your user-data script.
- Use the *AWS Systems Manager Patch Manager*—Install updates based on a patch baseline.

The first two options can be easily included in the user data of your EC2 instance. You can find the code in /chapter05/ec2-yum-update.yaml in the book's code folder. You install all updates as follows:

```
Instance:
  Type: 'AWS::EC2::Instance'
  Properties:
    # [...]
    UserData: !Base64 |
      #!/bin/bash -ex
      yum -y update
```

Installs all updates

To install only security updates, do the following:

```
Instance:
  Type: 'AWS::EC2::Instance'
  Properties:
    # [...]
    UserData: !Base64 |
      #!/bin/bash -ex
      yum -y --security update
```

Installs only security updates

The following challenges are still waiting for a solution:

- The problem with installing all updates is that your system might still be vulnerable. Some updates require a reboot (most notably kernel updates)!
- Installing updates on startup is not enough. Updates need to be installed continuously.

Before reinventing the wheel, it is a good strategy to research whether AWS provides the building blocks needed to get the job done. Luckily, AWS Systems Manager (SSM) Patch Manager is a good choice to make patching more robust and stable.

The AWS Systems Manager provides a toolbox that includes a core set of features bundled into capabilities. Patch Manager is one such capability. The following core SSM features are bundled together in the Patch Manager, as figure 5.2 shows:

- *Agent*—Preinstalled and autostarted on Amazon Linux 2 (also powers the Session Manager).
- *Document*—Think of a document as a script on steroids. We use a prebuild document named `AWS-RunPatchBaseline` to install patches.
- *Run Command*—Executes a document on an EC2 instance.
- *Association*—Sends commands (via Run Command) to EC2 instances on a schedule or during startup (bundled into the capability named State Manager).
- *Maintenance Window*—Sends commands (via Run Command) to EC2 instances on a schedule during a time window.
- *Patch baseline*—Set of rules to approve patches for installation based on classification and severity. Luckily, AWS provides predefined patch baselines for various operating systems including Amazon Linux 2. The predefined patch baseline for Amazon Linux 2 approves all security patches that have a severity level of critical or important and all bug fixes. A seven-day waiting period exists after the release of a patch before approval.

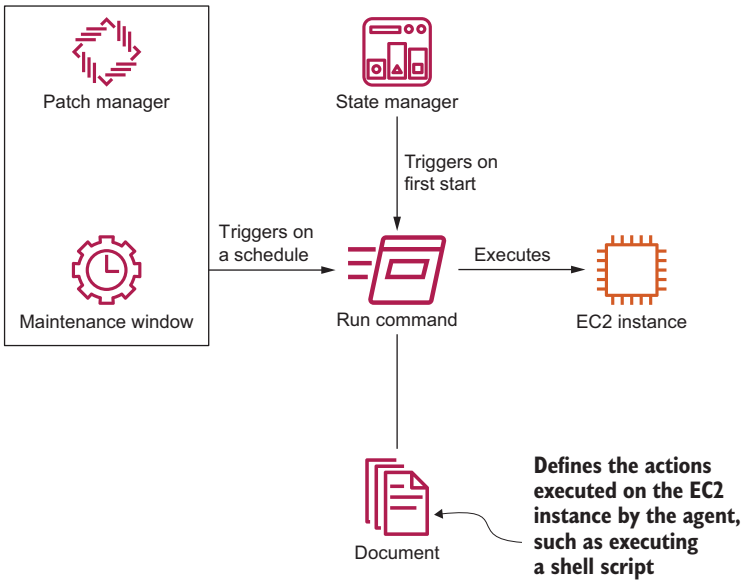


Figure 5.2 SSM features required for Patch Manager capability

The following CloudFormation snippet defines a maintenance window to patch on a schedule as well as an association to patch on startup:

```
MaintenanceWindow:
  Type: 'AWS::SSM::MaintenanceWindow'
  Properties:
    AllowUnassociatedTargets: false
    Duration: 2
    Cutoff: 1
    Name: !Ref 'AWS::StackName'
    Schedule: 'cron(0 5 ? * SUN *)'
    ScheduleTimezone: UTC
MaintenanceWindowTarget:
  Type: 'AWS::SSM::MaintenanceWindowTarget'
  Properties:
    ResourceType: INSTANCE
    Targets:
    - Key: InstanceIds
      Values:
      - !Ref Instance
    WindowId: !Ref MaintenanceWindow
MaintenanceWindowTask:
  Type: 'AWS::SSM::MaintenanceWindowTask'
  Properties:
    MaxConcurrency: '1'
    MaxErrors: '1'
    Priority: 0
    Targets:
    - Key: WindowTargetIds
      Values:
      - !Ref MaintenanceWindowTarget
    TaskArn: 'AWS-RunPatchBaseline'
    TaskInvocationParameters:
      MaintenanceWindowRunCommandParameters:
        Parameters:
          Operation:
          - Install
    TaskType: 'RUN_COMMAND'
    WindowId: !Ref MaintenanceWindow
AssociationRunPatchBaselineInstall:
  Type: 'AWS::SSM::Association'
  Properties:
    Name: 'AWS-RunPatchBaseline'
    Parameters:
      Operation:
      - Install
    Targets:
    - Key: InstanceIds
      Values:
      - !Ref Instance
```

The maintenance window is two hours long. You can patch more than one EC2 instance if you wish.

The last hour is reserved for commands to finish (all commands are started in the first hour).

The maintenance window is scheduled every Sunday morning at 5am UTC time. Learn more about the syntax at http://mng.bz/zmRZ.

Assigns one EC2 instance to the maintenance window. You can also assign EC2 instances based on tags.

The AWS-RunPatchBaseline document is executed.

The document supports parameters. Operation can be set to Install or Scan. By default, a reboot happens if required by any patch.

The association ensures that patches are installed on startup. The same document with the same parameters are used.

There is one prerequisite missing: the EC2 instance needs read access to a set of S3 buckets for Patch Manager to work, which is granted in the next snippet. Learn more at https://mng.bz/0ynz:

```
InstanceRole:
  Type: 'AWS::IAM::Role'
  Properties:
    #[...]
    Policies:
    - PolicyName: PatchManager
      PolicyDocument:
        Version: '2012-10-17'
        Statement:
        - Effect: Allow
          Action: 's3:GetObject'
          Resource:
          - !Sub 'arn:aws:s3:::patch-baseline-snapshot-${AWS::Region}/*'
          - !Sub 'arn:aws:s3:::aws-ssm-${AWS::Region}/*'
```

Patch Manager can also visualize the patches that are waiting for installation. To gather the data, another association is needed, as shown next:

Do not run on startup. Unfortunately, the document AWS-RunPatchBaseline crashes when running more than once at the same time. It avoids a conflict with the association defined in AssociationRunPatchBaselineInstall.

```
AssociationRunPatchBaselineScan:
  Type: 'AWS::SSM::Association'
  Properties:
    ApplyOnlyAtCronInterval: true
    Name: 'AWS-RunPatchBaseline'
    Parameters:
      Operation:
      - Scan
    ScheduleExpression: 'cron(0 0/1 * * ? *)'
    Targets:
    - Key: InstanceIds
      Values:
      - !Ref Instance
```

Uses the same document AWS-RunPatchBaseline…

…but this time, Operation is set to Scan.

Runs every hour

It's time for a demo. Create the CloudFormation stack with the template located at https://s3.amazonaws.com/awsinaction-code3/chapter05/ec2-os-update.yaml by clicking the CloudFormation Quick-Create Link (http://mng.bz/KlXn). Pick the default Virtual Private Cloud (VPC) and subnet, then wait for the stack creation to finish.

Visit the AWS Systems Manager management console at https://console.aws.amazon.com/systems-manager/. Open Patch Manager in the navigation bar, and you see a nice dashboard, as shown in figure 5.3.

You can also patch instances manually by pressing the Patch Now button, if needed.

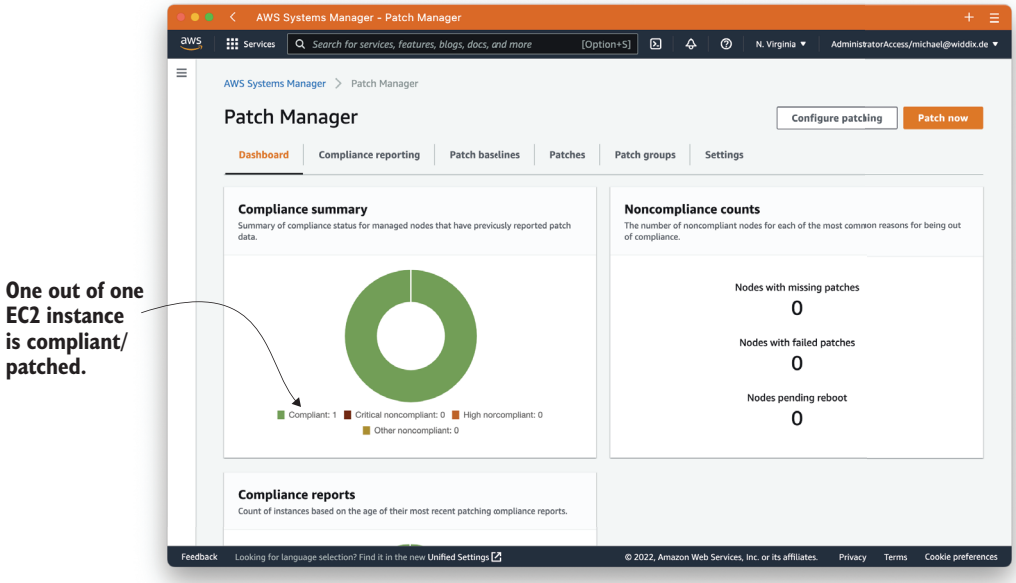


Figure 5.3 AWS Systems Manager Patch Manager dashboard

Cleaning up

Don't forget to delete your stack `ec2-os-update` after you finish this section, to clean up all used resources. Otherwise, you'll likely be charged for the resources you use.

5.3 Securing your AWS account

Securing your AWS account is critical. If someone gets access to your AWS account, they can steal your data, use resources at your expense, or delete all your data. As figure 5.4 shows, an AWS account is a basket for all the resources you own: EC2 instances, CloudFormation stacks, IAM users, and so on. Each AWS account comes with a root user granted unrestricted access to all resources. So far, you've used the AWS account root user to log in to the Management Console and the user `mycli`—created in section 4.2—when using the CLI. In this section, you will create an additional user to log in to the Management Console to avoid using the AWS account root user at all. Doing so allows you to manage multiple users, each restricted to the resources that are necessary for their roles.

To access your AWS account, an attacker must be able to authenticate to your account. There are three ways to do so: using the AWS account root user, using an IAM user, or authenticating as an AWS resource like an EC2 instance. To authenticate as a AWS account root user or IAM user, the attacker needs the username and

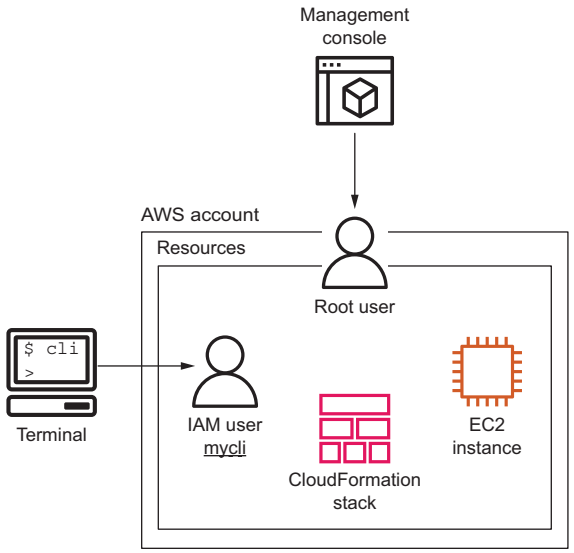


Figure 5.4 An AWS account contains all the AWS resources and comes with an AWS account root user by default.

password or the access keys. To authenticate as an AWS resource like an EC2 instance, the attacker needs access to the machine to communicate with the instance metadata service (IMDS).

To protect yourself from an attacker stealing or cracking your passwords or access keys, in the following section, you will enable multifactor authentication (MFA) for your AWS account root user to add an additional layer of security to the authentication process.

5.3.1 Securing your AWS account's root user

We advise you to enable MFA for the AWS account root user of your AWS account. After MFA is activated, you'll need a password and a temporary token to log in as the root user. Follow these steps to enable MFA, as shown in figure 5.5:

1. Click your name in the navigation bar at the top right of the Management Console.
2. Select Security Credentials.
3. Install an MFA app on your smartphone that supports the TOTP standard (such as Google Authenticator).
4. Expand the Multi-Factor Authentication (MFA) section.
5. Click Activate MFA.
6. Select Virtual MFA Device, and proceed with the next step.
7. Follow the instructions. Use the MFA app on your smartphone to scan the QR code that is displayed.

If you're using your smartphone as a virtual MFA device, it's a good idea not to log in to the Management Console from your smartphone or to store the AWS account root

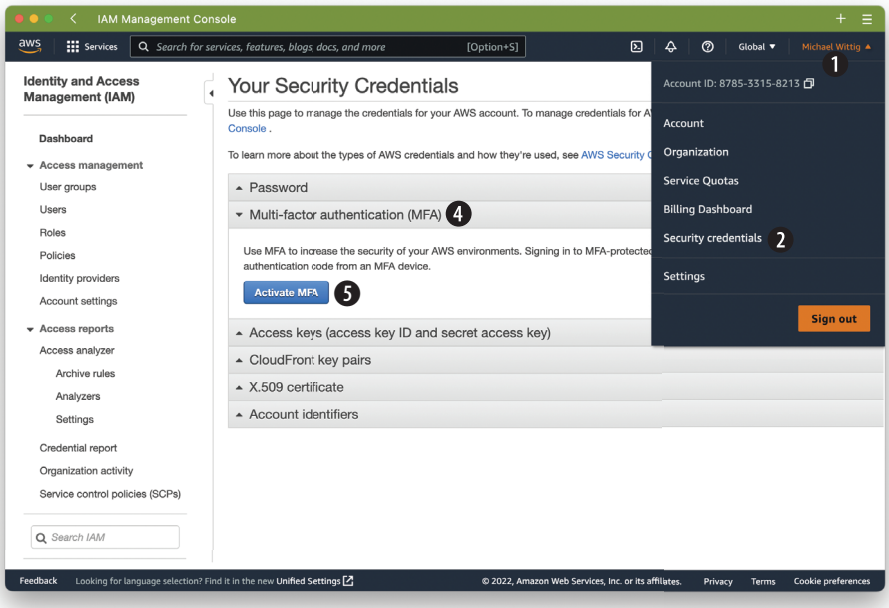


Figure 5.5 Protect your AWS account root user with multifactor authentication (MFA).

user's password on the phone. Keep the MFA token separate from your password. YubiKeys and hardware MFA tokens are also supported.

5.3.2 AWS Identity and Access Management (IAM)

Figure 5.6 shows an overview of all the core concepts of the *Identity and Access Management* (IAM) service. This service provides authentication and authorization for the AWS API. When you send a request to the AWS API, IAM verifies your identity and checks whether you are allowed to perform the action. IAM controls who (authentication) can do what (authorization) in your AWS account. For example, is the user allowed to launch a new virtual machine? The various components of IAM follow:

- An *IAM user* is used to authenticate people or workloads running outside of AWS.
- An *IAM group* is a collection of IAM users with the same permissions.
- An *IAM role* is used to authenticate AWS resources, for example, an EC2 instance.
- An *IAM identity policy* is used to define the permissions for a user, group, or role.

Table 5.1 shows the differences between users and roles. Roles authenticate AWS entities such as EC2 instances. IAM users authenticate the people who manage

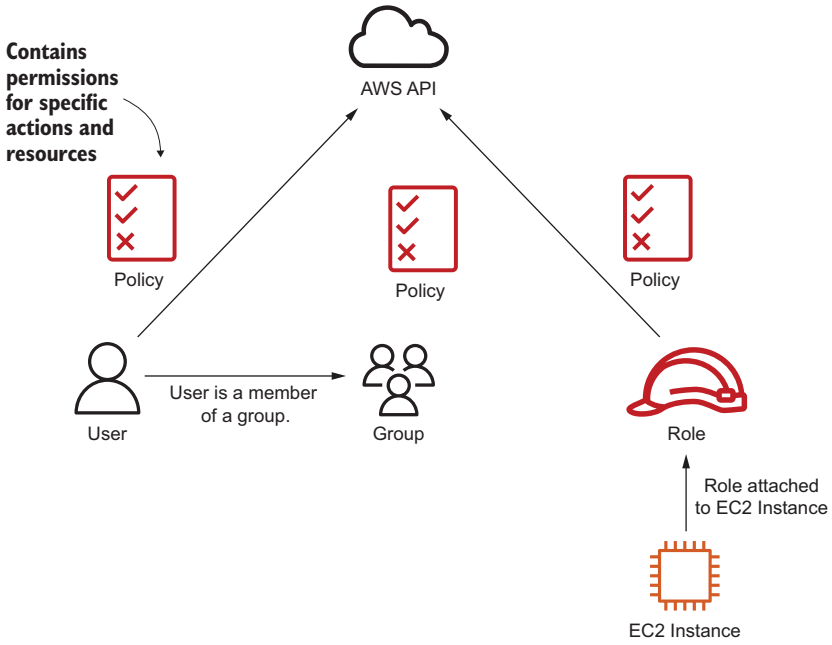


Figure 5.6 IAM concepts

AWS resources, for example, system administrators, DevOps engineers, or software developers.

Table 5.1 Differences among an AWS account root user, IAM user, and IAM role

	AWS account root user	IAM user	IAM role
Can have a password (needed to log in to the AWS Management Console)	Always	Yes	No
Can have access keys (needed to send requests to the AWS API (e.g., for CLI or SDK)	Yes (not recommended)	Yes	No
Can belong to a group	No	Yes	No
Can be associated with an EC2 instance, ECS container, Lambda function	No	No	Yes

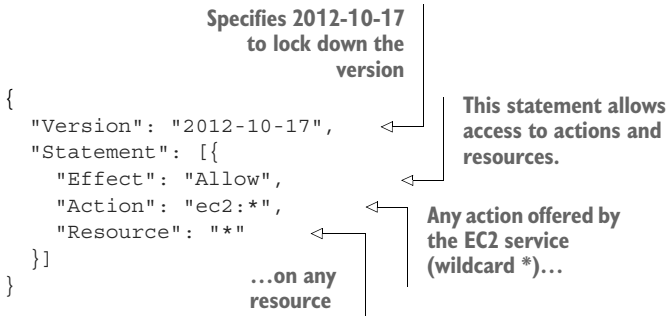
By default, users and roles can't do anything. You have to create an identity policy stating what actions they're allowed to perform. IAM users and IAM roles use identity policies for authorization. Let's look at identity policies next.

5.3.3 Defining permissions with an IAM identity policy

By attaching one or multiple IAM identity policies to an IAM user or role, you are granting permissions to manage AWS resources. Identity policies are defined in JSON and contain one or more statements. A statement can either allow or deny specific actions on specific resources. You can use the wildcard character `*` to create more generic statements.

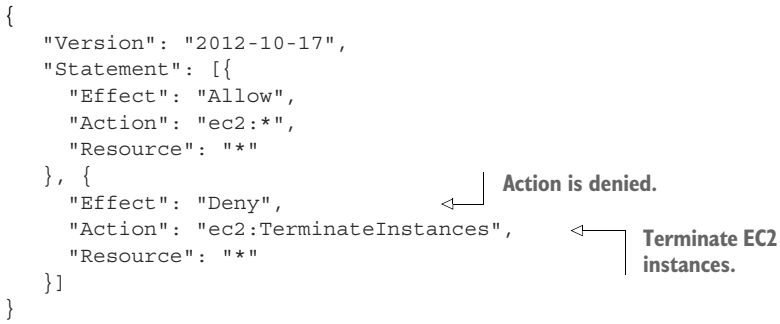
> **Identity vs. resource policies**
>
> IAM policies come in two types. *Identity policies* are attached to users, groups, or roles. *Resource policies* are attached to resources. Very few resource types support resource policies. One common example is the S3 bucket policy attached to S3 buckets.
>
> If a policy contains the property `Principal`, it is a resource policy. The `Principal` defines who is allowed to perform the action. Keep in mind that the principal can be set to public.

The following identity policy has one statement that allows every action for the EC2 service, for all resources:




```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": "ec2:*",
    "Resource": "*"
  }]
}
```

If you have multiple statements that apply to the same action, `Deny` overrides `Allow`. The following identity policy allows all EC2 actions except terminating EC2 instances:




```
{
   "Version": "2012-10-17",
   "Statement": [{
     "Effect": "Allow",
     "Action": "ec2:*",
     "Resource": "*"
   }, {
     "Effect": "Deny",
     "Action": "ec2:TerminateInstances",
     "Resource": "*"
   }]
}
```

The following identity policy denies all EC2 actions. The `ec2:TerminateInstances` statement isn't crucial, because `Deny` overrides `Allow`. When you deny an action, you can't allow that action with another statement:

```
{
   "Version": "2012-10-17",
   "Statement": [{
     "Effect": "Deny",
     "Action": "ec2:*",
     "Resource": "*"
   }, {
     "Effect": "Allow",
     "Action": "ec2:TerminateInstances",
     "Resource": "*"
   }]
}
```

Denies every EC2 action

Allow isn't crucial; Deny overrides Allow.

So far, the `Resource` part has been set to `*` to apply to every resource. Resources in AWS have an Amazon Resource Name (ARN); figure 5.7 shows the ARN of an EC2 instance.

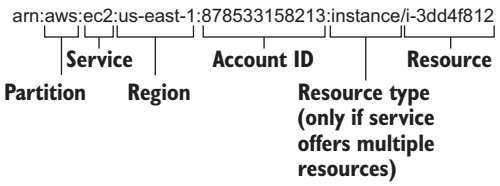


Figure 5.7 Components of an Amazon Resource Name (ARN) identifying an EC2 instance

To find out the account ID, you can use the CLI as follows:

```
$ aws sts get-caller-identity --query "Account" --output text
111111111111
```

Account ID always has 12 digits.

If you know your account ID, you can use ARNs to allow access to specific resources of a service like this:

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": "ec2:TerminateInstances",
    "Resource":
➥ "arn:aws:ec2:us-east-1:111111111111:instance/i-0b5c991e026104db9"
  }]
}
```

The list of all IAM actions of a service and possible resource ARNs can be found at http://mng.bz/Po0g.

The following two types of identity policies exist:

- *Managed policy*—If you want to create identity policies that can be reused in your account, a managed policy is what you're looking for. There are two types of managed policies:
  - *AWS managed policy*—An identity policy maintained by AWS. There are identity policies that grant admin rights, read-only rights, and so on.
  - *Customer managed*—An identity policy maintained by you. It could be an identity policy that represents the roles in your organization, for example.
- *Inline policy*—An identity policy that belongs to a certain IAM role, user, or group. An inline identity policy can't exist without the IAM role, user, or group that it belongs to.

With CloudFormation, it's easy to maintain inline identity policies; that's why we use inline identity policies most of the time in this book. One exception is the `mycli` user: this user has the AWS managed policy `AdministratorAccess` attached.

WARNING Using managed policies can often conflict with following the least-privilege principal. Managed policies usually set the `Resource` property to `*`. That's why we attach our own inline policies to IAM roles or users.

5.3.4 Users for authentication and groups to organize users

A user can authenticate using either a username and password or access keys. When you log in to the Management Console, you're authenticating with your username and password. When you use the CLI from your computer, you use access keys to authenticate as the `mycli` user.

You're using the AWS account root user at the moment to log in to the Management Console. You should create an IAM user instead, for the following reasons:

- Creating IAM users allows you to set up a unique user for every person who needs to access your AWS account.
- You can grant access only to the resources each user needs, allowing you to follow the least-privilege principle.

To make things easier if you want to add users in the future, you'll first create a group for all users with administrator access. Groups can't be used to authenticate, but they centralize authorization. So, if you want to stop your admin users from terminating EC2 instances, you need to change the identity policy only for the group instead of changing it for all admin users. A user can be a member of zero, one, or multiple groups.

It's easy to create groups and users with the CLI, as shown here. Replace `$Password` in the following with a secure password:

```
$ aws iam create-group --group-name "admin"
$ aws iam attach-group-policy --group-name "admin" \
➥ --policy-arn "arn:aws:iam::aws:policy/AdministratorAccess"
$ aws iam create-user --user-name "myuser"
```

```
$ aws iam add-user-to-group --group-name "admin" --user-name "myuser"
$ aws iam create-login-profile --user-name "myuser" --password '$Password'
```

The user `myuser` is ready to be used. But you must use a different URL to access the Management Console if you aren't using the AWS account root user: https://$accountId.signin.aws.amazon.com/console. Replace `$accountId` with the account ID that you extracted earlier with the `aws sts get-caller-identity` command.

Enabling MFA for IAM users

We encourage you to enable MFA for all users. To enable MFA for your users, follow these steps:

1. Open the IAM service in the Management Console.
2. Choose Users at the left.
3. Click the myuser user.
4. Select the Security Credentials tab.
5. Click the Manage link near the Assigned MFA Device.
6. The wizard to enable MFA for the IAM user is the same one you used for enabling MFA for the AWS account root user.

We recommend enabling MFA for all users, especially for users granted administrator access to all or some services.

WARNING Stop using the AWS account root user from now on. Always use `myuser` and the new link to the Management Console.

WARNING You should never copy a user's access keys to an EC2 instance; use IAM roles instead! Don't store security credentials in your source code. And never ever check them into your source code repository. Try to use IAM roles instead whenever possible, as described in the next section.

5.3.5 Authenticating AWS resources with roles

Various use cases exist where an EC2 instance needs to access or manage AWS resources. For example, an EC2 instance might need to do the following:

- Back up data to the object store S3
- Terminate itself after a job has been completed
- Change the configuration of the private network environment in the cloud

To be able to access the AWS API, an EC2 instance needs to authenticate itself. You could create an IAM user with access keys and store the access keys on an EC2 instance for authentication. But doing so is a hassle and violates security best practices, especially if you want to rotate the access keys regularly.

Instead of using an IAM user for authentication, you should use an IAM role whenever you need to authenticate AWS resources like EC2 instances. When using an IAM role, your access keys are injected into your EC2 instance automatically.

If an IAM role is attached to an EC2 instance, all identity policies attached to those roles are evaluated to determine whether the request is allowed. By default, no role is attached to an EC2 instance, and, therefore, the EC2 instance is not allowed to make any calls to the AWS API.

The following example will show you how to use an IAM role for an EC2 instance. Do you remember the temporary EC2 instances from chapter 4? What if we forgot to terminate those VMs? A lot of money would have been wasted because of that. You'll now create an EC2 instance that stops itself automatically. The following snippet shows a one-liner terminating an EC2 instance after five minutes. The command `at` is used to execute the `aws ec2 stop-instances` with a five-minute delay:

```
$ echo "aws ec2 stop-instances --instance-ids i-0b5c991e026104db9" \
➥ | at now + 5 minutes
```

The EC2 instance needs permission to stop itself. Therefore, you need to attach an IAM role to the EC2 instance. The role contains an inline identity policy granting access to the `ec2:StopInstances` action. Unfortunately, we can't lock down the action to the EC2 instance resource itself due to a cyclic dependency. Luckily, we can grant permissions with additional conditions. One such condition is that a specific tag must be present. The following code shows how you define an IAM role with the help of CloudFormation:

To attach an inline role to an instance, you must first create an instance profile, as shown in the following code snippet:

```
InstanceProfile:
  Type: 'AWS::IAM::InstanceProfile'
  Properties:
    Roles:
    - !Ref Role
```

The next code snippet shows how to attach the IAM role to the virtual machine:

```
Instance:
  Type: 'AWS::EC2::Instance'
  Properties:
     # [...]
    IamInstanceProfile: !Ref InstanceProfile
    UserData:
      'Fn::Base64': !Sub |
        #!/bin/bash -ex
        TOKEN=`curl -X PUT "http://169.254.169.254/latest/api/token" \
➥ -H "X-aws-ec2-metadata-token-ttl-seconds: 21600"`
        INSTANCEID=`curl -H "X-aws-ec2-metadata-token: $TOKEN" \
➥ -s "http://169.254.169.254/latest/meta-data/instance-id"`
        echo "aws ec2 stop-instances --region ${AWS::Region} \
➥ --instance-ids $INSTANCEID" | at now + ${Lifetime} minutes
```

Create the CloudFormation stack with the template located at https://s3.amazonaws.com/awsinaction-code3/chapter05/ec2-iam-role.yaml by clicking the CloudFormation Quick-Create Link (http://mng.bz/JVeP). Specify the lifetime of the EC2 instance via the parameter, and pick the default VPC and subnet as well. Wait until the amount of time specified as the lifetime has passed, and see whether your EC2 instance is stopped in the EC2 Management Console. The lifetime begins when the server is fully started and booted.

Cleaning up

Don't forget to delete your stack `ec2-iam-role` after you finish this section, to clean up all used resources. Otherwise, you'll likely be charged for the resources you use (even when your EC2 instance is stopped, you pay for the network-attached storage).

You have learned how to use IAM users to authenticate people and IAM roles to authenticate EC2 instances or other AWS resources. You've also seen how to grant access to specific actions and resources by using an IAM identity policy. The next section will cover controlling network traffic to and from your virtual machine.

5.4 Controlling network traffic to and from your virtual machine

You want traffic to enter or leave your EC2 instance only if it has to do so. With a firewall, you control ingoing (also called *inbound* or *ingress*) and outgoing (also called *outbound* or *egress*) traffic. If you run a web server, the only ports you need to open to the

outside world are ports 80 for HTTP traffic and 443 for HTTPS traffic. All other ports should be closed down. You should only open ports that must be accessible, just as you grant only the permissions you need with IAM. If you are using a firewall that allows only legitimate traffic, you close a lot of possible security holes. You can also prevent yourself from human failure—for example, you prevent accidentally sending email to customers from a test system by not opening outgoing SMTP connections for test systems.

Before network traffic enters or leaves your EC2 instance, it goes through a firewall provided by AWS. The firewalls inspects the network traffic and uses rules to decide whether the traffic is allowed or denied.

IP vs. IP address

The abbreviation IP is used for Internet Protocol, whereas an IP address describes a specific address like 84.186.116.47.

Figure 5.8 shows how an SSH request from a source IP address 10.0.0.10 is inspected by the firewall and received by the destination IP address 10.10.0.20. In this case, the firewall allows the request because a rule is in place that allows TCP traffic on port 22 between the source and the destination.

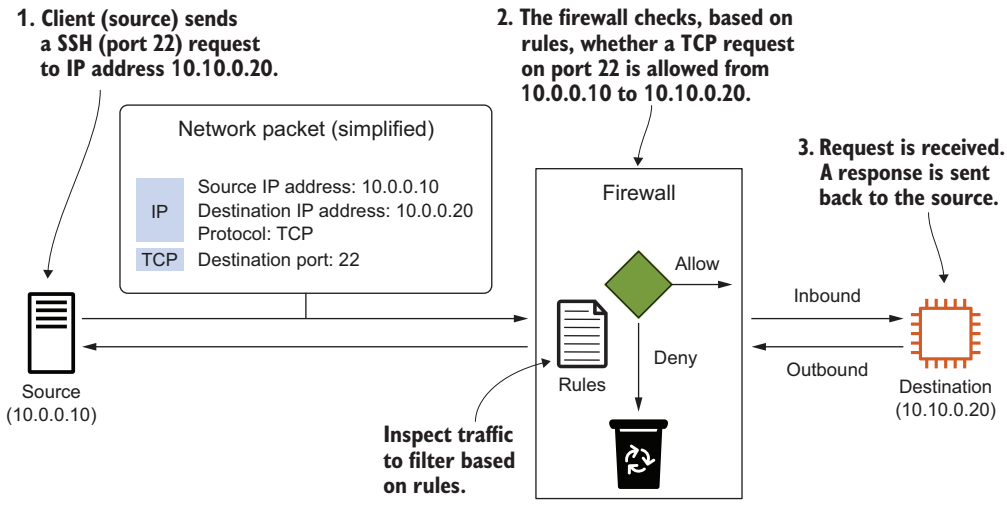


Figure 5.8 How an SSH request travels from source to destination, controlled by a firewall

AWS is responsible for the firewall, but you're responsible for the rules. By default, a security group does not allow any inbound traffic. You must add your own rules to allow specific incoming traffic. A security group contains a rule allowing all outbound traffic by default. If your use case requires a high level of network security, you should remove the rule and add your own rules to control outgoing traffic.

Debugging or monitoring network traffic

Imagine the following problem: your EC2 instance does not accept SSH traffic as you want it to, but you can't spot any misconfiguration in your firewall rules. The following two strategies are helpful:

- Use the VPC Reachability Analyzer to simulate the traffic and see if the tool finds the configuration problem. Learn more at http://mng.bz/wyqW.
- Enable VPC Flow Logs to get access to aggregated log messages containing rejected connections. Learn more at http://mng.bz/qoqE.

5.4.1 Controlling traffic to virtual machines with security groups

A security group acts as a firewall for virtual machines and other services. You will associate a security group with AWS resources, such as EC2 instances, to control traffic. It's common for EC2 instances to have more than one security group associated with them and for the same security group to be associated with multiple EC2 instances.

A security group consists of a set of rules. Each rule allows network traffic based on the following:

- Direction (inbound or outbound)
- IP protocol (TCP, UDP, ICMP)
- Port
- Source/destination based on IP address, IP address range, or security group (works only within AWS)

In theory, you could define rules that allow all traffic to enter and leave your virtual machine; AWS won't prevent you from doing so. But it's a best practice to define your rules so they are as restrictive as possible.

Security group resources in CloudFormation are of type `AWS::EC2::SecurityGroup`. The following listing is in /chapter05/firewall1.yaml in the book's code folder; the template describes an empty security group associated with a single EC2 instance.

Listing 5.1 CloudFormation template: Security group

```
---
[..]
Parameters:
  VPC:
    # [..]
  Subnet:
    # [..]
Resources:
  SecurityGroup:
    Type: 'AWS::EC2::SecurityGroup'
    Properties:
      GroupDescription: 'Learn how to protect your EC2 Instance.'
      VpcId: !Ref VPC
```

You'll learn about this in section 5.5.

Defines the security group without any rules (by default, inbound traffic is denied and outbound traffic is allowed). Rules will be added in the following sections.

```
      Tags:
      - Key: Name
        Value: 'AWS in Action: chapter 5 (firewall)'
  Instance:                                          <-- Defines the EC2 instance
    Type: 'AWS::EC2::Instance'
    Properties:
      # [...]
      SecurityGroupIds:                              <-- Associates the security group with the EC2 instance
      - !Ref SecurityGroup
      SubnetId: !Ref Subnet
```

To explore security groups, you can try the CloudFormation template located at https://s3.amazonaws.com/awsinaction-code3/chapter05/firewall1.yaml. Create a stack based on that template by clicking the CloudFormation Quick-Create Link (http://mng.bz/91e8), and then copy the `PublicIpAddress` from the stack output.

5.4.2 Allowing ICMP traffic

If you want to ping an EC2 instance from your computer, you must allow inbound Internet Control Message Protocol (ICMP) traffic. By default, all inbound traffic is blocked. Try `ping $PublicIpAddress` to make sure `ping` isn't working, like this:

```
$ ping 34.205.166.12
PING 34.205.166.12 (34.205.166.12): 56 data bytes
Request timeout for icmp_seq 0
Request timeout for icmp_seq 1
[...]
```

You need to add a rule to the security group that allows inbound traffic, where the protocol equals ICMP. The following listing is in /chapter05/firewall2.yaml in the book's code folder.

Listing 5.2 CloudFormation template: Security group that allows ICMP

```
SecurityGroup:
  Type: 'AWS::EC2::SecurityGroup'
  Properties:
    GroupDescription: 'Learn how to protect your EC2 Instance.'
    VpcId: !Ref VPC
    Tags:
    - Key: Name
      Value: 'AWS in Action: chapter 5 (firewall)'
    SecurityGroupIngress:                            <-- Rules allowing incoming traffic
    - Description: 'allowing inbound ICMP traffic'
      IpProtocol: icmp                               <-- Specifies ICMP as the protocol
      FromPort: '-1'                                 <-- ICMP does not use ports. -1 means every port.
      ToPort: '-1'
      CidrIp: '0.0.0.0/0'                            <-- Allows traffic from any source IP address
```

Update the CloudFormation stack with the template located at https://s3.amazonaws.com/awsinaction-code3/chapter05/firewall2.yaml and retry the `ping` command. It should work now:

```
$ ping 34.205.166.12
PING 34.205.166.12 (34.205.166.12): 56 data bytes
64 bytes from 34.205.166.12: icmp_seq=0 ttl=234 time=109.095 ms
64 bytes from 34.205.166.12: icmp_seq=1 ttl=234 time=107.000 ms
[...]
round-trip min/avg/max/stddev = 107.000/108.917/110.657/1.498 ms
```

Everyone's inbound ICMP traffic (every source IP address) is now allowed to reach your EC2 instance.

5.4.3 Allowing HTTP traffic

Once you can ping your EC2 instance, you want to run a web server. To do so, you must create a rule to allow inbound TCP requests on port 80, as shown in the next listing. You also need a running web server.

Listing 5.3 CloudFormation template: Security group that allows HTTP

```
SecurityGroup:
  Type: 'AWS::EC2::SecurityGroup'
  Properties:
    GroupDescription: 'Learn how to protect your EC2 Instance.'
    VpcId: !Ref VPC
    # [...]
    SecurityGroupIngress:
    # [...]
    - Description: 'allowing inbound HTTP traffic'
      IpProtocol: tcp
      FromPort: '80'
      ToPort: '80'
      CidrIp: '0.0.0.0/0'
Instance:
  Type: 'AWS::EC2::Instance'
  Properties:
    # [...]
    UserData:
      'Fn::Base64': |
        #!/bin/bash -ex
        yum -y install httpd
        systemctl start httpd
        echo '<html>...</html>' > /var/www/html/index.html
```

HTTP is based on the TCP protocol.

The default HTTP port is 80.

Adds a rule to allow incoming HTTP traffic

You can allow a range of ports or set FromPort = ToPort.

Allows traffic from any source IP address

Installs Apache HTTP Server on startup

Starts Apache HTTP Server

Update the CloudFormation stack with the template located at https://s3.amazonaws.com/awsinaction-code3/chapter05/firewall3.yaml. Enter the public IP address in your browser to see a very basic test page.

5.4.4 Allowing HTTP traffic from a specific source IP address

So far, you're allowing inbound traffic on port 80 (HTTP) from every source IP address. It is possible to restrict access to only your own IP address for additional security as well.

> **What's the difference between public and private IP addresses?**
>
> On my local network, I'm using private IP addresses that start with 192.168.0.*. My laptop uses 192.168.0.10, and my iPad uses 192.168.0.20. But if I access the internet, I have the same public IP address (such as 79.241.98.155) for my laptop and iPad. That's because only my internet gateway (the box that connects to the internet) has a public IP address, and all requests are redirected by the gateway. (If you want to know more about this, search for network address translation.) Your local network doesn't know about this public IP address. My laptop and iPad only know that the internet gateway is reachable under 192.168.0.1 on the private network.
>
> To find your public IP address, visit https://checkip.amazonaws.com/. For some of us, our public IP address changes from time to time, usually when you reconnect to the internet (which happens every 24 hours in my case).

Hardcoding the public IP address into the template isn't a good solution because your public IP address can change from time to time. You already know the solution: parameters. You need to add a parameter that holds your current public IP address, and you need to modify the `Security Group`. You can find the following listing in /chapter05/firewall4.yaml in the book's code folder.

Listing 5.4 Security group allows traffic from source IP

```
Parameters:
  WhitelistedIpAddress:                                  <-- Public IP address parameter
    Description: 'Whitelisted IP address'
    Type: String
    AllowedPattern: '^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}$'
    ConstraintDescription: 'Enter a valid IPv4 address'
Resources:
  SecurityGroup:
    Type: 'AWS::EC2::SecurityGroup'
    Properties:
      GroupDescription: 'Learn how to protect your EC2 Instance.'
      VpcId: !Ref VPC
      # [...]
      SecurityGroupIngress:
      # [...]
      - Description: 'allowing inbound HTTP traffic'
        IpProtocol: tcp
        FromPort: '80'
        ToPort: '80'
        CidrIp: !Sub '${WhitelistedIpAddress}/32'        <-- Uses WhitelistedIpAddress/32 as a value to turn the IP address input into a CIDR
```

Update the CloudFormation stack with the template located at https://s3.amazonaws.com/awsinaction-code3/chapter05/firewall4.yaml. When asked for parameters, type in your public IP address for `WhitelistedIpAddress`. Now only your IP address can open HTTP connections to your EC2 instance.

> **Classless Inter-Domain Routing (CIDR)**
>
> You may wonder what `/32` means. To understand what's going on, you need to switch your brain into binary mode. An IP address is 4 bytes or 32 bits long. The `/32` defines how many bits (32, in this case) should be used to form a range of addresses. If you want to define the exact IP address that is allowed, you must use all 32 bits.
>
> Sometimes it makes sense to define a range of allowed IP addresses. For example, you can use `10.0.0.0/8` to create a range between 10.0.0.0 and 10.255.255.255, `10.0.0.0/16` to create a range between 10.0.0.0 and 10.0.255.255, or `10.0.0.0/24` to create a range between 10.0.0.0 and 10.0.0.255. You aren't required to use the binary boundaries (8, 16, 24, 32), but they're easier for most people to understand. You already used `0.0.0.0/0` to create a range that contains every possible IP address.

Now you can control network traffic that comes from outside a virtual machine or goes outside a virtual machine by filtering based on protocol, port, and source IP address.

5.4.5 Allowing HTTP traffic from a source security group

It is possible to control network traffic based on whether the source or destination belongs to a specific security group. For example, you can say that a MySQL database can be accessed only if the traffic comes from your web servers, or that only your proxy servers are allowed to access the web servers. Because of the elastic nature of the cloud, you'll likely deal with a dynamic number of virtual machines, so rules based on source IP addresses are difficult to maintain. This process becomes easy, however, if your rules are based on source security groups.

To explore the power of rules based on a source security group, let's look at the concept of a load balancer/ingress router/proxy. The client sends requests to the proxy. The proxy inspects the request and forwards it to the backend. The backend response is then passed back to the client by the proxy. To implement the concept of an HTTP proxy, you must follow these two rules:

- Allow HTTP access to the proxy from 0.0.0.0/0 or a specific source address.
- Allow HTTP access to all backends only if the traffic source is the proxy.

Figure 5.9 shows that only the proxy is allowed to communicate with the backend over HTTP.

A security group allowing incoming HTTP traffic from anywhere needs to be attached to the proxy. All backend VMs are attached to a security group allowing HTTP traffic only if the source is the proxy's security group. Listing 5.5 shows the security groups defined in a CloudFormation template.

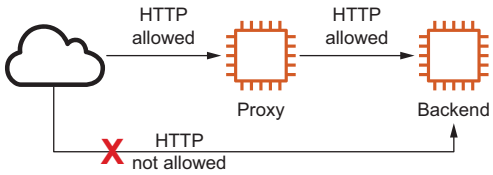


Figure 5.9 The proxy is the only HTTP entry point to the system (realized with security groups).

Listing 5.5 CloudFormation template: HTTP from proxy to backend

```
SecurityGroupProxy:                                    <-- Security group attached to the proxy
  Type: 'AWS::EC2::SecurityGroup'
  Properties:
    GroupDescription: 'Allowing incoming HTTP and ICPM from anywhere.'
    VpcId: !Ref VPC
    SecurityGroupIngress:
    - Description: 'allowing inbound ICMP traffic'
      IpProtocol: icmp
      FromPort: '-1'
      ToPort: '-1'
      CidrIp: '0.0.0.0/0'
    - Description: 'allowing inbound HTTP traffic'
      IpProtocol: tcp
      FromPort: '80'
      ToPort: '80'                                     <-- Security group attached to the backend
      CidrIp: '0.0.0.0/0'
SecurityGroupBackend:
  Type: 'AWS::EC2::SecurityGroup'
  Properties:
    GroupDescription: 'Allowing incoming HTTP from proxy.'
    VpcId: !Ref VPC
    SecurityGroupIngress:
    - Description: 'allowing inbound HTTP traffic from proxy'
      IpProtocol: tcp
      FromPort: '80'
      ToPort: '80'
      SourceSecurityGroupId: !Ref SecurityGroupProxy   <-- Allows incoming HTTP traffic only from proxy
```

Update the CloudFormation stack with the template located at https://s3.amazonaws.com/awsinaction-code3/chapter05/firewall5.yaml. After the update is completed, the stack will show the following two outputs:

- `ProxyPublicIpAddress`—Entry point into the system. Receives HTTP requests from the outside world.
- `BackendPublicIpAddress`—You can connect to this EC2 instance only from the proxy.

Copy the `ProxyPublicIpAddress` output and open it in your browser. A Hello AWS in Action! page shows up.

But when you copy the `BackendPublicIpAddress` output and try to open it in your browser, an error message will appear. That's because the security group of the backend instance allows incoming traffic only from the proxy instance, not from your IP address.

The only way to reach the backend instance is through the proxy instance. The Hello AWS in Action! page that appears when opening http://$ProxyPublicIpAddress in your browser originates from the backend instance but gets passed through the proxy instance. Check the details of the HTTP request, for example, with `curl`, as shown in the following code. You'll find a `x-backend` response header indicating that the proxy forwarded your request to the backend named `app1`, which points to the backend instance:

```
$ curl -I http://$ProxyPublicIpAddress
< HTTP/1.1 200 OK
[...]
< accept-ranges: bytes
< content-length: 91
< content-type: text/html; charset=UTF-8
< x-backend: app1
<html><title>Hello AWS in Action!</title><body>
➥ <h1>Hello AWS in Action!</h1></body></html>
```

The x-backend header is injected by the proxy and indicates the backend answered the request.

Cleaning up

Don't forget to delete your stack after you finish this section to clean up all used resources. Otherwise, you'll likely be charged for the resources you use.

5.5 Creating a private network in the cloud: Amazon Virtual Private Cloud (VPC)

When you create a VPC, you get your own private network on AWS. *Private* means you can use the address ranges 10.0.0.0/8, 172.16.0.0/12, or 192.168.0.0/16 to design a network that isn't necessarily connected to the public internet. You can create subnets, route tables, network access control lists (NACLs), and gateways to the internet or a VPN endpoint.

IPv6

Amazon Virtual Private Cloud supports IPv6 as well. You can create IPv4 only, IPv6 only, or IPv4 and IPv6 VPCs. To reduce complexity, we are sticking to IPv4 in this chapter.

Subnets allow you to separate concerns. We recommend to create at least the following two types of subnets:

- *Public subnets*—For all resources that need to be reachable from the internet, such as a load balancer of a internet-facing web application
- *Private subnets*—For all resources that should not be reachable from the internet, such as an application server or a database system

What's the difference between a public and private subnet? A public subnet has a route to the internet; a private subnet doesn't.

For the purpose of understanding how a VPC works, you'll create a VPC to replicate the example from the previous section. You'll implement the proxy concept from the previous section by creating a public subnet that contains only the proxy. You'll also create a private subnet for the backend servers. You will not be able to access a backend server directly from the internet because it will sit on a private subnet. The backend servers will be accessible only via the proxy server running in a public subnet.

The VPC uses the address space 10.0.0.0/16. To isolate different parts of the system, you'll add the next two public subnets and one private subnet to the VPC:

- *10.0.0.0/24*—Public subnet used later in this section to deploy a NAT gateway
- *10.0.1.0/24*—Public proxy subnet
- *10.0.2.0/24*—Private backend subnet

What does 10.0.0.0/16 mean?

10.0.0.0/16 represents all IP addresses in 10.0.0.0 and 10.0.255.255. It's using CIDR notation (explained earlier in the chapter).

Figure 5.10 shows the architecture of the VPC.

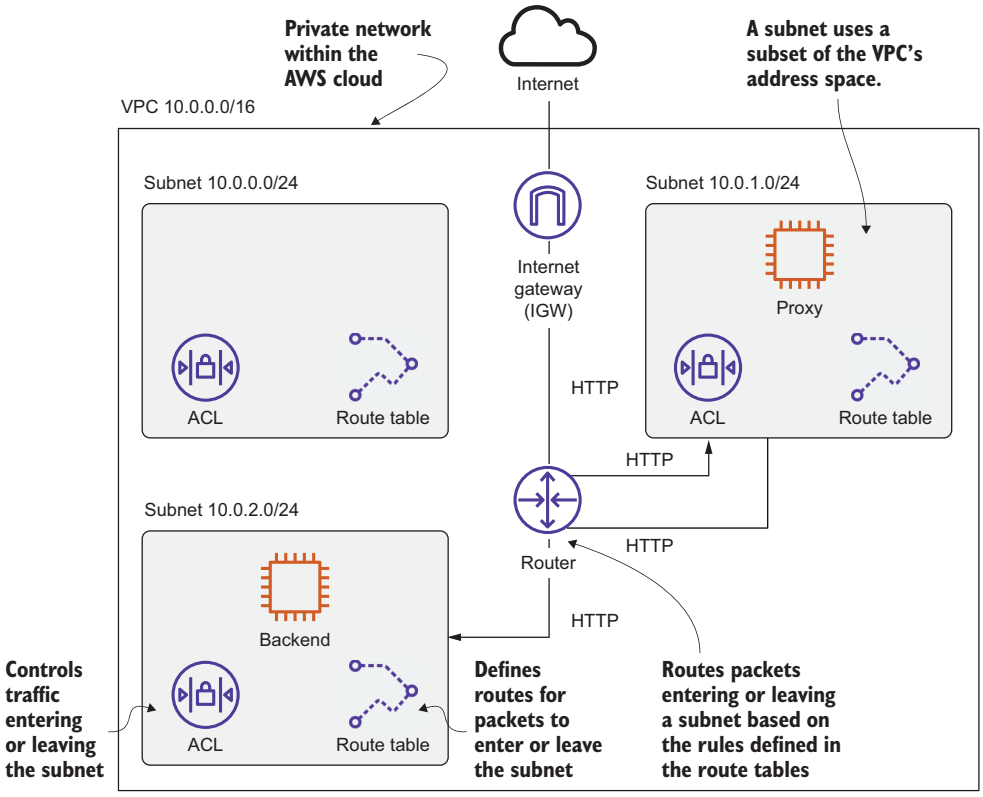


Figure 5.10 VPC with three subnets to secure a web application

You'll use CloudFormation to describe the VPC with its subnets. The template is split into smaller parts to make it easier to read in the book. As usual, you'll find the code in the book's code repository on GitHub: https://github.com/AWSinAction/code3. The template is located at /chapter05/vpc.yaml.

5.5.1 Creating the VPC and an internet gateway (IGW)

The first resources listed in the template are the VPC and the internet gateway (IGW). In the next code snippet, the IGW will translate the public IP addresses of your virtual machines to their private IP addresses using network address translation (NAT). All public IP addresses used in the VPC are controlled by this IGW:

```
VPC:
  Type: 'AWS::EC2::VPC'
  Properties:
    CidrBlock: '10.0.0.0/16'
    EnableDnsHostnames: 'true'
    Tags:
    - Key: Name
      Value: 'AWS in Action: chapter 5 (VPC)'
InternetGateway:
  Type: 'AWS::EC2::InternetGateway'
  Properties: {}
VPCGatewayAttachment:
  Type: 'AWS::EC2::VPCGatewayAttachment'
  Properties:
    VpcId: !Ref VPC
    InternetGatewayId: !Ref InternetGateway
```

The IP address space used for the private network

Adds a Name tag to the VPC

An IGW is needed to enable traffic to and from the internet.

Attaches the internet gateway to the VPC

Next you'll define the subnet for the proxy.

5.5.2 Defining the public proxy subnet

The EC2 instance running the proxy needs to be reachable via the internet. To achieve that, you'll need to complete the next four steps:

1. Create a subnet spanning a subsection of the IP address range assigned to the VPC.
2. Create a route table, and attach it to the subnet.
3. Add the `0.0.0.0/0` route pointing to the internet gateway to the route table.
4. Create an NACL, and attach it to the subnet.

To allow traffic from the internet to the proxy machine and from the proxy to the backend servers, you'll need the following NACL rules, shown in the next code snippet:

- Internet to proxy: HTTP from 0.0.0.0/0 to 10.0.1.0/24 is allowed.
- Proxy to backend: HTTP from 10.0.1.0/24 to 10.0.2.0/24 is allowed.

```
SubnetPublicProxy:
  Type: 'AWS::EC2::Subnet'
  Properties:
    AvailabilityZone: !Select [0, !GetAZs '']
    CidrBlock: '10.0.1.0/24'
    MapPublicIpOnLaunch: true
    VpcId: !Ref VPC
    Tags:
    - Key: Name
      Value: 'Public Proxy'
RouteTablePublicProxy:
  Type: 'AWS::EC2::RouteTable'
  Properties:
    VpcId: !Ref VPC
RouteTableAssociationPublicProxy:
  Type: 'AWS::EC2::SubnetRouteTableAssociation'
  Properties:
    SubnetId: !Ref SubnetPublicProxy
    RouteTableId: !Ref RouteTablePublicProxy
RoutePublicProxyToInternet:
  Type: 'AWS::EC2::Route'
  Properties:
    RouteTableId: !Ref RouteTablePublicProxy
    DestinationCidrBlock: '0.0.0.0/0'
    GatewayId: !Ref InternetGateway
  DependsOn: VPCGatewayAttachment
NetworkAclPublicProxy:
  Type: 'AWS::EC2::NetworkAcl'
  Properties:
    VpcId: !Ref VPC
SubnetNetworkAclAssociationPublicProxy:
  Type: 'AWS::EC2::SubnetNetworkAclAssociation'
  Properties:
    SubnetId: !Ref SubnetPublicProxy
    NetworkAclId: !Ref NetworkAclPublicProxy
```

Picks the first availability zone in the region. (You'll learn about availability zones in chapter 16.)

IP address space

Route table

Associates the route table with the subnet

Routes everything (0.0.0.0/0) to the IGW

Network access control list (NACL)

Associates the NACL with the subnet

There's an important difference between security groups and NACLs: security groups are stateful, but NACLs aren't. If you allow an inbound port on a security group, the corresponding response to requests on that port are allowed as well. A security group rule will work as you expect it to. If you open inbound port 80 on a security group, you can connect via HTTP.

That's not true for NACLs. If you open inbound port 80 on an NACL for your subnet, you still may not be able to connect via HTTP. In addition, you need to allow outbound ephemeral ports, because the web server accepts connections on port 80 but uses an ephemeral port for communication with the client. Ephemeral ports are selected from the range starting at 1024 and ending at 65535. If you want to make an HTTP connection from within your subnet, you have to open outbound port 80 and inbound ephemeral ports as well.

Another difference between security group rules and NACL rules is that you have to define the priority for NACL rules. A smaller rule number indicates a higher priority.

When evaluating an NACL, the first rule that matches a package is applied; all other rules are skipped.

The proxy subnet allows clients sending HTTP requests on port 80. The following NACL rules are needed:

- Allow inbound port 80 (HTTP) from 0.0.0.0/0.
- Allow outbound ephemeral ports to 0.0.0.0/0.

We also want to make HTTP and HTTPS requests from the subnet to the internet. The following NACL rules are needed:

- Allow inbound ephemeral ports from 0.0.0.0/0.
- Allow outbound port 80 (HTTP) to 0.0.0.0/0.
- Allow outbound port 443 (HTTPS) to 0.0.0.0/0.

Find the CloudFormation implementation of the previous NACLs next:

```
NetworkAclEntryInPublicProxyHTTP:
  Type: 'AWS::EC2::NetworkAclEntry'
  Properties:
    NetworkAclId: !Ref NetworkAclPublicProxy
    RuleNumber: '110'
    Protocol: '6'
    PortRange:
      From: '80'
      To: '80'
    RuleAction: 'allow'
    Egress: 'false'
    CidrBlock: '0.0.0.0/0'
NetworkAclEntryInPublicProxyEphemeralPorts:
  Type: 'AWS::EC2::NetworkAclEntry'
  Properties:
    NetworkAclId: !Ref NetworkAclPublicProxy
    RuleNumber: '200'
    Protocol: '6'
    PortRange:
      From: '1024'
      To: '65535'
    RuleAction: 'allow'
    Egress: 'false'
    CidrBlock: '0.0.0.0/0'
NetworkAclEntryOutPublicProxyHTTP:
  Type: 'AWS::EC2::NetworkAclEntry'
  Properties:
    NetworkAclId: !Ref NetworkAclPublicProxy
    RuleNumber: '100'
    Protocol: '6'
    PortRange:
      From: '80'
      To: '80'
    RuleAction: 'allow'
    Egress: 'true'
    CidrBlock: '0.0.0.0/0'
```

Allows inbound HTTP from everywhere

Rules are evaluated starting with the lowest-numbered rule.

Inbound

Ephemeral ports used for short-lived TCP/IP connections

Allows outbound HTTP to everywhere

Outbound

```
NetworkAclEntryOutPublicProxyHTTPS:
  Type: 'AWS::EC2::NetworkAclEntry'
  Properties:
    NetworkAclId: !Ref NetworkAclPublicProxy
    RuleNumber: '110'
    Protocol: '6'
    PortRange:
      From: '443'
      To: '443'
    RuleAction: 'allow'
    Egress: 'true'
    CidrBlock: '0.0.0.0/0'
NetworkAclEntryOutPublicProxyEphemeralPorts:
  Type: 'AWS::EC2::NetworkAclEntry'
  Properties:
    NetworkAclId: !Ref NetworkAclPublicProxy
    RuleNumber: '200'
    Protocol: '6'
    PortRange:
      From: '1024'
      To: '65535'
    RuleAction: 'allow'
    Egress: 'true'
    CidrBlock: '0.0.0.0/0'
```

Allows outbound HTTPS to everywhere

Ephemeral ports

We recommend you start with using security groups to control traffic. If you want to add an extra layer of security, you should use NACLs on top. But doing so is optional, in our opinion.

5.5.3 Adding the private backend subnet

As shown in figure 5.11, the only difference between a public and a private subnet is that a private subnet doesn't have a route to the IGW.

Traffic between subnets of a VPC is always routed by default. You can't remove the routes between the subnets. If you want to prevent traffic between subnets in a VPC, you need to use NACLs attached to the subnets, as shown here. The subnet for the web server has no additional routes and is, therefore, private:

```
SubnetPrivateBackend:
  Type: 'AWS::EC2::Subnet'
  Properties:
    AvailabilityZone: !Select [0, !GetAZs '']
    CidrBlock: '10.0.2.0/24'
    VpcId: !Ref VPC
    Tags:
    - Key: Name
      Value: 'Private Backend'
RouteTablePrivateBackend:
  Type: 'AWS::EC2::RouteTable'
  Properties:
    VpcId: !Ref VPC
```

Address space

No route to the IGW

```
RouteTableAssociationPrivateBackend:
  Type: 'AWS::EC2::SubnetRouteTableAssociation'
  Properties:
    SubnetId: !Ref SubnetPrivateBackend
    RouteTableId: !Ref RouteTablePrivateBackend
NetworkAclPrivateBackend:
  Type: 'AWS::EC2::NetworkAcl'
  Properties:
    VpcId: !Ref VPC
SubnetNetworkAclAssociationPrivateBackend:
  Type: 'AWS::EC2::SubnetNetworkAclAssociation'
  Properties:
    SubnetId: !Ref SubnetPrivateBackend
    NetworkAclId: !Ref NetworkAclPrivateBackend
```

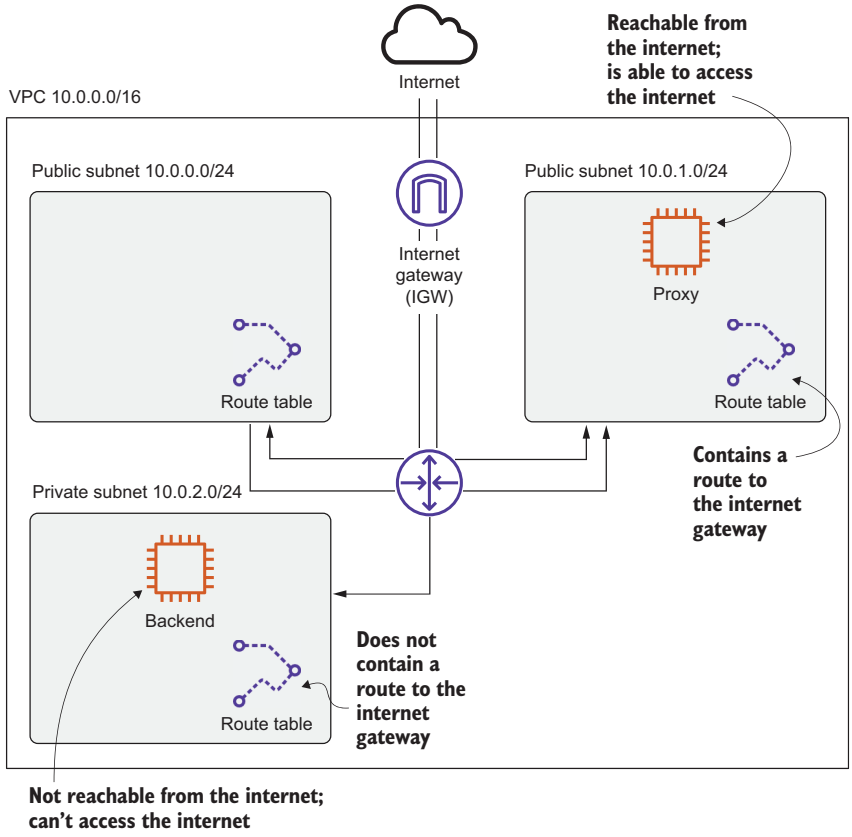


Figure 5.11 Private and public subnets

The backend subnet allows HTTP requests from the proxy subnet. The following NACLs are needed:

- Allow inbound port 80 (HTTP) from 10.0.1.0/24.
- Allow outbound ephemeral ports to 10.0.1.0/24.

We also want to make HTTP and HTTPS requests from the subnet to the internet. Keep in mind that we have no route to the internet yet. There is no way to access the internet, even with the NACLs. You will change this soon. The following NACLs are needed:

- Allow inbound ephemeral ports from 0.0.0.0/0.
- Allow outbound port 80 (HTTP) to 0.0.0.0/0.
- Allow outbound port 443 (HTTPS) to 0.0.0.0/0.

Find the CloudFormation implementation of the previous NACLs here:

```
NetworkAclEntryInPrivateBackendHTTP:
  Type: 'AWS::EC2::NetworkAclEntry'
  Properties:
    NetworkAclId: !Ref NetworkAclPrivateBackend
    RuleNumber: '110'
    Protocol: '6'
    PortRange:
      From: '80'
      To: '80'
    RuleAction: 'allow'
    Egress: 'false'
    CidrBlock: '10.0.1.0/24'
NetworkAclEntryInPrivateBackendEphemeralPorts:
  Type: 'AWS::EC2::NetworkAclEntry'
  Properties:
    NetworkAclId: !Ref NetworkAclPrivateBackend
    RuleNumber: '200'
    Protocol: '6'
    PortRange:
      From: '1024'
      To: '65535'
    RuleAction: 'allow'
    Egress: 'false'
    CidrBlock: '0.0.0.0/0'
NetworkAclEntryOutPrivateBackendHTTP:
  Type: 'AWS::EC2::NetworkAclEntry'
  Properties:
    NetworkAclId: !Ref NetworkAclPrivateBackend
    RuleNumber: '100'
    Protocol: '6'
    PortRange:
      From: '80'
      To: '80'
    RuleAction: 'allow'
    Egress: 'true'
    CidrBlock: '0.0.0.0/0'
NetworkAclEntryOutPrivateBackendHTTPS:
  Type: 'AWS::EC2::NetworkAclEntry'
  Properties:
    NetworkAclId: !Ref NetworkAclPrivateBackend
    RuleNumber: '110'
    Protocol: '6'
```

Allows inbound HTTP from proxy subnet

Ephemeral ports

Allows outbound HTTP to everywhere

Allows outbound HTTPS to everywhere

```
    PortRange:
      From: '443'
      To: '443'
    RuleAction: 'allow'
    Egress: 'true'
    CidrBlock: '0.0.0.0/0'
NetworkAclEntryOutPrivateBackendEphemeralPorts:
  Type: 'AWS::EC2::NetworkAclEntry'
  Properties:
    NetworkAclId: !Ref NetworkAclPrivateBackend
    RuleNumber: '200'
    Protocol: '6'
    PortRange:
      From: '1024'
      To: '65535'
    RuleAction: 'allow'
    Egress: 'true'
    CidrBlock: '10.0.1.0/24'
```

Ephemeral ports

5.5.4 Launching virtual machines in the subnets

Your subnets are ready, and you can continue with the EC2 instances. First you describe the proxy, like this:

```
SecurityGroup:
  Type: 'AWS::EC2::SecurityGroup'
  Properties:
    GroupDescription: 'Allowing all incoming and outgoing traffic.'
    VpcId: !Ref VPC
    SecurityGroupIngress:
    - IpProtocol: '-1'
      FromPort: '-1'
      ToPort: '-1'
      CidrIp: '0.0.0.0/0'
    SecurityGroupEgress:
    - IpProtocol: '-1'
      FromPort: '-1'
      ToPort: '-1'
      CidrIp: '0.0.0.0/0'
Proxy:
  Type: AWS::EC2::Instance
  Properties:
    ImageId: 'ami-061ac2e015473fbe2'
    InstanceType: 't2.micro'
    IamInstanceProfile: 'ec2-ssm-core'
    SecurityGroupIds:
    - !Ref SecurityGroup
    SubnetId: !Ref SubnetPublicProxy
    Tags:
    - Key: Name
      Value: Proxy
    UserData: # [...]
  DependsOn: VPCGatewayAttachment
```

This security group allows everything.

Launches in the proxy subnet

We need to manually tell CloudFormation about a dependency here. Without the attachment being created, the instance could launch but without access to the internet.

The private backend has a different configuration, as shown next:

```
Backend:
  Type: 'AWS::EC2::Instance'
  Properties:
    ImageId: 'ami-061ac2e015473fbe2'
    InstanceType: 't2.micro'
    IamInstanceProfile: 'ec2-ssm-core'
    SecurityGroupIds:
    - !Ref SecurityGroup
    SubnetId: !Ref SubnetPrivateBackend
    Tags:
    - Key: Name
      Value: Backend
    UserData:
      'Fn::Base64': |
        #!/bin/bash -ex
        yum -y install httpd
        systemctl start httpd
        echo '<html>...</html>' > /var/www/html/index.html
```

Launches in the private backend subnet

Installs Apache from the internet

Starts the Apache web server

You're now in serious trouble: installing Apache won't work because your private subnet has no route to the internet. Therefore, the `yum install` command will fail, because the public Yum repository is not reachable without access to the internet.

5.5.5 Accessing the internet from private subnets via a NAT gateway

Public subnets have a route to the internet gateway. You can use a similar mechanism to provide outbound internet connectivity for EC2 instances running in private subnets without having a direct route to the internet: create a NAT gateway in a public subnet, and create a route from your private subnet to the NAT gateway. This way, you can reach the internet from private subnets, but the internet can't reach your private subnets. A NAT gateway is a managed service provided by AWS that handles network address translation. Internet traffic from your private subnet will access the internet from the public IP address of the NAT gateway.

Reducing costs for NAT gateway

You have to pay for the traffic processed by a NAT gateway (see VPC Pricing at https://aws.amazon.com/vpc/pricing/ for more details). If your EC2 instances in private subnets will have to transfer huge amounts of data to the internet, you have two options to decrease costs:

- Moving your EC2 instances from the private subnet to a public subnet allows them to transfer data to the internet without using the NAT gateway. Use firewalls to strictly restrict incoming traffic from the internet.
- If data is transferred over the internet to reach AWS services (such as Amazon S3 and Amazon DynamoDB), use gateway VPC endpoints. These endpoints allow your EC2 instances to communicate with S3 and DynamoDB directly and at no additional charge. Furthermore, most other services are accessible from private subnets via interface VPC endpoints (offered by AWS PrivateLink) with an hourly and bandwidth fee.

WARNING NAT gateways are finite resources. A NAT gateway processes up to 100 Gbit/s of traffic and up to 10 million packets per second. A NAT gateway is also bound to an availability zone (introduced in chapter 16).

To keep concerns separated, you'll create a subnet for the NAT gateway as follows:

```
SubnetPublicNAT:
  Type: 'AWS::EC2::Subnet'
  Properties:
    AvailabilityZone: !Select [0, !GetAZs '']
    CidrBlock: '10.0.0.0/24'
    MapPublicIpOnLaunch: true
    VpcId: !Ref VPC
    Tags:
    - Key: Name
      Value: 'Public NAT'
RouteTablePublicNAT:
  Type: 'AWS::EC2::RouteTable'
  Properties:
    VpcId: !Ref VPC
RouteTableAssociationPublicNAT:
  Type: 'AWS::EC2::SubnetRouteTableAssociation'
  Properties:
    SubnetId: !Ref SubnetPublicNAT
    RouteTableId: !Ref RouteTablePublicNAT
RoutePublicNATToInternet:
  Type: 'AWS::EC2::Route'
  Properties:
    RouteTableId: !Ref RouteTablePublicNAT
    DestinationCidrBlock: '0.0.0.0/0'
    GatewayId: !Ref InternetGateway
  DependsOn: VPCGatewayAttachment
NetworkAclPublicNAT:
  Type: 'AWS::EC2::NetworkAcl'
  Properties:
    VpcId: !Ref VPC
SubnetNetworkAclAssociationPublicNAT:
  Type: 'AWS::EC2::SubnetNetworkAclAssociation'
  Properties:
    SubnetId: !Ref SubnetPublicNAT
    NetworkAclId: !Ref NetworkAclPublicNAT
```

10.0.0.0/24 is the NAT subnet.

The NAT subnet is public with a route to the internet.

We need a bunch of NACL rules to make the NAT gateway work.

To allow all VPC subnets to use the NAT gateway for HTTP and HTTPS, perform the following steps:

1. Allow inbound ports 80 (HTTP) and 443 (HTTPS) from 10.0.0.0/16.
2. Allow outbound ephemeral ports to 10.0.0.0/16.

To allow the NAT gateway to reach out to the internet on HTTP and HTTPS, perform these steps:

- Allow outbound ports 80 (HTTP) and 443 (HTTPS) to 0.0.0.0/0.
- Allow inbound ephemeral ports from 0.0.0.0/0.

Find the CloudFormation implementation of the previous NACL rules next:

```
NetworkAclEntryInPublicNATHTTP:                    <-- Allows inbound HTTP from all VPC subnets
  Type: 'AWS::EC2::NetworkAclEntry'
  Properties:
    NetworkAclId: !Ref NetworkAclPublicNAT
    RuleNumber: '100'
    Protocol: '6'
    PortRange:
      From: '80'
      To: '80'
    RuleAction: 'allow'
    Egress: 'false'
    CidrBlock: '10.0.0.0/16'
NetworkAclEntryInPublicNATHTTPS:                   <-- Allows inbound HTTPS from all VPC subnets
  Type: 'AWS::EC2::NetworkAclEntry'
  Properties:
    NetworkAclId: !Ref NetworkAclPublicNAT
    RuleNumber: '110'
    Protocol: '6'
    PortRange:
      From: '443'
      To: '443'
    RuleAction: 'allow'
    Egress: 'false'
    CidrBlock: '10.0.0.0/16'
NetworkAclEntryInPublicNATEphemeralPorts:          <-- Ephemeral ports
  Type: 'AWS::EC2::NetworkAclEntry'
  Properties:
    NetworkAclId: !Ref NetworkAclPublicNAT
    RuleNumber: '200'
    Protocol: '6'
    PortRange:
      From: '1024'
      To: '65535'
    RuleAction: 'allow'
    Egress: 'false'
    CidrBlock: '0.0.0.0/0'
NetworkAclEntryOutPublicNATHTTP:                   <-- Allows outbound HTTP to everywhere
  Type: 'AWS::EC2::NetworkAclEntry'
  Properties:
    NetworkAclId: !Ref NetworkAclPublicNAT
    RuleNumber: '100'
    Protocol: '6'
    PortRange:
      From: '80'
      To: '80'
    RuleAction: 'allow'
    Egress: 'true'
    CidrBlock: '0.0.0.0/0'
NetworkAclEntryOutPublicNATHTTPS:                  <-- Allows outbound HTTPS to everywhere
  Type: 'AWS::EC2::NetworkAclEntry'
  Properties:
    NetworkAclId: !Ref NetworkAclPublicNAT
    RuleNumber: '110'
```

```
    Protocol: '6'
    PortRange:
      From: '443'
      To: '443'
    RuleAction: 'allow'
    Egress: 'true'
    CidrBlock: '0.0.0.0/0'
NetworkAclEntryOutPublicNATEphemeralPorts:
  Type: 'AWS::EC2::NetworkAclEntry'
  Properties:
    NetworkAclId: !Ref NetworkAclPublicNAT
    RuleNumber: '200'
    Protocol: '6'
    PortRange:
      From: '1024'
      To: '65535'
    RuleAction: 'allow'
    Egress: 'true'
    CidrBlock: '10.0.0.0/16'
```

Ephemeral ports

Finally, you can add the NAT gateway itself. The NAT gateway comes with a fixed public IP address (also called an Elastic IP address). All traffic that is routed through the NAT gateway will come from this IP address. We also add a route to the backend's route table to route traffic to 0.0.0.0/0 via the NAT gateway, as shown here:

```
EIPNatGateway:
  Type: 'AWS::EC2::EIP'
  Properties:
    Domain: 'vpc'
NatGateway:
  Type: 'AWS::EC2::NatGateway'
  Properties:
    AllocationId: !GetAtt 'EIPNatGateway.AllocationId'
    SubnetId: !Ref SubnetPublicNAT
RoutePrivateBackendToInternet:
  Type: 'AWS::EC2::Route'
  Properties:
    RouteTableId: !Ref RouteTablePrivateBackend
    DestinationCidrBlock: '0.0.0.0/0'
    NatGatewayId: !Ref NatGateway
```

A static public IP address is used for the NAT gateway.

The NAT gateway is placed into the private subnet and associated with the static public IP address.

Route from the Apache subnet to the NAT gateway

Last, one small tweak to the already covered backend EC2 instance is needed to keep dependencies up-to-date, shown here:

```
Backend:
  Type: 'AWS::EC2::Instance'
  Properties:
    # [...]
  DependsOn: RoutePrivateBackendToInternet
```

Helps CloudFormation to understand that the route is needed before the EC2 instance can be created

WARNING The NAT gateway included in the example is not covered by the Free Tier. The NAT gateway will cost you $0.045 per hour and $0.045 per GB of

data processed when creating the stack in the US East (N. Virginia) region. Go to https://aws.amazon.com/vpc/pricing/ to have a look at the current prices.

Now you're ready to create the CloudFormation stack with the template located at https://s3.amazonaws.com/awsinaction-code3/chapter05/vpc.yaml by clicking the CloudFormation Quick-Create Link (http://mng.bz/jmR9). Once you've done so, copy the `ProxyPublicIpAddress` output and open it in your browser. You'll see an Apache test page.

Cleaning up

Don't forget to delete your stack after finishing this section to clean up all used resources. Otherwise, you'll likely be charged for the resources you use.

Summary

- AWS is a shared-responsibility environment in which security can be achieved only if you and AWS work together. You're responsible for securely configuring your AWS resources and your software running on EC2 instances, whereas AWS protects buildings and host systems.
- Keeping your software up-to-date is key and can be automated.
- The Identity and Access Management (IAM) service provides everything needed for authentication and authorization with the AWS API. Every request you make to the AWS API goes through IAM to check whether the request is allowed. IAM controls who can do what in your AWS account. To protect your AWS account, grant only those permissions that your users and roles need.
- Traffic to or from AWS resources like EC2 instances can be filtered based on protocol, port, and source or destination.
- A VPC is a private network in AWS where you have full control. With VPCs, you can control routing, subnets, NACLs, and gateways to the internet or your company network via a VPN. A NAT gateway enables access to the internet from private subnets.
- You should separate concerns in your network to reduce potential damage, for example, if one of your subnets is hacked. Keep every system in a private subnet that doesn't need to be accessed from the public internet, to reduce your attackable surface.

6 Automating operational tasks with Lambda

This chapter covers

- Creating a Lambda function to perform periodic health checks of a website
- Triggering a Lambda function with EventBridge events to automate DevOps tasks
- Searching through your Lambda function's logs with CloudWatch
- Monitoring Lambda functions with CloudWatch alarms
- Configuring IAM roles so Lambda functions can access other services

This chapter is about adding a new tool to your toolbox. The tool we're talking about, AWS Lambda, is as flexible as a Swiss Army knife. You don't need a virtual machine to run your own code anymore, because AWS Lambda offers execution environments for C#/.NET Core, Go, Java, JavaScript/Node.js, Python, and Ruby. All you have to do is implement a function, upload your code, and configure the execution environment. Afterward, your code is executed within a fully managed computing environment. AWS Lambda is well integrated with all parts of AWS,

enabling you to easily automate operations tasks within your infrastructure. We use AWS to automate our infrastructure regularly, such as to add and remove instances to a container cluster based on a custom algorithm and to process and analyze log files.

AWS Lambda offers a maintenance-free and highly available computing environment. You no longer need to install security updates, replace failed virtual machines, or manage remote access (such as SSH or RDP) for administrators. On top of that, AWS Lambda is billed by invocation. Therefore, you don't have to pay for idling resources that are waiting for work (e.g., for a task triggered once a day).

In our first example, you will create a Lambda function that performs periodic health checks for your website. This will teach you how to use the Management Console to get started with AWS Lambda quickly. In our second example, you will learn how to write your own Python code and deploy a Lambda function in an automated way using CloudFormation, which we introduced in chapter 4. Your Lambda function will automatically add a tag to newly launched EC2 instances. At the end of the chapter, we'll show you additional use cases like building web applications and Internet of Things (IoT) backends and processing data with AWS Lambda.

Examples are almost all covered by the Free Tier

The examples in this chapter are mostly covered by the Free Tier. There is one exception: AWS CloudTrail. In case you already created a trail in your account, additional charges—most likely just a few cents—will apply. For details, please see https://aws.amazon.com/cloudtrail/pricing/.

You will find instructions on how to clean up the examples at the end of each section.

You may be asking a more basic question: what exactly is AWS Lambda? Before diving into our first real-world example, let us introduce you, briefly, to Lambda and explain why it's often mentioned in the context of an architecture that's being called *serverless*.

6.1 Executing your code with AWS Lambda

Computing capacity is available at different layers of abstraction on AWS: virtual machines, containers, and functions. You learned about the virtual machines offered by Amazon's EC2 service in chapter 3. Containers offer another layer of abstraction on top of virtual machines; you will learn about containers in chapter 18. AWS Lambda provides computing power, as well, but in a fine-grained manner: it is an execution environment for small functions, rather than a full-blown operating system or container.

6.1.1 What is serverless?

When reading about AWS Lambda, you might have stumbled upon the term *serverless*. In his book *Serverless Architectures on AWS* (Manning, 2022; http://mng.bz/wyr5), Peter Sbarski summarizes the confusion created by this catchy and provocative phrase:

> *[…] the word serverless is a bit of a misnomer. Whether you use a compute service such as AWS Lambda to execute your code, or interact with an API, there are still servers running in the background. The difference is that these servers are hidden from you. There's no infrastructure for you to think about and no way to tweak the underlying operating system. Someone else takes care of the nitty-gritty details of infrastructure management, freeing your time for other things.*
>
> —Peter Sbarski

We define a serverless system as one that meets the following criteria:

- No need to manage and maintain virtual machines
- Fully managed service offering scalability and high availability
- Billed per request and by resource consumption

AWS Lambda certainly fits these definitions and is indeed a serverless platform in that AWS handles server configurations and management so the server is essentially invisible to you and takes care of itself. AWS is not the only provider offering a serverless platform. Google (Cloud Functions) and Microsoft (Azure Functions) are other competitors in this area. If you would like to read more about serverless, here is a free chapter from *Serverless Architectures on AWS*, second edition: http://mng.bz/qoEx.

6.1.2 Running your code on AWS Lambda

AWS Lambda is the basic building block of the serverless platform provided by AWS. The first step in the process is to run your code on Lambda instead of on your own server.

As shown in figure 6.1, to execute your code with AWS Lambda, follow these steps:

1. Write the code.
2. Upload your code and its dependencies (such as libraries or modules).
3. Create a function determining the runtime environment and configuration.
4. Invoke the function to execute your code in the cloud.

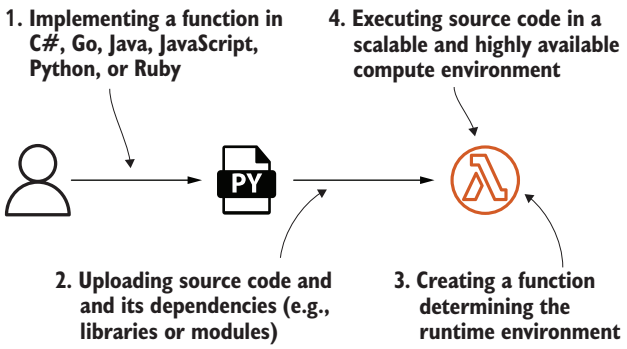


Figure 6.1 Executing code with AWS Lambda

Currently, AWS Lambda offers runtime environments for the following languages:

- C#/.NET Core
- Go
- Java
- JavaScript/Node.js
- Python
- Ruby

Besides that, you can bring your own runtime by using a custom runtime. In theory, a custom runtime supports any programming language. You need to bring your own container image and follow conventions for initializing the function and processing tasks. Doing so adds extra complexity, so we recommend going with one of the available runtimes.

Next, we will compare AWS Lambda with EC2 virtual machines.

6.1.3 Comparing AWS Lambda with virtual machines (Amazon EC2)

What is the difference between using AWS Lambda and virtual machines? First is the granularity of virtualization. Virtual machines provide a full operating system for running one or multiple applications. In contrast, AWS Lambda offers an execution environment for a single function, a small part of an application.

Furthermore, Amazon EC2 offers virtual machines as a service, but you are responsible for operating them in a secure, scalable, and highly available way. Doing so requires you to put a substantial amount of effort into maintenance. In contrast, when building with Lambda, AWS manages the underlying infrastructure for you and provides a production-ready infrastructure.

Beyond that, AWS Lambda is billed per execution, and not per second like when a virtual machine is running. You don't have to pay for unused resources that are waiting for requests or tasks. For example, running a script to check the health of a website every five minutes on a virtual machine would cost you about $3.71 per month. Executing the same health check with AWS Lambda will cost about $0.04 per month.

Table 6.1 compares AWS Lambda and virtual machines in detail. You'll find a discussion of AWS Lambda's limitations at the end of the chapter.

Table 6.1 AWS Lambda compared to Amazon EC2

	AWS Lambda	Amazon EC2
Granularity of virtualization	Small piece of code (a function).	An entire operating system.
Scalability	Scales automatically. A throttling limit prevents you from creating unwanted charges accidentally and can be increased by AWS support if needed.	As you will learn in chapter 17, using an Auto Scaling group allows you to scale the number of EC2 instances serving requests automatically, but configuring and monitoring the scaling activities is your responsibility.

Table 6.1 AWS Lambda compared to Amazon EC2 *(continued)*

	AWS Lambda	Amazon EC2
High availability	Fault tolerant by default. The computing infrastructure spans multiple machines and data centers.	Virtual machines are not highly available by default. Nevertheless, as you will learn in chapter 13, it is possible to set up a highly available infrastructure based on EC2 instances as well.
Maintenance effort	Almost zero. You need only to configure your function.	You are responsible for maintaining all layers between your virtual machine's operating system and your application's runtime environment.
Deployment effort	Almost zero due to a well-defined API	Rolling out your application to a fleet of virtual machines is a challenge that requires tools and know-how.
Pricing model	Pay per request as well as execution time and allocated memory	Pay for operating hours of the virtual machines, billed per second

Looking for limitations and pitfalls of AWS Lambda? Stay tuned: you will find a discussion of Lambda's limitations at the end of the chapter.

That's all you need to know about AWS Lambda to be able to go through the first real-world example. Are you ready?

6.2 Building a website health check with AWS Lambda

Are you responsible for the uptime of a website or application? We do our best to make sure our blog https://cloudonaut.io is accessible 24/7. An external health check acts as a safety net, making sure we, and not our readers, are the first to know when our blog goes down. AWS Lambda is the perfect choice for building a website health check, because you do not need computing resources constantly but only every few minutes for a few milliseconds. This section guides you through setting up a health check for your website based on AWS Lambda.

In addition to AWS Lambda, we are using the Amazon CloudWatch service for this example. Lambda functions publish metrics to CloudWatch by default. Typically, you inspect metrics using charts and create alarms by defining thresholds. For example, a metric could count failures during the function's execution. On top of that, EventBridge provides events that can be used to trigger Lambda functions as well. Here we are using a rule to publish an event every five minutes.

As shown in figure 6.2, your website health check will consist of the following three parts:

1. *Lambda function*—Executes a Python script that sends an HTTP request to your website (e.g., `GET https://cloudonaut.io`) and verifies that the response includes specific text (such as `cloudonaut`).
2. *EventBridge rule*—Triggers the Lambda function every five minutes. This is comparable to the cron service on Linux.

3 *Alarm*—Monitors the number of failed health checks and notifies you via email whenever your website is unavailable.

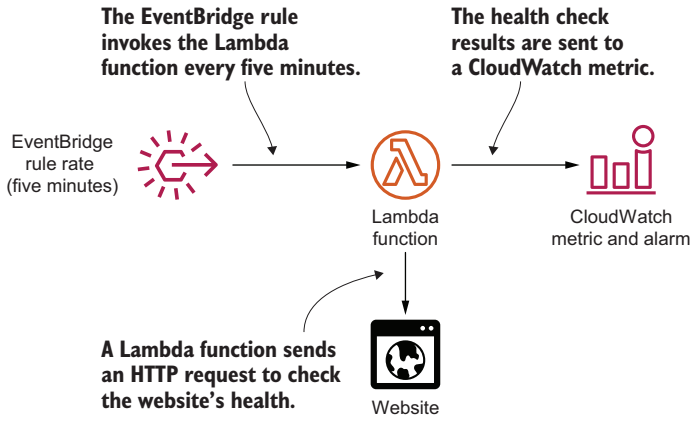


Figure 6.2 The Lambda function performing the website health check is executed every five minutes by a scheduled event. Errors are reported to CloudWatch.

You will use the Management Console to create and configure all the necessary parts manually. In our opinion, this is a simple way to get familiar with AWS Lambda. You will learn how to deploy a Lambda function in an automated way in section 6.3.

6.2.1 Creating a Lambda function

The following step-by-step instructions guide you through setting up a website health check based on AWS Lambda. Open AWS Lambda in the Management Console: https://console.aws.amazon.com/lambda/home. Click Create a Function to start the Lambda function wizard, as shown in figure 6.3.

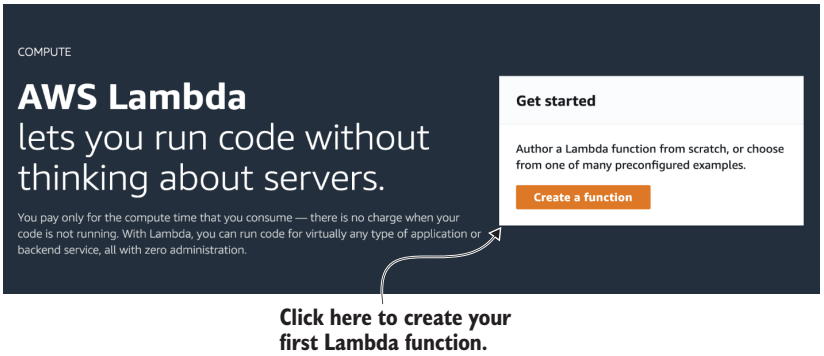


Figure 6.3 Welcome screen: Ready to create your first Lambda function

AWS provides blueprints for various use cases, including the code and the Lambda function configuration. We will use one of these blueprints to create a website health check. Select Use a Blueprint, and search for canary. Next, choose the lambda-canary blueprint. Figure 6.4 illustrates the details.

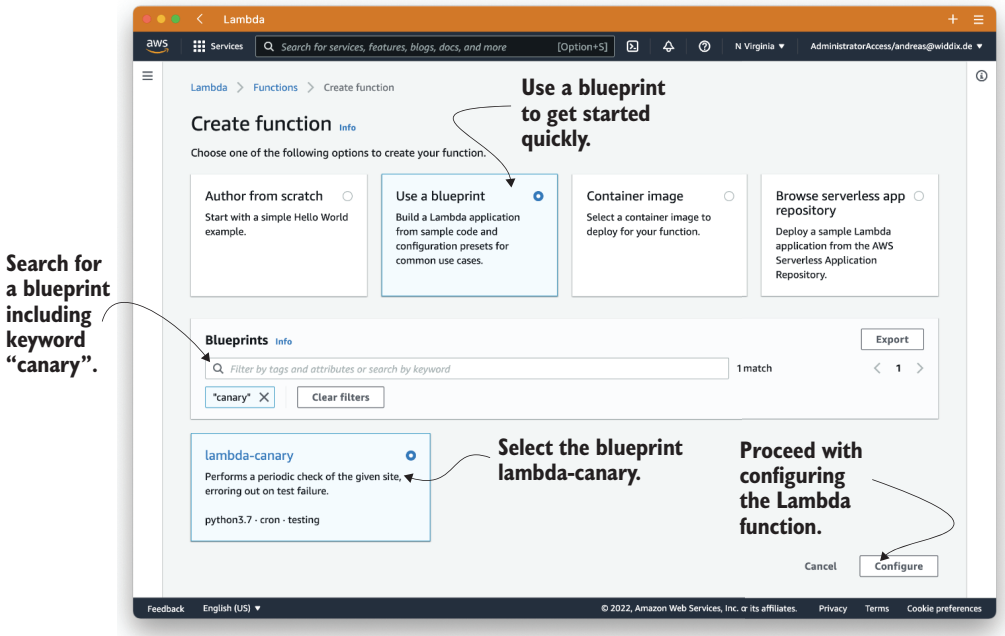


Figure 6.4 Creating a Lambda function based on a blueprint provided by AWS

In the next step of the wizard, you need to specify a name for your Lambda function, as shown in figure 6.5. The function name needs to be unique within your AWS account and the current region US East (N. Virginia). In addition, the name is limited to 64 characters. To invoke a function via the API, you need to provide the function name. Type in website-health-check as the name for your Lambda function.

Continue with creating an IAM role. Select Create a New Role with Basic Lambda Permissions, as shown in figure 6.5, to create an IAM role for your Lambda function. You will learn how your Lambda function uses the IAM role in section 6.3.

Next, configure the scheduled event that will trigger your health check repeatedly. We will use an interval of five minutes in this example. Figure 6.6 shows the settings you need:

1. Select Create a New Rule to create a scheduled event rule.
2. Type in website-health-check as the name for the rule.
3. Enter a description that will help you to understand what is going on if you come back later.

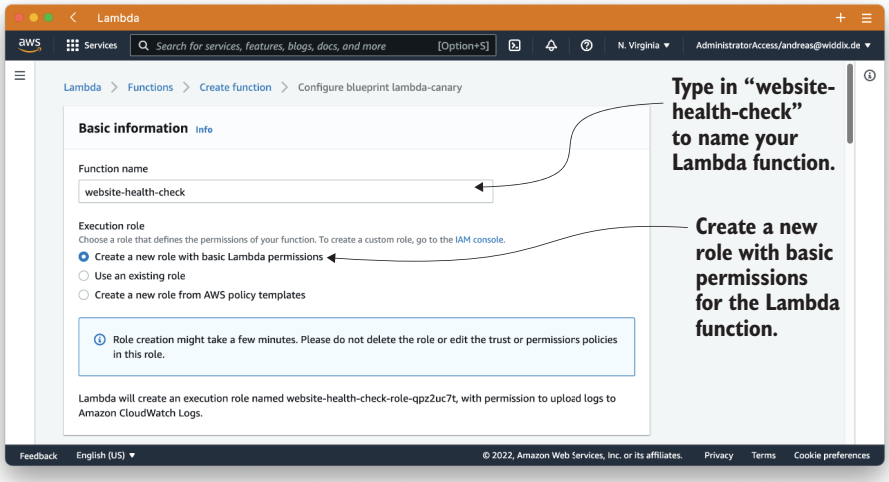


Figure 6.5 Creating a Lambda function: Choose a name and define an IAM role.

4 Select Schedule Expression as the rule type. You will learn about the other option, Event Pattern, at the end of this chapter.
5 Use rate(5 minutes) as the schedule expression.

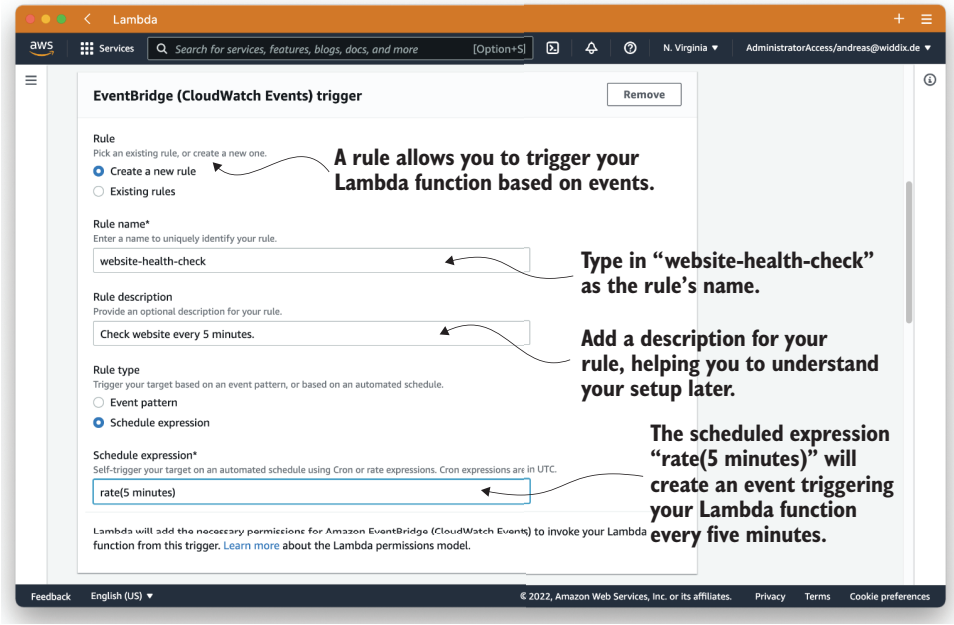


Figure 6.6 Configuring a scheduled event triggering your Lambda function every five minutes

Define recurring tasks using a *schedule expression* in form of `rate($value $unit)`. For example, you could trigger a task every five minutes, every hour, or once a day. `$value` needs to be a positive integer value. Use `minute`, `minutes`, `hour`, `hours`, `day`, or `days` as the unit. For example, instead of triggering the website health check every five minutes, you could use `rate(1 hour)` as the schedule expression to execute the health check every hour. Note that frequencies of less than one minute are not supported.

It is also possible to use the crontab format, shown here, when defining a schedule expression:

```
cron($minutes $hours $dayOfMonth $month $dayOfWeek $year)

 # Invoke a Lambda function at 08:00am (UTC) everyday
 cron(0 8 * * ? *)

 # Invoke a Lambda function at 04:00pm (UTC) every Monday to Friday
 cron(0 16 ? * MON-FRI *)
```

See "Schedule Expressions Using Rate or cron" at http://mng.bz/7ZnQ for more details.

Your Lambda function is missing an integral part: the code. Because you are using a blueprint, AWS has inserted the Python code implementing the website health check for you, as shown in figure 6.7.

The Python code references two environment variables: `site` and `expected`. Environment variables are commonly used to dynamically pass settings to your function. An environment variable consists of a key and a value. Specify the following environment variables for your Lambda function:

- `site`—Contains the URL of the website you want to monitor. Use https://cloudonaut.io if you do not have a website to monitor yourself.
- `expected`—Contains a text snippet that must be available on your website. If the function doesn't find this text, the health check fails. Use `cloudonaut` if you are using https://cloudonaut.io as your website.

The Lambda function is reading the environment variables during its execution, as shown next:

```
SITE = os.environment['site']
 EXPECTED = os.environment['expected']
```

After defining the environment variables for your Lambda function, click the Create Function button at the bottom of the screen.

Congratulations—you have successfully created a Lambda function. Every five minutes, the function is invoked automatically and executes a health check for your website.

You could use this approach to automate recurring tasks, like checking the status of a system, cleaning up unused resources like EBS snapshots, or to send recurring reports.

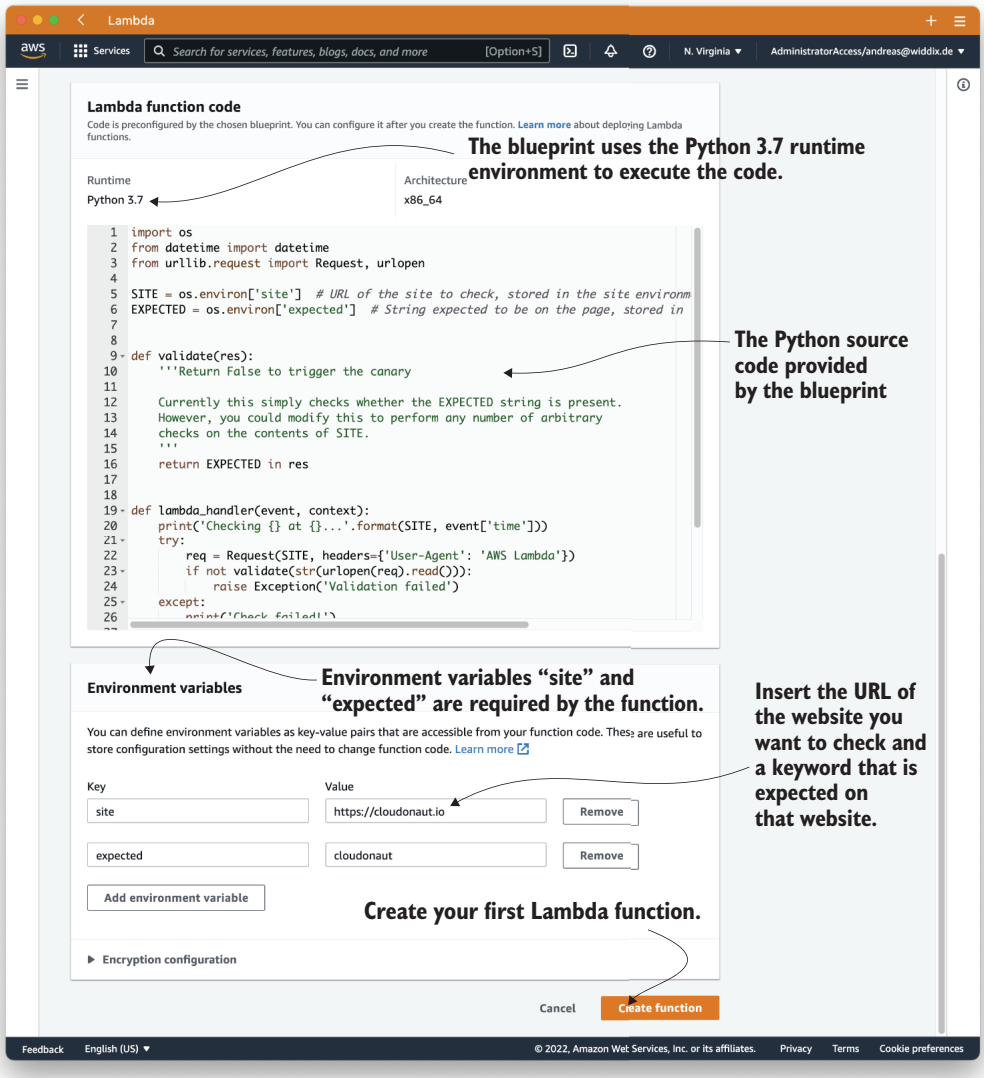


Figure 6.7 The predefined code implementing the website health check and environment variables to pass settings to the Lambda function

So far, you have used a predefined template. Lambda gets much more powerful when you write your own code. But before we look into that, you will learn how to monitor your Lambda function and get notified via email whenever the health check fails.

6.2.2 Use CloudWatch to search through your Lambda function's logs

How do you know whether your website health check is working correctly? How do you even know whether your Lambda function has been executed? It is time to look at

how to monitor a Lambda function. You will learn how to access your Lambda function's log messages first. Afterward, you will create an alarm notifying you if your function fails.

Open the Monitor tab in the details view of your Lambda function. You will find a chart illustrating the number of times your function has been invoked. Reload the chart after a few minutes, in case the chart isn't showing any invocations. To go to your Lambda function's logs, click View Logs in CloudWatch, as shown in figure 6.8.

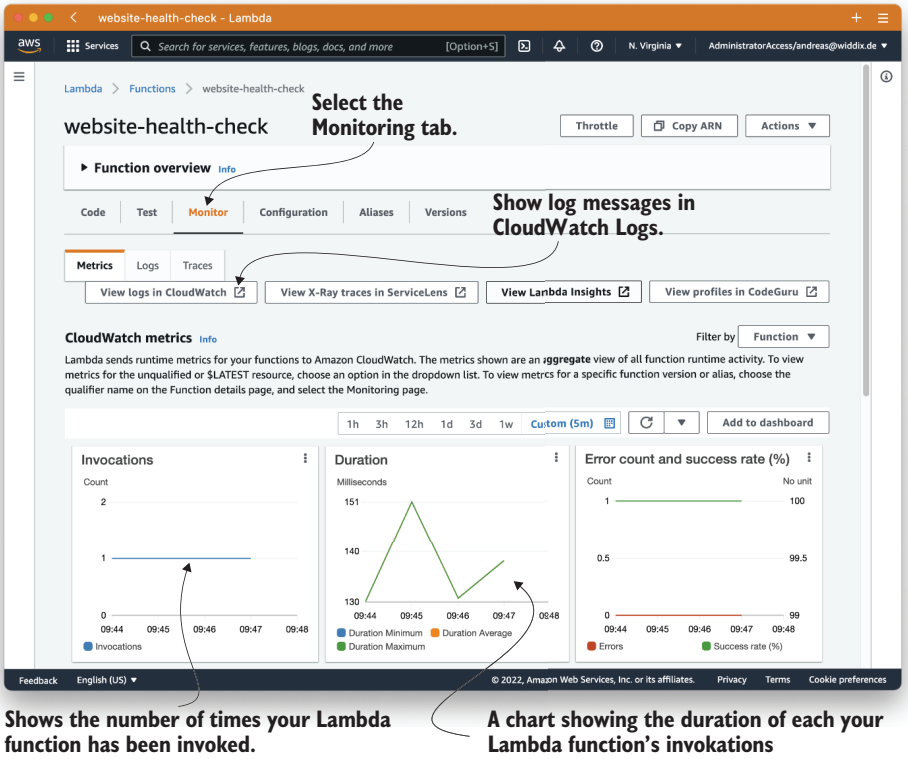


Figure 6.8 Monitoring overview: Get insights into your Lambda function's invocations.

By default, your Lambda function sends log messages to CloudWatch. Figure 6.9 shows the log group named /aws/lambda/website-health-check that was created automatically and collects the logs from your function. Typically, a log group contains multiple log streams, allowing the log group to scale. Click Search Log Group to view the log messages from all streams in one view.

All log messages are presented in the overview of log streams, as shown in figure 6.10. You should be able to find a log message `Check passed!`, indicating that the website health check was executed and passed successfully, for example.

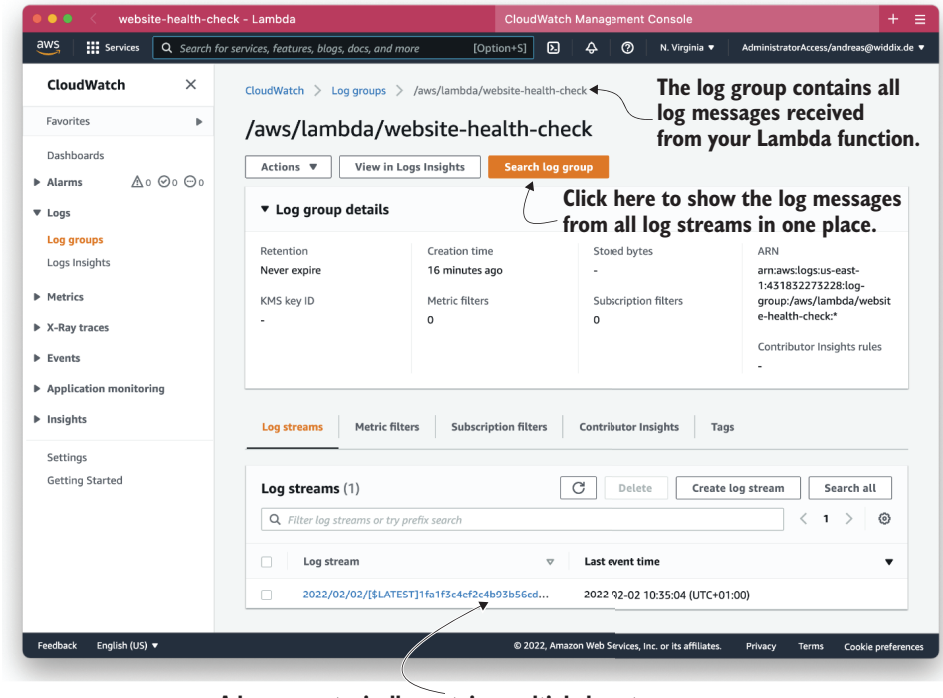


Figure 6.9 A log group collects log messages from a Lambda function stored in multiple log streams.

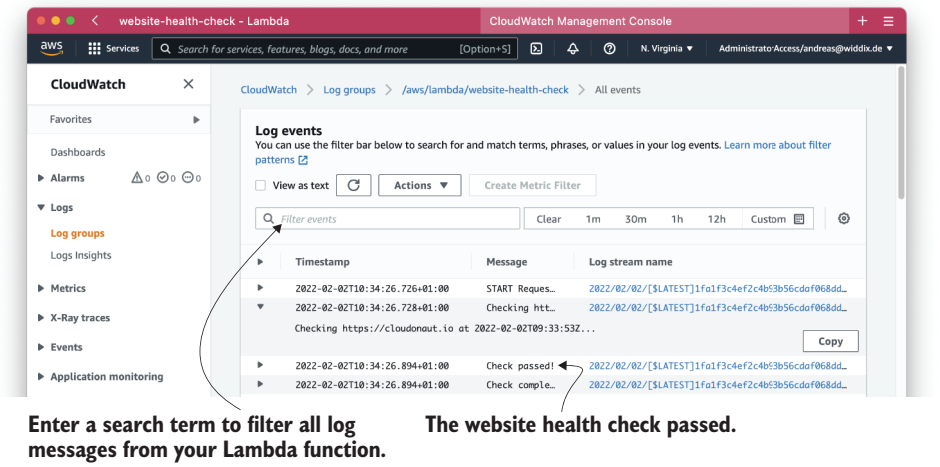


Figure 6.10 CloudWatch shows the log messages of your Lambda function.

The log messages show up after a delay of a few minutes. Reload the table if you are missing any log messages.

Being able to search through log messages in a centralized place is handy when debugging Lambda functions, especially if you are writing your own code. When using Python, you can use `print` statements or use the `logging` module to send log messages to CloudWatch.

6.2.3 Monitoring a Lambda function with CloudWatch metrics and alarms

The Lambda function now checks the health of your website every five minutes, and a log message with the result of each health check is written to CloudWatch. But how do you get notified via email if the health check fails? Each Lambda function publishes metrics to CloudWatch by default. Table 6.2 shows important metrics. Check out http://mng.bz/m298 for a complete list of metrics.

Table 6.2 The CloudWatch metrics published by each Lambda function

Name	Description
Invocations	Counts the number of times a function is invoked. Includes successful and failed invocations.
Errors	Counts the number of times a function failed due to errors inside the function, for example, exceptions or timeouts.
Duration	Measures how long the code takes to run, from the time when the code starts executing to when it stops executing.
Throttles	As discussed at the beginning of the chapter, there is a limit for how many copies of your Lambda function can run at one time. This metric counts how many invocations have been throttled due to reaching this limit. Contact AWS support to increase the limit, if needed.

Whenever the website health check fails, the Lambda function returns an error, which increases the count of the Errors metric. You will create an alarm notifying you via email whenever this metric counts more than zero errors. In general, we recommend creating an alarm on the Errors and Throttles metrics to monitor your Lambda functions.

The following steps guide you through creating a CloudWatch alarm to monitor your website health checks. Your Management Console still shows the CloudWatch service. Select Alarms from the subnavigation menu. Next, click Create Alarm, as shown in figure 6.11.

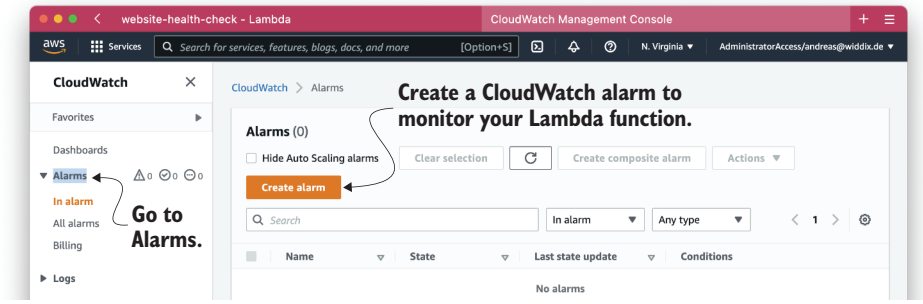


Figure 6.11 Starting the wizard to create a CloudWatch alarm to monitor a Lambda function

The following three steps guide you through the process of selecting the Error metric of your `website-health-check` Lambda function, as illustrated in figure 6.12:

1. Click Select Metric.
2. Choose the Lambda namespace.
3. Select metrics with dimension Function Name.

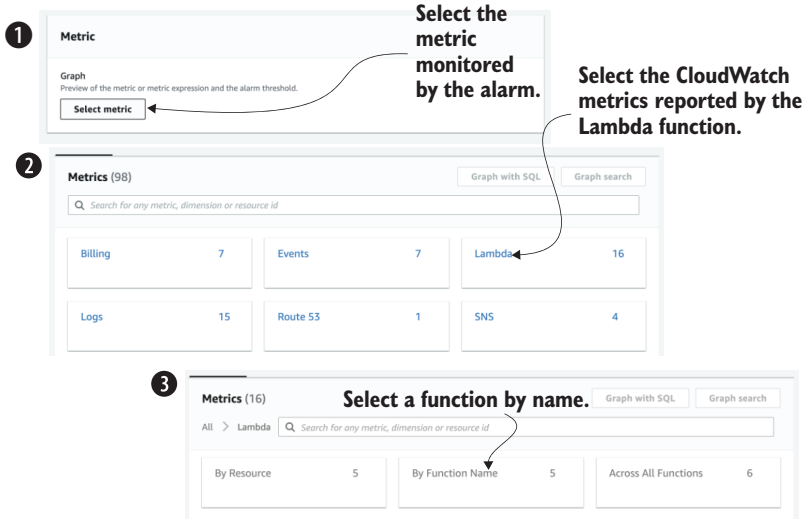


Figure 6.12 Searching the CloudWatch metric to create an alarm

Last but not least, select the Error metric belonging to the Lambbda function `website-health-check` as shown in figure 6.13. Proceed by clicking the Select Metric button.

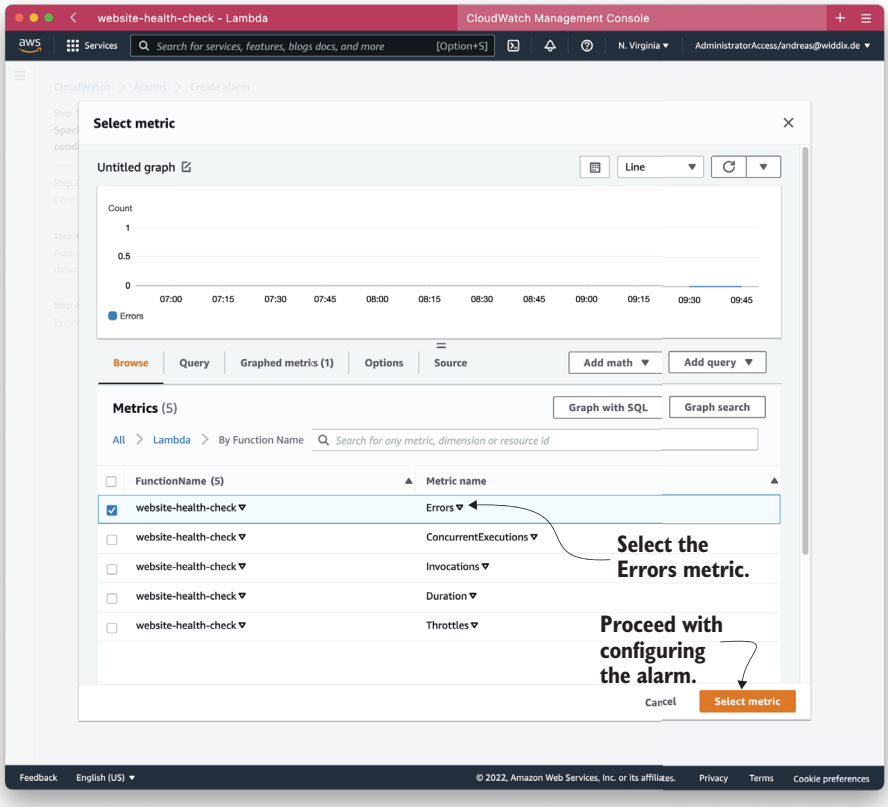


Figure 6.13 Selecting the Error metric of the Lambda function

Next, you need to configure the alarm, as shown in figure 6.14. To do so, specify the statistic, period, and threshold as follows:

1. Choose the statistic Sum to add up all the errors that occurred during the evaluation period.
2. Define an evaluation period of five minutes.
3. Select the Static Threshold option.
4. Choose the Greater operator.
5. Define a threshold of zero.
6. Click the Next button to proceed.

In other words, the alarm state will change to `ALARM` when the Lambda function reports any errors during the evaluation period of five minutes. Otherwise, the alarm state will be `OK`.

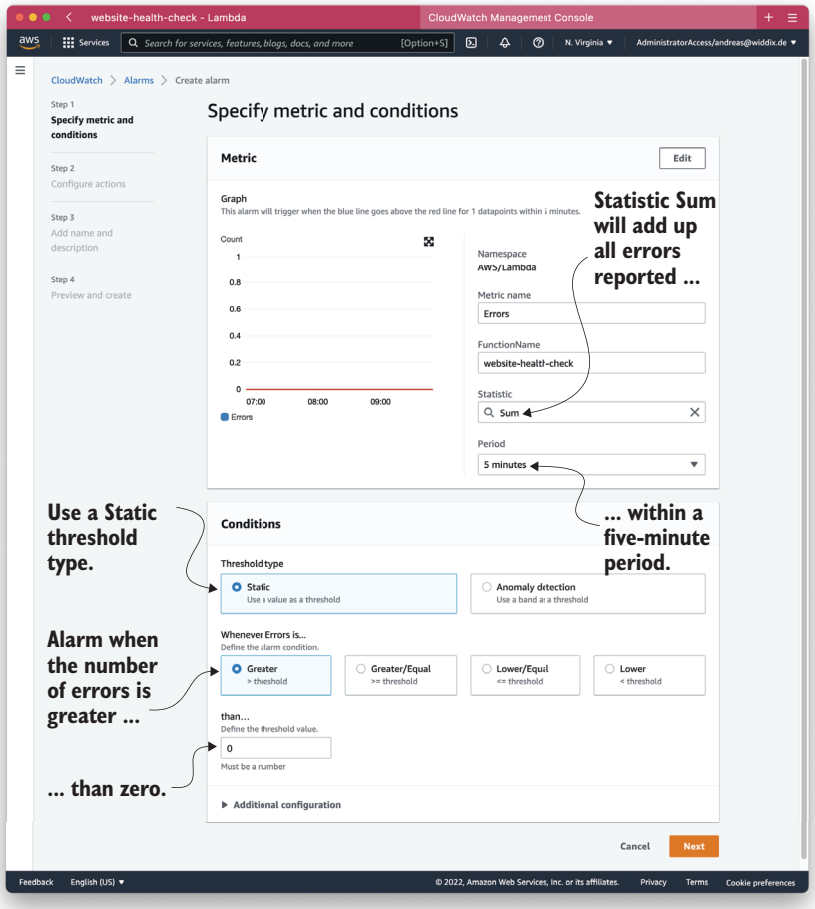


Figure 6.14 Selecting and preparing the metric view for the alarm

The next step, illustrated in figure 6.15, configures the alarm actions so you will be notified via e-mail:

1. Select In Alarm as the state trigger.
2. Create a new SNS topic by choosing Create New Topic.
3. Type in website-health-check as the name for the SNS topic.
4. Enter your email address.
5. Click the Next button to proceed.

In the following step, you need to enter a name for the CloudWatch alarm. Type in website-health-check-error. Afterward, click the Next button to proceed.

After reviewing the configuration, click the Create Alarm button. Shortly after creating the alarm, you will receive an email including a confirmation link for SNS. Check your inbox and click the link to confirm your subscription to the notification list.

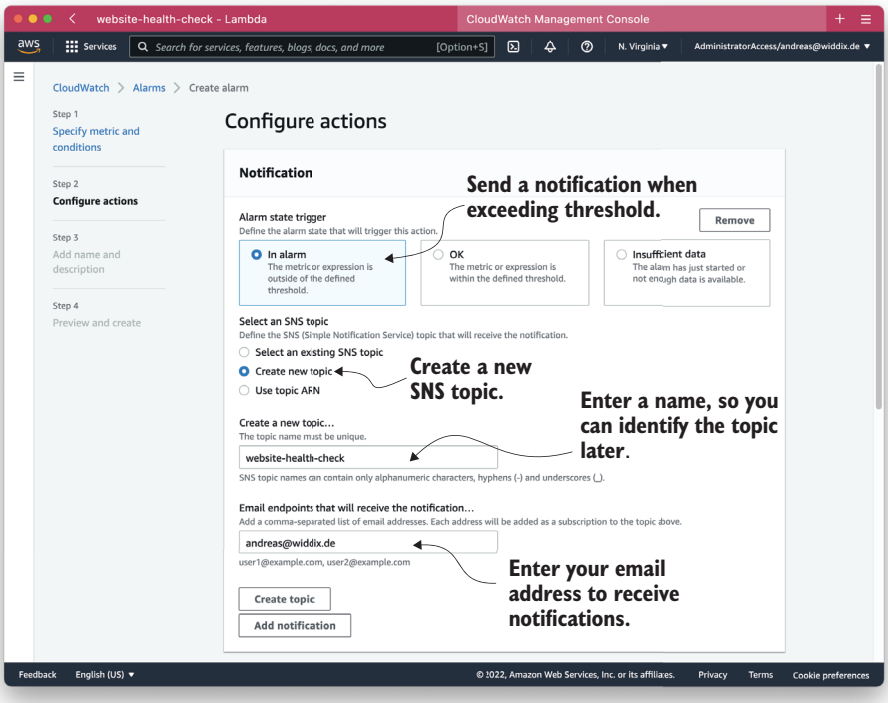


Figure 6.15 Creating an alarm by defining a threshold and defining an alarm action to send notifications via email

To test the alarm, go back to the Lambda function and modify the environment variable `expected`. For example, modify the value from `cloudonaut` to `FAILURE`. This will cause the health check to fail, because the word `FAILURE` does not appear on the website. It might take up to 15 minutes before you receive a notification about the failed website health check via email.

Cleaning up

Open your Management Console and follow these steps to delete all the resources you have created during this section:

1. Go to the AWS Lambda service and delete the function named website-health-check.
2. Open the AWS CloudWatch service, select Logs from the subnavigation options, and delete the log group /aws/lambda/website-health-check.
3. Go to the EventBridge service, select Rules from the subnavigation menu, and delete the rule website-health-check.

4 Open the CloudWatch service, select Alarms from the subnavigation menu, and delete the alarm website-health-check-error.
5 Jump to the AWS IAM service, select Roles from the subnavigation menu, and delete the role whose name starts with website-health-check-role-.
6 Go to the SNS service, select Topics from the subnavigation menu, and delete the rule website-health-check.

6.2.4 Accessing endpoints within a VPC

As illustrated in figure 6.16, by default Lambda functions run outside your networks defined with VPC. However, Lambda functions are connected to the internet and, therefore, are able to access other services. That's exactly what you have been doing when creating a website health check: the Lambda function was sending HTTP requests over the internet.

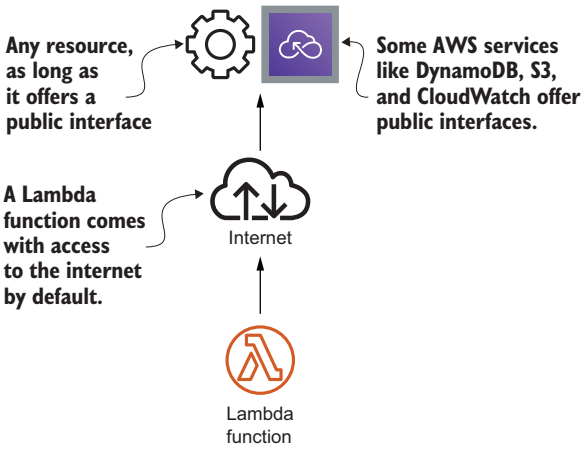


Figure 6.16 By default a Lambda function is connected to the internet and running outside your VPCs.

So, what do you do when you have to reach a resource running in a private network within your VPC, for example, if you want to run a health check for an internal website? If you add network interfaces to your Lambda function, the function can access resources within your VPCs, as shown in figure 6.17.

To do so, you have to define the VPC and the subnets, as well as security groups for your Lambda function. See "Configuring a Lambda Function to Access Resources in an Amazon VPC" at http://mng.bz/5m67 for more details. We have been using the ability to access resources within a VPC to access databases in various projects.

We do recommend connecting a Lambda function to a VPC only when absolutely necessary because it introduces additional complexity. However, being able to connect with resources within your private networks is very interesting, especially when integrating with legacy systems.

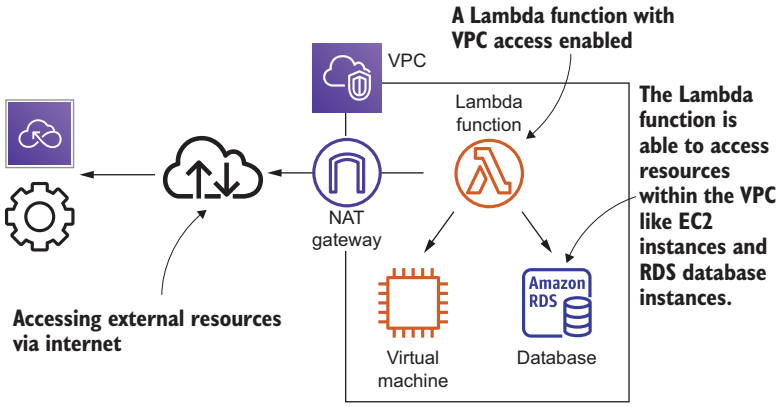


Figure 6.17 Deploying a Lambda function into your VPC allows you to access internal resources (such as database, virtual machines, and so on).

6.3 Adding a tag containing the owner of an EC2 instance automatically

After using one of AWS's predefined blueprints to create a Lambda function, you will implement the function from scratch in this section. We are strongly focused on setting up your cloud infrastructure in an automated way. That's why you will learn how to deploy a Lambda function and all its dependencies without needing the Management Console.

Are you working in an AWS account together with your colleagues? Have you ever wondered who launched a certain EC2 instance? Sometimes you need to find out the owner of an EC2 instance for the following reasons:

- Double-checking whether it is safe to terminate an unused instance without losing relevant data
- Reviewing changes to an instance's configuration with its owner (such as making changes to the firewall configuration)
- Attributing costs to individuals, projects, or departments
- Restricting access to an instance (e.g., so only the owner is allowed to terminate an instance)

Adding a tag that states who owns an instance solves all these use cases. A tag can be added to an EC2 instance or almost any other AWS resource and consists of a key and a value. You can use tags to add information to a resource, filter resources, attribute costs to resources, and restrict access. See "Tag your Amazon EC2 resources" at http://mng.bz/69oR for more details.

It is possible to add tags specifying the owner of an EC2 instance manually. But, sooner or later, someone will forget to add the owner tag. There is a better solution! In the following section, you will implement and deploy a Lambda function that automatically adds a tag containing the name of the user who launched an EC2 instance.

But how do you execute a Lambda function every time an EC2 instance is launched so that you can add the tag?

6.3.1 Event-driven: Subscribing to EventBridge events

EventBridge is an event bus used by AWS, third-party vendors, and customers like you, to publish and subscribe to events. In this example, you will create a rule to listen for events from AWS. Whenever something changes in your infrastructure, an event is generated in near real time and the following things occur:

- CloudTrail emits an event for every call to the AWS API if CloudTrail is enabled in the AWS account and region.
- EC2 emits events whenever the state of an EC2 instances changes (such as when the state changes from `Pending` to `Running`).
- AWS emits an event to notify you of service degradations or downtimes.

Whenever you launch a new EC2 instance, you are sending a call to the AWS API. Subsequently, CloudTrail sends an event to EventBridge. Our goal is to add a tag to every new EC2 instance. Therefore, we are executing a function for every event that indicates the launch of a new EC2 instance. To trigger a Lambda function whenever such an event occurs, you need a rule. As illustrated in figure 6.18, the rule matches incoming events and routes them to a target, a Lambda function in our case.

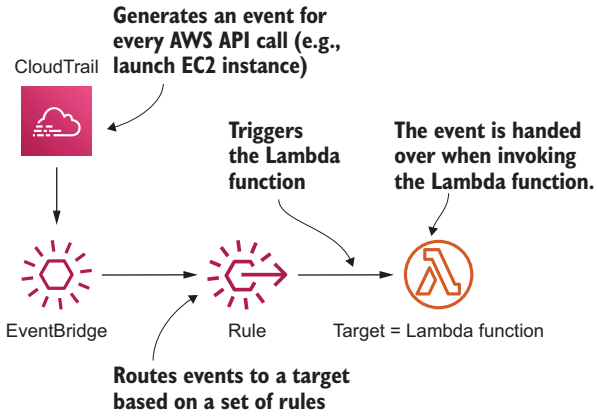


Figure 6.18 CloudTrail generates an event for every AWS API call; a rule routes the event to the Lambda function.

Listing 6.1 shows some of the event details generated by CloudTrail whenever someone launches an EC2 instance. For our case, we're interested in the following information:

- `detail-type`—The event has been created by CloudTrail.
- `source`—The EC2 service is the source of the event.
- `eventName`—The event name `RunInstances` indicates that the event was generated because of an AWS API call launching an EC2 instance.
- `userIdentity`—Who called the AWS API to launch an instance?

- `responseElements`—The response from the AWS API when launching an instance. This includes the ID of the launched EC2 instance; we will need to add a tag to the instance later.

Listing 6.1 Event generated by CloudTrail when launching an EC2 instance

```
{
  "version": "0",
  "id": "2db486ef-6775-de10-1472-ecc242928abe",
  "detail-type": "AWS API Call via CloudTrail",
  "source": "aws.ec2",
  "account": "XXXXXXXXXXXX",
  "time": "2022-02-03T11:42:25Z",
  "region": "us-east-1",
  "resources": [],
  "detail": {
    "eventVersion": "1.08",
    "userIdentity": {
      "type": "IAMUser",
      "principalId": "XXXXXXXXXXXX",
      "arn": "arn:aws:iam::XXXXXXXXXXXX:user/myuser",
      "accountId": "XXXXXXXXXXXX",
      "accessKeyId": "XXXXXXXXXXXX",
      "userName": "myuser"
    },
    "eventTime": "2022-02-03T11:42:25Z",
    "eventSource": "ec2.amazonaws.com",
    "eventName": "RunInstances",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "109.90.107.17",
    "userAgent": "aws-cli/2.4.14 Python/3.9.10 Darwin/21.2.0
      ➥ source/arm64 prompt/off command/ec2.run-instances",
    "requestParameters": {
      [...]
    },
    "responseElements": {
      "requestId": "d52b86c7-5bf8-4d19-86e8-112e59164b21",
      "reservationId": "r-08131583e8311879d",
      "ownerId": "166876438428",
      "groupSet": {},
      "instancesSet": {
        "items": [
          {
            "instanceId": "i-07a3c0d78dc1cb505",
            "imageId": "ami-01893222c83843146",
            [...]
          }
        ]
      }
    },
    "requestID": "d52b86c7-5bf8-4d19-86e8-112e59164b21",
    "eventID": "8225151b-3a9c-4275-8b37-4a317dfe9ee2",
    "readOnly": false,
    "eventType": "AwsApiCall",
```

CloudTrail generated the event.

Someone sent a call to the AWS API, affecting the EC2 service.

Information about the user who launched the EC2 instance

ID of the user who launched the EC2 instance

Event was generated because a RunInstances call (used to launch an EC2 instance) was processed by the AWS API.

Response of the AWS API when launching the instance

ID of the launched EC2 instance

```
    "managementEvent": true,
    "recipientAccountId": "XXXXXXXXXXXX",
    "eventCategory": "Management",
    "tlsDetails": {
      [...]
    }
  }
}
```

A rule consists of an event pattern for selecting events, along with a definition of one or multiple targets. The following pattern selects all events from CloudTrail generated by an AWS API call affecting the EC2 service. The pattern matches four attributes from the event described in the next listing: `source`, `detail-type`, `eventSource`, and `eventName`.

Listing 6.2 The pattern to filter events from CloudTrail

```
{
  "source": [
    "aws.ec2"
  ],
  "detail-type": [
    "AWS API Call via CloudTrail"
  ],
  "detail": {
    "eventSource": [
      "ec2.amazonaws.com"
    ],
    "eventName": [
      "RunInstances"
    ]
  }
}
```

Filters events from the EC2 service

Filters events from CloudTrail caused by AWS API calls

Filters events from the EC2 service

Filters events with event name RunInstances, which is the AWS API call to launch an EC2 instance

Defining filters on other event attributes is possible as well, in case you are planning to write another rule in the future. The rule format stays the same.

When specifying an event pattern, we typically use the following fields, which are included in every event:

- `source`—The namespace of the service that generated the event. See "Amazon Resource Names (ARNs)" at http://mng.bz/o5WD for details.
- `detail-type`—Categorizes the event in more detail.

See "EventBridge Event Patterns" at http://mng.bz/nejd for more detailed information.

You have now defined the events that will trigger your Lambda function. Next, you will implement the Lambda function.

6.3.2 Implementing the Lambda function in Python

Implementing the Lambda function to tag an EC2 instance with the owner's user name is simple. You will need to write no more than 10 lines of Python code. The programming

model for a Lambda function depends on the programming language you choose. Although we are using Python in our example, you will be able to apply what you've learned when implementing a Lambda function in C#/.NET Core, Go, Java, JavaScript/Node.js, Python, or Ruby. As shown in the next listing, your function written in Python needs to implement a well-defined structure.

Listing 6.3 Lambda function written in Python

The name of the Python function, which is referenced by the AWS Lambda as the function handler. The event parameter is used to pass the CloudWatch event, and the context parameter includes runtime information.

```
def lambda_handler(event, context):
    # Insert your code
    return
```

It is your job to implement the function.

Use return to end the function execution. It is not useful to hand over a value in this scenario, because the Lambda function is invoked asynchronously by a CloudWatch event.

> **Where is the code located?**
>
> As usual, you'll find the code in the book's code repository on GitHub: https://github.com/AWSinAction/code3. Switch to the chapter06 directory, which includes all files needed for this example.

Time to write some Python code! Listing 6.4 for `lambda_function.py` shows the function, which receives an event from CloudTrail indicating that an EC2 instance has been launched recently, and adds a tag including the name of the instance's owner. The AWS SDK for Python 3.9, named `boto3`, is provided out of the box in the Lambda runtime environment for Python 3.9. In this example, you are using the AWS SDK to create a tag for the EC2 instance `ec2.create_tags(...)`. See the Boto3 documentation at https://boto3.readthedocs.io/en/latest/index.html if you are interested in the details of `boto3`.

Listing 6.4 Lambda function adding a tag to EC2 instance

Creates an AWS SDK client for EC2

The name of the function used as entry point for the Lambda function

Logs the incoming event for debugging

Extracts the user's name from the CloudTrail event

```
import boto3
ec2 = boto3.client('ec2')

def lambda_handler(event, context):
  print(event)
  if "/" in event['detail']['userIdentity']['arn']:
      userName = event['detail']['userIdentity']['arn'].split('/')[1]
  else:
      userName = event['detail']['userIdentity']['arn']
  instanceId = event['detail']['responseElements']['instancesSet']
```

```
    ➥ ['items'][0]['instanceId']
    print("Adding owner tag " + userName + " to instance " + instanceId + ".")
    ec2.create_tags(Resources=[instanceId,],
    ➥ Tags=[{'Key': 'Owner', 'Value': userName},])
    return
```

Extracts the instance's ID from the CloudTrail event

Adds a tag to the EC2 instance using the key owner and the user's name as value

After implementing your function in Python, the next step is to deploy the Lambda function with all its dependencies.

6.3.3 Setting up a Lambda function with the Serverless Application Model (SAM)

You have probably noticed that we are huge fans of automating infrastructures with CloudFormation. Using the Management Console is a perfect way to take the first step when learning about a new service on AWS. But leveling up from manually clicking through a web interface to fully automating the deployment of your infrastructure should be your second step.

AWS released the *Serverless Application Model* (SAM) in 2016. SAM provides a framework for serverless applications, extending plain CloudFormation templates to make it easier to deploy Lambda functions. The next listing shows how to define a Lambda function using SAM and a CloudFormation template.

Listing 6.5 Defining a Lambda function with SAM within a CloudFormation template

A special resource provided by SAM allows us to define a Lambda function in a simplified way. CloudFormation will generate multiple resources out of this declaration during the transformation phase.

The CloudFormation template version, not the version of your code

Transforms are used to process your template. We're using the SAM transformation.

```
---
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: Adding an owner tag to EC2 instances automatically
Resources:
  # [...]
  EC2OwnerTagFunction:
    Type: AWS::Serverless::Function
    Properties:
      Handler: lambda_function.lambda_handler
      Runtime: python3.9
      Architectures:
      - arm64
      CodeUri: '.'
      Policies:
      - Version: '2012-10-17'
        Statement:
        - Effect: Allow
          Action: 'ec2:CreateTags'
          Resource: '*'
      Events:
```

The handler is a combination of your script's filename and Python function name.

Uses the Python 3.9 runtime environment

Choose the ARM architecture for better price performance.

The current directory will be bundled, uploaded, and deployed. You will learn more about that soon.

Authorizes the Lambda function to call other AWS services (more on that next)

The definition of the event invoking the Lambda function

```
EventBridgeRule:
  Type: EventBridgeRule
  Properties:
    Pattern:
      source:
      - 'aws.ec2'
      detail-type:
      - 'AWS API Call via CloudTrail'
      detail:
        eventSource:
        - 'ec2.amazonaws.com'
        eventName:
        - 'RunInstances'
```

Creates an EventBridge rule

Configures the filter pattern

Please note: this example uses CloudTrail, which records all the activity within your AWS account. The CloudFormation template creates a trail to store an audit log on S3. That's needed because the EventBridge rule does not work without an active trail.

6.3.4 Authorizing a Lambda function to use other AWS services with an IAM role

Lambda functions typically interact with other AWS services. For instance, they write log messages to CloudWatch allowing you to monitor and debug your Lambda function. Or they create a tag for an EC2 instance, as in the current example. Therefore, calls to the AWS APIs need to be authenticated and authorized. Figure 6.19 shows a Lambda function assuming an IAM role to be able to send authenticated and authorized requests to other AWS services.

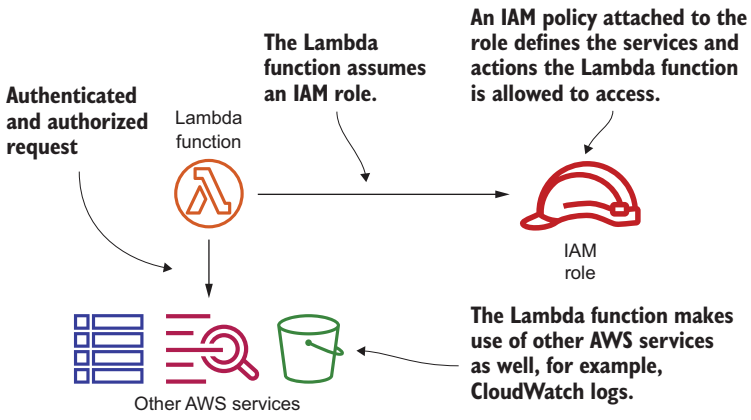


Figure 6.19 A Lambda function assumes an IAM role to authenticate and authorize requests to other AWS services.

Temporary credentials are generated based on the IAM role and injected into each invocation via environment variables (such as `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`, and `AWS_SESSION_TOKEN`). Those environment variables are used by the AWS SDK to sign requests automatically.

You should follow the least-privilege principle: your function should be allowed to access only services and actions that are needed to perform the function's task. You should specify a detailed IAM policy granting access to specific actions and resources.

Listing 6.6 shows an excerpt from the Lambda function's CloudFormation template based on SAM. When using SAM, an IAM role is created for each Lambda function by default. A managed policy that grants write access to CloudWatch logs is attached to the IAM role by default as well. Doing so allows the Lambda function to write to CloudWatch logs.

So far the Lambda function is not allowed to create a tag for the EC2 instance. You need a custom policy granting access to the `ec2:CreateTags`.

Listing 6.6 A custom policy for adding tags to EC2 instances

```
# [...]
 EC2OwnerTagFunction:
   Type: AWS::Serverless::Function
   Properties:
     Handler: lambda_function.lambda_handler
     Runtime: python3.9
     CodeUri: '.'
     Policies:
     - Version: '2012-10-17'
       Statement:
       - Effect: Allow
         Action: 'ec2:CreateTags'
         Resource: '*'
     # [...]
```

Defines a custom IAM policy that will be attached to the Lambda function's IAM role

The statement allows …

…creating tags…

…for all resources.

If you implement another Lambda function in the future, make sure you create an IAM role granting access to all the services your function needs to access (e.g., reading objects from S3, writing data to a DynamoDB database). Revisit section 5.3 if you want to recap the details of IAM.

6.3.5 Deploying a Lambda function with SAM

To deploy a Lambda function, you need to upload the deployment package to S3. The deployment package is a zip file including your code as well as additional modules. Afterward, you need to create and configure the Lambda function as well as all the dependencies (the IAM role, event rule, and so on). Using SAM in combination with the AWS CLI allows you to accomplish both tasks.

First, you need to create an S3 bucket to store your deployment packages. Use the following command, replacing `$yourname` with your name to avoid name conflicts with other readers:

```
$ aws s3 mb s3://ec2-owner-tag-$yourname
```

Execute the following command to create a deployment package and upload the package to S3. Please note: the command creates a copy of your template at output.yaml,

with a reference to the deployment package uploaded to S3. Make sure your working directory is the code directory chapter06 containing the template.yaml and lambda_function.py files:

```
$ aws cloudformation package --template-file template.yaml \
➥ --s3-bucket ec2-owner-tag-$yourname --output-template-file output.yaml
```

By typing the following command in your terminal, you are deploying the Lambda function. This results in a CloudFormation stack named `ec2-owner-tag`:

```
$ aws cloudformation deploy --stack-name ec2-owner-tag \
➥ --template-file output.yaml --capabilities CAPABILITY_IAM
```

You are a genius! Your Lambda function is up and running. Launch an EC2 instance, and you will find a tag with your username `myuser` attached after a few minutes.

Cleaning up

If you have launched an EC2 instance to test your Lambda function, don't forget to terminate the instance afterward. Otherwise, it is quite simple to delete the Lambda function and all its dependencies. Just execute the following command in your terminal. Replace `$yourname` with your name:

```
$ CURRENT_ACCOUNT=$(aws sts get-caller-identity --query Account \
➥ --output text)
$ aws s3 rm --recursive s3://ec2-owner-tag-${CURRENT_ACCOUNT}/
$ aws cloudformation delete-stack --stack-name ec2-owner-tag
$ aws s3 rb s3://ec2-owner-tag-$yourname --force
```

6.4 What else can you do with AWS Lambda?

In the last part of the chapter, we would like to share what else is possible with AWS Lambda, starting with Lambda's limitations and insights into the serverless pricing model. We will end with three use cases for serverless applications we have built for our consulting clients.

6.4.1 What are the limitations of AWS Lambda?

Executing a Lambda function cannot exceed 15 minutes. This means the problem you are solving with your function needs to be small enough to fit into the 900-second limit. It is probably not possible to download 10 GB of data from S3, process the data, and insert parts of the data into a database within a single invocation of a Lambda function. But even if your use case fits into the 900-second constraint, make sure that it will fit under all circumstances. Here's a short anecdote from one of our first serverless projects: we built a serverless application that preprocessed analytics data from news sites. The Lambda functions typically processed the data within less than 180 seconds.

But when the 2017 US elections came, the volume of the analytics data exploded in a way no one expected. Our Lambda functions were no longer able to complete within 300 seconds—which was the maximum back then. It was a show-stopper for our serverless approach.

AWS Lambda provisions and manages the resources needed to run your function. A new execution context is created in the background every time you deploy a new version of your code, go a long time without any invocations, or when the number of concurrent invocations increases. Starting a new execution context requires AWS Lambda to download your code, initialize a runtime environment, and load your code. This process is called a *cold start*. Depending on the size of your deployment package, the runtime environment, and your configuration, a cold start could take from a few milliseconds to a few seconds.

In many scenarios, the increased latency caused by a cold start is not a problem at all. The examples demonstrated in this chapter—a website health check and automated tags for EC2 instances—are not negatively affected by a cold start. To minimize cold-start times, you should keep the size of your deployment package as small as possible, provision additional memory, and use a runtime environment like JavaScript/Node.js or Python.

However, when processing real-time data or user interactions, a cold start is undesirable. For those scenarios, you could enable provisioned concurrency for a Lambda function. With provisioned concurrency, you tell AWS to keep a certain amount of execution contexts warm, even when the Lambda function is not processing any requests. As long as the provisioned concurrency exceeds the required number of execution contexts, you will not experience cold starts. The downside is you pay $0.0000041667 for every provisioned GB per second, whether or not you use the capacity. However, you will get a discount on the cost incurred for the actual term of the Lambda function.

Another limitation is the maximum amount of memory you can provision for a Lambda function: 10,240 MB. If your Lambda function uses more memory, its execution will be terminated.

It is also important to know that CPU and networking capacity are allocated to a Lambda function based on the provisioned memory as well. So, if you are running computing- or network-intensive work within a Lambda function, increasing the provisioned memory will probably improve performance.

At the same time, the default limit for the maximum size of the compressed deployment package (zip file) is 250 MB. When executing your Lambda function, you can use up to 512 MB nonpersistent disk space mounted to `/tmp`. Look at "Lambda Quotas" at http://mng.bz/vXda if you want to learn more about Lambda's limitations.

6.4.2 Effects of the serverless pricing model

When launching a virtual machine, you have to pay AWS for every operating hour, billed in second intervals. You are paying for the machines whether or not you are

using the resource they provide. Even when nobody is accessing your website or using your application, you are paying for the virtual machine.

That's totally different with AWS Lambda. Lambda is billed per request. Costs occur only when someone accesses your website or uses your application. That's a game changer, especially for applications with uneven access patterns or for applications that are used rarely. Table 6.3 explains the Lambda pricing model in detail. Please note: when creating a Lambda function, you select the architecture. As usual, the Arm architecture based on AWS Graviton is the cheaper option.

Table 6.3 AWS Lambda pricing model

	Free Tier	x86	Arm
Number of Lambda function invocations	First 1 million requests every month are free.	$0.0000002 per request	$0.0000002 per request
Duration billed in 1 ms increments based on the amount of memory you provisioned for your Lambda function	Using the equivalent of 400,000 seconds of a Lambda function with 1 GB is free of charge provisioned memory every month.	$0.0000166667 for using 1 GB for one second	$0.0000133334 for using 1 GB for one second

Free Tier for AWS Lambda

The Free Tier for AWS Lambda does not expire after 12 months. That's a huge difference compared to the Free Tier of other AWS services (such as EC2) where you are eligible for the Free Tier only within the first 12 months, after creating an AWS account.

Sounds complicated? Figure 6.20 shows an excerpt of an AWS bill. The bill is from November 2017 and belongs to an AWS account we are using to run a chatbot (see https://marbot.io). Our chatbot implementation is 100% serverless. The Lambda functions were executed 1.2 million times in November 2017, which results in a charge of $0.04. All our Lambda functions are configured to provision 1536 MB memory. In total, all our Lambda functions have been running for 216,000 seconds, or around 60 hours, in November 2017. That's still within the Free Tier of 400,000 seconds with 1 GB provisioned memory every month. So, in total we had to pay $0.04 for using AWS Lambda in November 2017, which allowed us to serve around 400 customers with our chatbot.

This is only a small piece of our AWS bill, because other services we used together with AWS Lambda—for example, to store data—add more significant costs to our bill.

Don't forget to compare costs between AWS Lambda and EC2. Especially in a high-load scenario with more than 10 million requests per day, using AWS Lambda will probably result in higher costs compared to using EC2. But comparing infrastructure costs is only one part of what you should be looking at. Consider the total cost of

▾ Lambda		$0.04
▾ US East (Northern Virginia) Region		$0.04
AWS Lambda Lambda-GB-Second		$0.00
AWS Lambda - Compute Free Tier - 400,000 GB-Seconds - US East (Northern Virginia)	331,906.500 seconds	$0.00
AWS Lambda Request		$0.04
0.0000000367 USD per AWS Lambda - Requests Free Tier - 1,000,000 Requests - US East (Northern Virginia) (blended price)*	1,209,096 Requests	$0.04

Figure 6.20 Excerpt from our AWS bill from November 2017 showing costs for AWS Lambda

ownership (TOC), including costs for managing your virtual machines, performing load and resilience tests, and automating deployments. Our experience has shown that the total cost of ownership is typically lower when running an application on AWS Lambda compared to Amazon EC2.

The last part of the chapter focuses on additional use cases for AWS Lambda besides automating operational tasks, as you have done thus far.

6.4.3 Use case: Web application

A common use case for AWS Lambda is building a backend for a web or mobile application. As illustrated in figure 6.21, an architecture for a serverless web application typically consists of the following building blocks:

- *Amazon API Gateway*—Offers a scalable and secure REST API that accepts HTTPS requests from your web application's frontend or mobile application.
- *AWS Lambda*—Lambda functions are triggered by the API gateway. Your Lambda function receives data from the request and returns the data for the response.

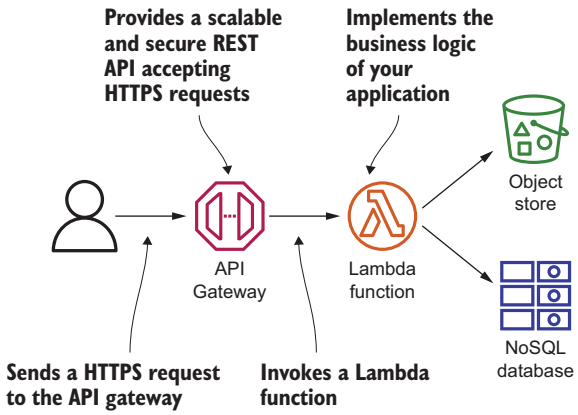


Figure 6.21 A web application build with API Gateway and Lambda

- *Object store and NoSQL database*—For storing and querying data, your Lambda functions typically use additional services offering object storage or NoSQL databases, for example.

Do you want to get started building web applications based on AWS Lambda? We recommend *AWS Lambda in Action* from Danilo Poccia (Manning, 2016).

6.4.4 Use case: Data processing

Another popular use case for AWS Lambda follows: event-driven data processing. Whenever new data is available, an event is generated. The event triggers the data processing needed to extract or transform the data. Figure 6.22 shows an example:

1. The load balancer collects access logs and uploads them to an object store periodically.
2. Whenever an object is created or modified, the object store triggers a Lambda function automatically.
3. The Lambda function downloads the file, including the access logs, from the object store and sends the data to an Elasticsearch database to be available for analytics.

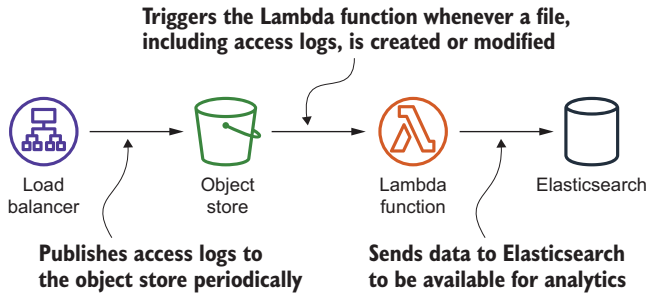


Figure 6.22 Processing access logs from a load balancer with AWS Lambda

We have successfully implemented this scenario in various projects. Keep in mind the maximum execution limit of 900 seconds when implementing data-processing jobs with AWS Lambda.

6.4.5 Use case: IoT backend

The AWS IoT service provides building blocks needed to communicate with various devices (things) and build event-driven applications. Figure 6.23 shows an example. Each thing publishes sensor data to a message broker. A rule filters the relevant messages and triggers a Lambda function. The Lambda function processes the event and decides what steps are needed based on the business logic you provide.

We built a proof of concept for collecting sensor data and publishing metrics to a dashboard with AWS IoT and AWS Lambda, for example.

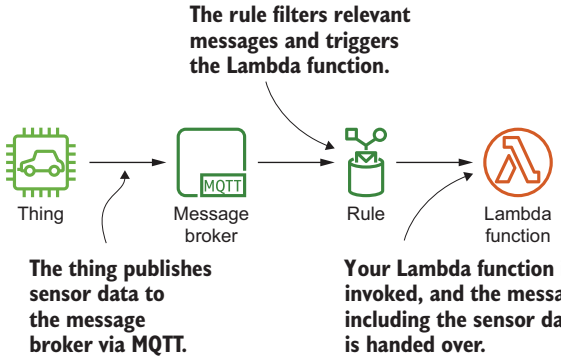


Figure 6.23 Processing events from an IoT device with AWS Lambda

We have gone through three possible use cases for AWS Lambda, but we haven't covered all of them. AWS Lambda is integrated with many other services as well. If you want to learn more about AWS Lambda, we recommend the following books:

- *AWS Lambda in Action* by Danilo Poccia (Manning, 2016) is an example-driven tutorial that teaches how to build applications using an event-based approach on the backend.
- *Serverless Architectures on AWS*, second edition, by Peter Sbarski (Manning, 2022) teaches how to build, secure, and manage serverless architectures that can power the most demanding web and mobile apps.

Summary

- AWS Lambda allows you to run your C#/.NET Core, Go, Java, JavaScript/Node.js, Python and Ruby code within a fully managed, highly available, and scalable environment.
- The Management Console and blueprints offered by AWS help you to get started quickly.
- By using a schedule expression, you can trigger a Lambda function periodically. This is comparable to triggering a script with the help of a cron job.
- CloudWatch is the place to go when it comes to monitoring and debugging your Lambda functions.
- The Serverless Application Model (SAM) enables you to deploy a Lambda function in an automated way with AWS CloudFormation.
- Many event sources exist for using Lambda functions in an event-driven way. For example, you can subscribe to events triggered by CloudTrail for every request you send to the AWS API.
- The most important limitation of a Lambda function is the maximum duration of 900 seconds per invocation.
- AWS Lambda can be used to built complex services as well, from typical web applications, to data analytics, and IoT backends.